Coaching an inexperienced agile team towards a continuous delivery methodology

Erik Gärtner, Malcolm Eriksson
{ine11ega, dat12mer}@student.lu.se

Mars 2015

Abstract

This study postulates that coaching an inexperienced agile team towards using a continuous delivery methodology is viable and preferred to a preconfigured setup. In the end the study concludes that this is indeed the case.

1 Introduction

The following report will explore the topic of teaching an inexperienced student team how to work agile and use Continuous Delivery/Deployment (CoDe). The main purpose was to study whether it was possible to coach an inexperienced agile team towards using Continuous Delivery during a very short project.

The background section below will introduce the reader to CoDe, agile development and why this is so relevant. Then the project section will describe the project and the course in greater detail. The method section will comment on how we tried to verify our hypothesis and research question presented in the thesis section. The result section and discussion will sum up what conclusions can be drawn from our work and how this study could be improved.

2 Research questions & hypothesis

We gave the team the added task to *perform a release after every complete story.* According to Poppendieck [5] teaching through iterative feedback loops increases learning by adjusting the focus. With this in mind we felt that the best way to teach the team about a CoDe system was to coach them through the setup of their own build system. Here arises a few interesting questions: how advanced system does a small inexperienced team need? When does the effort of setting up the system outweigh the gains? Does this approach further learning?

Our hypothesis was that: with modern continuous deployment systems it is possible to coach an inexperienced team towards a continuous release process. Furthermore this is preferable to giving them a preconfigured system since the team gains better insight into the process and appreciates the methodology more having worked without it.

3 Background

Before we discuss our research further let us quickly elaborate upon some important software development concepts that this study relates to.

3.1 eXtreme Programming

eXtremere Programming (XP) is a highly iterative agile development methodology[3]. The main objective of XP is to create high quality software that is susceptible to customer changes during the development process.

XP has twelve principles and five values that guides the development process that focuses on feature implementation in called stories. If should be possible to make a release after each story since they tend to be self-contained. The customer is part of the process and through the "planning game" prioritizes amongst stories and answers questions and gives feedback on the progress.

3.2 Continuous Delivery

For most projects the release process can be very complex and arduous. The developers need to manually test the software, compile and assemble all artifacts of the release. The purpose of the Continuous Delivery is to automate the build process through automated tests, automated build scripts and a well-defined release process.

By always running a battery of automated tests the number of bugs can be reduced providing they test the program properly. Automated tests also gives the developers instant feedback regarding the state of the code repository. Another benefit of using CoDe is that release the software is as easy as sending the latest compiled artifact, allow for quick releases upon request of the customer. Lastly it greatly decreases human errors by making the build process well-defined.

4 The project

The project was done as an assignment in the course two courses EDA260 and EDA270 provided at Lunds University. Specifics of the setup and the potential benefits have been evaluated by Hedin et al. [4]. This report is from previous iterations of the course, some details might vary slightly.

The assignment was to develop a combination of tools used to take times for different kinds of race events and later sort them by varying specifications. All the work was done during 6 iterations which consists of a 2 hour planning meeting followed by a work day where the team works on the code base. No work was allowed on the code base except during this day. Each iteration also had a pre-defined focus relating to working method.

The team also had a introduction meeting which was used to explain the scope of the project as well as allow the team to start getting to know each other.

The team worked in an agile sense more specifically an eXtreme Programming sense. The definition of functionality was provided primarily in the form of pre-defined stories from a customer. Sometimes the stories had to be redefined or new stories to be created/designed, this was done in co-operation between the customer and the developers with minimal guidance from the coaches. Due to this in-depth study the task of delivering a functional release was added to each story.

4.1 Planning meeting

During the planning meetings the team evaluated and estimated stories which then was prioritized by the customer according to expected performance of the team. The coaches maintained a passive role only making sure that the team didn't forget important points that was brought up during the evaluation and discussion of the stories since the customer was not present at that time.

During the planning meetings the team also evaluated the previous iterations workday performance in a general sense but primarily considering the iterations focus.

4.2 Workdays

The team worked on the code base once per week on the designated workdays. During these days they worked from 8.15 - 17.00 excluding a one hour lunch break, any natural breaks (e.g. coffee, toilet) and a fika-break. The workdays started with a discussion of the current situation of the project and a brief presentation by every pair of their spike. When the team needed to discuss either parts of the project or aspects relating to their working methods a stand-up meeting was called. Primarily these meetings were requested by the developers but at times the coaches requested them as well. A lot of reflection on working method was also done during the more casual fika-break which allowed the developers to spend some time to reflect on the day's problems and possible solution while the related experiences were fresh in their minds.

