Final Documentation

Test History Publisher Plugin for Jenkins

CS427 Team 1

Manan Ganhi	Suna K. Steffen
Min Li	Jessica Mullins
Fahad Mustafa	Wes Panther
Maxie Schmidt	Dongyan Zhang

Dec. 15th, 2013

Table of Contents

1. Introduction

1.1 Purpose

The goal of our project was to create/improve a dashboard to track test results over time. Although Jenkins shows an aggregate view of the results of all the tests in a project, we aimed to provide the user with more details on the pass/failures in a build. Jenkins did not provide historical information at a single test class or test case level in an easy to read view. Our intention was to extend Jenkins to show the result history for an individual test class and test case. To achieve our goal within the time frame of the project we decided to extend an existing Jenkins plugin. This allowed us to leverage existing code and functionality. In addition, the Jenkins code base is very large and by using a plugin we were able to reduce the amount of code we would have to become familiar with in order to complete our tasks.

We chose to use the Test Stability plugin, which was developed by eSailors IT Solutions [1]. This plugin gives details on the history of tests including stability and flakiness. The Test Stability plugin provided an excellent foundation for our project. We were able to use their test history data structure to create a table of test history for a single test, and moved forward from there to accomplish our goals. Some key features we developed include:

- Table of the history of a single test per build that includes commit history and flakiness
- History of the duration of a single test case
- Table history of a single test case per build
- Ability to configure how many builds to show on the single test case table
- History result table view of all test cases for the entire build

1.2 Scope

The scope of our project was defined in our “User Stories” as designed by the T1 team. These stories were estimated and then ranked in order of priority. Some stories were marked at “extra” and fell outside of the scope of the current project. The scope of the project was also limited due to time constraints. We had roughly 7 weeks of development on the user stories. We found challenges in learning the Jenkins and Test Stability code, mocking tests, and external factors such as balancing this project with other classes, all of which contributed to the time constraint on the project. Due to these factors we were unable to complete all of the user stories initially defined in the scope of the project. However, we were able to complete the most important ones to meet our goals for the key features.

1.3 Definitions

- **History of a test**: build pass/fail result and other attributes (such as duration) over past multiple builds
- **ClassResult**: corresponds to a test result of a test class.
- **CaseResult**: corresponds to a test result of a single test method.
- **Stability of a test**: percentage of how often the test fails
- **Flakiness**: the percentage of how often a test flips between pass and fails

1.4 References

- Test Stability plugin by eSailors IT Solutions. (https://wiki.jenkins-ci.org/display/JENKINS/Test+stability+plugin**).**
- Our user interface was modeled after Google's test history viewer (http://googletesting.blogspot.com/).

2. Architecture and Design

2.1 Use-Case View

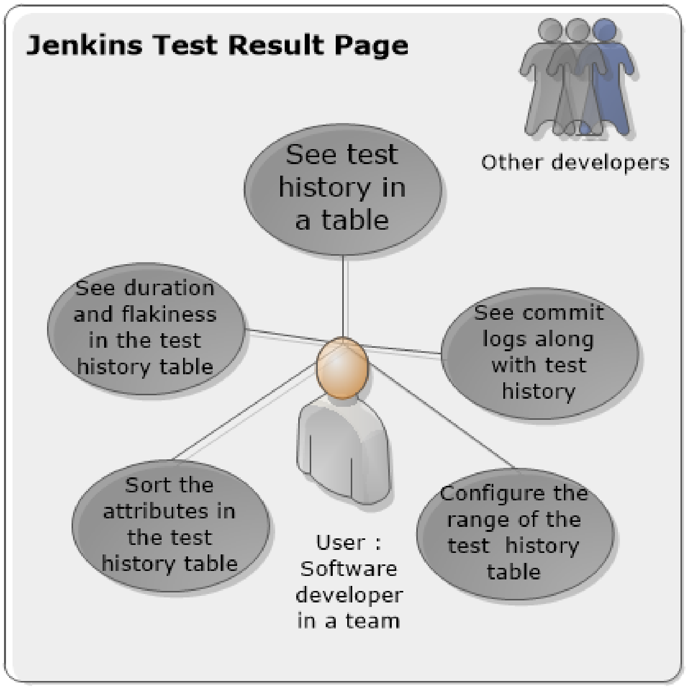

Jenkins Test Result Page
Other developers
See test history in a table
See duration and flakiness in the test history table
See commit logs along with test history
Sort the attributes in the test history table
Configure the range of the test history table
User : Software developer in a team

