Test Results Analyzer Plugin Extension

Varun Goverdhan, Sun Woo Kim, Chen Zhang, Shaan Iqbal, Patrick Sapin, Chris Fillak, Yisong Yue

CS 427: Software Engineering I, Department of Computer Science, UIUC

Darko Marinov

Date: December 8th, 2015

Table of Contents

1. Brief Description of the Project (pg 1)
2. Architecture & Design (pg 2)
3. Usage (pg 5)

1. Description of the Project

The Test Results Analyzer Plugin has been extended to improve the user experience. Previously, the user had to face difficulties navigating through the project in order to execute the plugin and was limited to solely viewing the build reports as charts and diagrams with no extra functionality. The project aims to fix the navigation issue that may prevent other users from using the plugin and to allow more flexibility for the users to view their build reports.

New functionality included specific build comparisons, displaying test status changes between builds, better integration with existing Jenkins test report pages by linking test outcomes, and other usability tweaks.

1.1. List of User Stories of the Project

S01: Display names of each newly passed/failed test on the main plugin page.

S02: Enable user to specify a certain number of past builds to display. Also, fix current filtering bug in plugin.

S03: Allow user to click on test status from plugin's results table to view associated Jenkins Test Report page directly.

S05: Allow user to compare test results and display differences in pass/fails for each run test and new/removed tests, etc.

S09: Allow user to show selected builds that are in non-consecutive order.

S10: Fix a bug in the current plugin to get build report successfully without having to go into the modules for the project.

S11: Display relevant source code for test cases.

2. Architecture & Design

Architecture:

S01 - We added functionality to find test results that were different between the two most recent builds of a project and displayed a list of these differences on the Test Results Analyzer Plugin page.

S02 - We added functionality on the plugin front page to show a condensed result of all the past builds. The number of total passed builds, the number of total failed builds and the number of the total N/A builds are calculated and shown as a table in a tree manner. It gives the user a general idea of all past builds conditions. The functionality as well as the tests are all implemented using javascript. A new JSON object is constructed based on the result of the package and used a chart generator to generate the table.

On the main plugin page we changed the options of the drop down menu for the number of builds to display. In the Index.jelly file, we expanded the original limited options of 10, 20, and 30 to every number between 1 and 30.

Our other task for this story was implementing the status filtering feature that had been laid out in the existing plugin but not properly implemented. This feature allows the user to choose a filter, passed, skipped or failed, and displays the test results that only fit that filter. This allows the user to view a clearer table of results based on his or her preference.

S03 - We also added functionality in the Handlebars table generator to make each result status displayed in the table a clickable link to direct the user to relevant Jenkins test result page for the specific test case or hierarchical list.

S05 - The only changes to the front end were made on the main plugin page. Dropdown menus were added to allow the build numbers to be provided as parameters to test comparing functions. This allows the user to compare two specific builds. Another front end modification was the adding of text that would tell the user about individual test cases according to the build numbers they provided. The backend was implemented entirely in JavaScript that takes in necessary parameters and compares test results. All of the implementation relied on the JSON build results object that is returned from the Java code. It contains all of the relevant information about the tests, and many of the computations were done using this object.

S09 - We added the checkboxes next to the build numbers in the table such that users can compare an arbitrary number of builds by checking the desired boxes and clicking the “Compare

Checked Builds" button. Implementing the checkboxes and its functionalities were entirely done in JavaScript. In the selectdiffresults.js file, we get the response object and remove the builds and results that weren't checked.

S10 - We made a change in the functionality of the sidebar link for the plugin when no module has been selected for the project. Originally it would display the same analyzer options, but nothing happened when anything was clicked. The test results are all linked to modules so without a module there was no defined behavior. This was reported multiple times as a bug to the plugin author but never really addressed. With our modifications, when there is no module selected the plugin now presents the user with a list of modules to analyze, where clicking a linked module name directs the user to the page where the user can generate builds and view the reports for the newly selected module.

