CS427 - Software Engineering

Test Regression Plugin - Jenkins

KGB Team

Ballet	Vincent
Gryska	Evan
Gupta	Shivam
Huang	Zifeng
Pecak	Mark
Peter	Simon
Pugliese	Daniel
Shin	Martin

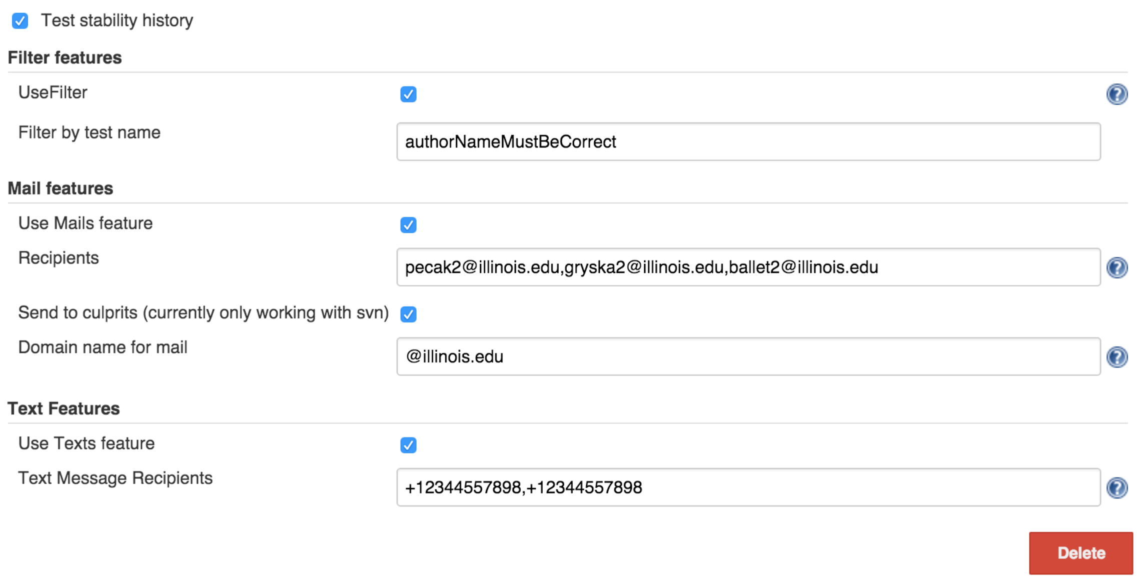
ILLINOIS

Department of Computer Science
University of Illinois at Urbana-Champaign

Contents

1 General Information

1.1 Overview


✓ Test stability history

Filter features

UseFilter	✓
Filter by test name	authorNameMustBeCorrect

Mail features

Use Mails feature	✓
Recipients	pecak2@illinois.edu,gryska2@illinois.edu,ballet2@illinois.edu
Send to culprits (currently only working with svn)	✓
Domain name for mail	@illinois.edu

Text Features

Use Texts feature	✓
Text Message Recipients	+12344557898,+12344557898

Figure 1.1: General plugin settings

This plugin provides statistics about the stability and flakiness of tests. Our additions have sought to increase the usefulness of these statistics by allowing the user more control over how they are calculated and displayed. One aspect of the added functionality is allowing users to filter out certain tests from the statistics, and to cancel builds that would compromise the integrity of the statistics (e.g. an experimental feature that does not influence the main code base). Other added features are showing a graph of the test status history, giving information about what had caused past failures that are now fixed, and giving information about individual tests within a test suite through a mouseover popup. The plugin also now gives notifications after each build with the number of regressed tests in that build to a specified mailing list, both over email and text message.

1.2 Architecture

Most of the plugin's functionality is from implementing the extension point `DataPublisher`. The plugin overrides the function `getTestData()` in `StabilityTestDataPublisher`, which is called as a post-build step of a Jenkins build. `getTestData()` takes as one of its parameters a `TestResult` object that represents the root of the entire project, which has as children and descendants other `TestResult` objects corresponding to the packages, classes, and test cases. The plugin then recursively moves through the tree of `TestResult` objects, building for each a `CircularStabilityHistory` - a data structure that will store all the information needed to later compute the stability and flakiness percentages - from the results of the current run and stored information about past runs. The `CircularStabilityHistory` objects are put into a map with the test name as key and `CircularStabilityHistory` as value. This map used to create a `StabilityTestData` object, which simply stores a copy of the map for later use. This `StabilityTestData` is the return value of `getTestData`, which is passed back into the Jenkins core code. Once the user navigates to a test results page, Jenkins will call `getTestAction()` on the previously returned `StabilityTestData` object to retrieve the appropriate `StabilityTestAction`. `getTestAction()` takes a `TestObject` as its parameter and checks to see if the `TestObject` is a kind of `TestResult` that we display statistics for (`PackageResult`, `ClassResult`, or `CaseResult`) and that the id of the test is a key in the map we created in `getTestData()`. If these conditions are met a list containing a single `StabilityTestAction` constructed from the stored `CircularStabilityHistory` is returned, otherwise an empty list is returned.

