Fail-Fast Feedback

Filip Lindqvist (ada10fl2@student.lu.se)
Jakob Berglund (ada10jbe@student.lu.se)
Lund Institute of Technology, Sweden

January 2014

Keywords: Feedback, Courage, Mistakes, Learning, Fail-Fast, Coaching

Abstract

This report is about encroaching mistakes in a mission to learn. By failing fast we can learn quickly, to correct and improve our practises. By proving valuable feedback in a student project, students learn to fail fast and learn from this. However, in the end it turns out that the project is to small to apply all of these paradigms, since it is already an extensive course in learning the eXtreme programming paradigms.

Introduction

Many say that a fear of making mistakes is the biggest mistake of all [7]. This is the problem that this project will focus on. The hypothesis is that if the PVG students are allowed to make many small mistakes and get feedback as soon as they make them they will avoid making large mistakes. In the end the goal is to give the students the courage to make mistakes and have the knowledge that it is okay to make them. Also that when the mistakes happens is not a time for anger or embarrassment but time for celebration over the fact that they found it early and that they now have learnt something new.

The core concept is to allow mistakes and understanding the power of reflecting upon them. Because making mistakes is only useful if one reflect and think about how to avoid to make the same mistake again. As John C. Maxwell said in the book "The Power of leadership"

> A man must be big enough to admit his mistakes, smart enough to profit from them and strong enough to correct them [4]

This is very true in this project since it is not a problem that someone is making a mistake the problem is if someone is hiding the mistake or if they do not learn from their mistakes.

Background

This project is carried out on a team programming course taken by students of the second year. They are put in teams of 8-10 student and practise agile XP-methods during eight-hour iterations each week during seven weeks. Each team is equipped with two coaches, that leads the team through the XP-practises. As team coaches it is possible to practice skills in coaching and also preform a in-depth project.

The Courses

The student course and coaching course is taut at *Lund Institute of Technology, Sweden* and the concept of the courses are described in *Teaching eXtreme Programming to large groups of students* [5].

The primary goal of these courses is to let the students learn the XP-programming paradigms and let the coaches learn to be a coach. Both courses are grade-less and only requires the attendance of the students. This opens up the chances of successfully learning the subject by making room for mistakes, which allows experiments with the objective to fail, reflect and learn.

> "Learning is the process whereby knowledge is created through the transformation of experience" [6]

Learning

One core principles of the PVG-project is to embrace the experimental learning cycle that was drafted by Kolb in 1984 [6]. One of the important principles in this paradigm is that learning is a continuous process. It also states that our knowledge is derived from our experience within the subject, not by simply reading literature. In short, learning by doing and reflecting upon mistakes and successful progress. Kolbs cycle starts with *concrete experience* were we are faced with a task which requires active involvement in the subject. We are required to interact, because it is hard to learn with passive involvement by simply reading about it. The next stage, *reflective observation*, requires a timeout from the active involvement, which requires to take a step back to reflect and verbalise the experience. It is a time of quiet thinking and completion of logs or diaries. Next we enter the *abstract conceptualisation*, where the toughs and diaries and being interpreted. It is a process of understanding what happened during the active involvement and figuring out the relation between the pasted events. The learner reflects and conceptualise a theory from this understanding, that can be applied in the next step. Finally the cycle reaches *active experimentation*, where the learner put the understanding and theory in practise. The learnt things are put into practise and this generates new experience which then bring the learner into cycle again.

The core of the PVG-project is that this learning cycle is repeated six times during seven weeks. During the eight hour iterations, the student is getting concrete experience from practising the XP-paradigms, getting introduced to new paradigms every week gradually during the total of seven weeks. After each iteration each student do a personal reflection upon the iteration and this forces the student to do a reflective observation. Later in the week, the first half of the planning meeting is spent on doing a reflection. Here it is very important as coaches, to align the discussions to make it arrive to an abstract conceptualisation, that is valid and can be tested in the following iteration. The reflection is summarised into a group reflection, but the most important artifact from this refection is the students personal understanding. The second half of the planning meeting is used to estimate and plan the following iteration. During the following iteration the student can use this understanding to improve the work and gain new experience. The repetition of this learning cycle is the key to learning student fast and efficient.

