Test Term Potpourri

Once you start looking for information on testing, you are bound to come across a plethora of names of testing styles and methods. This document is a quick reference to some of the more common terms.

A-B-A Testing

A-B-A Testing is really something of a misnomer. It might better be called A-B-A Debugging. It goes like this: (1) Identify a problem behavior *A*. (2) Change your code to the *B* condition and see if behavior *A* disappears. (3) Switch your code back to its original version to verify that the problem behavior *A* returns. This is a very useful technique, of course, but it isn't testing in the sense we are testing.

Acceptance Testing

Acceptance testing is the act of completing a set of tests to verify that you have met the customer's requirements (you are proving it works so they can accept your solution). This doesn't dictate a specific form of testing, though it is most often some combination of Manual Testing and System Testing.

Black Box Testing

Here you work purely with the external interface with no knowledge (or at least pretending to have no knowledge) of what is going on inside. This is a good approach to most levels of testing including Unit, Integration, and System Testing. Black box tests are typically more challenging to write than white box, but they are fairly immune to internal changes. For instance, you can easily test that an unordered list of items is properly ordered after a sorting operation without knowing what sorting was used.

Compliance Testing

This is a form of testing to verify that your project complies with some published standard (e.g. USB or Energy Star). Often there are industry test tools for this type of testing, and the testing is of the software and hardware combined. Thus, this testing almost always happens at the System Testing level. If your code itself must comply with government or industry standards, it is sometimes possible to work with the auditing body to arrive at a means for your test suite and automated verification tools to demonstrate coding compliance (see the Code Assurance document).

Exploratory Testing

This is a form of System Testing where one or more people manually exercise the overall system, trying out features in both standard and non-standard use-cases. Their goal is to discover unexpected behaviors that automated tests are unlikely to find.

Fuzz Testing

Fuzzing refers to generating a high volume of randomized input to stress test complex algorithms. The fuzzed input is usually created with some form of automation. Algorithms that operate on large data sets, a high number of input combinations, and/or include a high number of internal states are good candidates for this style of testing. Examples of applicable cases include de/compression algorithms or any system that processes user provided input (e.g. form fields or entire files).

Happy Day Testing

This term applies to any form of testing. It is testing for the ideal situation or how things are expected to work. Also known as Sunny Day Testing.

Integration Testing

Integration Testing is a level of granularity between Unit Testing and System Testing. It collects multiple related units together and tests how they work together. The goal of Integration Testing is to verify that entire subsystems behave appropriately. For example, the implementation of a protocol stack may be correct at the unit levels but without integration testing a problem in the logical layering could go undiscovered. This style of testing is usually only a small part of a given project and tends to be necessary only where many memory or I/O operations occur in aggregate.

Interaction Testing

Interaction Testing is a form of Unit Testing. It focuses on how a unit works when it is dependent on other units. The goal is not to verify all the units together, as in Integration Testing, but rather to verify that a particular unit behaves appropriately when working with code outside itself. Here we are insuring that the unit under test plays nice with others. Interaction Testing relies on techniques like mocking and stubbing. With these approaches you can test behavior often not easily testable in any other way. For instance, testing that your error handling gracefully deals with an out of memory exception can be very important to do but also really challenging to actually accomplish. The techniques of Interaction Testing allow you to exercise the logic and modules interactions in just such a scenario.

Manual Testing

Manual Testing is a form of System Testing that is done by hand. Usually a test plan is developed based on use-cases and requirements. Then one or more people manually work through the plan to verify that the system works as the plan dictates. This sort of testing tends to rely heavily on a physical or graphical UI.

Rainy Day Testing

This applies to any form of testing. These are tests where you are focusing on testing how a system handles errors or unusual input conditions. It is the opposite of Sunny or Happy Day Testing

State-Based Testing

State-Based Testing is a form of Unit Testing where we verify how functions alter a unit's state. These states can belong to traditional state machines or they can belong to any unit which has a predictable response to inputs: low-level drivers, math algorithms, storage mechanisms, etc.

Sunny Day Testing

See Happy Day Testing.

System Testing

System Testing is the highest level of testing. It tests a completed system (ideally release code running on target hardware). It tries to act like an end-user as much as possible, verifying that the entire system works end-to-end as required. Generally speaking, System Testing is unable to test all the nooks and crannies of your code base (that's what Unit Testing is for) but ensures that individual features composed of all those units work as intended.

Unit Testing

Unit Testing is the lowest level of testing. It tests a single unit of code at a time in isolation (e.g. a C module, a C++ class, a function/method, etc). Unit Testing is by necessity almost always implemented in the same language as the source code.

Validation Testing

Validation Testing is validating that the system meets the customers needs. It is another name for Acceptance Testing.

Verification Testing

Verification Testing is verifying that the system is compliant to all requirements. it is a combination of System Testing and Compliance Testing.

White Box Testing

White Box Testing relies on your knowledge of the inner workings of source code. It is applicable in Unit Testing or Integration Testing. White Box tests are typically easier to write than Black Box tests but suffer from brittleness as they are likely to break with any changes to the source code under test.