Figure 1 Use-Case View

2.2 Components of the System

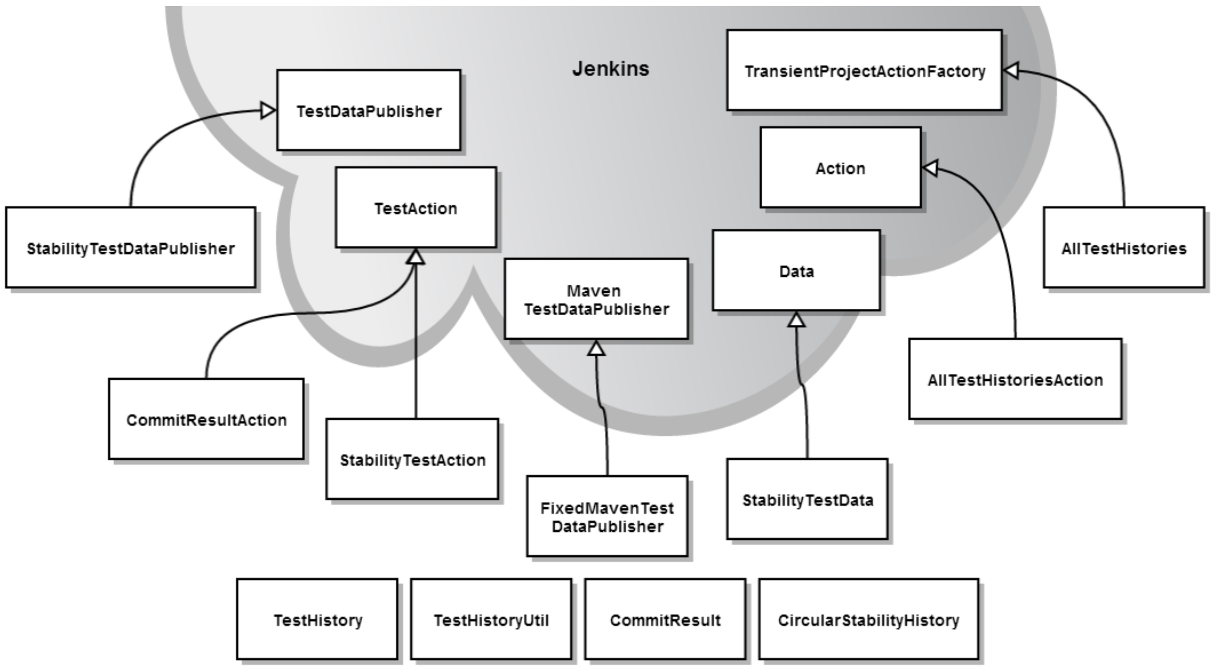


Figure 2 System Components

2.2.1 Classes extended from Jenkins

- **StablityTestDataPublisher.java** extends **hudson.tasks.junit.TestDataPublisher**
  - Populates previous history of tests from current build and returns StabilityTestData which contains that test history
  - hudson.tasks.junit.TestDataPublisher
    - Contributes TestActions to test results. This enables plugins to annotate test results and provide richer UI, such as letting users claim test failures, allowing people to file bugs, or more generally, additional actions, views, etc.
- **AllTestHistories.java** extends **hudson.model.TransientProjectActionFactory**
  - Reads all test history from a file saved from StabilityTestDataPublisher and returns an array list of abbreviated result. The list of results is to be fed to AllTestHistoriesAction to populate an all test history table.
  - hudson.model.TransientProjectActionFactory
    - Extension point for inserting transient Actions into AbstractProjects.
- **StablityTestData.java** extends **hudson.tasks.junit.TestResultAction.Data**
  - It basically returns specific actions to be taken for each given test object, ClassResult and CaseResult.
  - hudson.tasks.junit.TestResultAction.Data
    - Returns all TestActions for the testObject.
- **AllTestHistoriesAction.java** extends **hudson.model.Action**
  - Provides getter methods for an all test history table from the list of abbreviated result received from allTestHistories. It populates the table with associated index.groovy file.
  - hudson.model.Action

    - Object that contributes additional information, behaviors, and UIs to ModelObject (more specifically Actionable objects, which most interesting ModelObjects are.)
- **CommitResultAction.java** extends **hudson.tasks.junit.TestAction**
  - This action puts ClassResult test history in an array list and render on a table with associated summary.groovy file.
  - hudson.tasks.junit.TestAction implements Action
    - index.jelly: included at the top of the test page
    - summary.jelly: included in a collapsed panel on the test parent page
    - badge.jelly: shown after the test link on the test parent page
- **StabilityTestAction.java** extends **hudson.tasks.junit.TestAction**
  - This action initialize test history data structures and calculate flakiness per test class, which is original code from test-stability plugin.
- **FixedMavenTestDataPublisher.java** extends **MavenTestDataPublisher**
  - Original code from test-stability plugin which fixes bugs in MavenTestDataPublisher from Jenkins.
  - hudson.maven.MavenTestDataPublisher
    - Augments SurefireReport by executing TestDataPublishers.

2.2.2 Classes not extend from Jenkins

