UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

Team MIA Documentation

by

Yuto Otaguro, Kaushil Patel, Hanzhao Deng, Yushuo Lin,
Daniel Zhang, Ran Meng, Matthew Pifko, Alexander Merza

Documentation for the Jenkins Plugin developed
as a part of CS427

Fall 2015

Contents

1 Abstract **2**

2 Architecture **3**

2.1 Data Model / Storage 4

2.1.1 Data Storage 4

2.1.2 Data Model 5

3 Design **6**

3.1 Display Newly Passing and Failed Tests 6

3.2 Show Code for Specific Tests 6

3.3 Show Comment Tags for Test 7

3.4 Use History to Rank the Tests 7

3.5 Rank Tests by User Set Priority 7

3.6 Allow Graph Color Customization by User 8

3.7 Print Error Type 8

3.8 Filter Test Results 9

3.9 Select Specific Builds to View 9

4 Usage **10**

5 References Used **13**

Chapter 1

Abstract

Testing is a vital aspect of programming. As projects grow larger, test results become increasingly muddled and information that could be gained becomes lost due to the sheer volume of tests that exist. Furthermore, since continuous integration is necessary for any large-scale software project, a robust tool which allows developers to do analytics of test results during continuous integration is urgently needed.

Jenkins is an ideal working environment for continuous integration, and there are many plugins of Jenkins available on its official Webpage. One of the plugins, Test Result Analyser, which demonstrate the test results and corresponding analytics of each build, is a potentially helpful tool with lots of rooms for improvements. In this project, our team added many new features so as to enhance the usability and to make the analytics more demonstrative. Our goal was to allow users to do more customization and to make the analytics more interactive, informative, and efficient.

Chapter 2

Architecture

Figure 2.1 depicts the high-level system architecture. The system will be constructed from multiple distinct components:

- **User Interface**: The table that displays plugin's data and allows user to filter it. It also supplies the chart interface with this data.
- **Chart Interface**: User can select data from the table and generate chart with it. They can also personalize with the charts such as setting colors.
- **Data model**: The tree model that organizes the raw data and make it usable for the user interface.
- **Data Storage**: The interface for storing raw data supplied by Jenkins.

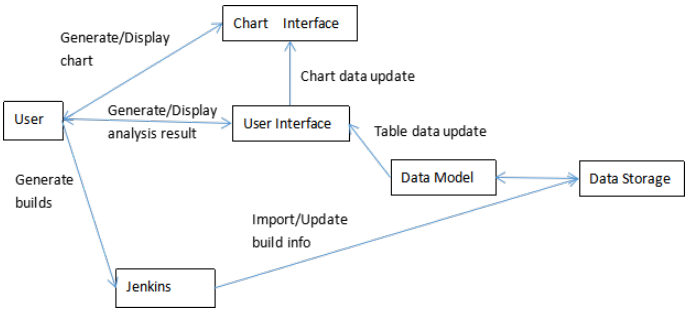

FIGURE 2.1: High-level architecture

System Operation

Figure 2.2 is the typical sequence of events that occur during a session. User make a build and Jenkins will supply the build info for the plug in. The plug in displays this information in a table. User can then select data to generate charts. The chart interface will get data from the table and display charts.

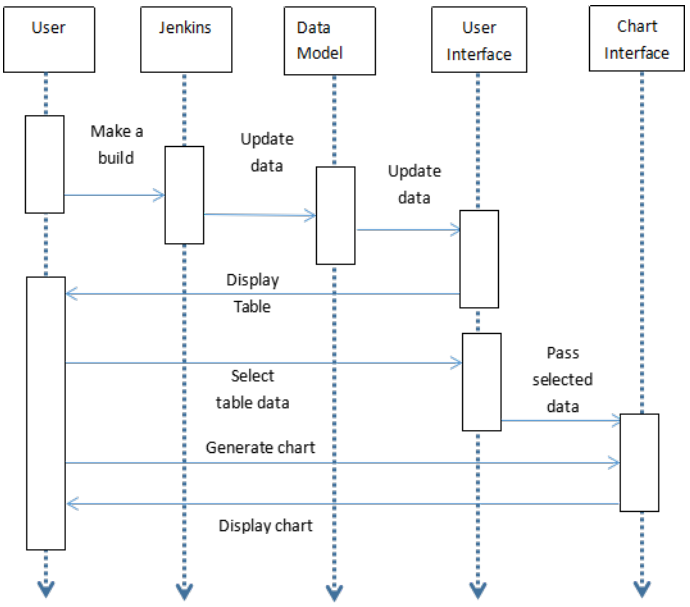

FIGURE 2.2: Sequence of Events

2.1 Data Model / Storage

Data storage contains Junit test data for every build. Data model constructs a tree based on the data in the data storage.

