Vaadin TestBench Testing Practices.

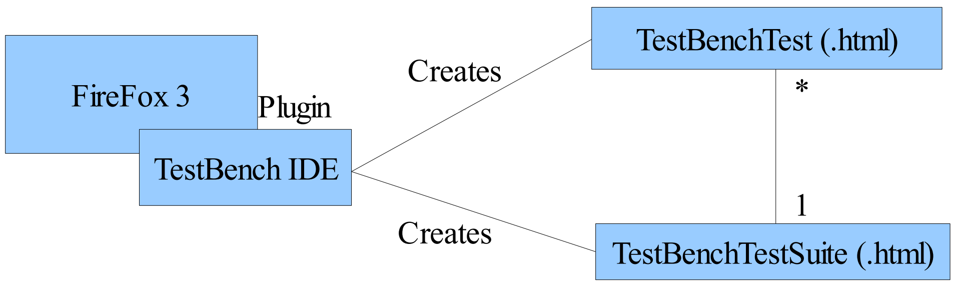
Glossary:

Hub	-	Controls communication between RC and the running test. Part of Vaadin TestBench Grid.
IDE	-	Vaadin TestBench IDE plugin.
RC	-	Controls browsers and executes test actions. Part of Vaadin TestBench Grid.
Test(s)	-	Vaadin TestBench .html test made with the IDE
Test suite	-	Vaadin TestBench .html test suite made with the IDE

Recording tests

Vaadin TestBench tests are recorded using the Vaadin TestBench IDE plugin for FireFox3.
The Vaadin TestBench IDE creates both TestBenchTests and TestBenchTestSuites.

A TestBenchTestSuite contains one or more TestBenchTests that are executed in order.



Tests should be created on a server that has the same deployment directory/path structure as the test server used during build tests.

Before starting to record a test, it would be advisable to plan what the test should do. This includes deciding on the initial state of the system at the start of the test (users, data etc.), will be long enough to benefit from cutting into smaller tests and what will this test do in general and how it should be done.

The initial state should be saved after it has been made/reached so that the system can be initialized to it before the test is executed on the build server.

Then when the system has the correct initial state go to the page from where the test should start and open the IDE. After this the test should be recorded.

If a test is long it would be advisable to make the test in multiple parts and put them in order as a test suite. This will help in debugging and finding where a test fails much easier. (it is possible to make the test as multiple files on the fly by selecting [New Test] from the IDE File menu, enabling recording and removing the first open that comes before the first command. This way you'll get the test suite made on the same run too. Remember to save the tests and suite when recording is finished.)

Test suites should always be combined from an empty test suite if there is a need to reorganize test order. This is due to the IDE not supporting reordering of tests.

Good practices

Before starting testing it would be good to define standards and naming conventions for the tests. This will make maintenance easier.

When a bug is found in the code or UI. If possible create a test that finds the bug and fails. After this, fix the bug so that the test passes. This way you get a new test that helps against regression and you get more code coverage.

All tests (and suites) should be made with the system initialized to a predetermined state.

Rules of Thumb:

1. Always test using sandbox data
   Tests should work on sandbox data that is initialized for each test (suite) so that there are no dependencies.
2. Stand-alone tests (Atomic tests)
   Minimize interaction between tests.
3. Conditions
   Prefer waiting for conditions to occur instead of using pauses in tests.
4. Prefer smaller tests
   Separate tests per business module. Each test should only test one part of the system.
5. Single responsibility
   One behavior, multiple methods → One test
   One method, multiple behaviors → Multiple tests
6. Tests should be self-descriptive
   The names of the tests should be informative and descriptive.
7. Use consistent test data
   Test data should be consistent for all runs.
8. Isolated tests (test suites)
   Tests (test suites) should not be depend on the state of a previous test (suite). (no state sharing)
   Different execution order => same results.
9. Environment isolation (Mock)
   Tests should be isolated from environmental influences.

To take into consideration when creating tests(test suites)

All tests (test suites) should be consistent. Multiple invocations on a test (test suite) should consistently return true or consistently return false if no changes have been made. This means that tests (test suites) can not depend on the environment (different dates, operating systems and hardware configurations) or use methods that return random data.

Tests (test suites) should only test behaviors not methods. This means that one test (test suite) may have multiple assertions. This often makes tests more readable and easier to understand.