- **TestHistory.java**
  - Represents a test history for a single test(CaseResult), which is stored in map with build number as key and single test result as value. This class contains a private class TestHistoryData to represent a single test result.
- **TestHistoryUtil.java**
  - This class constructs a previous history from current CaseResult
- **CommitResult.java**
  - . A data class that contains author ID, timestamp, commit message.
- **CircularStabilityHistory.java**
  - A linked list of test result per ClassResult. It contains build number, build result, and commit result

3. Future Plans and Features

3.1 Future Plans for the Plugin

If we had more time with the project we would continue to implement user stories (see 3.3.) and improve our overall code. We had plans to integrate our data structures with TestStability's data structures and due to time constraints and more pressing tasks we were unable to complete that goal.

3.2 Future Code to Implement

There were several updates to the code to help organize the separate features added to the plugin as a part of the final project. The `TestHistoryData` class inside of the added `TestHistory` class contains a stub for storing the associated `CommitResult` class data. Once the commit data is stored with the other data in the `TestHistoryData`, it is easily accessed by the GUI scripts.

3.3 Extra User Stories

The following list provides descriptions of "extra" user stories that were drafted as a part of the CS 427 final project. These extra user stories provide several additional features for the plugin that may be implemented in the future.

- T1_S8: As a user, I can only display flaky tests (with marks or be highlighted) in the tests list so that I could find flaky tests easily.
- T1_S9: As a user, I can see the history of tests grouped by class/package so that I can better assess what area of the project needs to work on or where changes are being made.
- T1_S10: As a user, I can display a test run time, test failure/pass info as a graph.
- T1_S11: Add tags for test types (new test, regression test, etc.) so the plugin can display test statistics specific to these tag types.
- T1_S12: As a user, I can see a package hierarchy(exploring) window on the side along with the test result so that I will have better understanding of the test result.
- T1_S13: As a user, I can see the test code of failed assertion and the change/history of the associated source code from the last successful build so that I can quick review the bug and judge the problem.
- T1_S14: As a user, I can see only failed build in the history of a single test.
- T1_S15: As a user, I can narrow down tests by regexp filter in the history of all tests.
- T1_S17: As a user, I can only see tests by author in a package hierarchy window.

