Identifying project warning signs

Jenkins in an agile environment

Victor Englund, ada08ven@student.lu.se
Niklas Lindvall, ada08nli@student.lu.se
Karl Nilsson, dat12kni@student.lu.se

March 6, 2016

EDA270
Coaching of programming teams
Faculty of Engineering
Lund University

Abstract

In this study we will investigate how to use the continuous integration tool Jenkins to identify project warning signs. When developing in larger team, issues such as simultaneous updates, shared data, double maintenance and fast changing specifications will occur. To be able to mitigate some of these problems it is necessary to have an objective overview of the project and Jenkins provide these capabilities. The study shows that Jenkins will increase the awareness and produce a helpful overview of the project through a couple of plugins, displaying code coverage and unit test statistics.

1 Introduction

This study is done as a part of the course "Coaching of programming teams" (EDA270 [2]) at the Faculty of Enginnering (LTH), Lund University. The purpose of the study is to explore the possibilities to use Jenkins[4] as a tool to identify warning signs as a coach in an agile development project. Warning signs can be helpful on determining when active coaching is required.

At first, we will describe the setting of the project, which includes the roles in the project and a description of what Jenkins is. Moving on, we will describe the problem that we are trying to address with this study using the works of previous coaches in the same field, as well as a description of Test Driven Development (TDD) in the course. This is used to compose our question formulation, and our methods and results of those will be explained after that. Finally, we will present our analysis of the data gathered from our methods.

2 Background

This section contains a description of the project setting in which this study was conducted. We will also cover what Jenkins is and what purpose it fills.

2.1 The course project

During the second year of the Computer Science Engineering program at Lund Faculty of Engineering a course called "Software Development in Teams - Project" (EDA260[1]) is given. The goal of the course is for a team consisting of 10-12 students to develop a Java system for registering and sorting times for an enduro contest. The product is developed over the course of 6 weeks through an 8 hour programming session each Monday, a 2 hour planning meeting on Wednesdays and a 4 hour spike. This is the first practical agile development in a university setting that the students experience. Git and Bitbucket is used as the platform for version control and the central repository, and for most of the students it was be the first time that they get to use a version control tool for a project.

2.2 Coaching

Our role as coaches in the project is to coach the team, as a part of the course "Coaching of programming teams" (EDA270). As coaches, our purpose is to make the team functionally using our own experiences and what we've learned about agile development and team theory. In our case, we were coaching two teams. Two of us were coaching one team each, and the third coach was coaching both teams simultaneously. This was feasible due to the fact that the teams were working in the same room, divided in half. During the course a study was performed regarding the coaches' overview of a project in the agile course environment.

2.3 Jenkins

Jenkins is an open source continuous integration tool for automating repetitive tasks. A lot of well known companies has seen the benefits from using Jenkins: Sony, eBay, Netflix, Facebook and many more [5]. The basic use case of Jenkins is executing unit tests, but it can perform most repetitive tasks such as calculate test code coverage, code analysis, generate java documentation, build releases, generate reports and much more. When running tasks Jenkins provides a clean environment, on an external machine, verifying the build working outside an developer environment. The build will be assigned a score for a build depending on success at the tasks, indicating the general health of the project for a certain commit in the project. The results is easily accessible by a webpage and can be reviewed by anyone. By automating task developers' time can be spent on production rather then repetitive monotone tasks - generating feedback.

3 Problem description

When introducing a group to collaborative software development, a lot of issues will occur. For example; building a sustainable software architecture, handling collaboration where different minds approach problems differently and putting all individual parts together. According to Wayne Babich the three core problems with software collaboration is; simultaneous updates, shared data and double maintenance[12], which can lead to problems like regression of functionality, indirect code conflicts and additional maintenance work.

On top of this the team is also introduced to the methodology of fast changing specifications, constantly evolving source code and continuous releases. Holmström Olsson et al [15] discuss common barriers when transitioning to a continuous delivery workflow. Often occurring barriers are unreliable internal verification of the quality of the system, lack of transparency of process and undistributed information.

To be able, as coaches, to mitigate the problems occurring when introducing these new practices it is necessary to have an overview of the project's status and health. Keeping an updated project overview can be an tedious task for the team. Constant status reports from team members can be highly subjective and consume a lot of time.