During a few of these days a release was expected to be delivered to and tested by the customer with the assistance of two developers.

4.3 The team

In this section we will give a brief summary of the different members of the team to give a background for possible social and professional experiences that might affect the project. The team consist of 11 developers two coaches and one (pretend) customer.

Both of the coaches are currently enrolled in a course EDA270 at LTH aimed at teaching the concepts of coaching developers. Both of the coaches knew each other from before but had not worked on any projects together previously. They also had experience on various development projects of varying size as well as experience from different projects with a more social focus.

The developers consisted of 11 students currently enrolled in a course called EDA260 at LTH with the goal of teacing agile programming and in specific XP-programming. Amongst the developers only one of them had previous professional experince, while the rest only had experince from a variety of university courses and a few private projects.

Some of the developers and coaches knew each other from before the course. This of course affects the team building phase of the project.

The customer had experience as a customer from previous iterations of the project. The customer also possessed more technical knowledge than most likely can be expected in a real situation, sometimes that expertise might have affected the decisions of the customer.

5 Methods used

During the course we've used different methods to guide the team towards different technical tools and methodology that assisted in setting up the development environment used as well as the continuous delivery setup.

5.1 Spikes

One of the main methods we used to guide the team towards suitable research was in the shape of spikes. This was minor tasks timeboxed for roughly 2-4 hours of work time in pairs. The spike tasks was primarily designed by the coaches with the influence of suggestions from the team.

The spikes were primarily focused around aspects such as working methods, architectural designs, general programming methods and the design and implementation of the continuous delivery system used. Every week a baking-spike was also assigned with the intent of team building and maintaining morale in the team.

5.1.1 Wiki

One of the sub-tasks involved in every spike (excluding the baking-spike) was to write a short (0.5-1 page) summary of the spike, the format of the wiki posts was not pre-defined due to the need to vary the format to fit the specific spike.

All the developers were also tasked with reading through the new wiki-inputs every week to make sure that information spread through the team efficiently.

5.1.2 Important spikes

Amongst the spike assignments a few were more important (due to our study) since they were aimed at teaching the concepts of CoDe and building different parts of the CoDe system.

The basic spike setup used for the different areas of the system were the same. For each area regarding the delivery first a spike were handed out with the intent to research the possibilities of benefit from tools in said area, what options were available as well as a personal recommendation of a tool. This information was then discussed briefly with the team in order to chose what tool and features we as a team would pursuit during the next spike.

After this a spike with an initial setup was designed for the team primarily by the coaches but with input of the developers on the specifics of feature implementation. After a tool was initially setup the team continuously evaluated the current setup to establish any part of the current setup should be modified or if new features could be added with tools already set up.

If such an aspect was identified a maintenance spike was designed to improve / maintain the current setup.

The primary areas relating to the delivery system that was spiked on was control of commits and push (through the use of *git hooks*), automation of the build process (through the use of *ant-script*) and an automation of the build process (through the use of *Codeship*). The final setup of these tools will be further explored in the results section.

5.2 Stand-up meetings

During the course we used short informal stand-up meetings as needed to discuss and evaluate the current situation of the project. The subject of the meetings varied between things such as system architecture, working practices and questions of functionality especially what aspects to bring to the customer.

We also had two somewhat formal meetings inspired by the same model. The first was a stand-up meeting at the start of the day. During this meeting we had a somewhat casual start where we checked how people were doing and then made everyone make a short presentation of their spike-task. If the task was more practical than theoretical the presentation consisted of a summary of what they had produced. We ended this meeting with a deeper architectural discussion those days when it was needed (e.g. first meeting or if the team had identified a larger refactoring). During these meetings we also introduced the focus of the week except for a few times when the coaches forgot.

The second meeting that was standardized was the post release delivery fika-meeting which was slightly longer. During these meetings the team reflected on the days work with focus on working practices and the focus of the week. During these more casual meetings we usually got a lot of suggestions and insight of problematic work practices that needed to be changed or improved.

6 Results

In the end we managed to coach the team towards setting up a continuous delivery system within the time frame with the added benefit of deeper understanding of the concept.

6.1 System setup

The first part of their setup was an Ant-script[6] that compiled the program into multiple executables. They later improved upon this script to run all JUnit-tests and copy and compress the documentation along with the executables into a single archive file.