4. Appendix

4.1 Installing and Running the Plugin

1. Building the Test-Stability Plugin

In this section, we will build our test stability plugin from the source.

First, we need to get our plugin from subversion. You can do this by opening the terminal and issuing the command:

svn co https://subversion.ews.illinois.edu/svn/fa13-cs427/_projects/T1/test-stability/trunk/

This will create the "trunk" directory in your current folder. Go into this directory with the command:

cd trunk

Once in this directory, we can build the plugin by using the command:

mvn install

This will create the file "test-stability.hpi" in the target folder which is the plugin and all of its required files.

2. Running Jenkins and Installing

Once the plugin has been built, we need to install it in Jenkins. First, open and terminal and navigate to the folder with the jenkins.jar file. Start Jenkins with the following command:

java -jar jenkins.war

Open up a web browser and go to the http://localhost:8080/pluginManager to see the Jenkins plugin manager. At the top, you should see something like this:

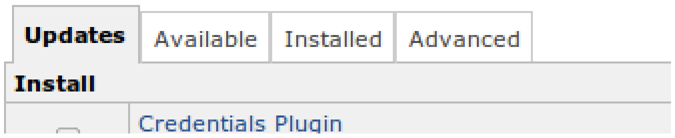
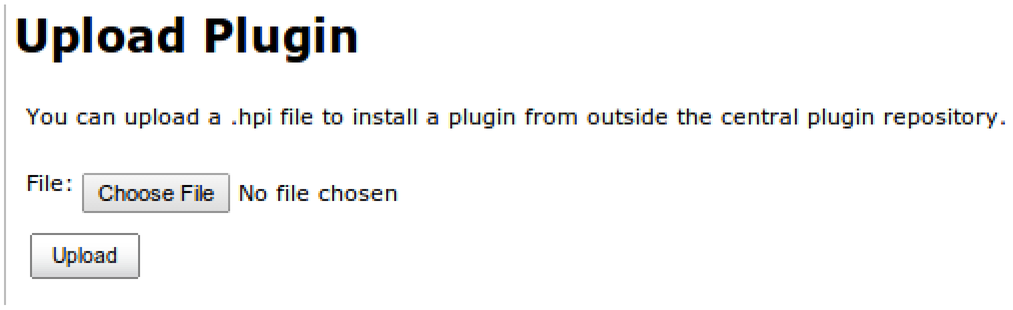


Figure 3 Plugin Manager page for Jenkins

Click on the "Advanced" tab. Down the page, you should see


Upload Plugin

You can upload a .hpi file to install a plugin from outside the central plugin repository.

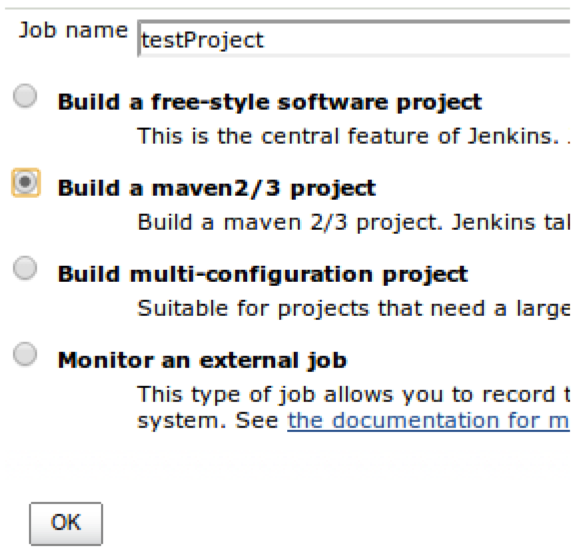
Figure 4 Upload Plugin page

Click "Choose File" and navigate into the target directory and find the "test-stability.hpi" file. Now click upload, and wait for the plugin to install. Once this is finished, go to http://localhost:8080/restart in the web browser and wait for Jenkins to restart. You should now have Jenkins running with the plugin installed.

