

LEE KONG CHIAN FACULTY OF ENGINEERING & SCIENCE

MAY TRIMESTER 2016

UECS2363

SOFTWARE CONSTRUCTION AND CONFIGURATION

Group Project

Team – 06

|  |  |  |
| --- | --- | --- |
| **No.** | **ID** | **Name** |
| 1 | 1406399 | Cheong Tymm |
| 2 |  | Lim Jia Hui |
| 3 |  | See Hong Min |
| 4 | 1300023 | Shashindran Vijayan |
| 5 | 1407677 | Yap Boon Keng |

Lecturer: Dr. Tay Yong Haur

# Overview of the project

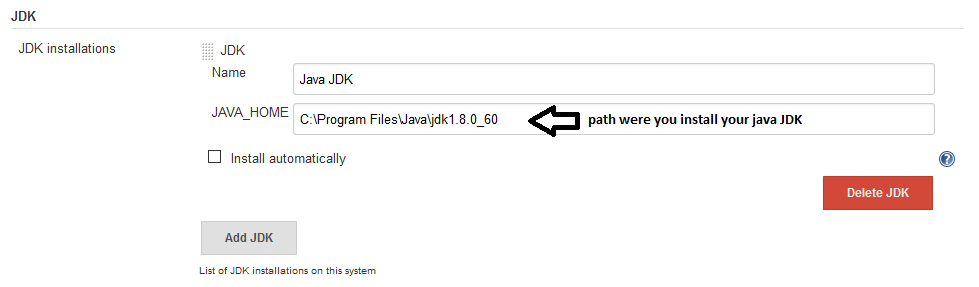
The project that we chose for our Assignment work is Apache/Maven project. Maven is a software project management and comprehension tool that can be used for building and managing any Java-based project. Maven can manage a project's build, reporting and documentation from a central piece of information based on the concept of a project object model (POM). Maven allows a project to build by using POM to provide a uniform build system so that it can make the build process easy in the shortest period of time. It also provides the quality information about the project and guidelines for best practices development.

# Continuous Integration Server (CI)

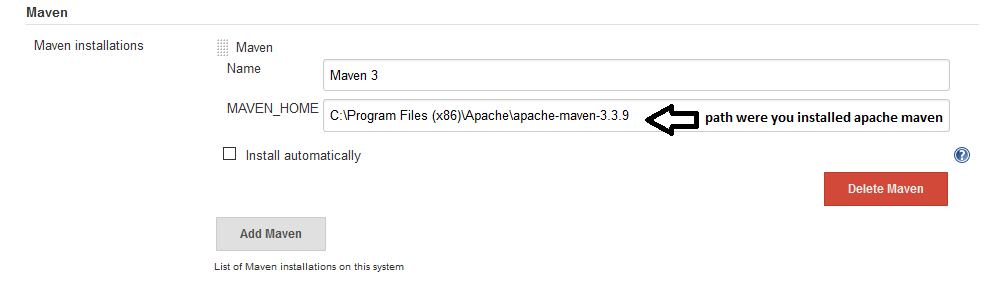
We used Jenkins as our continuous integration (CI) server for this project. This because Jenkins is one of leading open source automation server written in Java. Besides, it also easy and free to install and use in any operating system. It can support the source control management (SCM) such as Git, CVS, RTC and also can execute Apache Maven based projects. Moreover, Jenkins can be configured via its web interface. It also provides help and error check with the use of built-in help. On top of that, Jenkins has a lot of plugins which is very handy in the continuous integration and continuous delivery. The plugins can add the new functionality and change the way looks of the Jenkins. The test reports will be generated in various formats supported by plugins when the project build. It also can display the reports and show the trend in GUI.

# Overall configuration of CI

For this project to work well, apache maven and Java JDK files have to installed and configured correctly in the operating system. Besides that, under the ‘Global Tool Configuration’ in Jenkins, the maven path and JDK path has to be specified as shown below.