The order of execution of tests (test suites) should not/cannot matter. Tests (test suites) should be independent from one another and changes in one test (test suite) cannot affect the result of other tests (test suites). When tests (test suites) share states it becomes more difficult to find the cause of a failure, because a test can fail even though behavior of tested functionality is correct.

Common actions used often or in multiple tests should be moved to separate html test files. This also makes the code/function easier to reuse.

Testing should be fully automated and not require manual involvement. Special environments should be generated automatically before tests (test suites) are run.

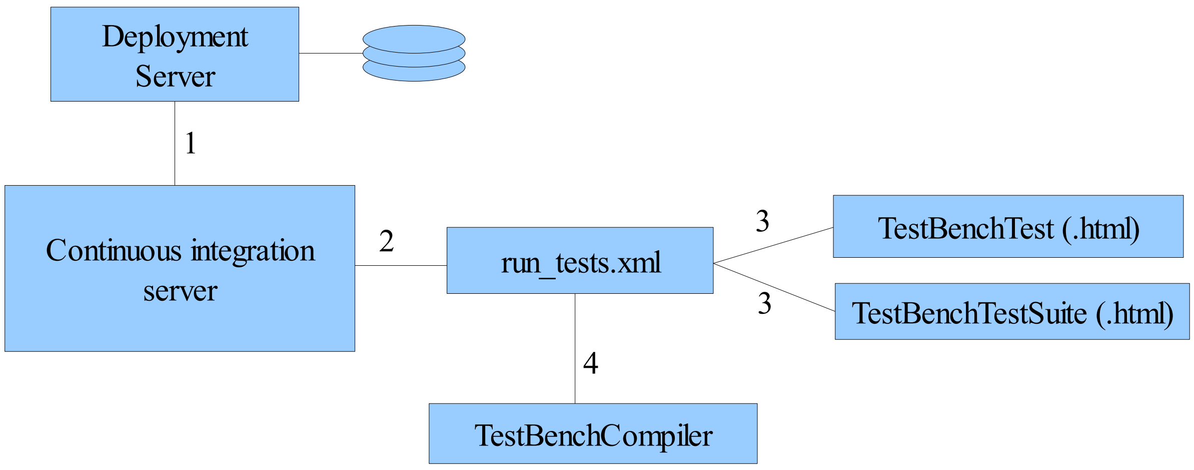
Production code should not include in itself any test logic if possible to move it elsewhere.

Required database and other setups for tests would be best to do using a special servlet that is deployed on the development server with the test build. Note that this servlet should never be added to a release build (not even as a disabled module).

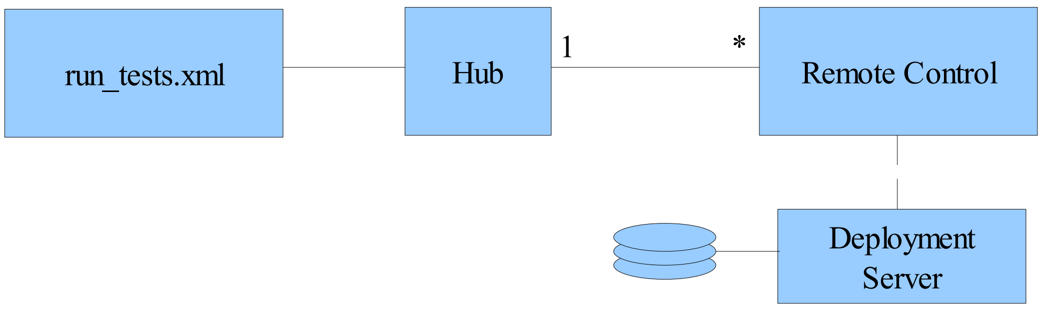
Running with a build system

After a build has completed the integration server should (1) deploy the application to a Deployment server used specially for testing. This build should include the initialization servlet.

After the application has been deployed, running and JUnit and other smoke tests have completed (2) run the TestBench run_tests.xml ant script (build.xml in the examples directory). The ant script will (3) collect .html test and test suites from the designed locations (to select locations edit the `fileset` with `id="html-test-files"`). After the files have been collected they are then (4) converted and compiled, after which the tests are executed.



The tests are executed on the TestBench Grid that consists of a Hub and one(or multiple) Remote Control(s). Remote controls may support the same or different browsers and run under different operating systems. The remote control(s) needs to be able to connect to the deployment server.



run_tests.xml
Hub
1
*
Remote Control
Deployment Server