The `StabilityTestAction` is constructed by checking if the `CircularStabilityHistory` passed to it is null. If it is non-null then the percentages for stability and flakiness, along with a count of how many times the test has failed over the course of the stored history. These numbers are found by scanning over each result in the `CircularStabilityHistory` and comparing the number of failures and the number of times the test switches from passing to failing or vice versa to the total number of runs. Once these values are retrieved they are stored in instance variable so they can be used later. Regardless of whether the `CircularStabilityHistory` is null or not the constructor then generates the description string and finds and stores the names and percentages for the descendants of this test that are the most flaky and least stable. If the `CircularStabilityHistory` is null this signifies that the test has not failed in the recorded history, and these fields are filled accordingly. The GUI then loads one or more of the three groovy files - index.groovy, badge.groovy, and summary.groovy - in divs within the webpage for the test result. These can access values from the `StabilityTestAction` object corresponding to the appropriate test result by using the sytax `my.someField` which will get the value returned by `StabilityTestAction.getSomeField()`. In this was the calculated values are passed to the groovy file, and displayed to the user.

The plugin also implements the `BuildWrapper` extension point for a small part of its functionality. It overrides the function `setUp()` in `StabilityTestBuildWrapper`. One of the parameters to `setUp()` is the `AbstractBuild` representing the current build. `setUp()` is run after the checkout from the project's SCM takes place, but before the build actually starts. Thus, it is able to look at the latest commit - retrieved by looking at the first entry in the list returned by `build.getChangeSet().getItems()` - and see if there is anything in the commit which should stop the build. The two possibilities are that the author of the commit has been flagged as a user whose commits should not be built, or that the commit message contains a specified keyword saying that this commit should not be built. If the build should continue (neither of these conditions are met) then `setUp()` returns a `StabilityTestBuildWrapper.Environment` containing the commit's author, message, and timestamp. If the build should not continue then `build.doStop()` is called, which causes the build to cease running. However, `build.doStop()` can sometimes take too long to trigger, and so `setUp()` also throws an `InterruptedException` which prevents the build from starting before the `build.doStop` call takes effect. This results in the build displaying on Jenkins as aborted with a grey ball. In contrast, having `setUp()` return null would also stop the build from running, but would cause it display in Jenkins as failed, with a red ball.

The configuration settings of both `StabilityTestDataPublisher` and `StabilityTestBuildWrapper` are laid out in their respective `config.jelly` files, and the values entered by users are passed to the constructors annotated with `@DataBoundConstructor`. These constructors simply store the values in instance variables so they can be referenced later during execution of the plugin.

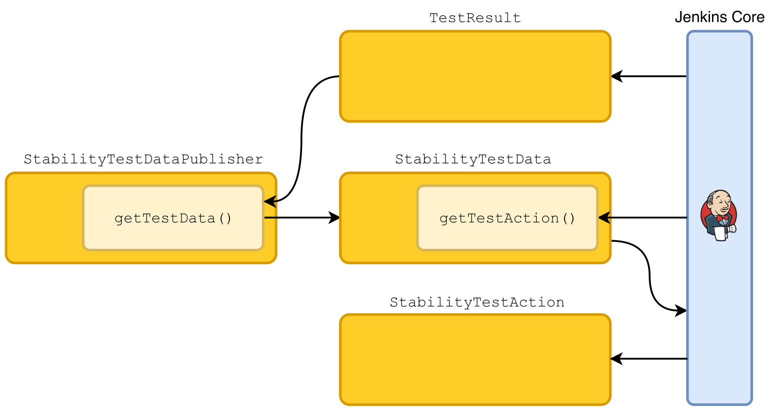
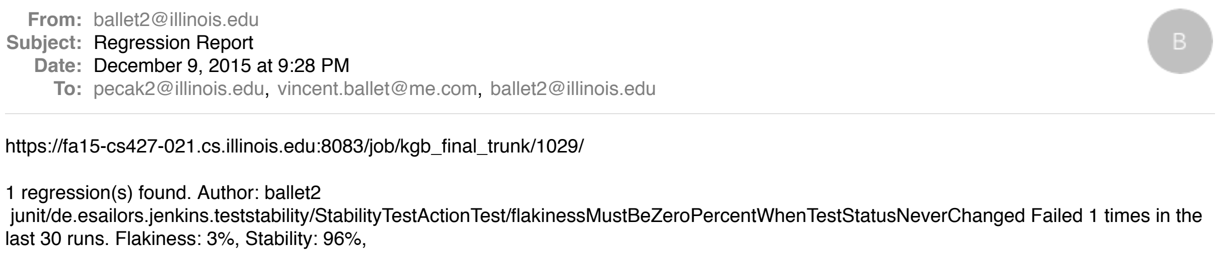