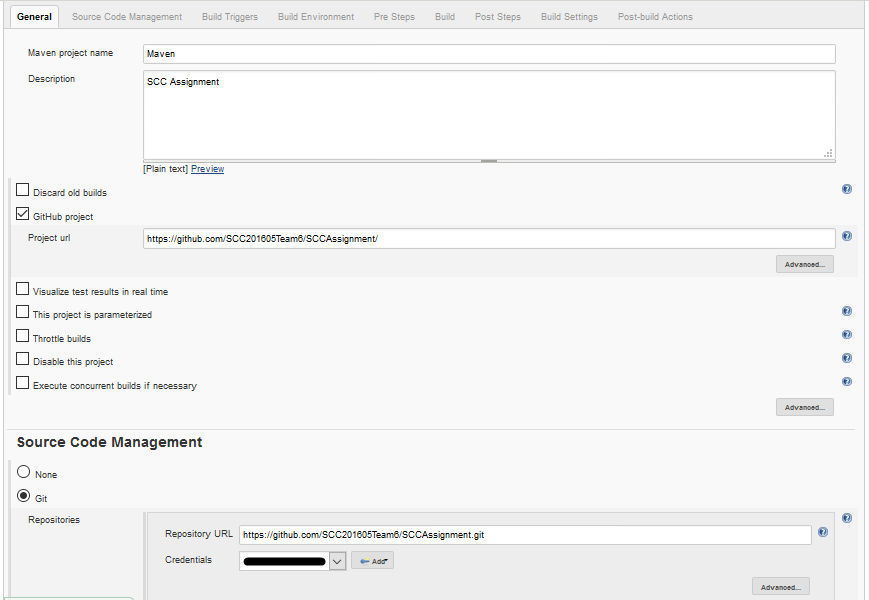


**Figure 1: JDK setting in Global Tool Configuration**

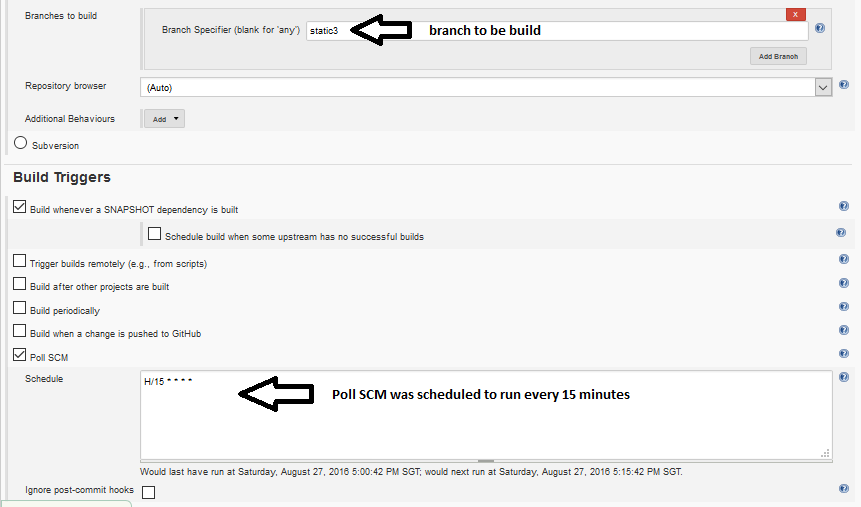


**Figure 2: Maven setting in Global Tool Configuration**

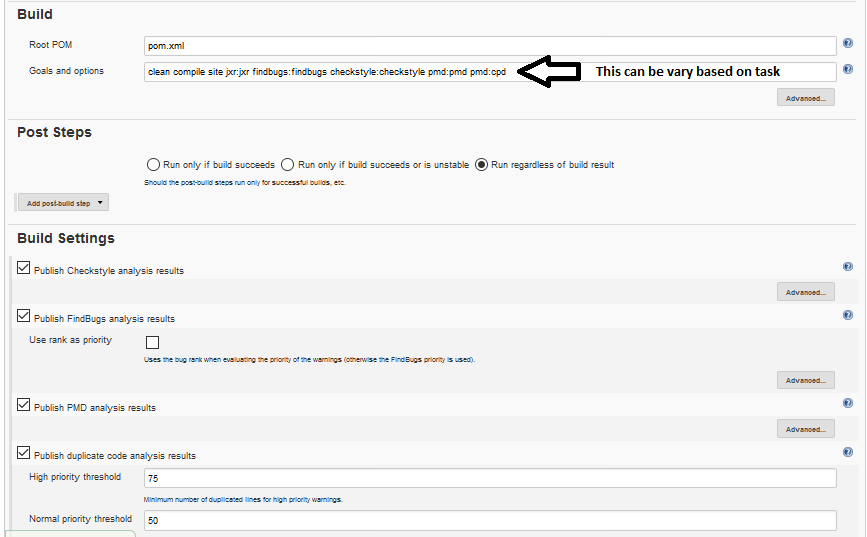
Besides that, we also installed plugin called green ball to get the green light to indicate the successful build. We also have to include the maven path in the system path configuration so that we can execute the maven commands through CMD. The Jenkins maven project was configured has followed:



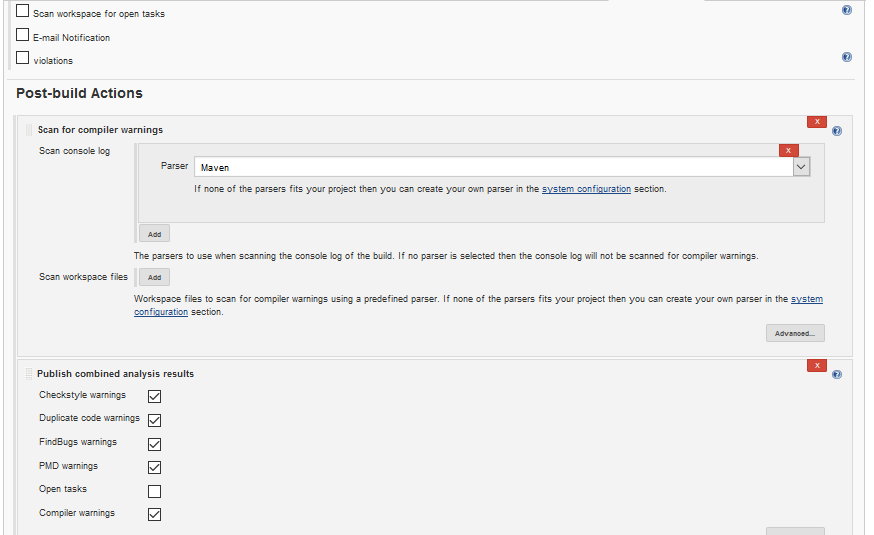
**Figure 3: Maven project configuration 1**



**Figure 4: Maven project configuration 2**



**Figure 5: Maven project configuration 3**



**Figure 6: Maven project configuration 4**

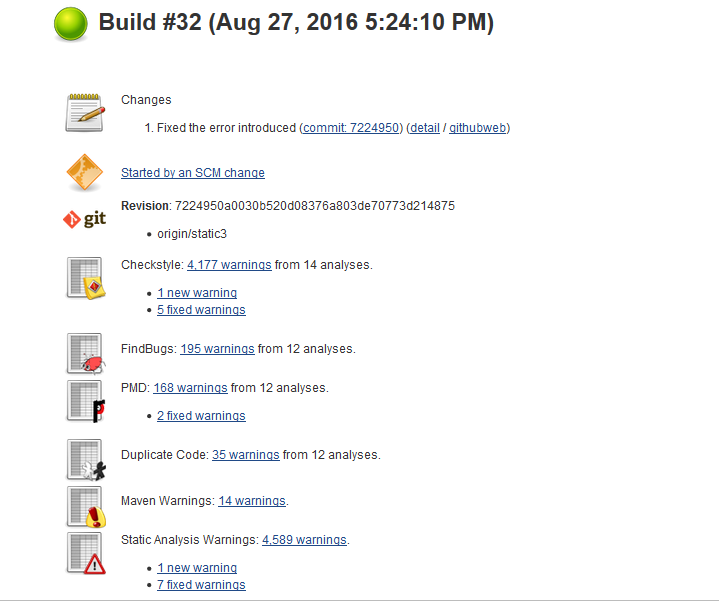
Moreover, the pom.xml files have to edit so that the file can use the intended plugins.

# Process of CI

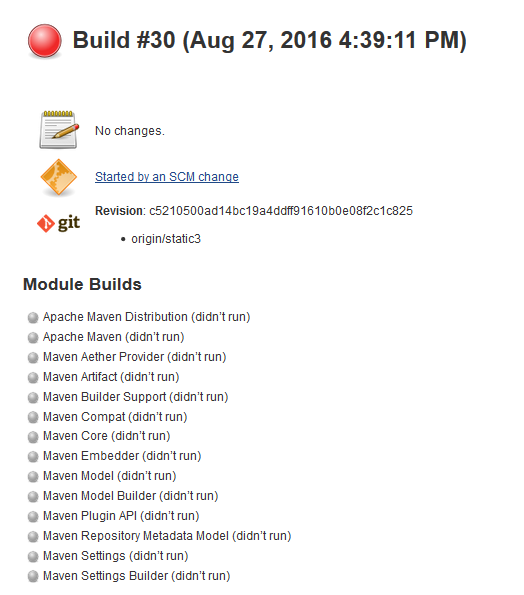
The Jenkins server will check for the changes in respective branches that configured in Jenkin project every 15 minutes. If there are any changes pushed to the repository, Jenkins server will pull the repository and start to build. The build script will be varied for each member for specific tasks. For example, for static analysis, the build script that used was “clean compile site findbugs:findbugs jxr:jxr checkstyle:checkstyle pmd:pmd pmd:cpd”. This line will execute and run the plugins present in the project. For testing however, build scripts that are needed are: clean package checkstyle:checkstyle. This line is sufficient for the test to run properly.

# Action of CI

The Jenkins will produce a detailed data of the build result. If the project is built successfully, the Jenkins will show green ball and all the result according to the build script will be formed meanwhile if the build is failed the Jenkins will display the red ball and additional detail about the error can be view from console out.