2.1.1 Data Storage

The data model involves 4 classes:

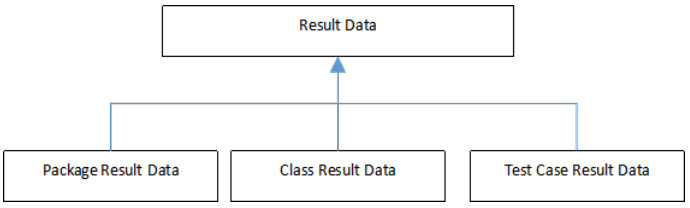

FIGURE 2.3: Sequence of Events

2.1.2 Data Model

- Result Data - the abstract class for the other 3. It contains basic information about the test/class/package.
- Package Result Data - Result data for a particular package.
- Class Result Data - Result data for a particular class.
- Test Case Result Data - It contains the data for a particular test in a particular build.

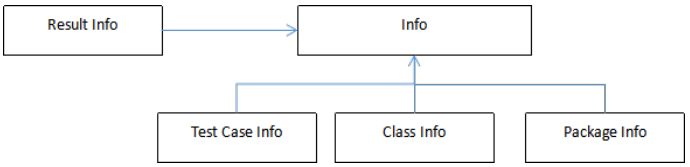

FIGURE 2.4: Model of the Data

Chapter 3

Design

3.1 Display Newly Passing and Failed Tests

This feature displays newly passing and newly failing tests.The functionality is implemented in Java and is primarily located in "Info.java"

The `getBuildJson` function in "Info.java" loops through all the "buildResults" and compares the previous build result with the current build result to determine if a test is newly passing or newly failing. This result is then saved and updated into the current build result's "status" field. The count or number of newly passing and newly failing tests are also stored in this process.

The table will display the newly passing and newly failing tests to the user based on what is saved in status.

3.2 Show Code for Specific Tests

This feature changes file names displayed in the test results table into hyperlinks to their source code. This is implemented in both Java and Javascript, primarily in "PackageInfo.java", "test-result-analyzer-template.js" and "get-url.js".

The `getFilePath` function in "PackageInfo.java" executes a 'find' bash command that reads the local file path via a BufferedReader. This file path is added to the description field of the `classInfo` JSON object which allows us to access the data on the client side.

The File Path is parsed and displayed to the user in "test-result-analyzer-template.js" by wrapping the class names in `<a>` tags. The `href` value is set using a Handlebars helper function called `getURL` which references to the function in "get-url.js". The result is an

easily accessible hyperlink to each class's source code to save developers time of having to navigate to the file themselves.

3.3 Show Comment Tags for Test

Written in ResultInfo.java, the file is retrieved and parsed to see whether any of the relevant lines have "@TAG:" contained within a comment structure. If the tag exists within a function, this method will extract it to a variable that holds the tags for the function, and if it exists outside of it, it will be extracted to a variable that holds the tags for the entire file.

On the webapp, the table is generated by a library called handlebars. The outputs that are created from ResultInfo.java is passed into handlebars to be populated in its appropriate place; if the tag exists in a function, the function description will be updated and if the tag exists outside, the test suite description will be changed.

3.4 Use History to Rank the Tests

This feature allows the user to rank by history. History here means for each filter user selected, the table will show them in decreasing order based on how many this selected filter occurs among all the builds. For example, if user wants to rank by "Failed", the table will show the packages in decreasing order based how many total failure builds they have among all builds for this project. The feature is mainly implemented in Java and primarily located in "Info.java" and "JsTreeUtil.java".

All information for each test case/class/package is collected in Info.java. Because Info.java is the class that traversal all builds for each test case/class/package, all information is collected here. Once they are all calculated and stored in the attributes, JsTreeUtil.java can get this information by obtaining the Json object for a particular test case/class/-package class. Once it get this data, it will be sorted using Bubble sort and display them in decreasing order in the final tree structure that will be used to generate the table.

3.5 Rank Tests by User Set Priority

This feature allows the user to reorganize the testcases according to priorities set by the user. The primary files in which this feature is implemented were ResultInfo.java and JsTreeUtil.java. Furthermore, Info.java, ClassInfo.java, and TestCaseInfo.java were also modified. In particular, a priority field was added to the JSON tree, which is used to

store the priority for each testCase (can be extended for testClass or testPackage) as provided by the user (the default value is -1).