3.1 Previous studies on Jenkins in this course

In 2013, an in-depth study[16] was made by two coaches, J. Hembring and P-G Stenberg. The purpose of the report was to show that usage of an automated build server - in this case, Jenkins - will reduce the amount of failed builds as well as tests during development. Another goal that they tried to accomplish using this was to lower stress that builds up within the team when a release is just around the corner. They had a Jenkins server set up, which fetched the code from the development team's Subversion repository, ran a test suite to ensure that the code was fully functional, and then used some plug-ins such as FindBugs and Javadoc in order to give feedback to the developers. Finally, the server then generated a release build if everything was in order. This build could then be fetched from the server by anyone in the team. After the project was finished, they handed out a survey to the other teams to see how the experienced the release process.

3.1.1 Results

Their results consisted of two sections. In one of them they showed that the team frequently used Jenkins in order to get quick feedback on broken builds, and thus were able to communicate this to the rest of the team quickly. The other part was about Continuous Integration (CI) among the other teams, and they concluded that the teams that used CI built up a lot more stress during the releases.

3.1.2 Critique

We felt that one of the aspects that could be explored further from this is the coaches' perspective and interaction with the Jenkins server, as the study never mentioned how the coaches experienced the teams progress and problems. This can be of importance since the coaches don't interact directly with the repository like the team does, thus maybe posing the risk of not having insight in the current state of the repository.

3.2 TDD in the course

While the course is designed to follow all XP practices, we have experienced that usually some of them are taken a bit more lightly. This is usually a combination of laziness, time pressure and especially a lack of understanding. Examples of this is TDD, where a lack of understanding usually was the cause during our project, and Pair programming where the pair switching sometimes was delayed due to laziness and time pressure. Time pressure was also a factor that sometimes caused the code in the repository to be uncompilable.

A consequence of lacking TDD can result in a lowered code quality and regression of functionality[14]. The lack of understanding of TDD usually boils down to lack of feedback. As the project grows, we found ourselves usually just running our own tests, instead of testing everything, as well as not running the tests just before push/commit or just after pull. Slacking on the TDD will usually lead to a lower test coverage of the code.

4 Question formulation

In Section 3.1.2 there is a mention of coaches having less insight in the repository than the team does, and that we want to investigate that matter further. Using an automated build system that can continuously produce data will hopefully give an objective overview of the health of the project to the coaches. It might identify warning signs of project problems such as failing test, decrease in test coverage and broken builds. Receiving an unbiased, computer generated status report can indicate when it is a suitable time to assist the team. As Cavano and McCall discuss, "software testing alone does not produce or ensure good software, it only gives an indication of error frequency that can be expected"[11]. These indicators will help management set an agenda for discussing reoccurring problems for the project and helping the team improve on problem areas of collaboration.

The questions we would like to answer in this study are the following:

- Will the use of an automated build server - in our case Jenkins - improve the overview the coach has of the project?
- Will an increased overview help the coaches to react to broken builds as quickly as the team does?

5 Method

In this section we will cover how we set up our Jenkins server, and how it was used. We will also show the questionnaire about project overview we handed out to the coaches of the other teams.

We use Jenkins in order to continuously build releases, perform tests and establish statistics for code quality. As a commit is sent to BitBucket master branch, Jenkins will start a build on the specific commit.

Jenkins marks a build failed, unstable or successful depending on different parameters. If the build is broken and can not compile, the build is marked failed. If one or more test does not pass or the test coverage of the code is below a certain limit (which is decided by the team), the build is marked as unstable. If the build can be compiled and all tests pass, the build is marked as successful and the corresponding commit is merged with our stable branch and pushed to the repository as illustrated in Figure 1. This practice provides us with a branch that we know has a valid build and functional test cases. Regardless the status of the build, a report of the gathered data will always be published.

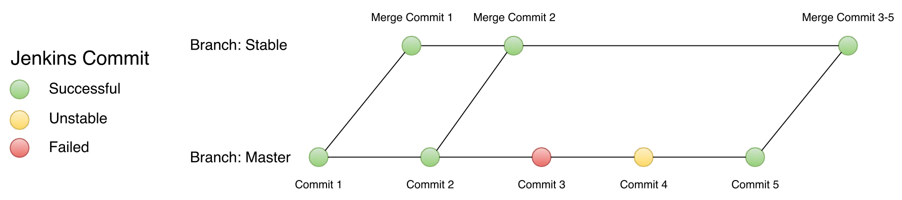

Figure 1: Master to Stable merging Jenkins

We configured Jenkins with the plugins JUnit[9] and Clover[7] to make it easier to gather the data produced by the unit testing with JUnit[4] and the test coverage produced by Atlassian Clover[6]. These plugins aggregates the data so it can be displayed as a graph over time.