S11 - A "code" button is displayed in each table row header cell in the build report. On clicking the button, the user will see the source code associated with that particular test render in a box at the top of the page. The functionality as well as the tests are implemented in javascript in testcodes.js. After "Get Build Report" is clicked and the table is constructed, the plugin asynchronously retrieves the source code file for each test case. This triggers a callback that after retrieval extracts the code snippet of a test case as an array of strings containing the content of each line of code. The array is mapped to the associated test case name in a javascript object. This object will be later used as a dictionary for looking up code for each test case.

Design

S01 - Getting and displaying the result differences between builds was done in javascript with a call to getDiffs from the populateTemplate function in testresults.js. In differentresults.js, getDiffs iterates through the plugin's JSON object containing the test result information. The result status of selected builds are compared in addDifferentResultsToList and a string describing the change is added to an HTML list to be presented to the user on the main plugin page.

S02 - The backend was implemented by modifying the HTML select element with `id="noofbuilds"` representing this dropdown field. We replaced the 10, 20, 30 hardcoded options in TestResultsAnalyzerAction/index.jelly with 30 new option tags for the intermediate values 1 through 30.

For the status (passed, failed) filtering feature (which was intended, but actually not implemented by the original developers of the plugin), populateTemplate function in testresult.js is modified, where a string variable "statusFilter" is retrieved from the drop down selection on the plugin page, and passed into getTreeResult function. This function eventually calls getJsTree and createJSON in JsTreeUtil.java, to both of which a String parameter "statusFilter" is added to

filter out properly some test cases from the JSON object generated as the content of the table on the plugin page.

S03 - To make the results in the table links we added anchor tags into the table generation handlebars string in test-result-analyzer-template.js for each test status. The createURL function crafts the URLs for each cell's associated Jenkins test result page based on their hierarchy, build number, and name information.

S05 - The addOptions function in selectdiffresults.js analyzes the JSON test result object to find the appropriate number of options to put into the drop down menus. This function modifies the HTML select elements on the page by appending new options for each build. showDiffs, along with the helper functions, gets the selected values from the two drop down menus and makes a call to getDiffs in differentresults.js to find and display the differences between the two tests, again using the results JSON object.

S09 - In selectdiffresults.js, createCheckboxButton creates the button for getting the build report respective to the checked checkboxes. It returns a string of HTML to be used in the jelly file. showCheckedBuilds and its helper functions gets the builds with checked checkboxes and removes all other builds and results that were not checked from the array of items. The checkboxes were created by using the each helpers and providing them each an 'id' in the variable tableBody containing html code. A checkbox is placed next to each build number in the table headers. The id's were used later to filter/remove unchecked builds from the chart.

S10 - To change how the main action in the sidebar behaves, we made a new function, getModuleList, in the TestResultsAnalyzerPluginAction.java file that looks at the current AbstractProject object and its Collection of jobs using the getAllJobs function. If one job was returned, a module has been selected so the plugin acts normally. When there are multiple jobs we create and return an HTML string that lists the display names of each job and links to the plugin page for each list element. We call getFullName and getDisplayName on each element and the full name without a '/' character is known as the project and not a module. The remaining elements are wrapped in anchor, h2, and list item tags and comprise the HTML list of modules for the project when no module has been selected. This string is passed to a div in the index.jelly of the plugin action and there is a check in javascript to see if the string was empty. If it was not empty, the default behavior is muted by replacing the main-panel div's HTML content with this module list. Clicking a module name directs the user to the plugin page for the selected module.

S11 - A new file, testcodes.js, is added which contains functions that initialize a javascript dictionary object "testCodes", which maps class names of each table-row containing a testcase to arrays of strings. Each of the strings is a line of source code of the testcase. The main function initTestCodes is invoked by populateTemplate in testresults.js (must be after analyzerTemplate is called). A button labeled "code" is displayed alongside the test names in the table. When this button is clicked it calls a function displayCode in