When the team had a working build script they started working on a continuous integration and delivery setup. After researching the alternatives they settled on Codeship[7]. It is an online web service that allows for both continuous integration and delivery/deployment. The first setup used Codeship to run all tests each time a commit was pushed to the repository. If the build failed the team members would receive an email.

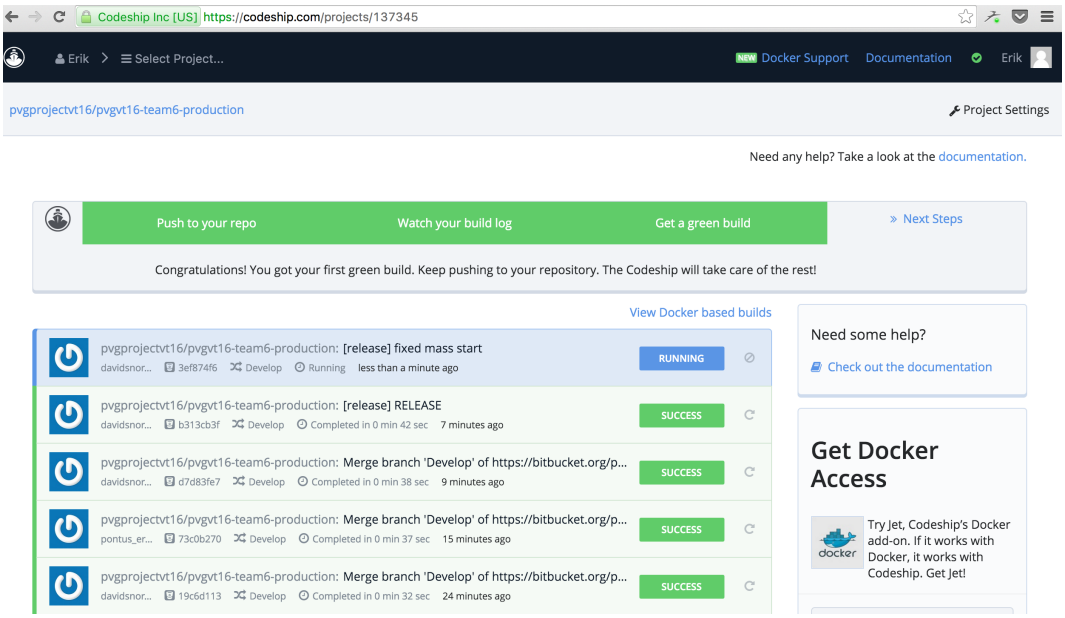

Figure 1: The main interface of Codeship. It shows the build status for each commit.

However the team still had a problem with members forgetting to run their tests before committing and the repository would sometimes contain broken code. To counteract this they decided to install git hooks (scripts that run on certain commands) that ran all tests and only allowed committing if all tests passed.

Their repository was now always green but they frequently had half-implemented stories in the master branch and merge conflicts was very common.

So the team adopted a more advanced git branching strategy. The master branch contained the latest stable release, develop contained the latest work and they had branches for each story. When a story was completed it was merged into develop.

This new branching strategy was complemented with an updated configuration on Codeship. Now when develop was pushed Codeship would run all tests and if they passed it would merge develop into master. Then it would build a release that was archived and uploaded to a release

repository that contained all releases. This way they guaranteed that the master branch passed all tested and they could always download the latest release from their release repository.

6.2 System knowledge

To measure the result regarding the team's understanding of a CoDe methodology and their system we sent out two surveys. One to our team and another to three other teams that used Jenkins. See the appendix for the raw data and the exact question formulation in the survey.

The main question of the survey was to explain why teams use continuous integration. This question was in both surveys. From the correctness of their answers we rated their understanding from lacking to deep.

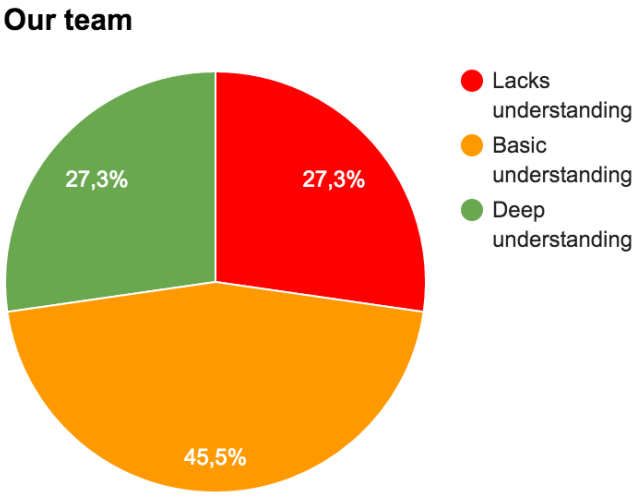

Figure 2: Our teams understanding of why continuous integration is used.