Figure 1.2: Architecture

2 Design and Usage of Features

2.1 Email Notifications



From: ballet2@illinois.edu
Subject: Regression Report
Date: December 9, 2015 at 9:28 PM
To: pecak2@illinois.edu, vincent.ballet@me.com, ballet2@illinois.edu

https://fa15-cs427-021.cs.illinois.edu:8083/job/kgb_final_trunk/1029/

1 regression(s) found. Author: ballet2
junit/de.esailors.jenkins.teststability/StabilityTestActionTest/flakinessMustBeZeroPercentWhenTestStatusNeverChanged Failed 1 times in the last 30 runs. Flakiness: 3%, Stability: 96%,

Figure 2.1: Mail feature

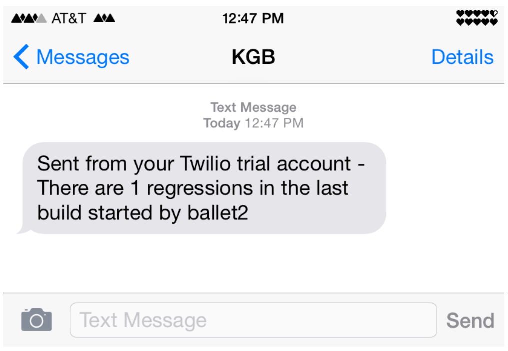
The mail notifications feature is meant to send a mail to anyone who enters their mail address into Jenkins configuration that says if there are any regressed tests in the last build, who is responsible of this build, and what are the regressed tests. This helps to keep everyone updated on the current status of the Jenkins job. We also provide an other feature: send the mail to the `svn` user which commit made the tests regress.

To get all of the mail addresses, there is a form to fill out on the Configure Jenkins page. You enter the mail addresses as a comma separated list of the form "`address1@someDomain.com,address2@someOtherdomain.fr`" and check a box to enable or disable mail notifications. The "send to culprit" feature is also enabled/disabled by checking a box. The mail addresses are passed from the front end to the `mailReport` method. When a build runs, it will check if you have mail notifications enabled and if you do, it will send all recipients a mail saying how many regressions there are in the current build, which tests regressed, and who is responsible for that.

2.2 Text Message Notifications

The text message notifications feature is meant to send a message to anyone who enters their phone number into Jenkins configuration that says if there are any regressed tests in the last build. This helps to keep everyone updated on the current status of the Jenkins job.

To get all of the phone numbers, there is a form to fill out on the Configure Jenkins page. You enter your phone numbers as a comma separated list of the form "`+11234567890,+15556667777`" and check a box to enable or disable text notifications. The phone numbers are passed from the front end to the `textReport` method. When a build runs, it will check if you have text notifications enabled and if you do, it will send all recipients a message saying how many regressions there are in the current build. The number of regressed tests is acquired from a `CircularStabilityHistory` object.



Sent from your Twilio trial account - There are 1 regressions in the last build started by ballet2

Figure 2.2: Text preview

To send the text message, we use the `Twilio API` (`https://www.twilio.com/`) which requires an account to be set up on the website. In order to send messages to an unverified number, you must pay to upgrade your account or buy a `Twilio` phone number. Otherwise, you must manually verify each phone number you wish to send a message to.