Feedback

Providing feedback in a good way is not always easy. There is however some guidelines for how to provide quality feedback [8]. We tried to use these whenever we gave the students feedback on their work. The most important and the thing that is at the essence of this project, is to provide feedback in a positive way, both positive feedback and negative. If the students feel that they get punished or get an unpleasant feeling when they get feedback on their mistakes then they might hide the mistakes. This would only make the mistake larger

and more problematic since it is simple to solve a mistake in the beginning but harder and more expensive the longer time it takes. Therefore if the students feel safe and believe that it is positive when they get feedback they will not hide or get uncomfortable when they make mistakes. Both mistakes and successes should be celebrated since both imply that the team or student have learnt something.

When giving negative feedback it is important to make it clear that it is not the person that is wrong or is a failure but the actions that where made. And try to learn something from the actions.

When giving feedback to students it is important to be clear about what was wrong. It is however it is also important to perhaps not always give all the answers and tell the person what was wrong but create a discussion about the problem and allow the student to think for them self. This is the moment people learn when they are not told what is right or wrong but when they figure out by them self what is.

It is also important to give feedback as soon as possible. This is important in this project since it is all about fast failures. If the students do not get feedback about what they do correctly and what they do incorrectly within minutes the students will forget what they did. This is how the brain works it prioritise special occasions and save these for later. This is the same thing as Pavlov's Dogs[9] if the dogs did not get the food directly they would never have created the connection between the bell and the treat. Just as the students brain will not connect the mistake/success with the celebration or the reflection.

Method

Several different approaches was used to teach the PVG students that making mistakes is good and should be a natural part of software development. Some parts are already built in to XP that all students uses during the PVG project, like Pair programming and Test driven development.

Pair Programming

Because they work in pairs the students will always have someone that will inspect the code that they write [10]. Because of this, they will get instant feedback as soon as they do something wrong. This eliminates both small errors like spelling or forgetting a comma. In addition to larger structural errors, since the students have someone to talk to and discuss design with. Since the students in this project are mostly second year students they are not all that familiar with programming and because of this, the fact that they are always a pair should improve the final code. However, there are problems with this method since it can produce more stress than what would be the case if they where working by them self. Especially students with less programming experience can feel that they are stressed when they are working with a better student. This should not be a problem in this course since there is plenty of time and the students are

supposed to learn new things and challenge their comfort zones.

Test Driven Development

Test driven development (TDD) [3] is one of the hardest parts of the course to learn and will be hard for the students to get familiar with. It is also hard for the students to understand the benefits. However there are several benefits to doing TDD from a feedback view point. Because if the test coverage is high and the tests are well written. When the students do the implementation they will already have thought about the design and will see the progress of the implementation since more and more tests will succeed. This progress will give them feedback about the state of their code and if the implementation is correct or not. Later a different team will be able to see if the implementation is correct by only looking at the test. Since test normally is easier to understand than code it will be a lot faster to just inspect the test.

Atlassian Clover

Clover [2] was used as an test coverage tool. Every time the team did a commit that commit resulted in a test coverage report. The students and coaches where then able to study the report and quickly find problematic areas that was untested. It was also possible to check if it was useful to test a specific method or if it was better to let it be. Since high test coverage will not give the complete story. Making useless tests only to get high test coverage is just as bad as not making any tests at all. Because in best case these tests will only take space and make it harder to find things in the test classes. However in worst case they will fail because something is changed in the code. Then it will only take time to debug and find the error witch have cost that the customer will not want to pay. A useless test would for example be to test an unused method from an interface. This method would lower the test coverage if it is not tested but there is no point in testing it.

SonarQube

Sonar[11] is a code complexity and static code analysis tool. It checks the code for common problems that might create issues in the future. However most of the problems it discovered was small problems like not following the Java conventions. It is possible to remove errors that is not important in the project and thereby filter out the real problems that will impact the code. It also give some statistics over how many lines of code that has been produced and similar metrics. All this information give the team feedback in different ways. The issues the software finds helps the developers to reduce small errors that can be problematic in the future. One of the most useful issues found is when the software throws exceptions but do not give the user any feedback of the error. When the customer later points out that errors need to be logged. Then a list of places to do the implementation is done and ready to be used this reduces the time it takes until the software is fixed.