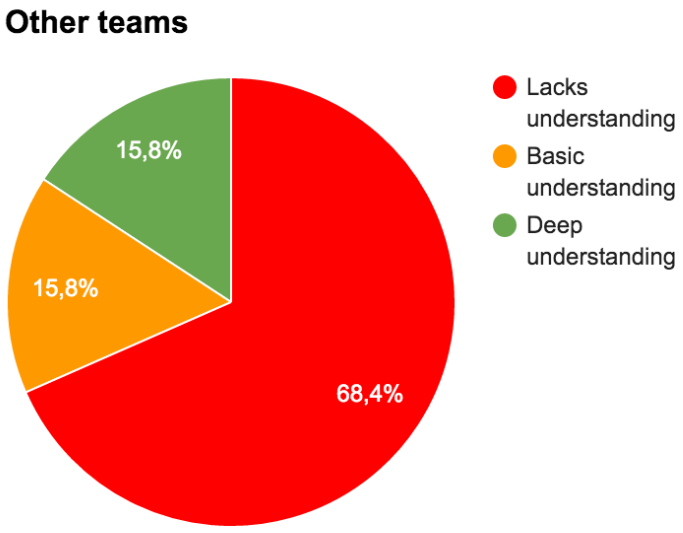

Figure 3: The other teams understanding of why continuous integration is used.

We also asked the other teams that hadn't setup their systems whether they thought it would be possible for them to setup their system themselves within the scope of the course.

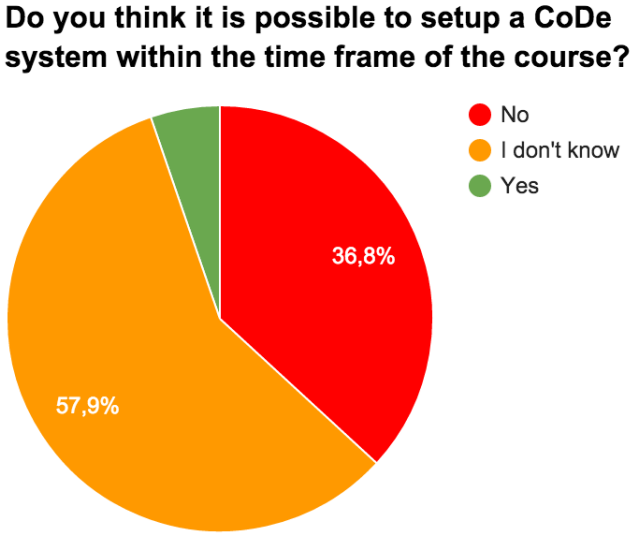

Figure 4: The other teams answers regarding if they thought it possible to setup their system themselves with in the scope of the course.

7 Discussion

The data collected through the surveys clearly speaks in favor of our hypothesis. It is both possible and advantageous to coach the team towards setting up the release environment on their own. The number of students having a deep or basic understanding of the system was around 70% compared to around 32%. This in it self is not as surprising as the fact that they managed to learn continuous integration and setup the system within in the time frame of the course. Our survey shows that the team that less than 10% of the student not in our team thought it was possible to setup the release system by themselves. This is indicative of two things: the other teams didn't fully understand the system, otherwise they would have realized that it is very possible. Secondly: to an inexperienced team setting up a continuous integration system seems like a daunting task.

By coaching the team towards setting up the system they gained deeper knowledge and by saying that it is possible we dispelled the fear that the task was impossible. It didn't matter that we ourselves didn't know if they would be able to setup the system, by authoritatively saying it was possible gave the team the necessary courage.

Another strong indication that our team truly knew how to configure the system was that during the last iteration they had to reconfigure Codeship from the ground up due to problems that forced us to create a new account. Had they not understood how it worked they wouldn't have able to configure the system in 30 minutes.

Their end system was not as advanced as a fully configured Jenkins build server. It didn't for example test for code coverage though it was theoretically possible to extend the system the team found a middle ground where maintenance and setup was well worth the effort. The opted for a branching model that used feature/story-branches, a stable main branch and a development branch. This is also not the most advanced branching strategy, they had for example no release branches but again they found a middle ground that suited the team.

