



METAOPTION

An Article on Testing Techniques
[By: Testing Team METAOPTION]

Software Testing Techniques

What is testing?

In general testing is finding out how well something works. In terms of human beings, testing tells us what levels of knowledge or skills have been achieved. In computers testing is used at key checkpoints in the overall process to determine whether objectives are being met.

In software development process, product objectives are sometimes tested by product user representative. When the design is complete, coding begins and the finished code is then tested at the unit/module level by each programmer, at the component level by the group of programmers involved and at the system level when all the components are combined together. At the early or later stages, a product or service may also be tested for usability.

At the system level, the manufacturer or independent reviewer may subject a product or service to one or more performance tests, possibly using one or more benchmarks. A Website site can also be tested in various ways. (a) By observing user experiences. (b) By asking questions of users. (c) By timing the flow through specific usage scenarios and by comparing it with other sites.

Testing Objectives

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has a high probability of finding an as yet undiscovered error.
- A successful test is one that uncovers an as yet undiscovered error.
- As a secondary benefit, testing demonstrates that software functions appear to be working according to specifications, that behavioral and performance requirements appear to have been met.

Testing Principles

Software testing is an extremely creative and intellectually challenging task. When testing follows certain things should be consider.

- **Testing must be done by an independent party.**
Testing should not be performed by the person or team that developed the software since they tend to defend the correctness of the program.
- **Test is not debugging.** Debugging has the goal to remove errors. The existence and the approximate location of the error are known. Debugging is not documented. There is no specification and there will be no record (log) or report. Debugging is the result of testing but never a substitution for it.

- **Test is a formal activity.** It involves a strategy and a systematic approach. The different stages of tests supplement each other. Tests are always specified and recorded.
- **Assign best personnel to the task.**
Because testing requires high creativity and responsibility, only the best personnel must be assigned to design, implement, and analyze test cases, test data and test results.
- Testing should not be planned under the unspoken assumption that no errors will be found.
- **Test for invalid and unexpected input conditions as well as valid conditions.**
The program should generate correct messages when an invalid test is encountered and should generate correct results when the test is valid.
- **Keep software static during test.**
The program must not be modified during the implementation of the set of designed test cases.
- **Document test cases and test results.**
- **Provide expected test results if possible.**
A necessary part of test documentation is the specification of expected results, even if providing such results is impractical.

Overview of Test Methods

Static Tests

The software is not executed but analyzed offline. It is generally not detailed testing, but checks mainly for the good sense of the code, algorithm, or document. It is primarily syntax checking of the code or and manually reading of the code or document to find errors. This type of testing can be used by the developer who wrote the code, in isolation. Code reviews, inspections and walkthroughs are also used.

Dynamic tests

This requires the execution of the software or parts of the software (using stubs). It can be executed in the target system, an emulator or simulator. Within the dynamic tests the state of the art distinguishes between structural and functional tests.

Structural tests

These are so called "White-box tests". Using white box testing methods, the software engineer can derive test cases that (a) guarantee that all independent paths within a module have been exercised at least once, (b) exercise all logical decisions on their true and false sides, (c) execute all loops at their boundaries and within their operational limits, (d) exercise internal data structures to ensure their validity.

Functional tests

These are the so called "Black-Box" tests. It is used to demonstrate that software functions are operational, that input is properly accepted and output is correctly produced, and the integrity of external information (database) is maintained. A black box test examines some fundamental aspect of a system with little regard for the internal logical structures of the software.

Test by progressive Stages

The various tests are able to find different kinds of errors. Therefore it is not enough to rely on one kind of test and completely neglect the other. E.g. white-box tests will be able to find coding errors. To detect the same coding error in the system test is very difficult. The system malfunction which may result from the coding error will not necessarily allow conclusions about the location of the coding error. Test therefore should be progressive and supplement each other in stages in order to find each kind of error with the appropriate method.