**Figure 7: Build Success**



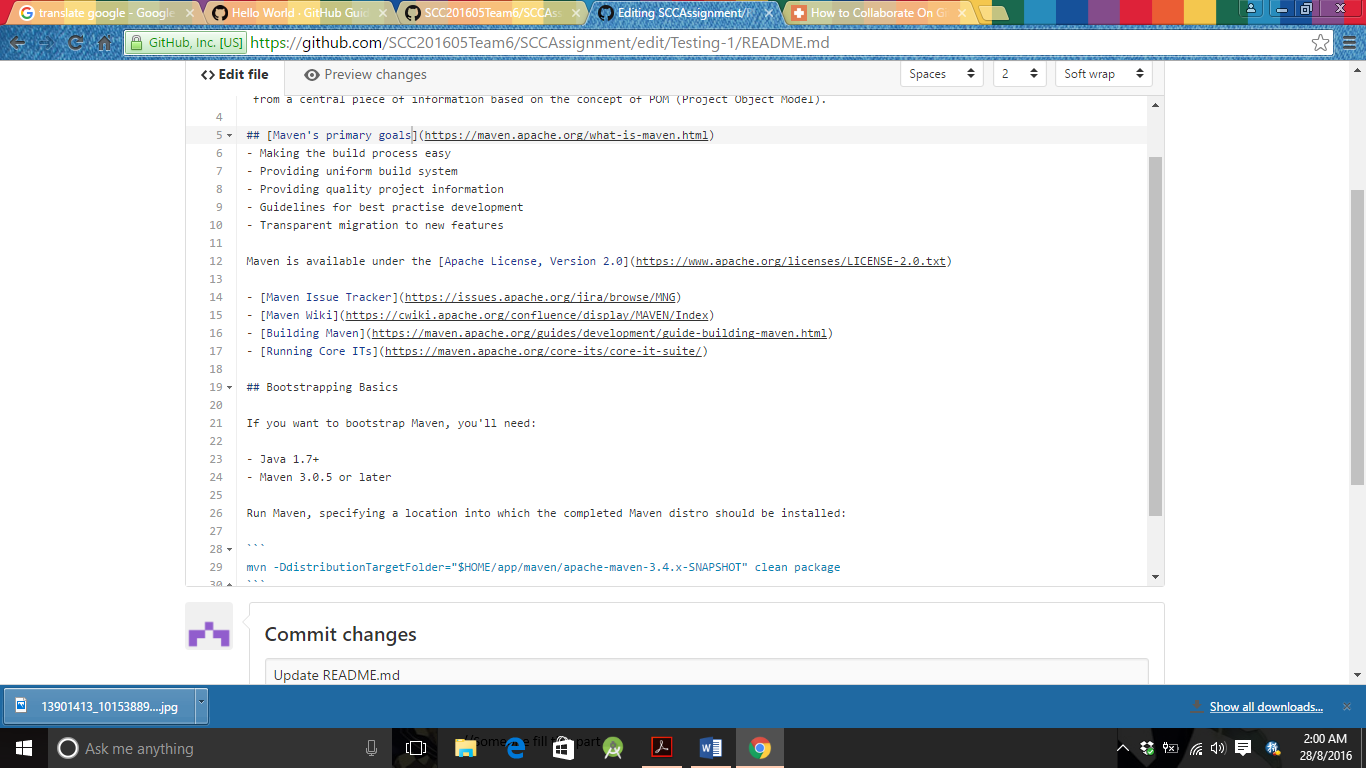
**Figure 8: Build Failed**

# Branch handling

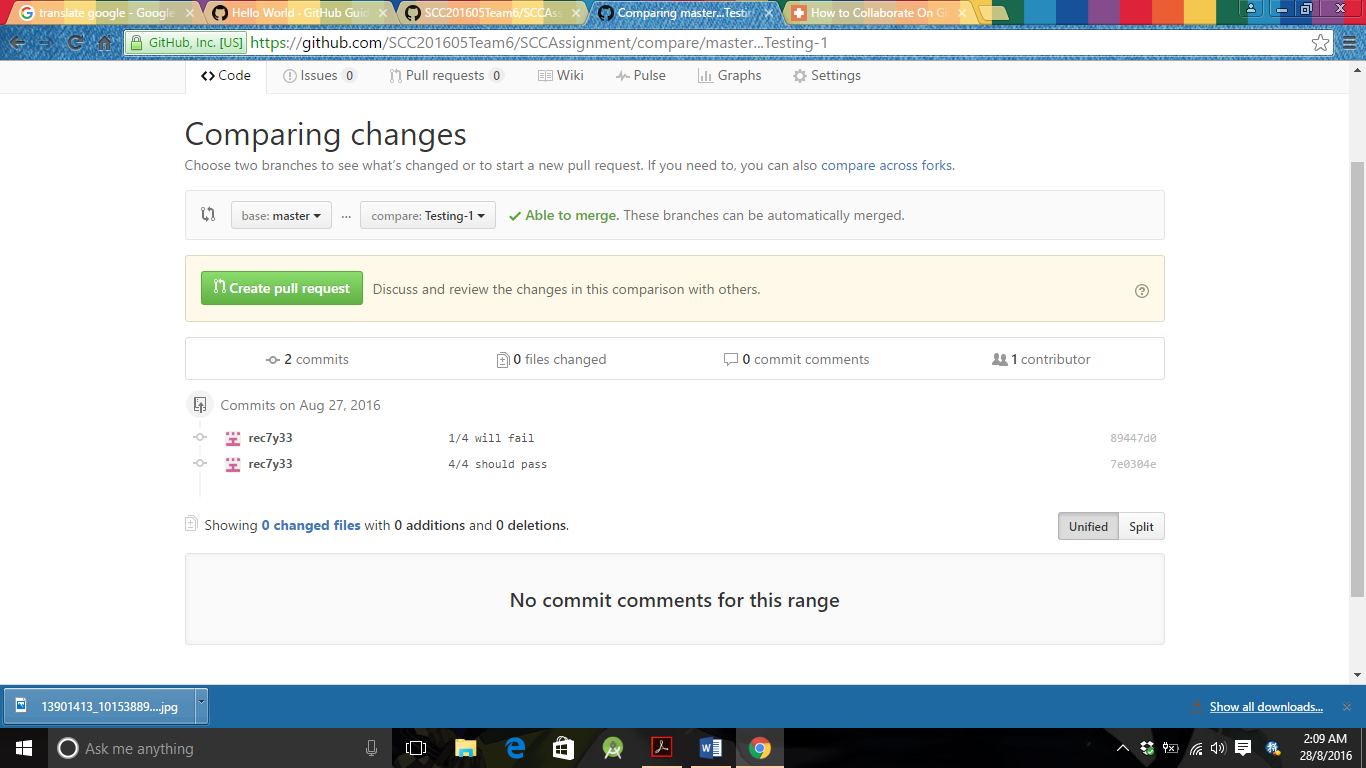
Each new feature and bug is handled in new branch and the branch was tested using Jenkins for correctness before merging with master. If the Jenkins builds the branch successfully, a pull request will be sent. The project manager will check again for the correctness and approve the pull request if the branch did not produce any error on other files.