To use Jenkins, the team needed to create an Ant[10] build script in order to build the project. The build script is used to compile the code, run unit tests, build runnable Java Archives, JAR's, and produce compressed zip-files with documentation and code.

During the third iteration of the project, the Jenkins server was set up and introduced to the team. This allowed us to always have the possibility to quickly look at the status of a build and acted on it if possible or appropriate. The developers were provided the address to the server and were free to use it to check the data. They were however not actively encouraged by us coaches to use it.

5.1 Test coverage

One of our major metrics for measuring the health status of the project was the test coverage reports generated by the Clover plugin. Our purpose for doing this was to see if the teams were getting sloppy with their unit tests.

5.2 Questionnaire

In order to allow us to compare our experiences to those of the coaches who did not use any assisting tools for project overview, we created a questionnaire containing the following questions.

1. *I have had an overview of the project status and its progress (i. e. red code in the repo, code quality, test quality)* **(0-5)**
2. *What type of methods have you used in order to get a status overview of the project?* **(Free text answer)**
3. *How quickly do you, the coaches, discover if the repository is "red"?* **(0-5)**
4. *How quickly do you perceive that your team discovers if the repository is "red"?* **(0-5)**

6 Results

In this section we will first present our own results from using Jenkins, both for overviewing the project and for discovering broken builds. After that, we will present the results from our questionnaire given to the other teams.

6.1 Overview

Since we always had Jenkins running on at least one of the computers that us coaches used, we constantly had a good overview of the teams' respective repositories, including the current code test coverage. This overview made us discover broken/unstable builds before the team recognized most of the time.

6.2 Red repository

Due to each build in Jenkins being graphically represented by an icon in the colors of red (broken), yellow (unstable) and blue (stable) it was very easy for us to see the current state of the repository. In most of the cases, we managed to discover that a commit with failing tests or uncompilable code had been pushed to the repository before the team did.

6.3 Data from Jenkins

The Jenkins server had at the end of the project been involved in between 80 and 120 builds for each of the three teams that used the Jenkins server. The graphs for the test coverage up until Iteration 5 is presented in Appendix A, B and C. The three lines in the Code Coverage graphs describe the following[17]:

- Conditional: Represents how much of the if- and case-statements that are covered.
- Method: Represents how much of the methods in the code that are covered.
- Statement: Represents how many of the variants of if- and case-statements that are covered.

It's a bit hard to distinguish the difference between conditional and statement coverage. The difference is that statement coverage considers all outcomes of a statements. For example, if the code has an if-statement without an else case, the statement coverage will count the non-existing else-case as not covered by the code and thus lower the coverage for statements.

The three Build Status graphs show the total number of tests executed for each push, and how many of the tests that passed/failed. Any amount of failed tests results in the build being marked as unstable. Some of the builds shown have 0 executed tests, and that is due to the pushed code not compiling at all. This results in no tests ever being executed.

6.4 Data from questionnaire

In Figure 2 and 3 below you will find the data from three of the questions in the questionnaire. Worth noting when studying these graphs is that Team 6 and Team 7 were conducting studies on CI as well.

