Viewing Tests and Configuration Problems

In this section you will read about:

Viewing Problems on Project Overview Page

The project overview page also displays the number of tests failed in build configurations as well as other problems. When the data in the problematic build configuration is not expanded, the link under the tests number takes you to the Current Problems page (see below).

When you expand the build configuration data, the problem summary appears. The presentation of problems is shortened if your browser does not have enough horizontal space.

Using Current Problems Tab

To view problematic build configurations and tests in your project, open the Project Home page and go to the Current Problems tab.

By default, the Current Problems tab displays data for all build configurations within a project. To limit the data displayed, use the Filter by build configuration drop-down.

From this page you can view problems in your project divided into the following groups:

- build configuration problems,
- failed tests,
- muted test failures,
- build problems.

Each of the sections is expandable and you can further drill down to the smallest relevant item using the up and down arrows .

The links appearing in build configuration problems and build problems sections enable you to monitor a great deal of useful data, e.g. you can navigate to build results, view build log, changes, etc. More options are available when you click the arrow next to a link.

The failed test and muted failures sections have grouping options and allow viewing the test stacktrace available when clicking the test name link. You can also view the test history, open the test in the active IDE, start investigating a particular test failure, fix and unmute a test or start investigating a particular test failure.

Flaky Tests

TeamCity detects flaky tests and displays them on the dedicated tab for a given project.

A flaky test is a test that is unstable (can exhibit both a passing and a failing result) with the same code.

Flaky test detection is based on the following heuristics:

1. High flip rate (Frequent test status changes). A flip in TeamCity is a test status change — either from OK to Failure, or vice versa. The Flip Rate is the ratio of such "flips" to the invocation count of a given test, measured per agent, per build configuration, or over a certain time period (7 days by default). A test which constantly fails, while having a 100% failure rate, will have its flip rate close to zero; a test which "flips" each time it is invoked will have the flip rate close to 100%.
   If the flip rate is too high, TeamCity will consider the test flaky.
2. Different test status for build configurations with the same VCS change: if two builds of different configurations are run on the same VCS change and the test result is different in these builds, the test is reported as flaky. This may be an indication of environmental issues.
3. If the status of a test 'flipped' in the new build with no changes, i.e. a previously successful test failed in a build without changes or a previously failing test passed in a build without changes, TeamCity will consider the test flaky.
4. Different test status for multiple invocations in the same build (flaky failure): if the same test is invoked multiple times and the test status flips, TeamCity will consider the test flaky.
   This heuristic is supported for TestNG unit tests with `invocationCount` [1][2] greater than `1`.

There is a known issue with parameterized test invocations with TestNG: TeamCity relies on the Surefire and Failsafe Maven plugins for the purposes of test result reporting, and these plugins ignore test parameters in their XML output. As a result, parameterized test invocations within the same build are erroneously treated as multiple identical test invocations, and, if some of the invocations fail, the test is marked as flaky with the Flak y Failure reason. See the related issue.