# Collaboration

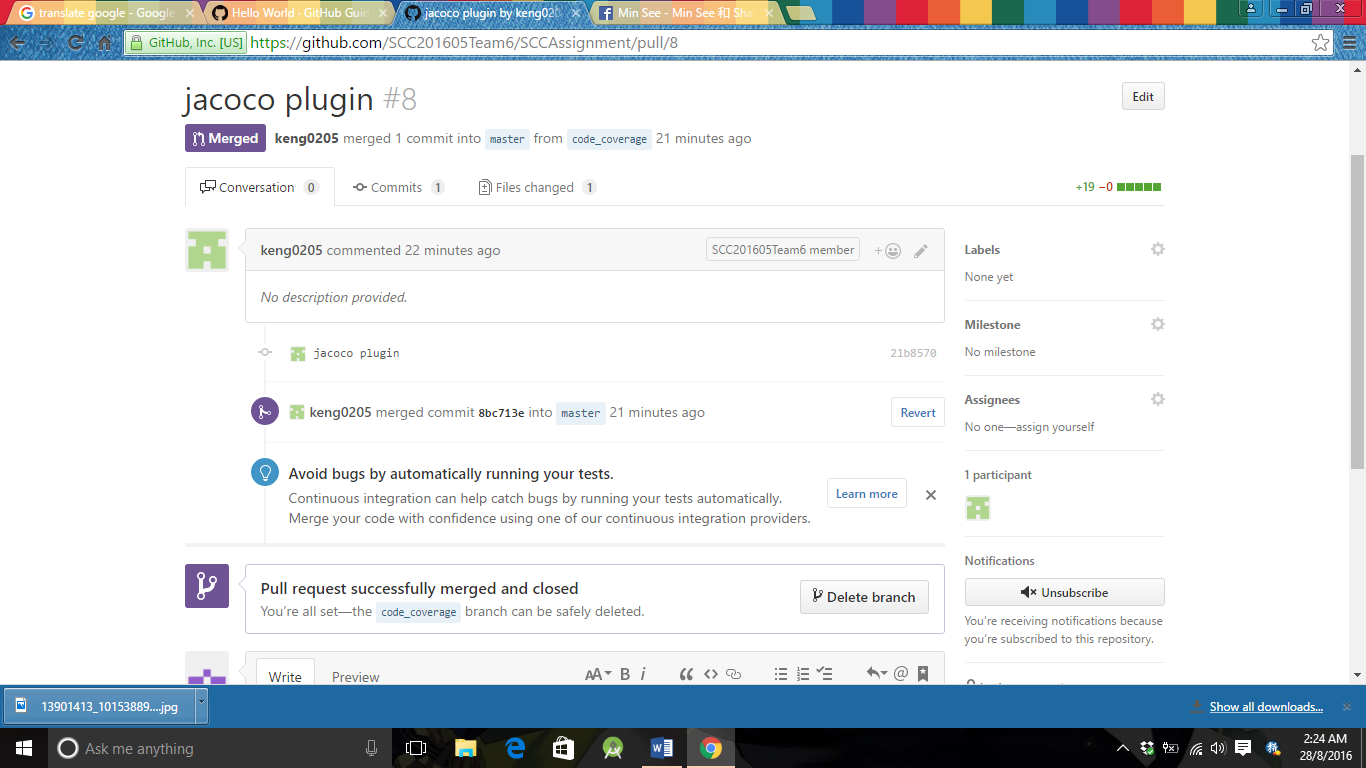
Before we start to contribute the project, we fork the target project to our own group repository and we clone it to our own local machine. We create new branches to work on different versions of same repository at one time. Then, we make and commit changes by following our roles. We need to create a pull request by using different viewer on Github. The requested will make when we open the pull request. The pull request is then merge into the master branch and the topic branch will be deleted from the upstream repository.



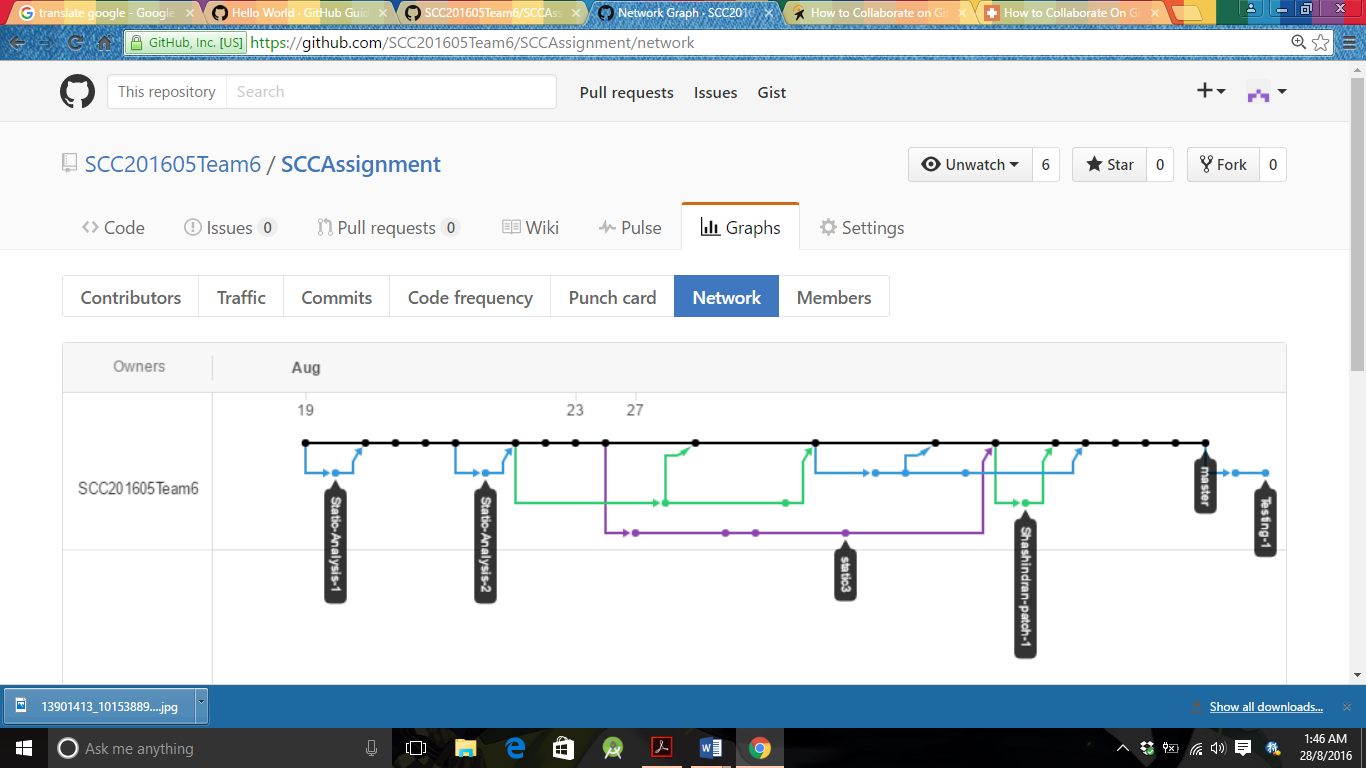
* The README.md files will change and commit in testing-1 branch, so its content is different from master branch.



* Look over and compare the changes from master branch when the pull request is created.