3. Adding and Building a Test Project

Now that Jenkins is running and the plugin is installed, we need some kind of test project we can build to show the changes the plugin has made. In a web browser, go to http://localhost:8080 to get to the Jenkins home page. Here you can add your own test project, or use the our team uses by following the steps here.

On the sidebar to the left, click on "New Job". This will bring you to the page to create a new job. Fill out the job name with some name such as "testProject", and click on the "Build a maven2/3 project" bubble. The page should look like this:


Job name testProject

( ) Build a free-style software project
This is the central feature of Jenkins. .

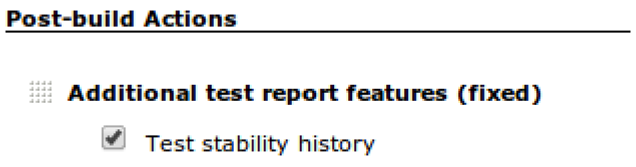
(●) Build a maven2/3 project
Build a maven 2/3 project. Jenkins tal

( ) Build multi-configuration project
Suitable for projects that need a large

( ) Monitor an external job
This type of job allows you to record t
system. See the documentation for m

Figure 5 Build new test job

Now click "OK" and you will be taken to the job configuration page. Under "Source Code Management" click the bubble for Subversion. In the Repository field, enter https://subversion.ews.illinois.edu/svn/fa13-cs427/_projects/T1/testProject/ and click somewhere on the screen. This tells Jenkins where to find our test project. Under the "Build Triggers" section, make sure all of the options are unchecked. Under the "Build" section in the "Goals and options" field, add the word "test". At the bottom of the page under "Post-build Actions", click on the "Add post-build action" drop-down menu and choose "Additional test report features (fixed)". Check the box next to "Test stability history" that appears. The page should look like this:


Post-build Actions

Additional test report features (fixed)

[✓] Test stability history

Figure 6 Test stability checkbox

Click "Save" at the bottom of the page. You will now be taken to the testProject page. From here you can verify with the tests below.

4.2 Testing

4.2.1 Manual Tests

- **Test #1 - Empty Test History**

Here we test that the test history page is displayed and that it works in the case that there are no tests yet run. When the project is first added, there should be nothing in the build history on the left side of the page. Verify that it looks like this:

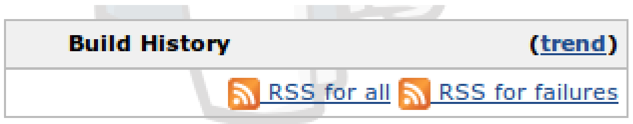
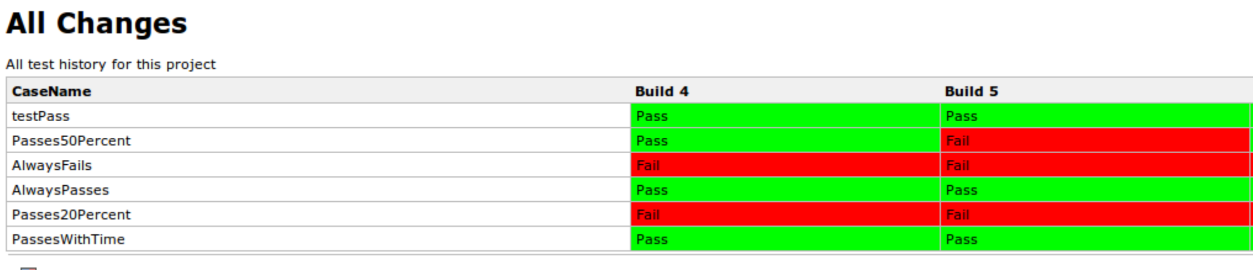


Figure 7 Clean build history for new project

The plugin should have added a button that says "test history" on the sidebar. Verify that this button exists. Click this button and you should be taken to the test-history page for the project. The text "Please build the job first" should appear on the screen. This shows us that the plugin is working, and correctly detecting that there is no test history.