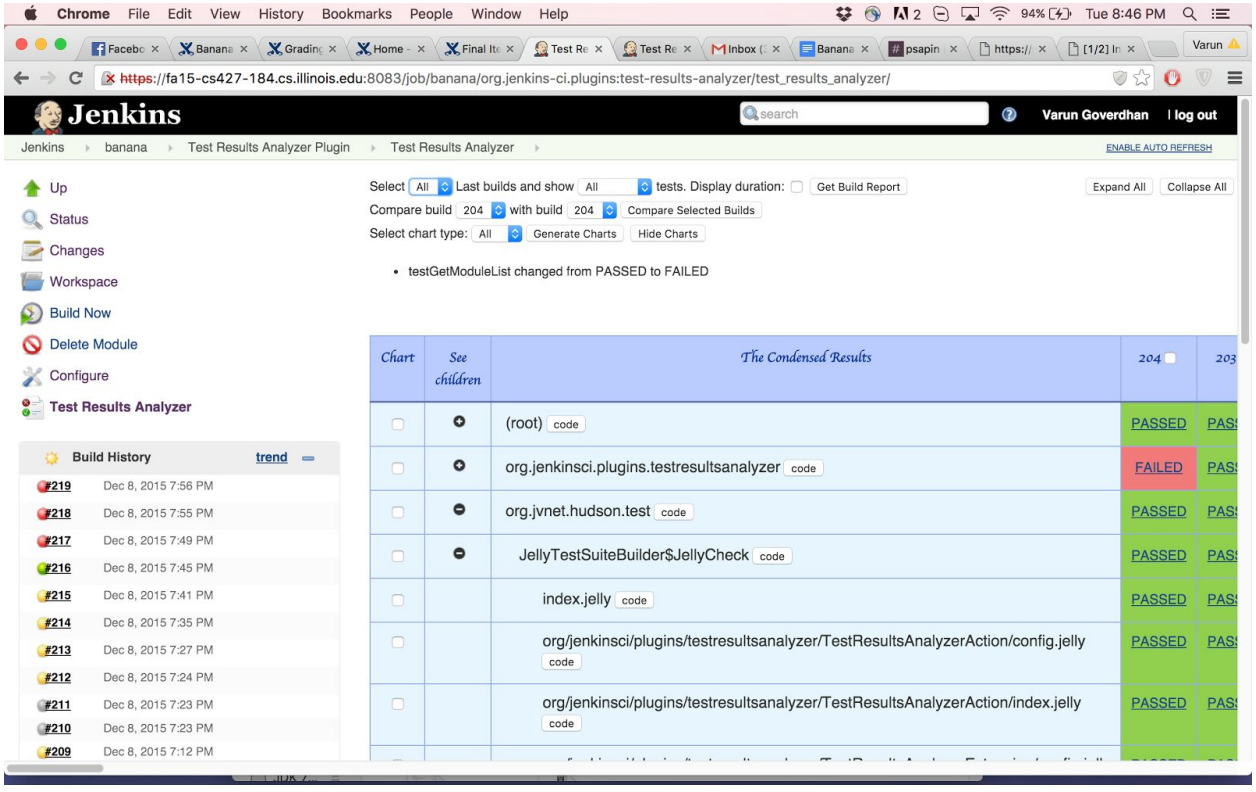
test-result-analyzer-template.js that passes a key to “testCodes” and displays the associated source code on the plugin page.

3. Usage

Once the plugin is installed, select a job on jenkins and click on the Test Results Analyzer plugin option on the left side.

User story 1 Feature:

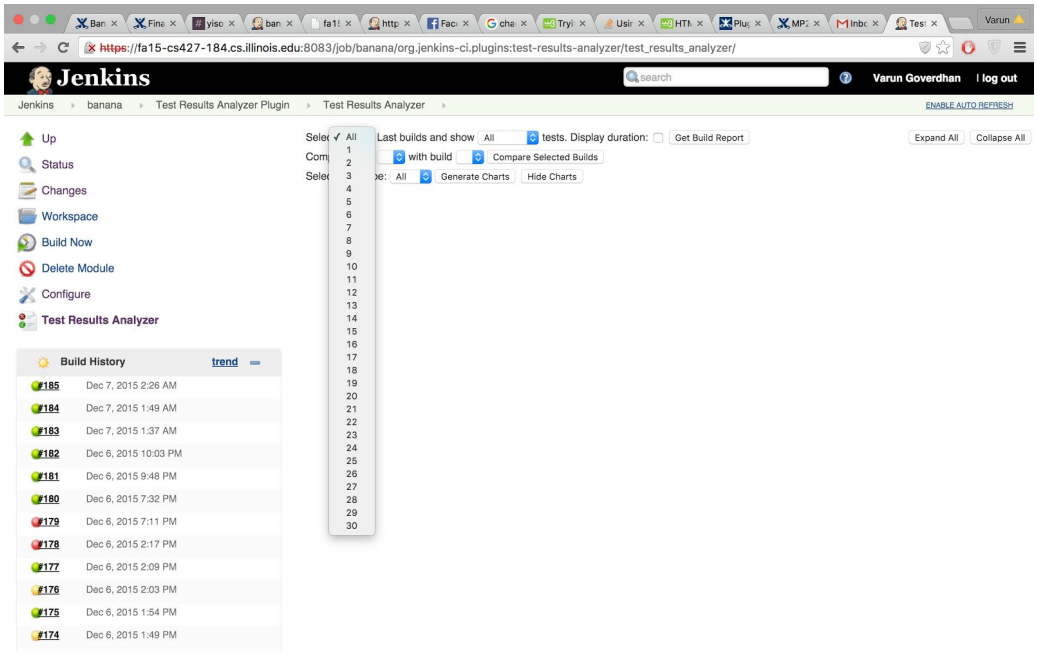
This feature is displayed in the screenshot below. The differences between the last two builds are displayed in text when the “get build report” button is clicked and the table is shown.



Screenshot 1

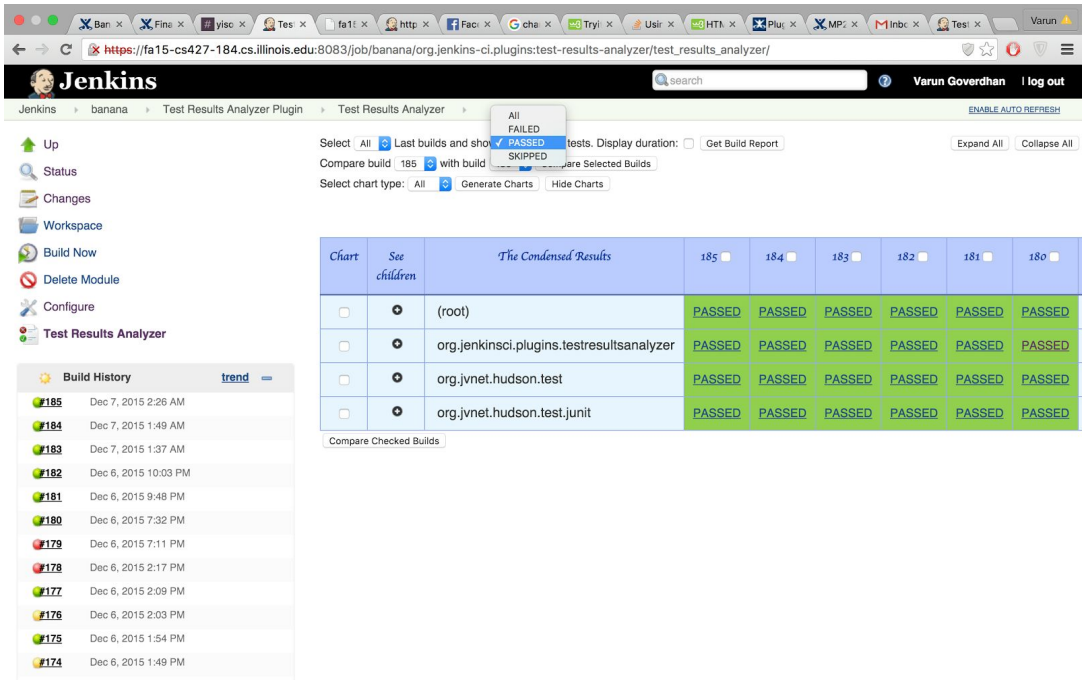
User Story 2 Feature:

This feature gives the user a wide choice of numbers to specify the previous builds to view. The drop down menu gives the user options between 1 and 30 last builds. This is the screen the user is directed to, after selecting the module discussed above.