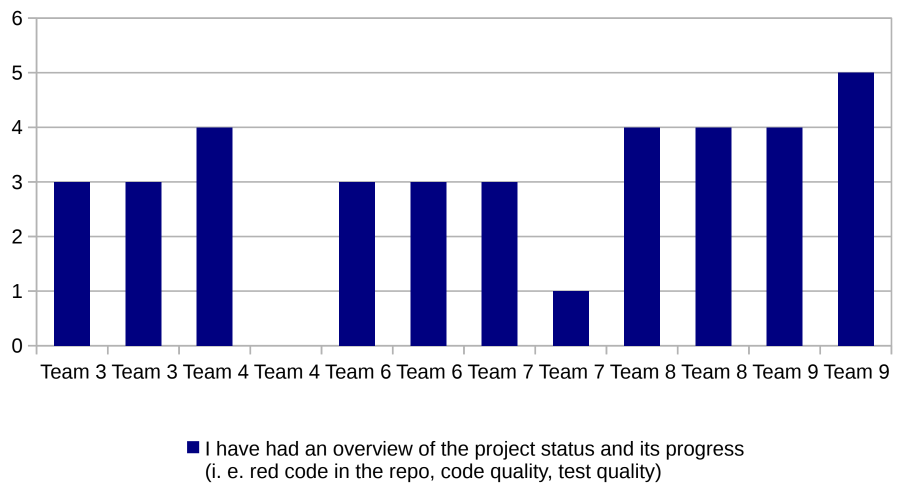

Figure 2

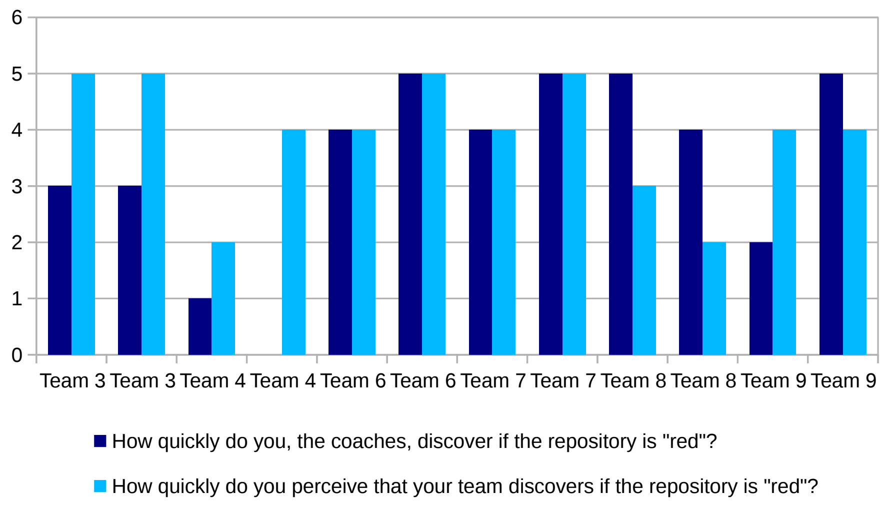

Figure 3

The free text question *What type of methods have you used in order to get a status overview of the project?* resulted in a couple of different answers but most of the coaches used short team meetings and verbal communication to get an overview. Two teams said they used some form of continuous integration tool (CircleCI[18] and Codeship[19]) and the rest got an overview by hand using other type of tools like code coverage and/or unit tests plugins to Eclipse.

The raw data from this questionnaire can be found in Appendix D.

7 Analysis

In this section we will provide an analysis of the results presented in the previous section. Before discussing our findings, we introduce the constraints we used and which problems occured during the execution of our methods.

7.1 Constraints

Time is always a factor, and was so in this project as well. The fact that the project was run over just six iterations limited us a lot. Had there been more time we would probably introduced a lot more functionality to the project, such as fully automated release process on Jenkins, rejection hooks for BitBucket pushes and automated GUI tests for JUnit.

A big fact that makes this course differ a lot from an industrial setting is that it's just a course. The main goal is to learn, and it's not really possible to fail the course unless the students are absent from the labs.

Along the lines of that, we would also like to point out that the team, coaches included, were having other courses of differing workload simultaneously. The course is also limited to the scheduled time, and the students are not allowed to work over-time. Had this been in an industrial setting, and the release was approaching, the team would most likely not go home at 17:00 with half finished work just before a release.

As mentioned earlier, we decided to not introduce Jenkins until the third iteration of the project in order to not overwhelm the team. This decision gave us some useful data about our own experiences before and after the Jenkins introduction, but it also shrunk the time window we had to work with and experience Jenkins.

Another constraint we were forced to put on ourselves due to time limits was to set up the Jenkins server to just concern the master branch. Even though most of the pushes were headed directly to the master branch, a couple of branches, mainly for refactoring, were created. We do not have any data for Jenkins regarding these.

7.2 Problems

One of the first problems we encountered when using Jenkins was the use of the Ant scripts. The teams had the task of creating an Ant script assigned to them by the customer. The script worked well in the work environment the team was developing in, but when executed in the sterile environment of the Jenkins server it was discovered that some of the libraries used weren't properly included. This was due to the teams being inexperienced with Ant, and the problem could be solved after a few hours of work.

Another problem that one of the teams encountered during the second release of the project was that the Clover plugin we used on the Jenkins server seemed to inject some code into the JARs, making the program unexecutable for the customer. This delayed the release for about an hour. This was later discovered to be the fault of the Clover plugin in Jenkins, but we managed to find a way to work around it later on.

During the fifth iteration of the project, the Jenkins server crashed due to it running out of storage space. The time from the initial crash until the server was up and running again was about three hours.