In the ResultInfo.java file, the getFilePath functions excutes a command line "find" command given a from directory and the name of the file to find the test class file. The from directory is chosen to be " /.jenkins/jobs/" as that is where Jenkins stores the workspace. The file name is found by finding the class name and attaching ".java" to it. The found filepath is then used by the setPriorites function along with the TestCaseInfos associated with the the current testClass. The setPriorities functions then opens the files and parses it line by line. If a line contains the phrase "Priority (number)", then the line immediately after is checked for containing one of the testCase names. Depending on the matching testCase, it's priority is set to user specified one.

Once the priorities are set, when the JSON tree is formulated by the getJsTree function in the JsTreeUtil.java which further calls the createJson function, the children are sorted based on the priorites set earlier. This is done via the sort_by_priority function which takes in length and childrens as inputs. The sort is simply done by swapping elements.

This is the default configuration that is displayed by the plugin.

3.6 Allow Graph Color Customization by User

The files that were utlized in this feature are "chart-generator.js" and "color-change.js". They will be referred to from this point on as chart-generator and color-change respectively.

There are four color boxes that can be accessed on the webpage of the plugin to edit the graph colors of "Passing", "Failed", "Skipped", and "Total". These values are stored as a JSON cookie on the client computer through the function `setCookie` in color-change. At this point, unless the cookie is removed, the color scheme for that project will remain in place each time the user returns.

When the graphs are generated, the color cookie is retrieved via the `getColors` function in color-change and is passed into chart-generator in the `generateChart` function. If no such cookie could be found, the colors default to the standard green/red color scheme to represent passing and failing.

3.7 Print Error Type

This feature allows for the user to instantly view what went incorrectly in a test which saves the time of looking for it and also keeps it in a place so that it may be compared to later.

There are two files that were modified to allow for display of errors in a test. The first is "test-result-analyzer-template.js" located in the webapp folder (referred to as analyzer-template from now on) and "TestCaseResultData.java" located in the java folder.

To allow the changes to be seen in the front end, the `applyvalue` helper function from analyzer-template adds the failure data to a string so that it may be shown in the table. To pass this failure data to the front end TestCaseResultData.java includes the failure data in the JSON tree constructed if there is a failed test. We see the results for this in the table on the class level after the packages and classes are expanded upon.

3.8 Filter Test Results

This feature allows the user to filter and display test results. This can be done by choosing either "SUCCESS", "FAILURE", "NEWLY PASSED", or "NEWLY FAILED" in the drop down box on the Jenkins page. The feature is mainly implemented in Java and primarily located in "JsTreeUtil.java".

All the test case/class/info statuses must be updated before generating the tree structure in JsTreeUtil.java. This process ensures every build will be labeled correctly. When the user selects a particular filter on the user interface, his or her selection will be recorded in the variable "teststatus", and TestResultsAnalyzerAction.java will get the value for this variable and pass it to JsTreeUtil.java. Once JsTreeUtil obtains the value for the desired status, it will use this variable to traverse all the builds for every test case/class/package and only select the builds where the status matches the test status value and put them in the final tree structure.

3.9 Select Specific Builds to View

This feature allows to user to specify which builds you want to view. This can be done by entering the build numbers separated by commas, or by entering a range of build numbers (ex. "1-7") in the "Select Specific Build" textbox on the Test Results Analyzers plugin page. The default value is "All" which displays all the builds.

This feature implemented by making changes to multiple files: 1) index.jelly under TestResultsAnalyzerAction directory, which added the textbox to the plugin page. 2) testresults.js, through which the value entered in the textbox is passed on to the java backend. 3) TestResultsAnalyzerAction.java, where the input is parsed by the getSelectedBuilds function. 4) JsTreeUtil, where the correct JSON object is returned by the getJsTree function based on the selected buils.

Chapter 4

Usage

First the user must enter the job they wish to view test results for. At this instance, the user should see something akin to Figure 4.1.

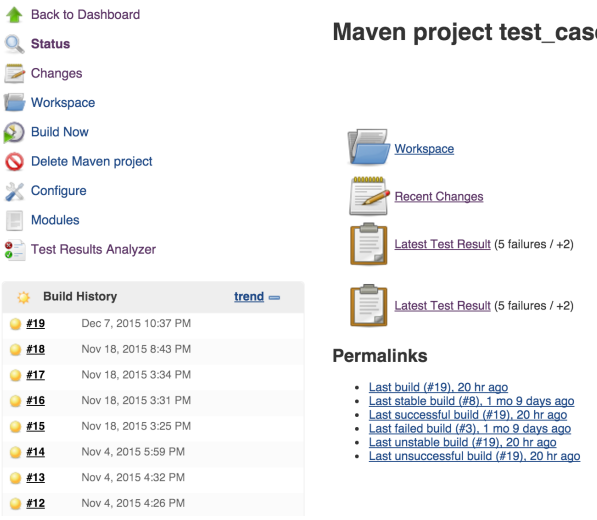

FIGURE 4.1: Sequence of Events

From this step, the user cannot immediately access the plugin because the user has not selected a build they wish to see. Consequently, the user must first click on "Latest Test Result." If there are two such options, select the first of the two. The user should now be in a page that resembles Figure 4.2.