Screenshot 2.1

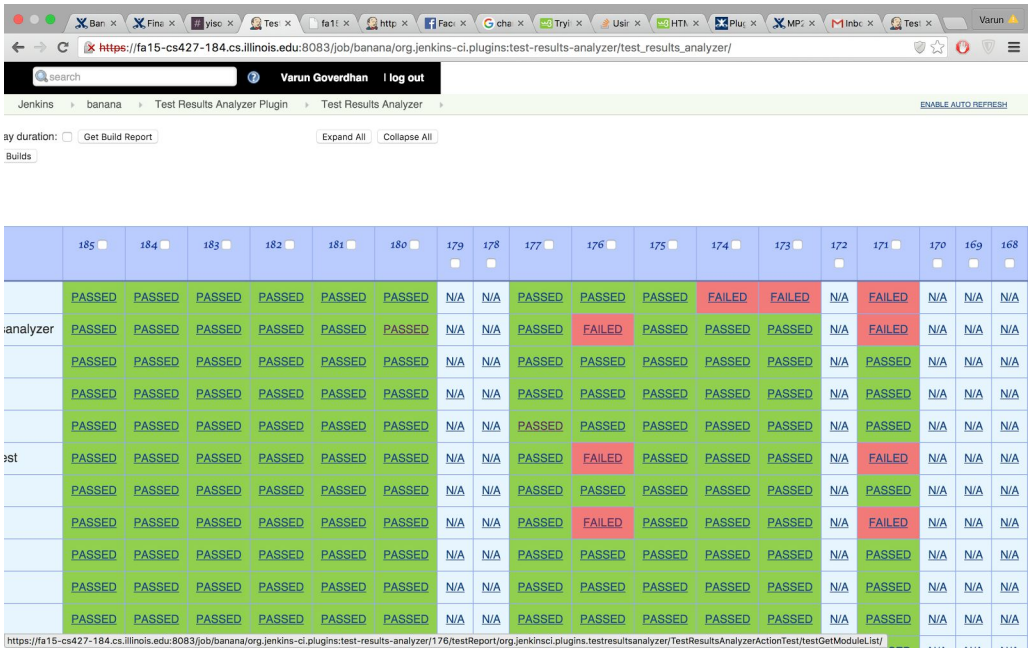
Another feature of story 2 allows the user to filter the results based on Failed/Passed/Skipped reports. This feature was intended in the original plugin but was not functional.



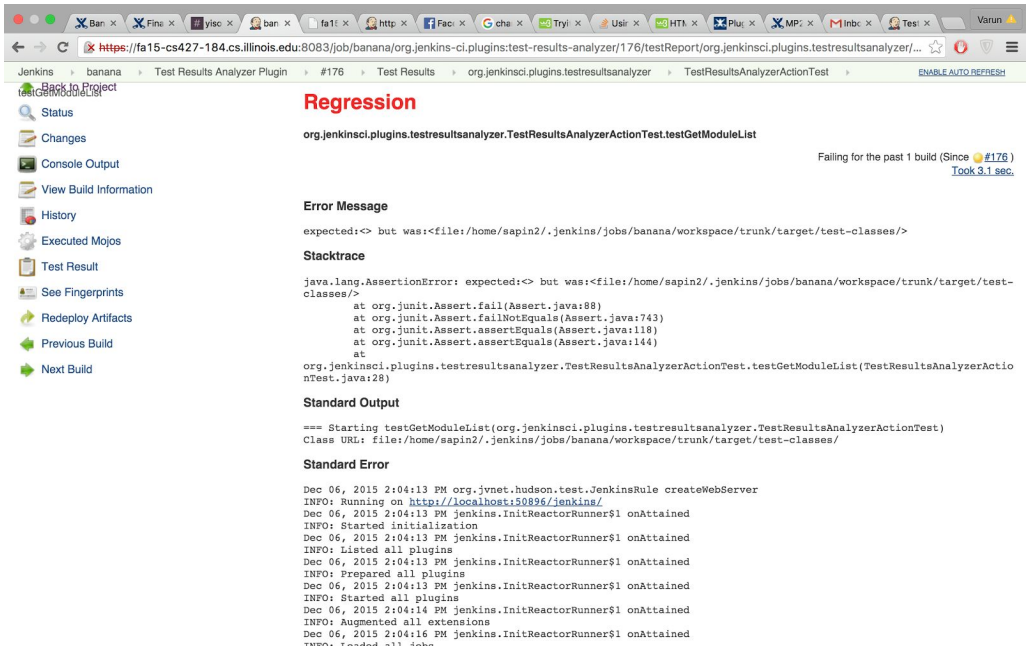
Screenshot 2.2

User Story 3 Feature:

The next two screenshots display the feature in story 3. Links are created for each cell that directs the user to the test report of that particular test and for that particular build.