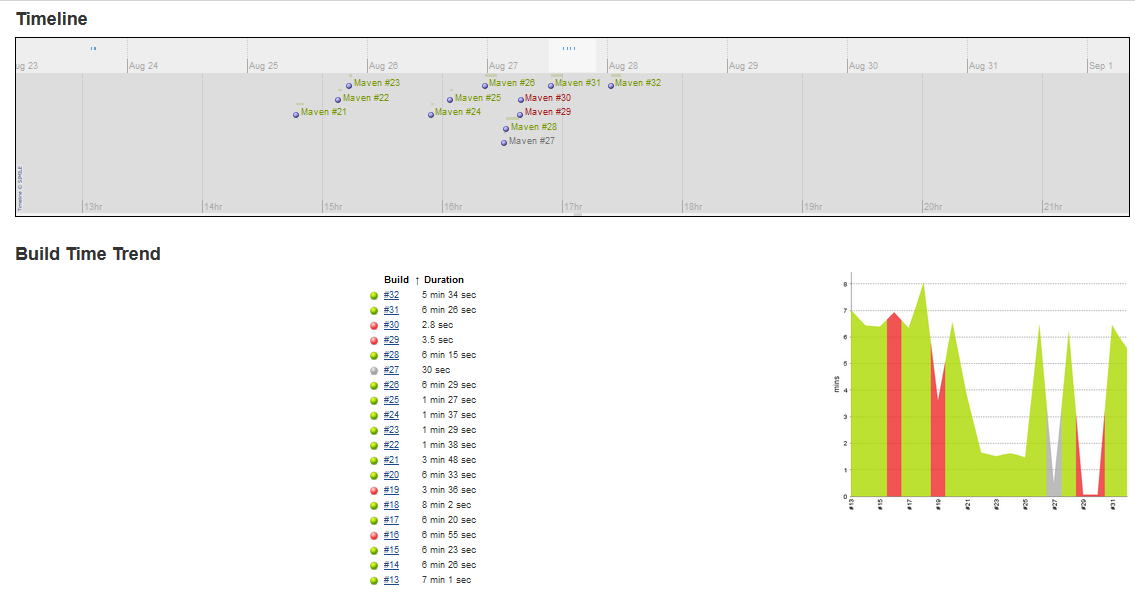
* The pull request has successfully merged into master from code\_coverage branch.



* The network graph shows the branch history of the branches of the root repository and branches of forks that contain commits unique to the network.

# CI server’s report

## Build History

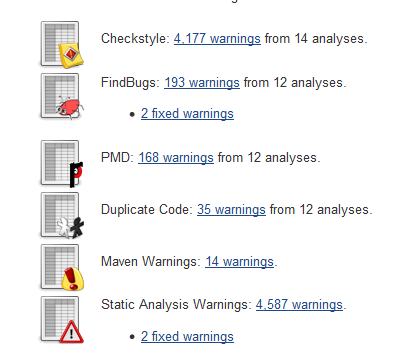


**Figure 9: Build History**

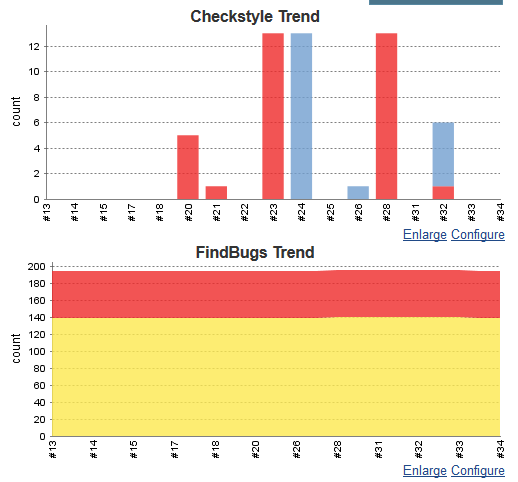
## Static Analysis

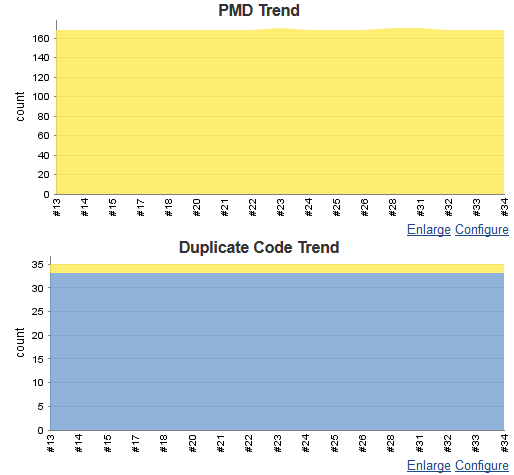
The static analysis was carried out using these plugins:

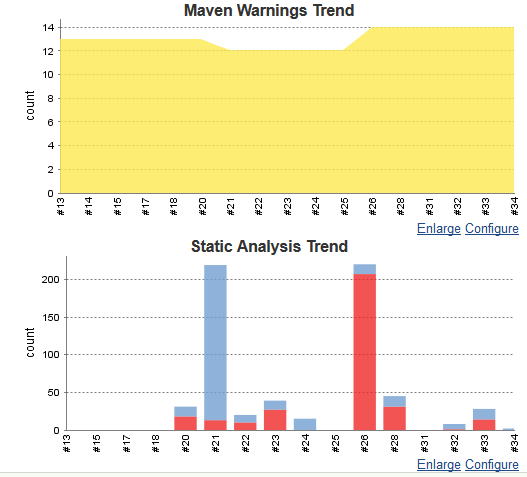
* Findbugs
* Pmd
* Checkstyle
* Dry
* Warning



**Figure 10: Static Analysis Report for build**







**Figure 11: Static Analysis Graph**

## Testing (Junit)

For the unit testing of this project, we used the Junit plugin for Jenkins. Simply build the project and the Junit tests will be run after Jenkins has finished building the project.