Jenkins went down once again during the final iteration. The reason for this was a late requirement from the customer, which was to make the code compatible with Java 6. Using the Java 6 compilation broke our Clover plugin on the server, and we were forced to remove it. This is the reason for us not having any code coverage data for iteration 6. Had there been more time we would probably have still compiled the program using Java 8 with a different Ant script on the Jenkins server.

7.3 Questionnaire

The data set from the questionnaire is fairly small as there are few teams by design in this coarse. In total there are nine teams, minus our own two teams and another team that chose not to answer. This means we can only speculate about any patterns that are emerging.

In Figure 2 and 3, we can see that only because you are using a CI tool (CircleCI or Codeship) the overview of the project is not necessarily increased. However, the teams using a CI tool tend to have more of a similar response time, in respect of red repository, between the coaches and the developers.

The teams using verbal communication, and no specific tools to help, had more differences between how quickly the coaches respective the developers discover if the repository was red. This could mean that using a CI tool could be helping in getting a shared view of the project.

7.4 Discussion

Using a build server such as Jenkins enables the coaches to get an increased and helpful overview of the project. Being able to spot a "Red repository", in many cases before the team itself, did allow coaches to be prepared and ready to coach the situation if needed. By consulting the Jenkins report, you can easily identify what change has caused the issue and what kind of problem has occurred; uncompilable files, incorrectly solved merge conflict, failing test cases and drastic drops in code coverage. Usually the developers showed the behaviour of pulling source code just before pushing their own, delaying the impact of the potential issues in in the shared repository. The instant feedback provided by Jenkins was an invaluable teaching tool.

The stable branch that Jenkins pushed to after successful builds was a great asset when a release was approaching. Having access to a branch were every commit was a potential release candidate helped during releasing. Even though it's possible to use the stable branch pattern without an automated build server, it was good to know that the tests were always carried and passed by Jenkins.

As mentioned in Section 5, we introduced Jenkins during the third iteration of the project. Although we felt that it would be a good experience for the team to carry out their first release without the support from the Jenkins server's stable branch, this late introduction combined with our time window of six iteration might have hindered their workflow from fully adopting Jenkins. The tools that was introduced earlier on in the project such as Git, Eclipse and JUnit, got a greater adoption than Jenkins and Clover which was introduced later on. This somewhat reduced the feedback we got from the teams on the server. Getting more feedback from the teams regarding what they would've been interested in having on Jenkins in terms of graphs, plugins or other suggestions to further enhance our experiences and thus the results for this study.

When the Jenkins server went down during the fifth iteration, our overview went down with it. Since we were used to having this comfortable way of telling whether the repository was in good shape or not, we were somewhat perplexed by the situation. Luckily, the teams didn't seem to have any major issues during that time anyway, so we could sit down and sort out the problems with the Jenkins server relatively undisturbed. This experience taught us a lot how much we were relying on the Jenkins server, and that it could be worth having some sort of backup plan for situations like this. The teams that didn't use tools such as Jenkins mostly relied on oral communication and stand-up meetings.

8 Conclusion

From the data gathered, both from our own usage of Jenkins and the questionnaire handed out to the other teams, we feel confident to conclude that using any CI tool will make the coaches and the teams equally aware of the objective status of the repository. Our own experience with Jenkins gave us the impression that we had a better overview of the project overall, not just the repository. This is however something that needs to be investigated further, as we would like to have more data on other coaches using Jenkins for the same purpose as we did. Another thing that might be of interest for future works is to explore if the team will benefit from their coaches having the same (or better) status knowledge of the repository as the team.

References

[1] EDA260 - Software Development in Teams – Project. (n.d.). Retrieved March 03, 2016, from `https://kurser.lth.se/lot/?val=kurs&kurskod=EDA260`

[2] EDA270 - Coaching of Programming Teams. (n.d.). Retrieved March 03, 2016, from `https://kurser.lth.se/lot/?val=kurs&kurskod=EDA270`

[3] Bitbucket — The Git solution for professional teams. (n.d.). Retrieved February 23, 2016, from `https://bitbucket.org/`

[4] Jenkins . (n.d.). Retrieved February 23, 2016, from `http://jenkins-ci.org/`

[5] Who is using Jenkins? (n.d.). Retrieved February 26, 2016, from `https://wiki.jenkins-ci.org/pages/viewpage.action?pageId=58001258`