Atlassian Bamboo

Both Sonar and Clover would not be useful in the context of this report without a build system like Bamboo [1]. When source code is committed to the repository Bamboo will notice this and make an update and compile of the software. When the build is done it will also run all tests and give the results of the tests. If any test fail it will clearly show this failure. During the labs this information was the mostly used by the students since they quickly understood that just because the test works on their compute it doesn't mean that they will work on every computer.

As soon as the students discovered this they quickly became interested in the results of the build. Since the build was done in under a minute most students looked at screen that displayed the build results until the results were showed on the screen. Thereby they quickly got feedback on potential problems within their code and tests.

Bamboo also helped in the later releases with producing a release on every commit. This reduced some of the stress within the team but it also meant that the students for the second release thought that they were able to add more in the last minutes before the release than what was really possible. This was not ideal.

Coaches

Since the students in the project are both inexperienced programmers and inexperienced members of a team. The coaches have an important role [5]. In the beginning of the project the coaches role is to structure and lead the students using an instructing leadership. However as the project progresses the students are given more responsibility. Therefor it is important that the coaches teach the students to take the correct path and to quickly give the students feedback when they are on the wrong path. However, it is also important to allow the students to make errors because that is how one learns the fastest. All these factors make the coaches job a balancing act. In the end it is important to allow the students to find their own answers. As the famous saying "Give a man a fish and you feed him for a day; teach a man to fish and you feed him for a lifetime" this is also true for the students. If the coaches give the student the answer the student will solve his current problem but if the coaches teaches the student to find answers the student will never have to ask again. Therefor it is important to create an environment for the students that inspires them to think for them self.

Observation will be used to collect information about how well the different method work. Observations will be made both during the labs and during the planing meetings.

Results

The observations will be written in a weekly diary form. Each week consists of observations from both the lab and the planing meetings. They will also contain some initial analysis from the observations.

Observations

Week 1

The first iteration was full of confusion and but the students quickly learnt that communication is very important. Both within each programming pair and between the pairs. They were also quickly introduced to the build system Bamboo and test analysis tool Clover. However there was no focus on the tools at this lab since the team first needs to make some of the social mistakes like communication before they can start using the tools.

The lack initial communication between the teams lead to some misunderstanding of the stories. The small parts produced by the first initial tasks was not integrated into something useful. By the end of the iteration it was clear to everyone that the communication between the pairs had to be improved, in other words they had learnt from their mistakes.

Week 2

Since the focus of this iteration was all about Test Driven Development the students were shown a screen that showed the test status of the project. In the beginning the students made some mistakes and the repository was red for almost an hour. However after this had been pointed out to the students they learnt that they need to check the screen since what works on their computer might not work on all computers. An interesting phenomenon was then observed when the students stopped and looked at the screen an waited for the results of the automatic tests to complete before they continued with the next story.

Week 3

This week the team started to appreciate the feedback that the build system was giving by running the tests in a separate environment. After every commit the authoring pair waited a minute to see the build come out clean on the dashboard screen. This gave them confidence and courage that the tests and all dependencies were correctly submitted to the versioning system. One problem that occurred during the second half of the iteration was that the tests became unstable and altered between red and green. This forced the team to rerun each build several times to make sure that is was green. This lowered the courage and the team could no longer rely on this feedback. The problem originated from a faulty test that was not using atomic timestamps to compare the outcome of a test. The assertion timestamp was taken when the test suite was initiated and the result was asserted in the end of a test case. The solution was to delegate

one pair to track down the test and repair it. After a quick introduction to the test statistics in Bamboo, the pair narrowed the fault down to a single test case. Which was repaired, to make the timestamp just before the assertion instead. After this fix the feedback was restored and the team quickly regained the confidence in the tests.