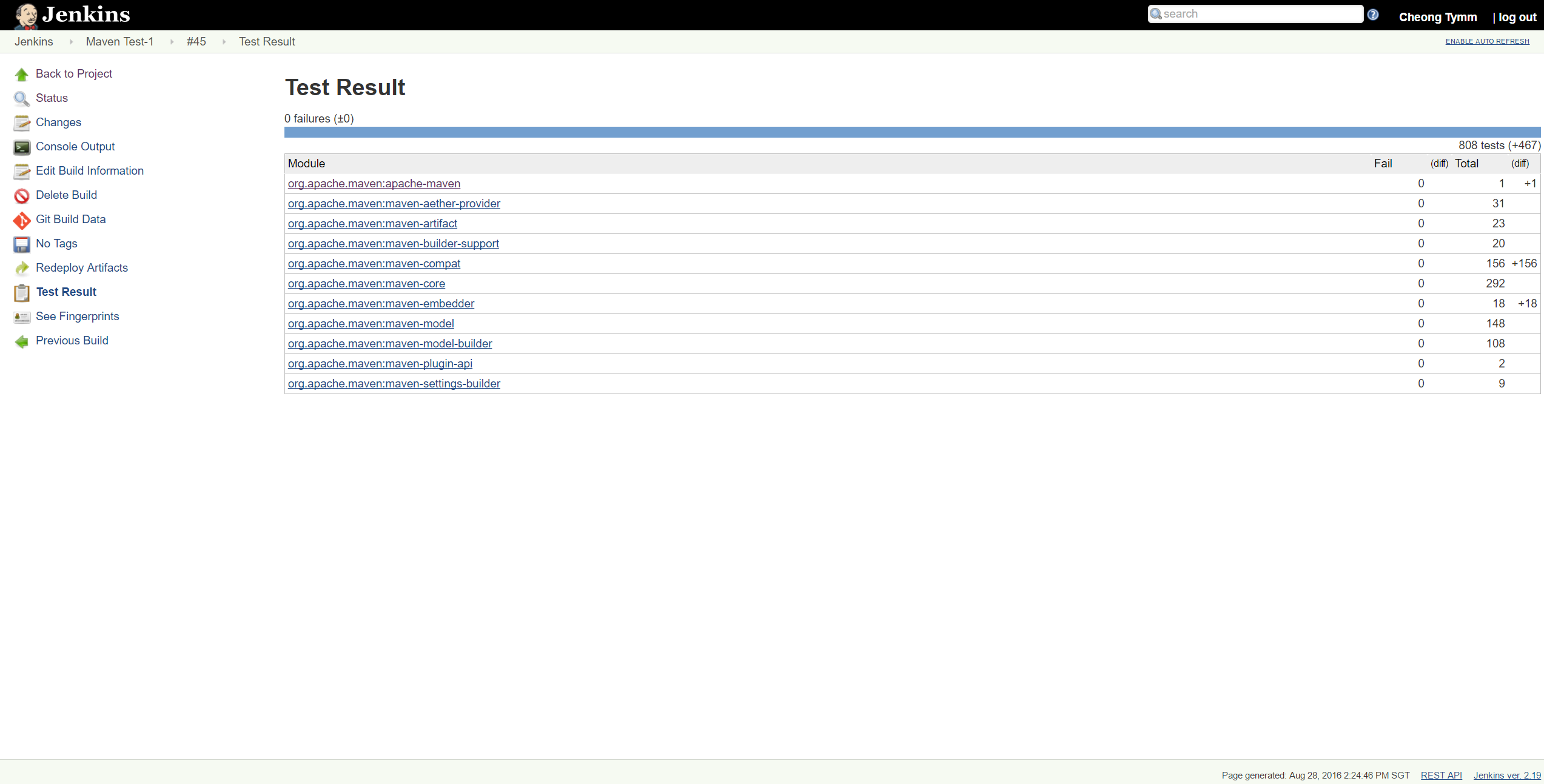


Figure 12: Test Result (all passed)

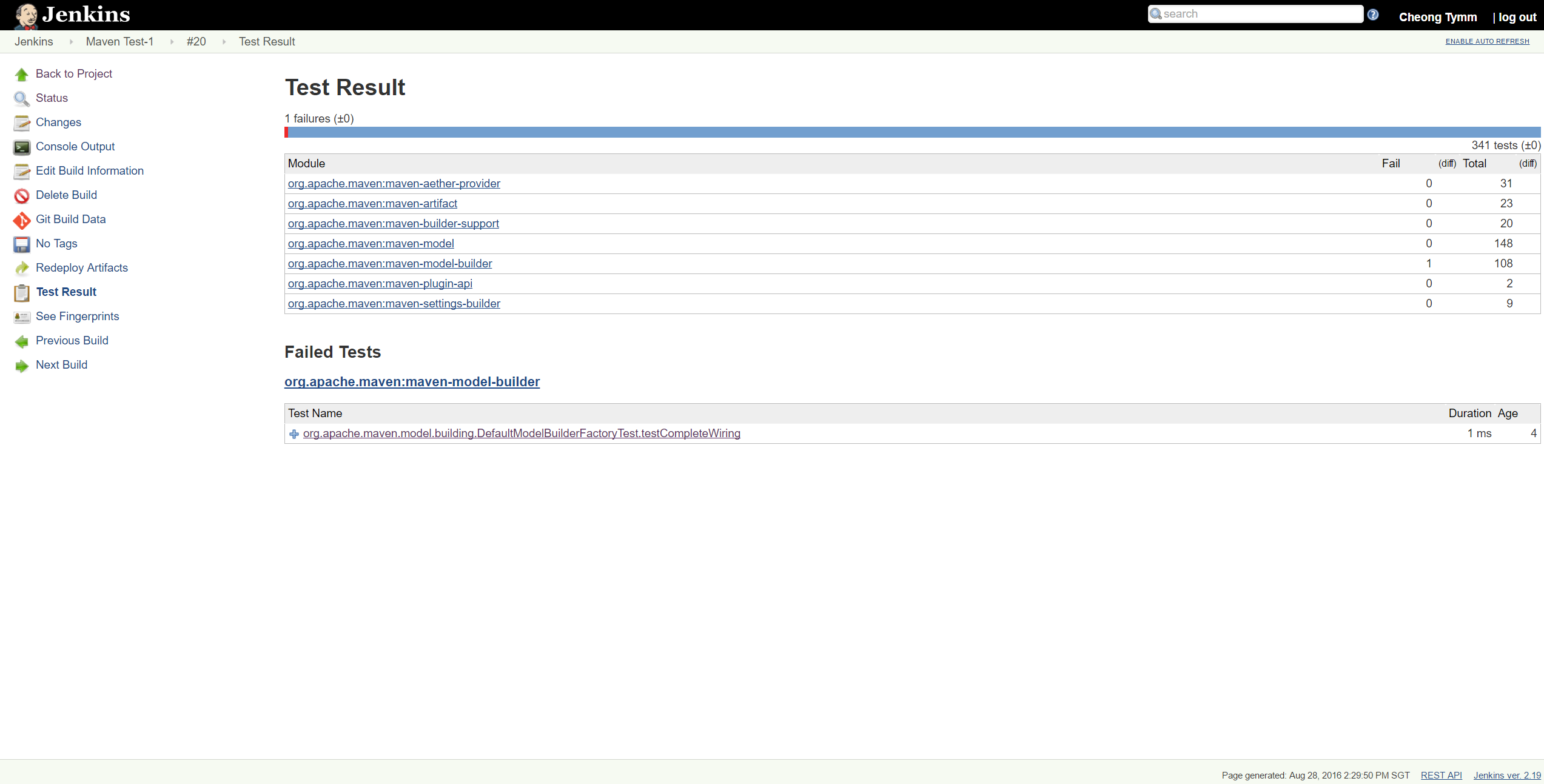


Figure 13: Test Result (with fail)