[6] Java and Groovy code coverage. (n.d.). Retrieved February 23, 2016, from `https://www.atlassian.com/software/clover/overview/`

[7] Clover Plugin. (n.d.). Retrieved February 23, 2016, `https://wiki.jenkins-ci.org/display/JENKINS/Clover+Plugin`

[8] JUnit - About. (n.d.). Retrieved February 23, 2016, from `http://junit.org/`

[9] JUnit Plugin. (n.d.). Retrieved February 23, 2016, `https://wiki.jenkins-ci.org/display/JENKINS/JUnit+Plugin`

[10] Apache Ant. (n.d.). Retrieved February 23, 2016, from `http://ant.apache.org/`

[11] Cavano, J. P., & Mccall, J. A. (1978). A framework for the measurement of software quality. Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues -.

[12] Babich, W. (1986). Software configuration management: Coordination for team productivity. Reading, Mass.: Addison-Wesley.

[13] Keyes, J. (2004). Introduction to Software Configuration Management. In *Software configuration management.* Boca Raton, Fla.: Auerbach Publications

[14] Extreme programming pocket guide. (2003). Beijing: O'Reilly.

[15] Olsson, H. H., & Bosch, J. (2014). Climbing the "Stairway to Heaven": Evolving From Agile Development to Continuous Deployment of Software. Continuous Software Engineering, 15-27.

[16] J. Hembrink, P-G Stenberg, (2013). Continuous Integration with Jenkins, Coaching of Programming Teams.

[17] About Code Coverage, Clover 4.1, Atlassian Documentation. (n.d.). Retrieved March 02, 2016, from https://confluence.atlassian.com/display/CLOVER/About+Code+Coverage

[18] "Ship Better Code, Faster." Continuous Integration and Delivery- CircleCI. Web. 03 Mar. 2016.

[19] "Continuous Delivery with Codeship: Fast, Secure and Fully Customizable." Continuous Delivery with Codeship: Fast, Secure and Fully Customizable. Web. 03 Mar. 2016.