During this iteration a lot of stories was introduced concerning the user interface of the program. These were not completely defined, with purpose to force the team to communicate with the customer. However the team was not ready to communicate with customer, which resulted in a complete story being implemented the "'wrong"' way. We as coaches noticed that the pair implementing this story was not embracing the XP-paradigm "Stories are promise of conversation". Since one of our project goals is to embrace "'Learning by failure"' we let the pair continue with this and release the solution to the customer. After the release the customer feedback was mainly concerning the details in the user interface and most of the implementation had to be rewritten. However in the personal reflection the pair members reflected over the fact that the communication with the customer had to be increased and that stories should be discussed with the customer before they were released. Our goal as coaches were achieved.

Week 4

During this iteration the communication with the customer was a continuous process throughout the day, with a lot of questions concerning the user interface. Since the team concluded that the graphical interface is very important to the costumer, and that stories was a promise for conversation. The team even convinced the customer that their solution was easier and cheaper than his original design. This is one of the important strengths of XP, having a on-site consumer to get feedback from during the iteration. Unfortunately this is not always the case in the reality. To get this feedback a team might have to conduct user testing or contact a distant costumer.

The iteration had a heavy focus on delivering stories. As the release passed, the feedback from Clover and Sonar showed that the technical depth and complexity had increased noticeable. In the personal reflections some admitted that the code had become more complex and that it was hard to implement changes. In the group reflection we asked the team what could be done to avoid this in the future and how we could simplify the existing code. One of the suggestions resulted in a spike targeting a checklist of refactoring tasks.

Week 5

This iteration the students were told that they will need to handle all of the planing in the iteration them self. They were told that the release should happen at 3pm and that they should send the release to the customer and a different team. After this they hold a quick meeting to plan the iteration and when they should have an release ready. During this meeting the students planed a new

meeting at 2pm to decide what the release should contain.

The students made a common error when it comes to planning which is that they thought it was better to insert unfinished and untested stories to make the customer happy. This error was made to a large degree because the students get stressed when they are close to a release and think that they should add things just because they like to have something to show for the customer. They are never told that the customer rather like a high quality release than more features. Since the students are not thought what the goal they should have with the releases they believe that features are the most important.

On the planing meeting the release was discussed and the students agreed that it was stupid to insert unfinished functionality. For the last iteration they will focus on quality and make sure that the program works flawlessly.

Week 6

This iteration one voluntary student was picked to be chairman at the morning meeting. The team discussed what had to be done during the iteration, in order to improve upon the feedback from the customer and the reviewing team. No new features was implemented and most of the day was spent on correcting the program and the documentation. Problems with file paths using different operating systems lowered the team morality during morning. Afterwards we organised a lightweight retrospective, which covered the different iterations and the focuses that they targeted. The goal from us as coaches was to give the students an understanding of why we use the different practises and how they apply in the real world. It is hard to learn without an understanding of why we apply certain practises. For each of theses focuses we coached the team discussion, to navigate down to the core of why we use the XP practises.

Analysis

From the diary we see that the confidence grew in the group for every iteration. As coaches we also allowed them to take more responsibility. In the beginning the students were inexperienced and did not really know how to work in a team. However, already after the first iteration they learnt their first valuable lesson. Communication is very important not only in every pair but also between different pairs. This lesson impacted their thinking throwout the course. Everybody felt that it was critical that the communication went perfect. Of course the students was not able to create a situation were everybody knew everything and where there were no communication problems. This was especially noticeable when approaching a release since the stress level grew and some pairs felt that they needed to get one more thing done even if they sacrificed quality.

The third and fourth iteration taught the group to write tests and make automatic builds and releases. This posed two challenges, both learning to embrace test driven development and learning the tools to do automatic builds. Test driven development is a real challenge to teach and learn during the few

weeks. Learning to design and structure code by writing the test first is hard and take years to grasp. The build tool ANT itself is very unfamiliar to the students and it took them quite a lot of time to grasp why and how the tool is used. Furthermore the releases fine-tune the release process, making the student aware of the fact that they can not put in stories at the last minutes. It also introduces a lot of problems with maintaining the repository in a clean state.