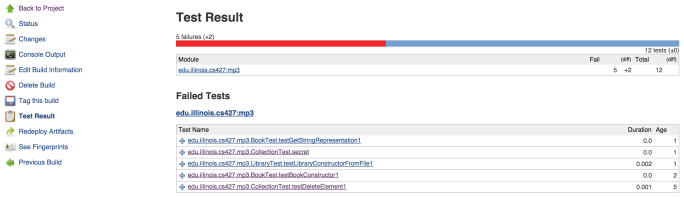

FIGURE 4.2: Sequence of Events

At this point, the user must select the module they wish to view the test results for. Using the example from Figure 4.2, this would mean the user should click on the module "edu.illinois.cs427" right underneath the Test Results bar. Now that a module was selected for examining, the user may return to the main job page in Figure 4.1 and select "Test Results Analyzer" to enter the page that will resemble Figure 4.3. To construct the build table, the user should click on "Get Build Report." Figure 4.3 is essentially the main navigation point for this plugin once the builds have been loaded.

FIGURE 4.3: Table showing test results

When at Figure 4.3, there are a variety of actions the user can take. The first is a filter that will filter the table so that only certain build stats will be displayed. Selecting the appropriate status from the "Last Builds and show" will allow the user to access this feature.

There is also an option to "Select Specific Build" which allows the user to input specific builds they would like to view. Users can specify additional builds by seperating them with commas, or a range of builds seperated by hyphens (Ex. "1,2,5-7").

The user also has the option to rearrange the test cases within a particular testsuite by adding a comment such as, "/*Priority #*/" to the line before the testcase declaration in test suite file. This will make it so that the particular test case appears at the numbered position in the table. Furthermore, the user has the ability to add a "@TAG" comment anywhere in the test suite file and it will appear alongside the name of the testcase or the test suite based on where the the tag is placed. It appears next to the testcase name if it were placed within the body of the testcase and next to the test suite name if it were placed elsewhere.

The block of text that is visible on Figure 4.3 is the error that caused the test case to fail. From this state, the table can be expanded to view more concise details about the testing suite. If the row is able to be expanded, the information that is displayed under the "Build Number" column displays the summary of the testing suite. When the table displays the smallest unit of information (ie the test methods), it will show the expectation failures. An example of what should be seen is given in Figure 4.4.

One important thing to take note of in Figure 4.4 is the hyperlinking of the test suites. Clicking on the suite will redirect the user to the file that contains the test suite, so the user will be able to view the file on their browser for easier access.

FIGURE 4.4: Test Results Table Expanded

Going back to the page represented by Figure 4.3, if the user chooses to "Generate Charts", graphs that resemble Figure 4.5 should be generated. These graphs represent the information displayed on the table, but in a format that is much easier to digest. If the user wishes to change the color scheme for the graph, they simply have to click on "Edit Colors" as shown on Figure 4.5, select the colors they wish, click "Save Colors", and finally refresh the page.

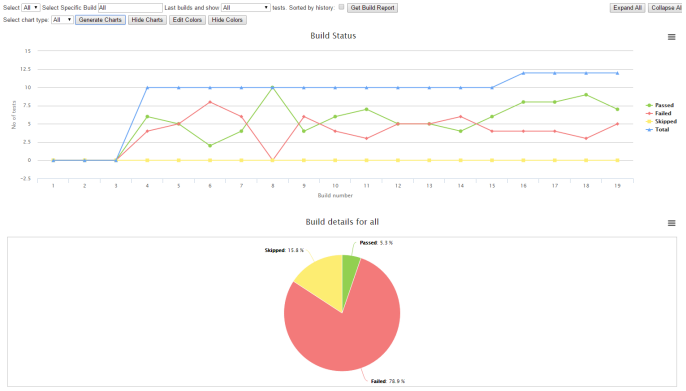

FIGURE 4.5: Test Results Table Expanded

Most of the features are centered around supplying information to users who desire to know more about the specifics, and yet at the same time the summaries that are offered have plenty of implicit information stored within them as well.

Chapter 5

References Used

https://www.cs.drexel.edu/~dpn52/Therawii/design.pdf

http://www.atilim.edu.tr/~dmishra/se112/sdd_template.pdf