Module test

A module is the smallest unit of source code. Often it is too small to allow functional tests (black-box tests). However it is the ideal candidate for white-box tests. These have to be first of all static tests followed by dynamic tests to check boundaries, branches and paths. This will usually require the employment of stubs and special test tools.

Component test

This is the black-box test of modules or groups of modules which represent certain functionality. There are no rules about what can be called a component. It is just what the tester defined to be a component; however it should make sense and be a testable unit. Components can be step by step integrated to bigger components and tested as such.

Integration test

Integration testing is a systematic approach for constructing the program structure while at the same time testes to uncover errors associated with interfacing.

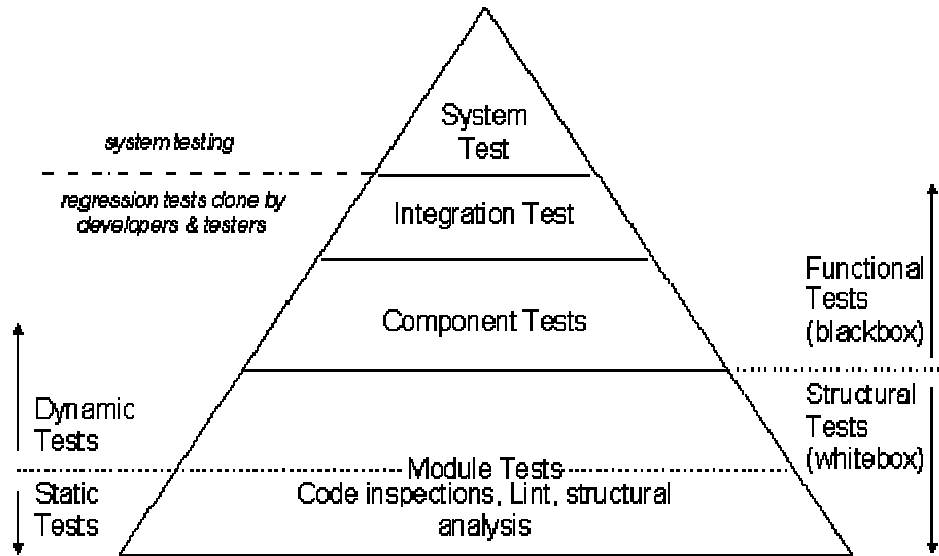
For example: one module can have an inadvertent, adverse affect on another, sub functions when combined, may not produce the desired major function.

System test

This is a black-box test of the complete software in the target system. The environmental conditions have to be realistic (complete original hardware in the destination environment).

System testing is performed on the entire system in the perspective of a functional requirements specification(s) (FRS) and/or system requirements specification (SRS). System testing is an investigatory testing phase, where the focus is to have almost a destructive attitude and tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s). System testing includes the Load testing and Stress Testing. Once the Load testing and Stress testing is completed successfully, the next level of Alpha Testing or Beta test starts.

Diagrammatic representations of Test methods



Which Test finds which Error?

Possible error	Can be best found by	Example
Syntax errors	Compiler, Lint	Missing semicolons, Values defined but not initialized or used, order of evaluation disregarded.
Data errors	Software inspection, module tests	Overflow of variables at calculation, usage of inappropriate data types, values not initialized, values loaded with wrong data or loaded at a wrong point in time, lifetime of pointers.
Algorithm and logical errors	Software inspection, module tests	Wrong program flow, use of wrong formulas and calculations.
Interface errors	Software inspection, module tests, component tests.	Overlapping ranges, range violation (min. and max. values not observed or limited), unexpected inputs, wrong sequence of input parameters.
Operating system errors, architecture and design errors	Design inspection, integration tests	Disturbances by OS interruptions or hardware interrupts, timing problems, lifetime and duration problems.
Integration errors	Integration tests, system tests	Resource problems (runtime, stack, registers, memory, etc.)
System errors	System tests	Wrong system behavior, specification errors

How to write Effective Bug Reports

How often do we see the developers requiring more information on the bug reports filed by the tester? How often do we need to spend more time investigating on the issue after the bug report has been filed? How often do we get to hear from the developers that the bug is not reproducible on their end and we need to improve on the Steps to Reproduce? In a broad sense, we end up spending more time on resolving these issues instead of investing more time testing the system.