2.3 Graphing of Test History

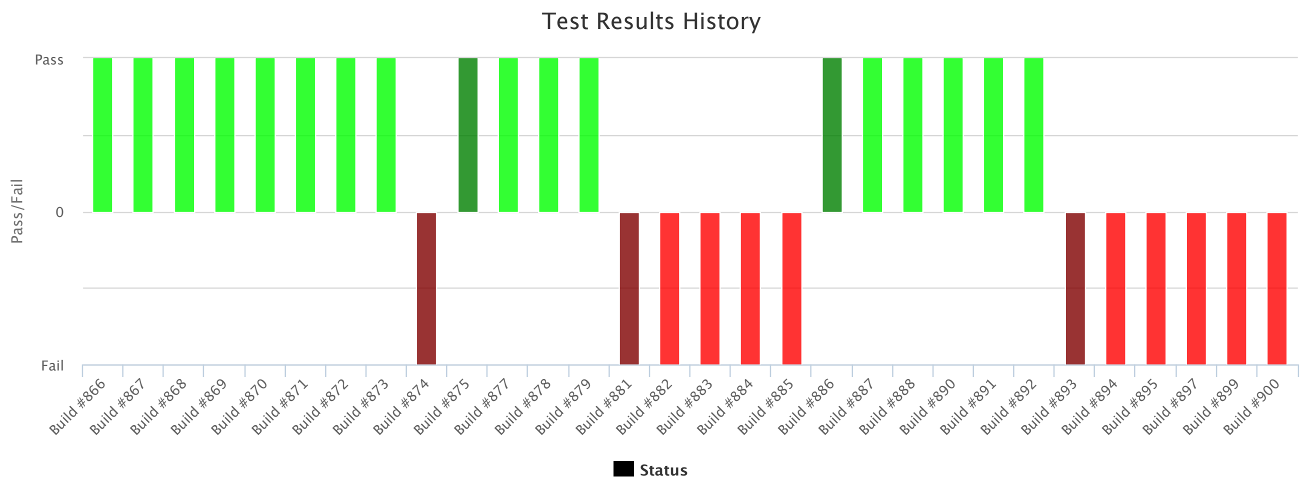
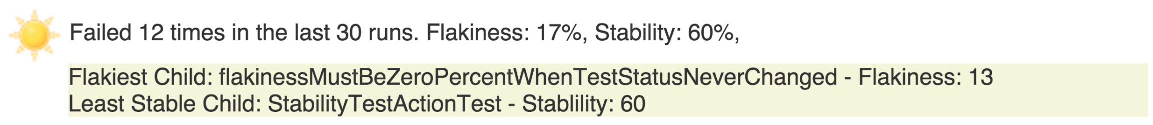


Figure 2.3: Test history graph

The graph feature was designed to give a visual representation of the status of the project's build history by showing you on which build a unit test (or package) failed, passed, regressed, or was fixed. The history data from the graph is built up in `StabilityTestAction` by going through the `CircularStabilityHistory` and determining the status for each build and the status of the previous build to see if the build has been fixed or regressed. The data is then created as a JSON object and added to a JSON array. The number of builds to store in the history is determined by the user. The graph is then generated using the Highcharts API (documentation: `http://api.highcharts.com/highcharts`) with some custom features specified in the package's `webapp` folder. A javascript file defines the properties of the graph, with the data being passed to this javascript file as a JSON array and then parsed. The javascript file is referenced in `StabilityTestAction` resources folder in the `summary.groovy` file.

2.4 Displaying Mouseover Information



Failed 12 times in the last 30 runs. Flakiness: 17%, Stability: 60%,

Flakiest Child: flakinessMustBeZeroPercentWhenTestStatusNeverChanged - Flakiness: 13
Least Stable Child: StabilityTestActionTest - Stablility: 60

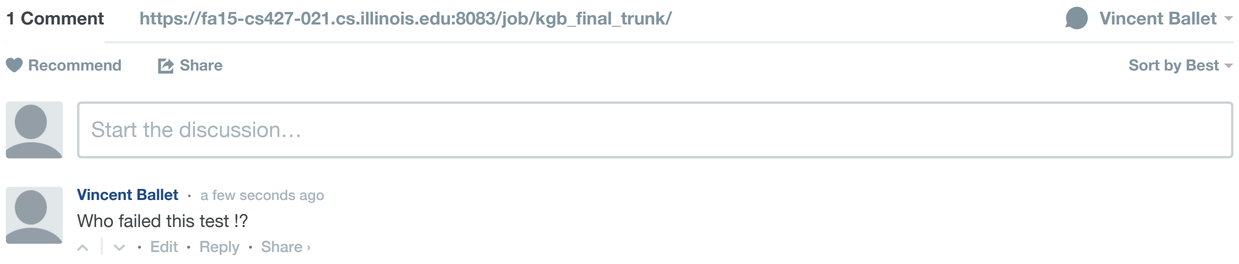
Figure 2.4: Mouseover feature

The mouseover feature allows for users to see more detailed information about the results within a test suite. There is a piece of text displaying the stability and flakiness statistics for the test suite, and when the user hovers their mouse over this text, a popup appears which says which test cases are contributing the greatest amount of instability and flakiness.
This feature makes use of the hierarchical structure of `CircularStabilityHistory` objects to look through all the leaf history objects that are descendants of the test suite in question. These represent the individual case results, and the ancestor which represents the test suite loops over them to find the one with highest flakiness and the one with lowest stability. The names and percentages of these case results are then retrieved by the corresponding `StabilityTestAction` and then passed to the front end and displayed.