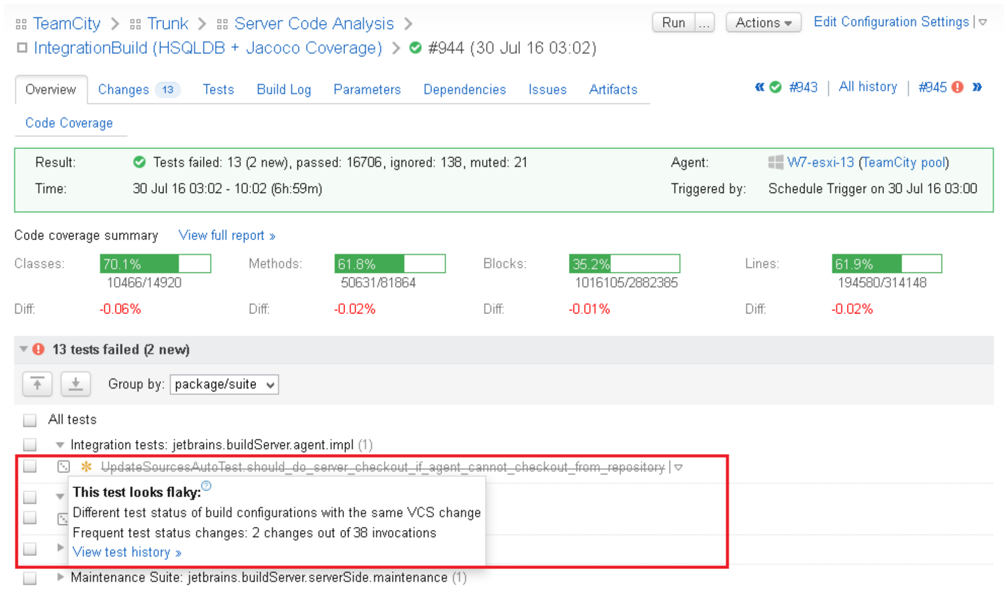
For JUnit tests, a 3rd party tempus-fugit library can be used together with JUnit. It is sufficient to annotate a test with `@Intermittent` and use the `IntermittentTestRunner` test runner, as in the minimal example below:

```
import org.junit.Test;
import org.junit.runner.RunWith;

import com.google.code.tempusfugit.concurrency.IntermittentTestRunner;
import com.google.code.tempusfugit.concurrency.annotations.Intermittent;

@RunWith(IntermittentTestRunner.class)
public class MultipleViaTempusFugit {
    @Test
    @Intermittent(repetition = 10)
    public void test() {
        // ...
    }
}
```

If such a test flakes at least once at multiple invocations within a single build, the Flaky Failure heuristic will be triggered.

Such tests are displayed on the dedicated project tab, Flaky Tests, along with the total number of test failures, the flip rate for the given test and reasons for qualifying the test as a flaky one.

You can also see if the test is flaky when viewing the expanded stacktrace for a failed test on the build results page.


TeamCity > Trunk > Server Code Analysis >
IntegrationBuild (HSQLDB + Jacoco Coverage) > #944 (30 Jul 16 03:02)
Run
Actions
Edit Configuration Settings
Overview
Changes 13
Tests
Build Log
Parameters
Dependencies
Issues
Artifacts
Code Coverage
#943
All history
#945

Result: Tests failed: 13 (2 new), passed: 16706, ignored: 138, muted: 21
Time: 30 Jul 16 03:02 - 10:02 (6h:59m)
Agent: W7-esxi-13 (TeamCity pool)
Triggered by: Schedule Trigger on 30 Jul 16 03:00

Code coverage summary View full report »

Classes:	70.1% 10466/14920	Methods:	61.8% 50631/81864	Blocks:	35.2% 1016105/2882385	Lines:	61.9% 194580/314148
Diff:	-0.06%	Diff:	-0.02%	Diff:	-0.01%	Diff:	-0.02%

13 tests failed (2 new)

Group by: package/suite

All tests
Integration tests: jetbrains.buildServer.agent.impl (1)
~~UpdateSourcesAutoTest.should_do_server_checkout_if_agent_cannot_checkout_from_repository~~

This test looks flaky:
Different test status of build configurations with the same VCS change
Frequent test status changes: 2 changes out of 38 invocations
View test history »

Maintenance Suite: jetbrains.buildServer.serverSide.maintenance (1)

As with any failed test, you can assign investigations for a flaky test (or multiple tests). For flaky tests the resolution method is automatically set to 'Manual'; otherwise the investigation will be automatically removed once the test is successful, which does not mean that the flaky test has been fixed.

Note that if branches are configured for a VCS Root, flaky tests are detected for the default branch only.

See also:

Concepts: Testing Frameworks
User's Guide: Working with Build Results