Appendices

A Team 1

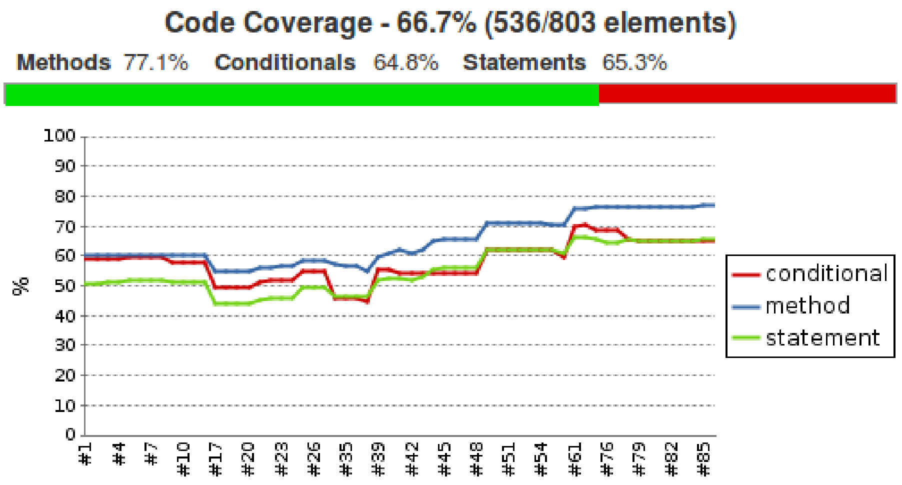

Figure 4: Test coverage for Team 1

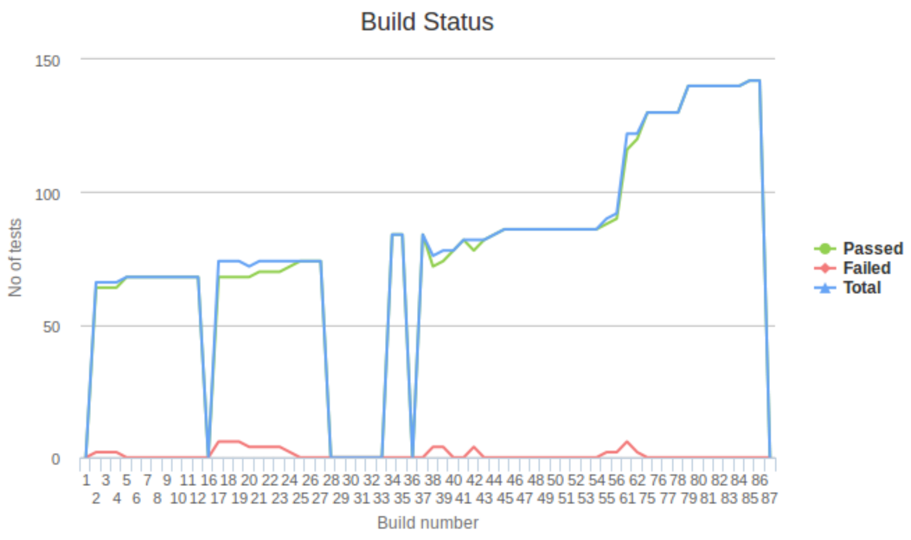

Figure 5: Number of passed test cases for Team 1

B Team 2

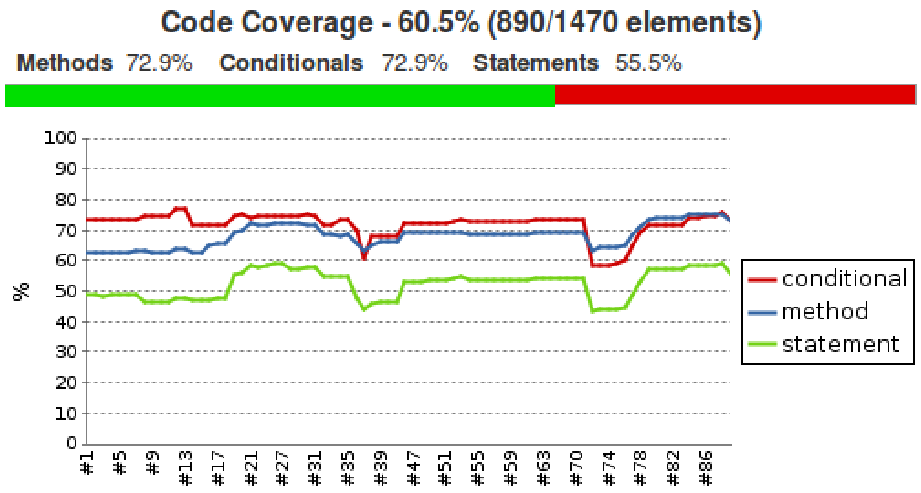

Figure 6: Test coverage for Team 2

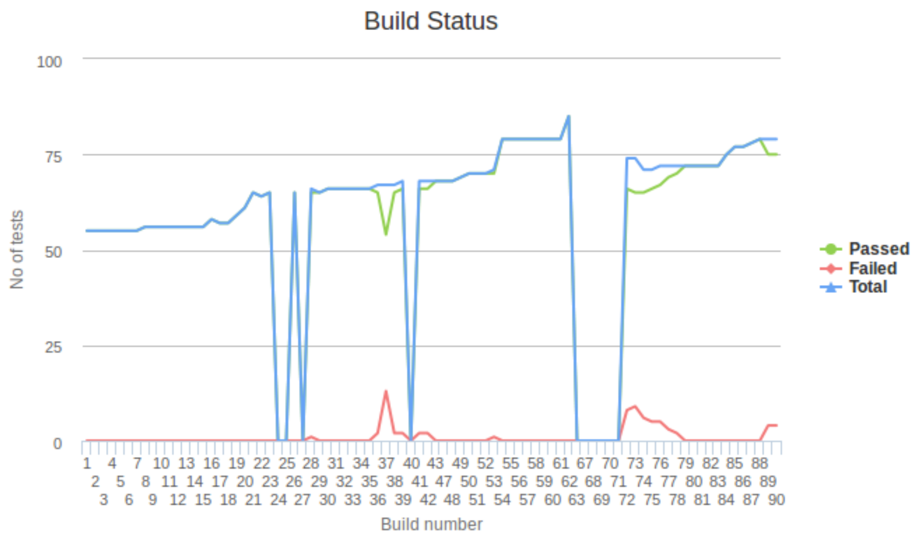

Figure 7: Number of passed test cases for Team 2

C Team 5

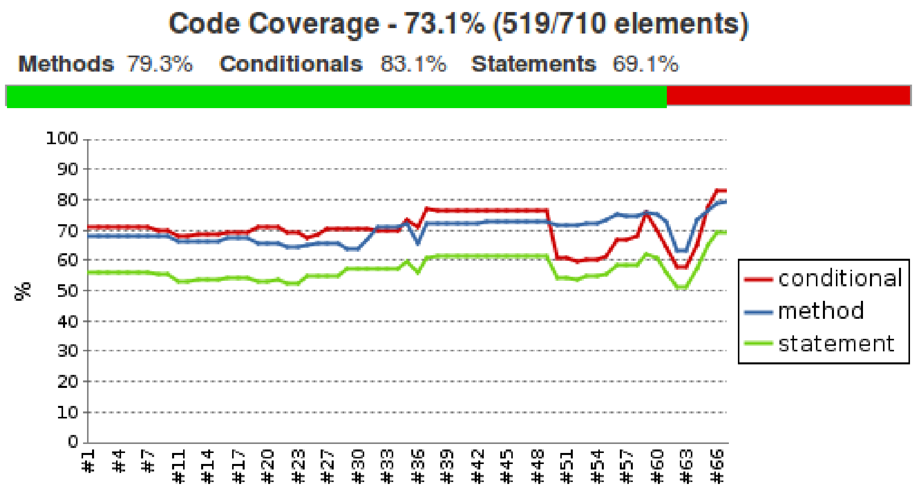

Figure 8: Test coverage for Team 5

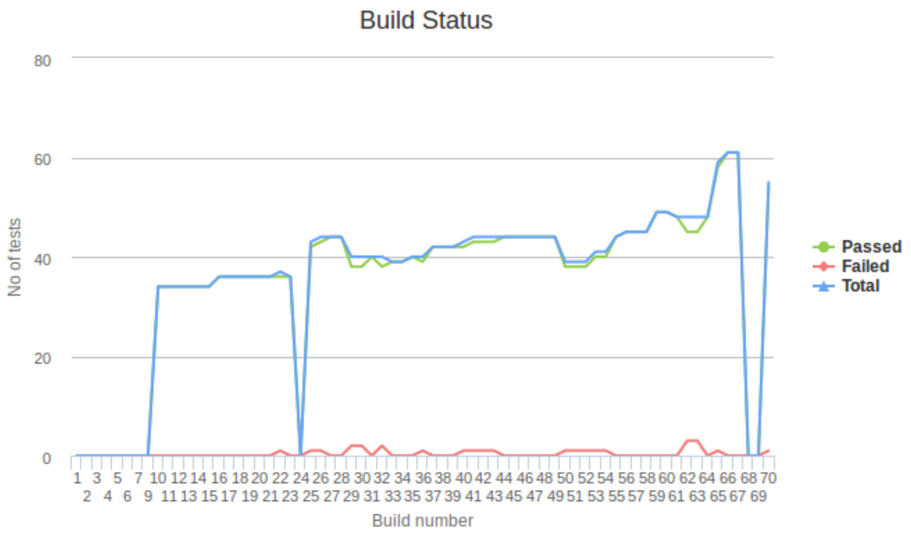

Figure 9: Number of passed test cases for Team 5

D Questionnaire raw data

Table 1: Raw data from questionnaire

Vilket team coachar du?	Jag har haft en överblick av projektet status och dess utveckling?	Vilka metoder har ni använt för att få en överblick?	Hur snabbt upptäcker ni coacher att det är rött i repot?	Hur snabbt upplever ni att teamet upptäcker att det är rött i repot?
Team 6	3	Standup, muntlig, codeship.	4	4
Team 4	4	Tjuvkikat i repot och pullat	1	2
Team 4	0	Jag: Inga. Andra coachen: Studerat koden	0	4
Team 3	3	Standup, Kommunikation	3	5
Team 6	3	Primärt muntlig kommunikation / standup. Vi har även fått teamet att utveckla ett system som tillexempel nu blockerar git push till dev branch om inte alla tester är gröna.	5	5
Team 3	3	Muntlig kommunikation främst	3	5
Team 8	4	Standup, muntlig kommunikation	5	3
Team 8	4	junit, standup	4	2
Team 9	4	Aldrig tittat i själva koden, men har lyssnat vid diskussioner, stand-up möten etc. Samt då och då kört igenom våra tester samt efteråt kollat code-coverage med EclEmma.	2	4
Team 7	3	CircleCI, Coveralls	4	4
Team 7	1	Standup, muntligt, CircleCI	5	5
Team 9	5	Muntlig kommunikation, lyssna vid standup, EclEmma, skumma igenom ny kod	5	4