2.5 Filter Out Specified Tests

The filter feature allows the user to filter a test in the front end as well as the back end. For instance, a filtered failing test won't affect stability and flakiness calculations of a package. At the same time this test will be hidden from the test report in the front end.
The method `getTestData()` in class `StabilityTestDataPublisher` gets passed as an argument the test's results. From there, our recursive method `addResultToMap()` will filter the test whose name matches our`testName` (this variable is being passed the value of the to-be-filtered test from the GUI).
We then use Javascript to hide this filtered test from the front end. We weren't able to access the front end of the plug-in from our `Extension Point`.

2.6 Display Comments on Tests Results


1 Comment https://fa15-cs427-021.cs.illinois.edu:8083/job/kgb_final_trunk/ Vincent Ballet

Recommend Share Sort by Best

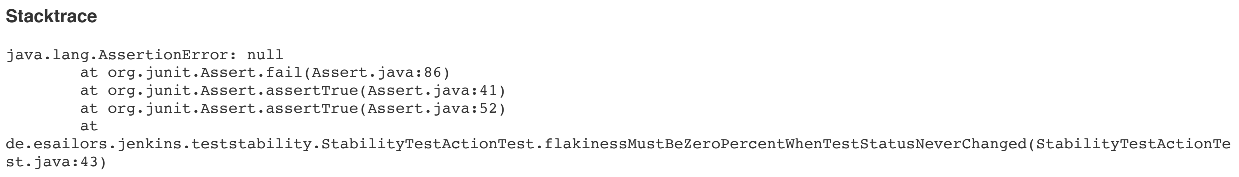
Start the discussion…

Vincent Ballet · a few seconds ago
Who failed this test !?
Edit · Reply · Share ›

Figure 2.5: Disqus feature

This feature allows all the developers to comment on a particular build and to discuss on certain build failures. `disqus_config` in `disqus.js` sets things like `page_identifier` to the `testName`. `disqusTest.js` tests this feature on `disqusTest.html` This is a standalone feature.

2.7 Display Stack Trace of Last Failure After a Test is Fixed


Stacktrace

```
java.lang.AssertionError: null
	at org.junit.Assert.fail(Assert.java:86)
	at org.junit.Assert.assertTrue(Assert.java:41)
	at org.junit.Assert.assertTrue(Assert.java:52)
	at
de.esailors.jenkins.teststability.StabilityTestActionTest.flakinessMustBeZeroPercentWhenTestStatusNeverChanged(StabilityTestActionTe
st.java:43)
```

Figure 2.6: Strack trace feature

This feature is used to track what errors in tests have been fixed. In vanilla Jenkins a failed test will display the stack trace of the failure, but with this plugin you can also see the stack trace on a test that has been fixed. This helps track what was wrong and what has been fixed in the new build.

When a build runs, the plugin looks for a test case that has been fixed in the current build and then retrieves the error stack trace of the test from the previous run. After retrieving the `CaseResult` of the fixed test, `getPreviousResult()` is called to pull the old `CaseResult` which then returns a stack trace string after calling `getErrorStackTrace()`. This string is then placed in the corresponding `CircularStabilityHistory` of this test, which can then be accessed by the `StabilityTestAction`. The front end of the plugin, which receives the appropriate `StabilityTestAction`, can then display the error stack trace of any tests that were fixed in the latest build.

2.8 Ignore Builds Based on SCM

This feature is used to stop a build from finishing based on the commit message from the builds svn log. This is useful in the case that system resources are being conserved and a build has had only minor changes such as adding comments to the code base and a build is not deemed necessary at the moment.

When changes are submitted to svn, Jenkins detects this and starts a build(assuming auto run has been enabled) however when it parses the log message an abort signal is used to kill the build and it never continues. The `StabilityTestBuildWrapper` is specifically the class responsible for checking the svn commit message in the beginning stages of a build and stopping the build if it is required.