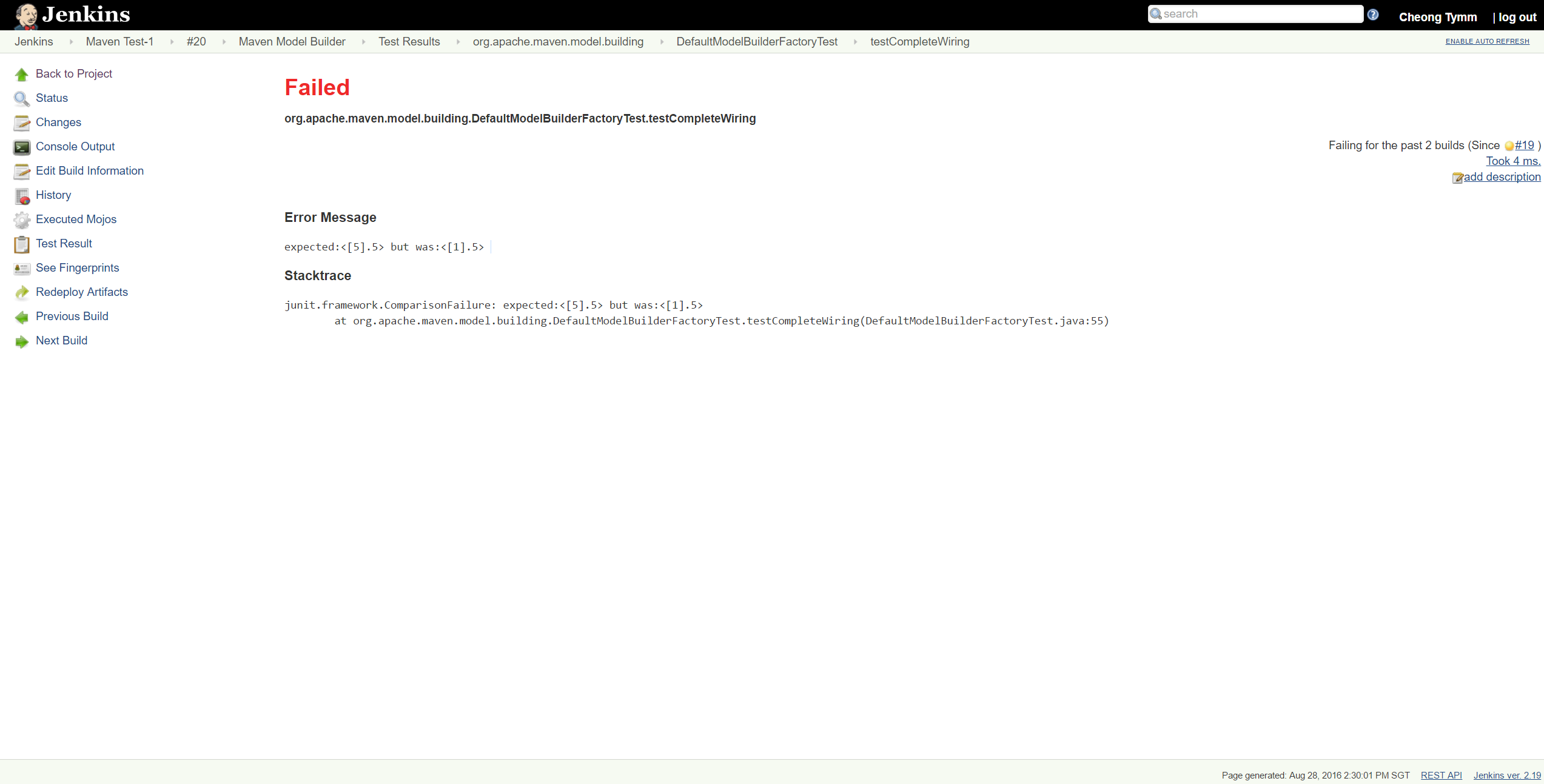


Figure 14: The failed test and its error messages

## Code Coverage

### “Code coverage is a measure used to describe the degree to which the source code of a program is tested by a particular test suite.” (Bergmann, 2015) The more portion of the source code is being covered by the test code, the least likely that program will run into bugs or glitches. This is what is being emphasized on with the new test driven development where test codes are to be written first before the actual code is put down. In this project, the tools that have been tried are JaCoCo and Cobertura. Jacoco plugin cause the build to become unsuccessful while Cobertura successfully build the code coverage report. Figure 15 shows the output from Cobertura coverage tool. Figure 16 figure Shows the Coverage Breakdown by Package produced by Coberture.