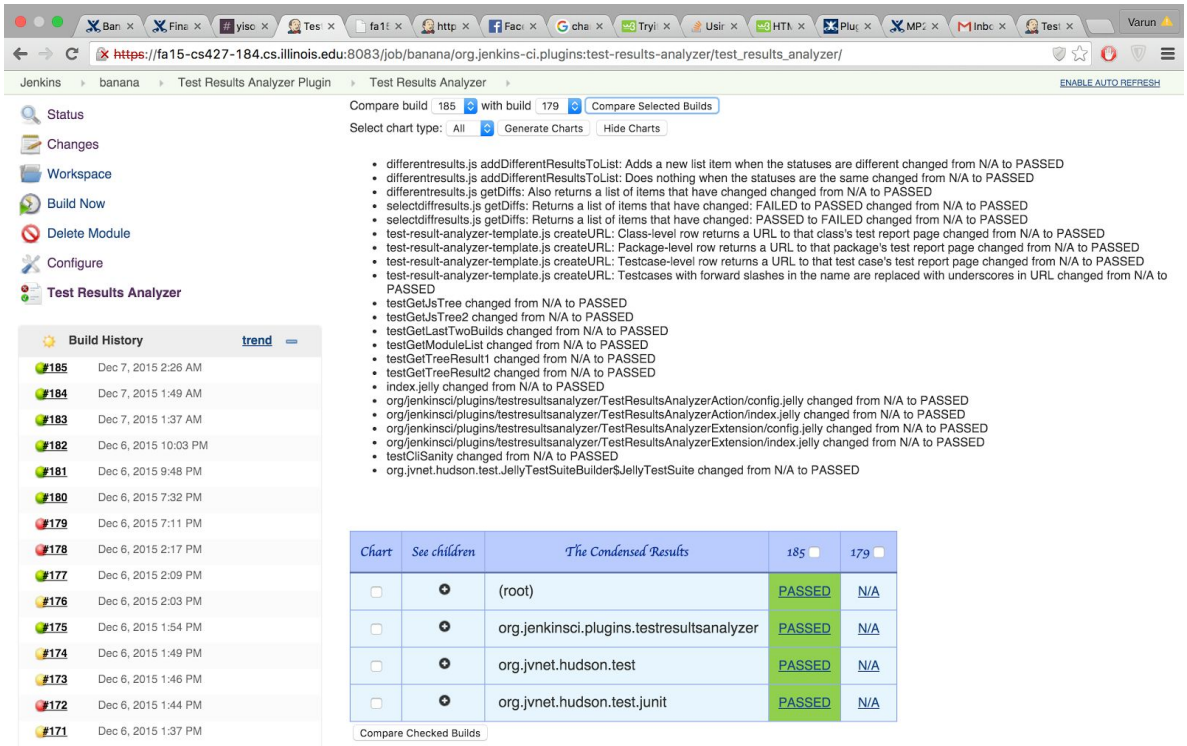
Screenshot 3.1



Screenshot 3.2

User Story 5 Feature:

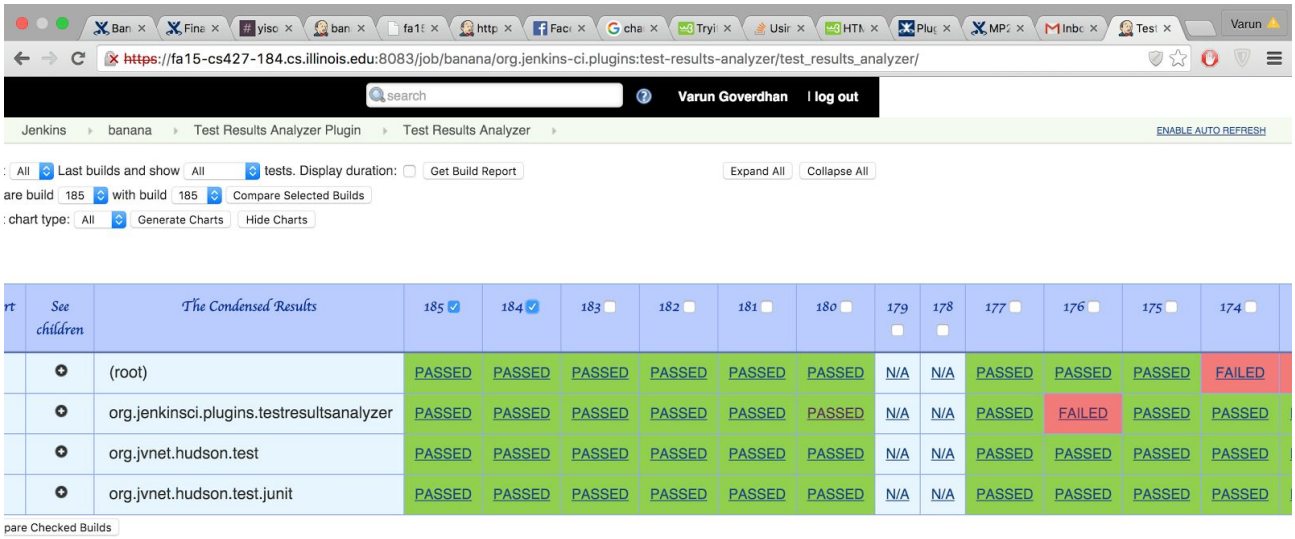
This story allows the user select builds in non consecutive order and also displays the difference between the two builds. This is done through the drop down menu and then clicking the ‘Compare Selective Builds’ button.



Screenshot 5

User Story 9 Feature:

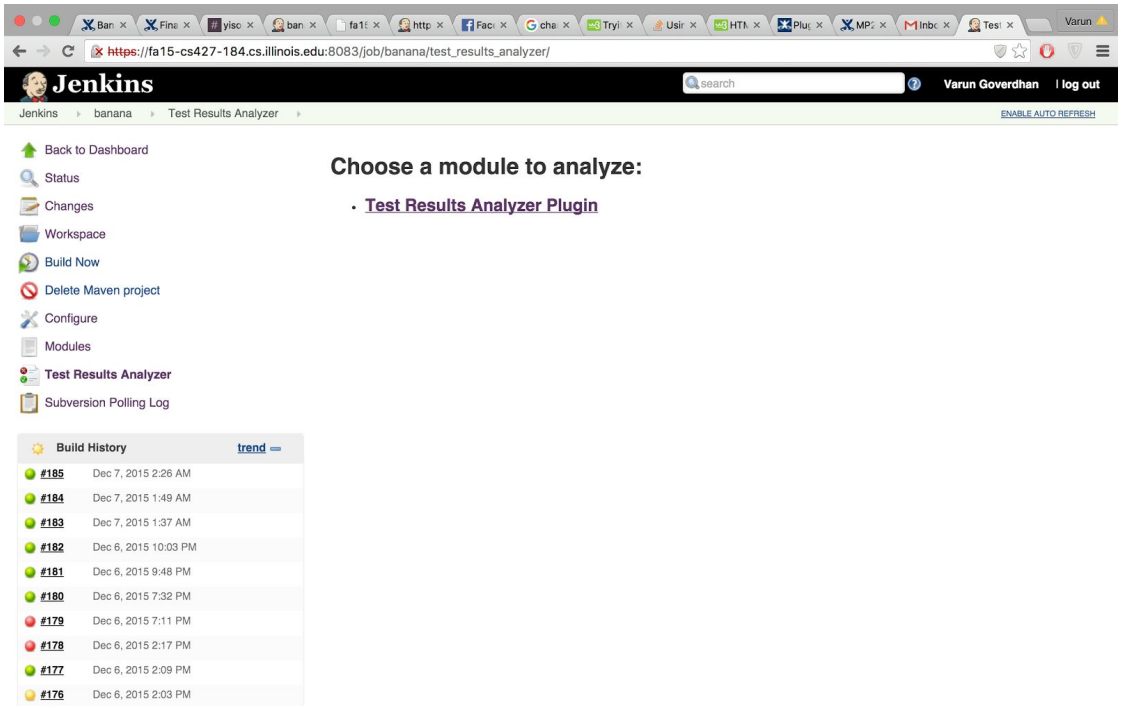
This feature allows the user to choose multiple non consecutive builds to display. This is achieved through the check boxes seen by the build numbers.