- **Test #2 - Populated Test History**

Now we want to test that the history page is correctly populated. On the left sidebar, you should see a button with the text "Build Now". Click this button. A build should appear under the build history on the bottom left side. You can build any number of times. In this example, we will click the "Build Now" button again so that we have two builds to display. After this is finished, click the "test history" button the left side. This will bring you to the test history page. Verify that the builds are displayed with some results. It will look something like the following:



All Changes

All test history for this project

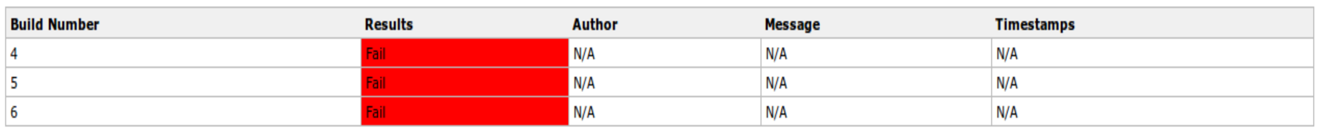
CaseName	Build 4	Build 5
testPass	Pass	Pass
Passes50Percent	Pass	Fail
AlwaysFails	Fail	Fail
AlwaysPasses	Pass	Pass
Passes20Percent	Fail	Fail
PassesWithTime	Pass	Pass

Figure 8 Test History page for the plugin

The build numbers will probably be different, and the tests results may not be exactly the same. Just verify that the table is there and populated, with Fails being red and Passes being Green.

- **Test #3 - Populated Class Results**

Now that we have some builds, we can verify that the plugin will show us a table of the test result for a class. On the left side under "Build History" click on the link next to #2 to show the results for the second build. On this page, click on "Test Result" on the left side of the page. Here, click on the test project link under "Module". This will bring us to the test report page for the project. Under "All Tests" choose one of the classes. For this, I will choose "TestClass.java". This will bring us to the test page for that particular test. Verify that a table exists with each of the builds you scheduled earlier. The table should look similar to this:



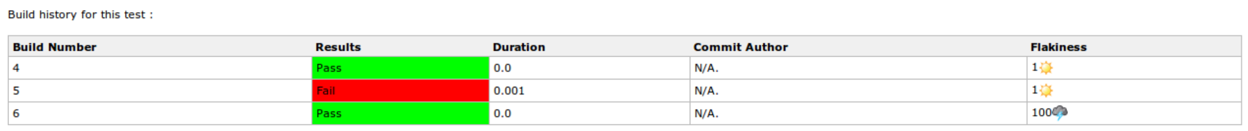
Build Number	Results	Author	Message	Timestamps
4	Fail	N/A	N/A	N/A
5	Fail	N/A	N/A	N/A
6	Fail	N/A	N/A	N/A

Figure 9 Class Result page for the plugin

Your table will have different build numbers. The Author, Message, and Timestamps columns are all for commit results corresponding to each build. Since we have not committed anything to the test project, these should all be "N/A".

- **Test #4 - Populated Case Results**

Similar to the last test, this test will verify that a table is created for each individual case. After the previous test, we should be on the class results page. At the bottom, you will "All Tests". Under this, there is a list of specific cases. Choose one of these and click on the name. I will choose "Passes50Percent". This will bring you to the page showing the history for this specific test case. Verify that a table exists with all of the build numbers you added earlier and the results. The table should look similar to the following:



Build history for this test :

Build Number	Results	Duration	Commit Author	Flakiness
4	Pass	0.0	N/A.	1
5	Fail	0.001	N/A.	1
6	Pass	0.0	N/A.	100

Figure 10 Case Result page for the plugin

Your table should look similar, but will probably have different values and only two builds.