At the two last iterations the students were more or less self going. The students held the last stand-up meeting by them self and were not told what to do. They then went trow the spikes and also planed the iteration. This was valuable for the students since they were allowed to think by them self. While doing the retrospective at the end of the day one of the students also said that the most fun part of the PVG project was that they were allowed to think by them self. This is something that is often missed in education today. Most things that is taught at Universities are not designed to allow students to think by them self and do research without knowing that there is a nice answer. Even if these skills are the most important skills a student could have when starting work in the real world.

The most used tool that we provided the students was Bamboo first as a clean test environment. They quickly understood that just because tests worked on their computers did not mean that they will work on every computer. This feedback forced them to think different when they wrote their tests. They had to make them robust enough so that they would work on every computer but they still needed to be a good test. This was clearly one of the most appreciated tools. Both proving a quality indicator of the repository test state and providing complete releases with every commit being made.

Clover however they started to use a bit later mostly because they did not have any control of what code was tested and what code was not tested. With clover they easily found where they had missed to add tests. This is clearly illustrated in clover. The students appreciated the fact that the tool was web based and not running on their computer mostly since they always got the same results but also since they knew that they always had a report to look at. The students did learn that it is hard to get perfect code coverage not only because some things are hard to test but also because even small changes to the code could impact a lot of tests. This was one of the things that the students felt was the biggest problem with TDD. Because even if they did follow TDD and write all the tests they often ended up having to rewrite or remove tests to be able to commit.

It was during the last iteration that the students were really introduced to Sonar. They then used Sonar to find problem in the code. They looked at the list of issues Sonar provided them and tried to fix these. After this exercise the students felt that some of the issues Sonar had found were a bit stupid and did not create any real problems in the code. This is an issue because if the students spend a long time fixing non issues just to make Sonar happy then the moral in the group would decrease. Therefor it is important that every tool that is added to a development process do not just create extra work for the developer but also really help the developers in their work. In Sonar it is possible to remove

issues that the team feels are non issues. This is important to do because it keeps the moral up.

Conclusion

One of our core goals in this practice project was to give the student feedback on their work. Both fast and wide feedback on the project outcome. We also tried to encourage retrospection and learning throughout the different parts. The students learnt a lot throwout the project. They went from not having an image of how real software is developed to being familiar with both the concepts of XP and Continues Integration (CI). They have not learnt the tools; Bamboo, Clover or Sonar, but they have learnt why tools like this can and are useful in real development. They also learnt that a good idea is not good until they get feedback from the customer. This they learnt several times when developing the GUI since they in the beginning did not care about asking the customer however in the end as soon as they liked to change something they went and got feedback from the customer.

In the end is this project a bit to small and intense for the students to really learn and use all the tools they had at their disposal. In a larger project with more time it would have been possible to drive the Fail Fast Feedback concept even harder. However in the available time it was not possible to get enough focus from the students since they were stressed about different stories that the customer had given them.

It is also important not just to add new tools to a development process because if the developers do not understand how the tool would be useful they will not use tool or even worse the tool might decrease the moral in the team. This is something that could not only hurt the team on the current iteration it could hurt for a long time. Therefor a good rule is to never add a tool if it is not seen as usable by the developers.

References

[1] Atlassian. Bamboo: Continuous integration & build server, 2014.

[2] Atlassian. Clover: Java code coverage, 2014.

[3] Kent Beck. *Test-driven Development: By Example*. 2003.

[4] Joel Garfinkle. *Ten Ways to Provide Quality Feedback*. 2005.

[5] Boris Magnusson Görel Hedin, Lars bendix. *Teaching eXtreme Programming to large groups*. 2003.

[6] D.A. Kolb. *Experiential learning: experience as the source of learning and development*. 1984.

[7] D.A. Kolb. *Code Complete*. 2004.

[8] John C. Maxwell. *The Power Of Leadership*. 2001.

[9] Saul McLeod. Pavlov's dogs, 2013.

[10] Johan Rix Mia Nyström, Karin Wanhainen. *En studie om parprogrammering i praktiken*. 2002.

[11] Sonarqube.org. Sonarcube, 2014.