7.1 Product quality

Due to low code coverage and malformed tests the team unfortunately didn't gain the full advantage of continuous delivery. During both release two and three the team inteded to use the latest release created by Codeship but found that the releases where very buggy. This resulted in late releases due to having to fix the problems before the team wanted to deliver the release the product to the customer. Had their tests been better this could have been avoided.

7.2 Reliability

This study had a very small test group which of course affects the reliability of the result. Another problem is estimating the knowledge level of the students. We rated the students responses from how they explained what continuous integration is. It is of course hard to accurately assess the knowledge level of anyone by only looking at survey answers. However the fact that our students could setup the system again during the last iteration at least support that they understood their system regardless.

Despite these contraindications we believe that our study strongly indicates that having the team setting up the system is both possible and preferable.

8 Related works

We were not able to find reports on the subject of teaching continuous delivery systems specifically. We did however find several texts about the related area of continuous integration.

In our opinion the results of the studies of teaching continuous delivery systems were not properly consolidated with enough proof. One example of this is Bowyer et al. [1] where we feel that both the provided setup as well as the result section were somewhat lacking. In the result section a "deep understanding" was claimed to be achieved by the students with the only criteria was achieving a passing grade on the course. From the description of the setup everything was automated and given in a running setup, nothing here indicates that the students understood anything of the actual setup of the tools provided but rather that they only understood how it functioned on basic level.

In another study by Billingsley et al. [2] a continuous integration system was provided with the intent of reducing the cost of education by making it easier to monitor performance. But in this study the build system was provided from the start in a similar manner as in the other study, which might teach the students of how to use a given build system but will not educate them in the basis of how the system functions. Also from the study a tenfold increase of commits was observed during the last week which also indicates that a lack of continuous commits were had.

It should be stated that neither of these studies were primarily aimed at teaching CoDe rather that a deeper understanding of continuous integration was claimed as a benefit from the use of the systems provided.

9 Summary

As seen from the data in the results section as well as the conclusions in the discussion section we feel that we have verified our hypothesis that coaching the team through the setup procedure is beneficial in a learning sense.

References

[1] Jon Bowyer, Janet Hughes *Assessing Undergraduate Experience of Continuous Integration and Test-Driven Development*, ICSE '06 Proceedings of the 28th international conference on Software engineering, 2006.

[2] Jörn Guy Süß, William Billingsley *Using Continuous Integration of Code and Content to Teach Software Engineering with Limited Resources*, ICSE '12 Proceedings of the 34th International Conference on Software Engineering, 2012

[3] M. Fowler, J. Highsmith *The Agile Manifesto*, Dr. Dobb's Journal, Aug 2001.

[4] G. Herdin, L. Bendix, B. Magnusson *Teaching extreme programming to large groups of students*, Journal of Systems and Software 74(2), 2005

[5] Mary Poppendieck: Lean Software Development, Dr. Dobb's Journal, Oct 2003.