4.2.2 Automated Tests

Here we look at running our automated tests from the terminal. First, we need to get our plugin from subversion. You can do this by opening the terminal and issuing the command:

svn co https://subversion.ews.illinois.edu/svn/fa13-cs427/_projects/T1/test-stability/trunk/

This will create the "trunk" directory in your current folder. Go into this directory with the command:

cd trunk

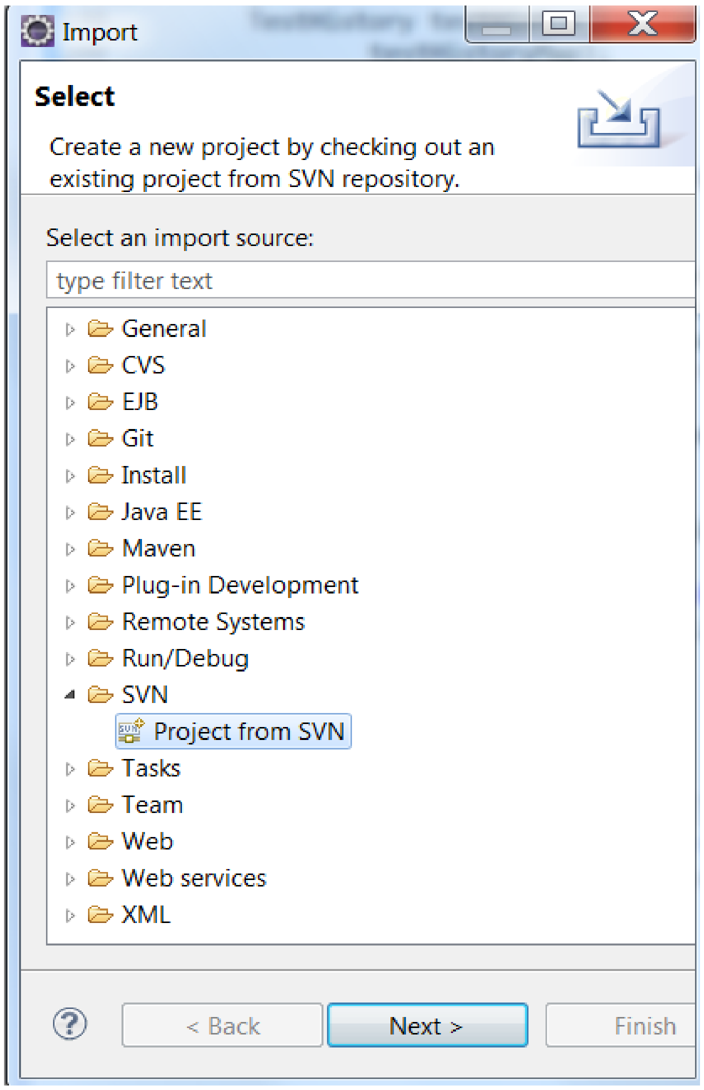
Once in this directory, we can run only the tests by using the command:

mvn test

This should run the junit tests without building the plugin.