Screenshot 9

User Story 10 Feature:

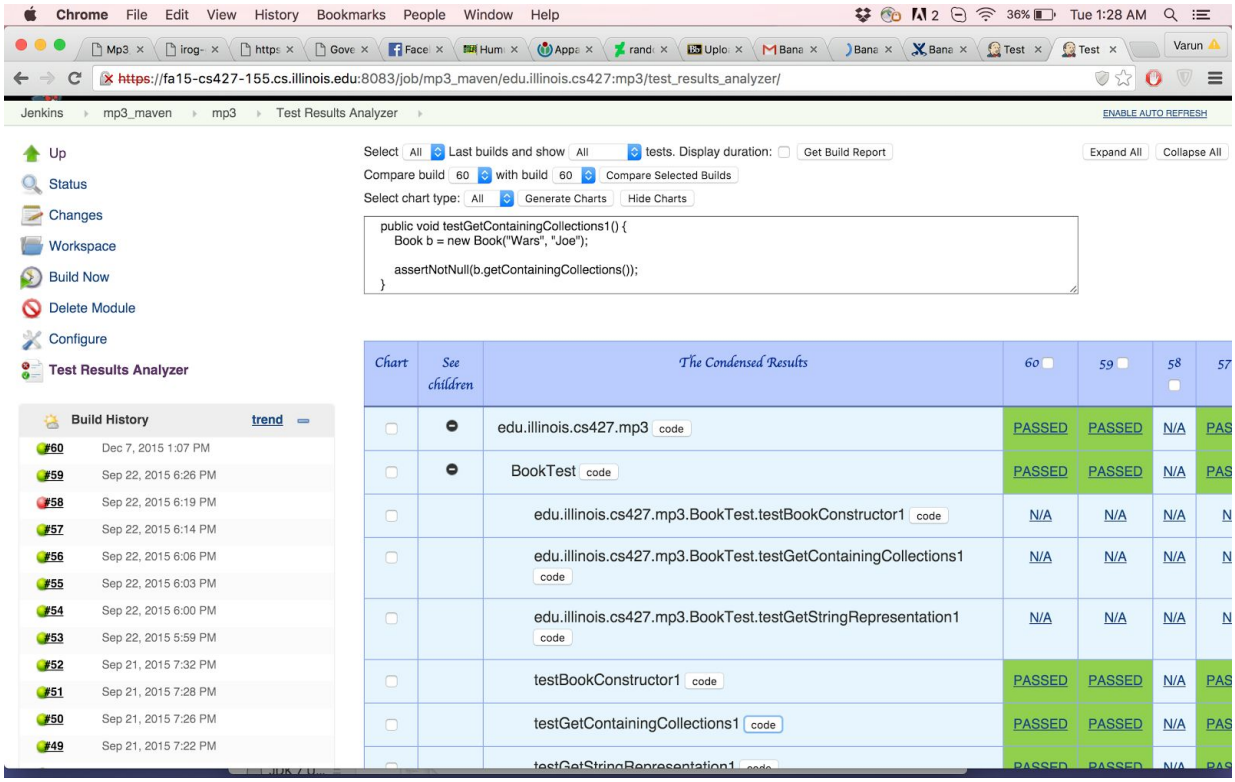
This feature allows the user the select which module to analyze. When the user clicks on the ‘Test Results Analyzer’ option, they are first directed to this screen to select the module before being taken to the main plugin page.



Screenshot 10

User story 11 Feature:

This feature is displayed in the screenshot below. There is an included button called ‘code’ next to each test name. When clicked it displays the source code for that specific test. If there is no associated source code, an error image is displayed.



Screenshot 11