The problem lies in the quality of bug reports. Here are the some areas where we can improve upon to achieve a perfect bug report.

Purpose of a Bug Report

- When we uncover any defect first of all we need to inform the developers about it. Bug report is a medium of such communication between testers and developers.
- The primary purpose of the bug report is to let the developers know the exact cause of the defect and see the failure with their own eyes. In all cases you can't be with them to make it fail/ show defect in front of them, give them detailed instructions so that they can make it fail for themselves.
- The bug report is a document that explains the gap between the expected result and the actual result. It also explains how to reproduce the scenario.

Finding the Defect

- When you are sure that you have found a bug, draft the bug report. It should be done the moment you notice the bug and not at the end of the day. It might be possible that you might miss out some point and the exact cause of the issue. Worse, you might miss the bug itself.
- Invest some more time to diagnose the defect you are reporting. Go through the same issue and try to reproduce on some other system. Think all the possible causes and you might come across uncovering some more defects.
- Mention what steps you have taken to produce the defects in your bug reports. The programmer will only be happy seeing that you have made their job easier.

Defect Summary

The summary of the bug report is the reader's first interaction with your bug report. The fortune of your bug heavily depends on the attraction grabbed by the summary of your bug report. It should be written in such a manner just like it seems to reader that he is going through some advertisement. A good summary will not be more than 50-60 characters. Also a good summary should carry some pictorial representation of the defects.

Communication and Language

- Keep the report language simple and straight. You are not writing an essay or an article.
- Keep your target audience in mind while writing the bug report. The report you are writing might be observed by the developers, fellow testers, managers, or in some cases even the customers. The bug report should be understandable by all of them.
- Do not augment the defect through the bug report.
- However malicious the bug might be, do not forget that it is the bug that is nasty not the programmer. Never offend the efforts of the programmer.

Steps to Reproduce

The flow of the steps to reproduce errors should be logical.

Mention the pre-requisites clearly.

Write standard detail steps for example: if a user wants to save file in .jpg format then write proper steps like open file menu, click on option save as, select .jpg format and give name to a file.

Test your steps to reproduce on a fresh system. You might find some steps which are missing or might be some other reason of the defect.

Pictures/Screenshots

Screenshots are a quite essential part of the bug report. A picture makes up for a thousand words. But do not make it a habit to attach unnecessary screenshot with every bug report. The bug report should be effective enough and enable the developers to reproduce the same at their end. Screen shots should just be a medium for verification.

If you attach screen shots to your bug report, ensure that the size of file should be smaller and the format should be .jpg or gif and definitely not .bmp.

Keep proper annotations on screen shots to pin point at the problems. This will ease the task of the developers and the developers can easily locate the problem.

Priority/Severity

Thoroughly analyze the impact of bug before setting the severity. If you think that your bug should be fixed with a high priority, justify it in the bug report. This justification should be mentioned in the description section of the bug report.

Logs

Make it a point to attach logs or quotation from the logs. This will help the developers to analyze and debug the system easily.

Other Concerns

- You should always add the exact steps to reproduce the issue later anytime (or anyone else) try to reproduce it. This will also come to rescue when someone else reports this issue, especially if it is a serious issue.
- Please mention the version number and build numbers in the bug report. This will ease up the task of developers to locate the exact source of defects.

- Please mention the platforms on which the issue is reproducible. Briefly mention the platform on which it is not reproducible or it is not being tested on that platform.
- Keep as eagle eye if you come across the same issue with a little difference in the cause. Mention those as a single bug and write down the exact case.

If the test environment on which the bug is reproducible is accessible to the developers then mention the details of accessing the setup. This will save the time of doing a new setup/ environment. Also if feasible do sharing of systems.