4.3 Setting Up the Project in Eclipse

1. In Eclipse, do Import > Project from SVN


Import

Select

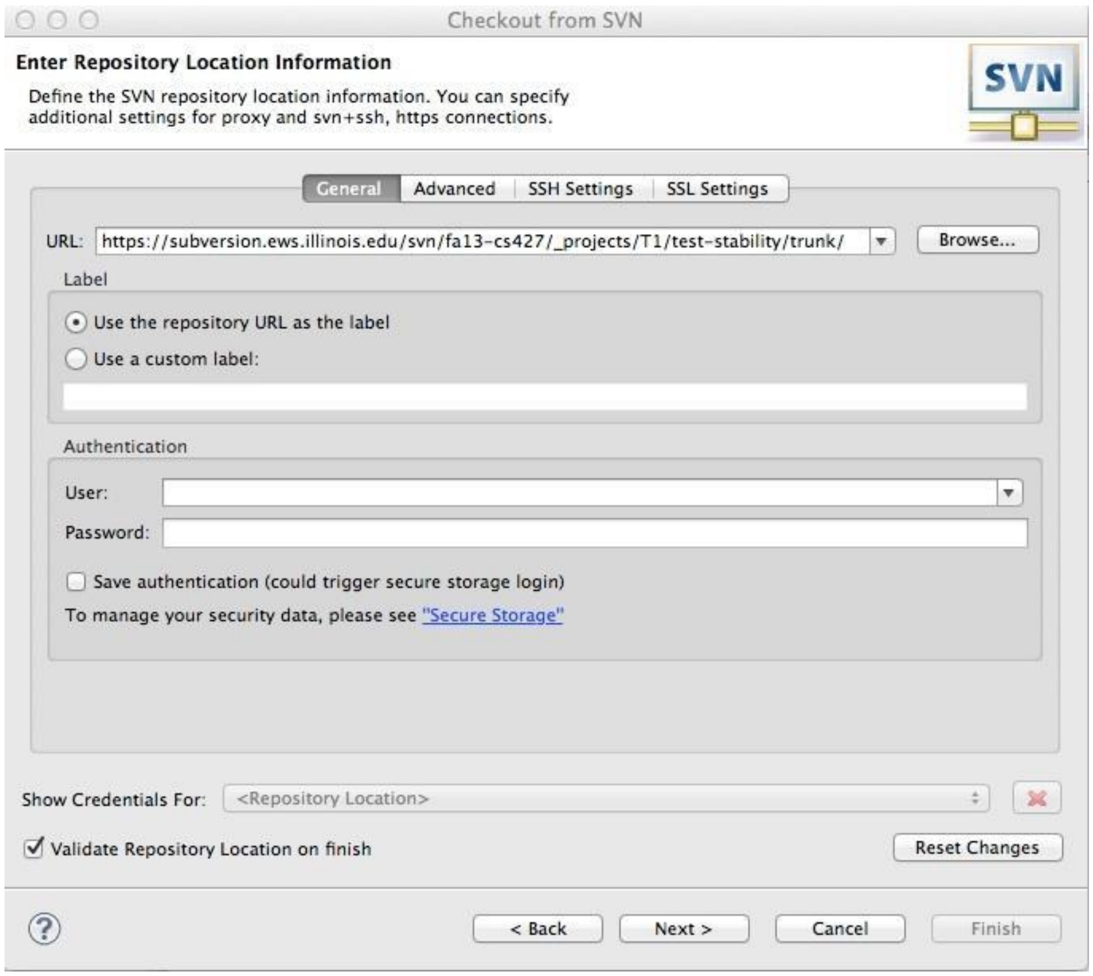
Create a new project by checking out an existing project from SVN repository.

Select an import source:

type filter text

Figure 11 Import page for Eclipse

2. For repository location use: https://subversion.ews.illinois.edu/svn/fa13-cs427/_projects/T1/test-stability/trunk/



Enter Repository Location Information

Define the SVN repository location information. You can specify additional settings for proxy and svn+ssh, https connections.

URL: https://subversion.ews.illinois.edu/svn/fa13-cs427/_projects/T1/test-stability/trunk/

Label

(●) Use the repository URL as the label

( ) Use a custom label:

Authentication

User:

Password:

[ ] Save authentication (could trigger secure storage login)

To manage your security data, please see "Secure Storage"

Show Credentials For: <Repository Location>

[✓] Validate Repository Location on finish

Figure 12 Check out page for Eclipse

3. Use defaults, select Next, Finish.
4. If you already have something in your workspace called "test-stability" rename the project to something else. Click "Finish"
5. You should now have the “test-stability” set up in your eclipse

** Documentation Style Reference :
http://www.ts.mah.se/RUP/wyliecollegeexample/courseregistrationproject/artifacts/analysisndesign/sadoc.htm
** Jenkins class information :
http://javadoc.jenkins-ci.org/