[6] Apache Ant Website, `http://ant.apache.org/`, last visited 2016-03-04

[7] Codeship Website `http://codeship.com`, last visited 2016-03-04

10 Appendix

10.1 Unmodified answers

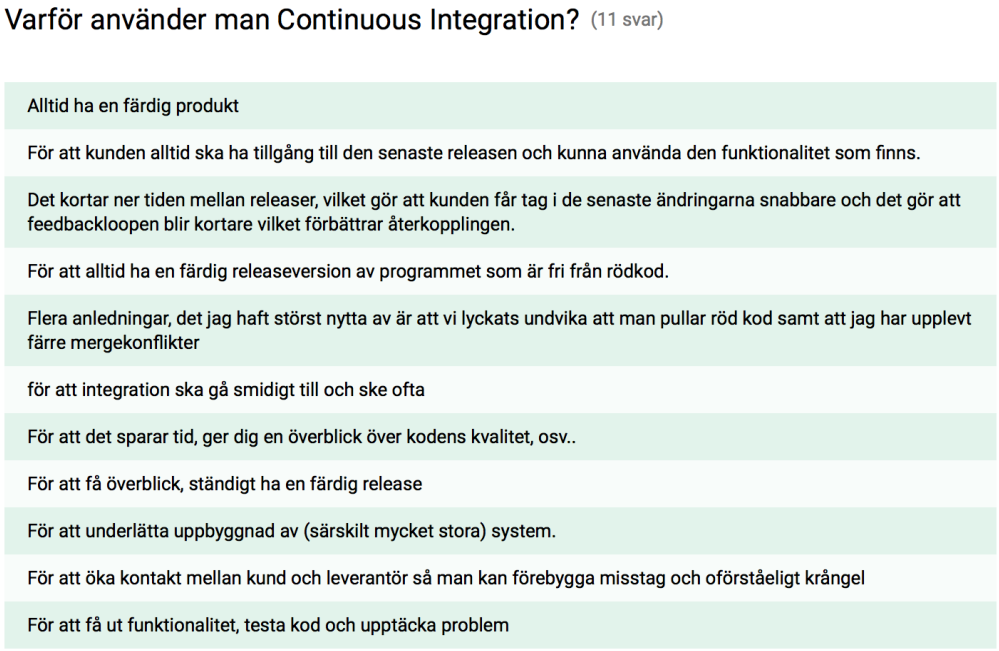

Figure 5: Our team's answers on why you use continuous integration.

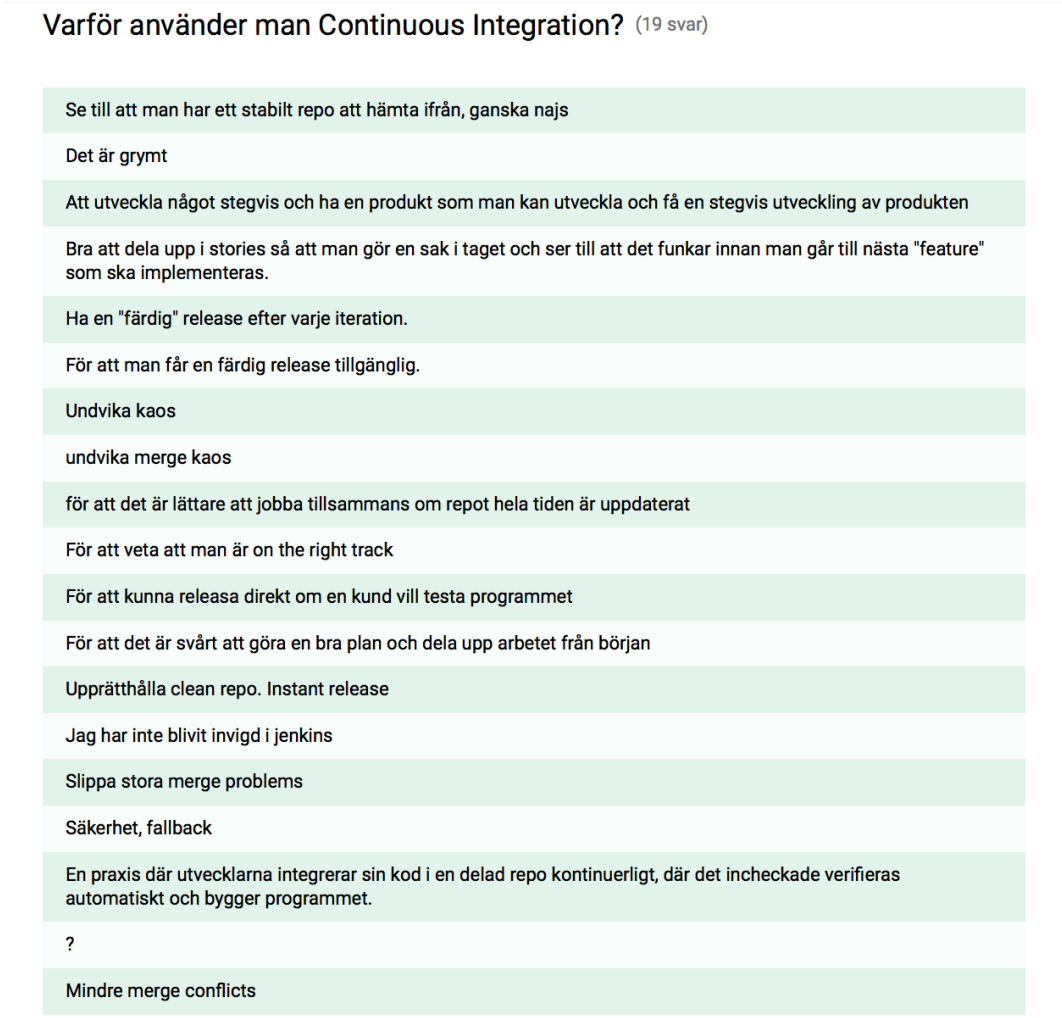

Figure 6: The other team's answers on why you use continuous integration.