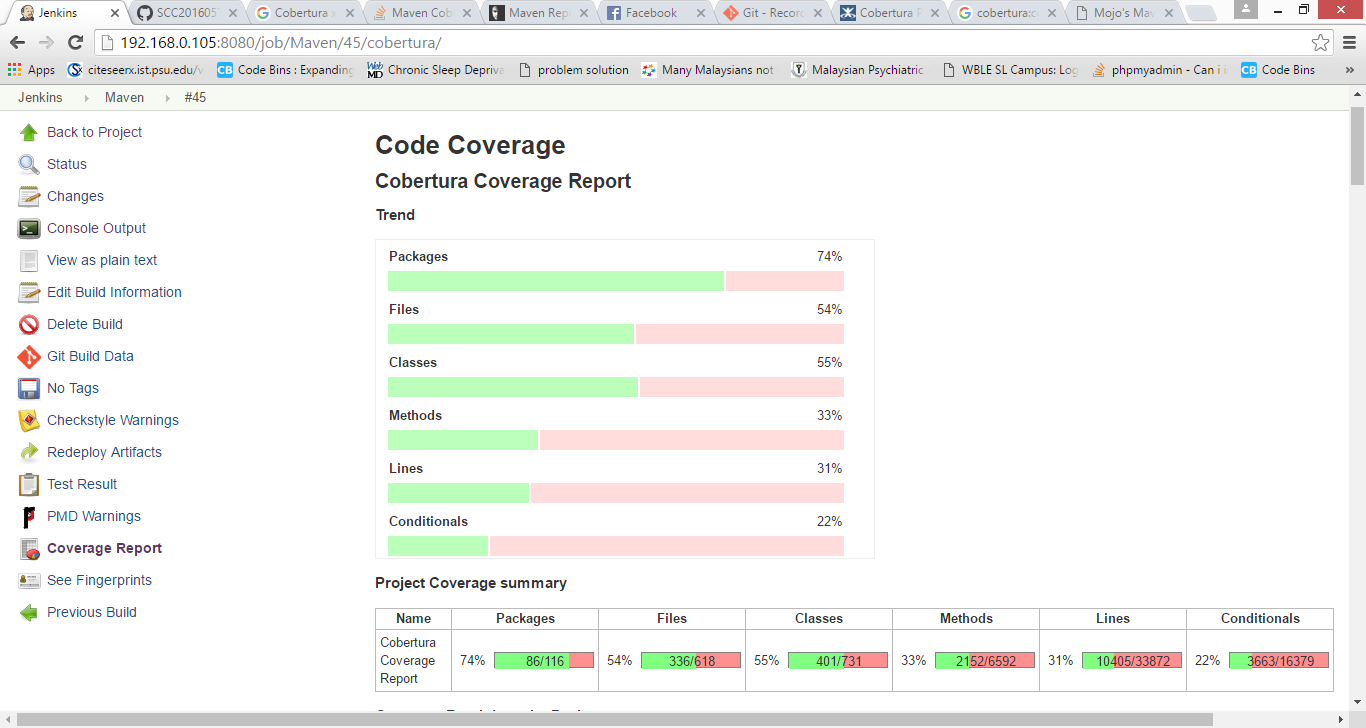


Figure 15: Trend of Coberture code coverage and Project Coverage summary

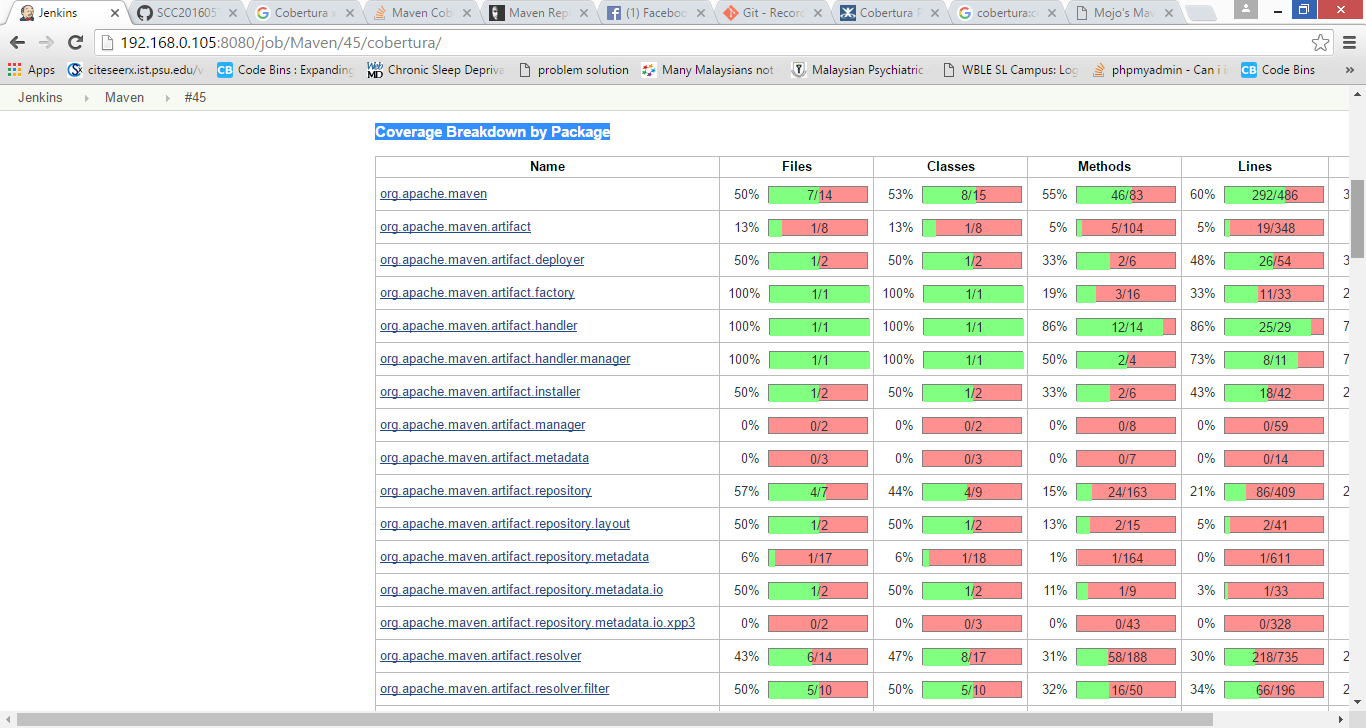
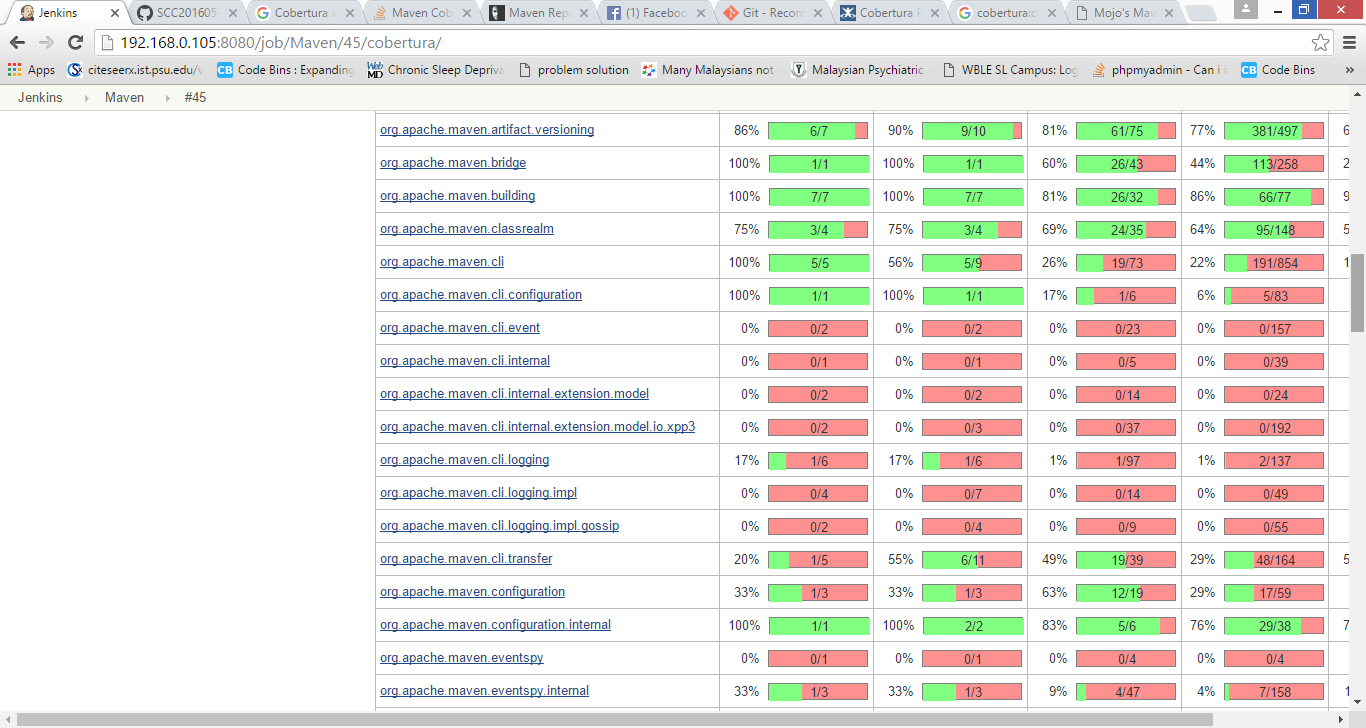


Figure 16: Coverage Breakdown by Package



Cont: Figure 16: Coverage Breakdown by Package

**Goals and options**

clean test -fae -Dskip-test-harness pmd:pmd checkstyle:checkstyle cobertura:cobertura -Dcobertura.report.format=xml

where

clean

Remove all files generated by the previous build

test

Run tests using a suitable unit testing framework

-fae

Fail at end; only fail the build at the end of the build

-Dskip-test-harness

Skip test harness

pmd:pmd

Creates a PMD report (The Apache Software Foundation, 2015)

checkstyle:checkstyle

Performs Checkstyle analysis and generates a report on violations (The Apache Software Foundation, 2015)

cobertura:cobertura

 Instrument the compiled classes, run the unit tests and generate a Cobertura report.

-Dcobertura.report.format=xml

generate a Cobertura report.in xml format.

# Roles of each team member in the project

|  |  |  |
| --- | --- | --- |
| No. | Name | Role |
| 1 | Cheong Tymm | Testing |
| 2 | Lim Jia Hui | Static Analysis |
| 3 | See Hong Min | Testing |
| 4 | Shashindran Vijayan | Static Analysis |
| 5 | Yap Boon Keng | Code Coverage |