Software Engineering *For* BCA 4th Semester Lecture 4 [Various Software Testing Approaches in Software Engineering]	Compiled *By* Sakhi Bandyopadhyay Dept. of Computer Science & BCA, Kharagpur College, Kharagpur 721305

Software testing

Software testing is an investigation conducted to provide stakeholders with information about the quality of the software product or service under test. Software testing can also provide an objective, independent view of the software to allow the business to appreciate and understand the risks of software implementation. Test techniques include the process of executing a program or application with the intent of finding software bugs (errors or other defects), and verifying that the software product is fit for use.

Software testing involves the execution of a software component or system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,
- performs its functions within an acceptable time,
- it is sufficiently usable,
- can be installed and run in its intended environments, and
- Achieves the general result its stakeholder's desire.

As the number of possible tests for even simple software components is practically infinite, all software testing uses some strategy to select tests that are feasible for the available time and resources. As a result, software testing typically (but not exclusively) attempts to execute a program or application with the intent of finding software bugs (errors or other defects). The job of testing is an iterative process as when one bug is fixed, it can illuminate other, deeper bugs, or can even create new ones.

Software testing can provide objective, independent information about the quality of software and risk of its failure to users or sponsors.

Purpose of Testing

A primary purpose of testing is to detect software failures so that defects may be discovered and corrected. Testing cannot establish that a product functions properly under all conditions, but only that it does not function properly under specific conditions.[4] The scope of software testing often includes the examination of code as well as the execution of that code in various environments and conditions as well as examining the aspects of code: does it do what it is supposed to do and do what

it needs to do. In the current culture of software development, a testing organization may be separate from the development team. There are various roles for testing team members. Information derived from software testing may be used to correct the process by which software is developed

Various Testing Approaches

White-box testing

White-box testing verifies the internal structures or workings of a program, as opposed to the functionality exposed to the end-user. In white-box testing, an internal perspective of the system (the source code), as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the appropriate outputs. This is analogous to testing nodes in a circuit, e.g., in-circuit testing (ICT).

While white-box testing can be applied at the unit, integration, and system levels of the software testing process, it is usually done at the unit level. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it might not detect unimplemented parts of the specification or missing requirements.

Techniques used in white-box testing include:

- API testing – testing of the application using public and private APIs (application programming interfaces)
- Code coverage – creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
- Fault injection methods – intentionally introducing faults to gauge the efficacy of testing strategies
- Mutation testing methods
- Static testing methods

Black-box Testing

Black-box testing (also known as functional testing) treats the software as a "black box," examining functionality without any knowledge of internal implementation, without seeing the source code. The testers are only aware of what the software is supposed to do, not how it does it. Black-box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, state transition tables, decision table testing, fuzz testing, model-based testing, use case testing, exploratory testing, and specification-based testing.

One advantage of the black box technique is that no programming knowledge is required. Whatever biases the programmers may have had, the tester likely has a different set and may emphasize different areas of functionality.

Grey-box Testing

Grey-box testing (American spelling: gray-box testing) involves having knowledge of internal data structures and algorithms for purposes of designing tests while executing those tests at the user, or black-box level. The tester will often have access to both "the source code and the executable binary." Grey-box testing may also include reverse engineering (using dynamic code analysis) to determine, for instance, boundary values or error messages. Manipulating input data and formatting output do not qualify as grey-box, as the input and output are clearly outside of the "black box" that we are calling the system under test. This distinction is particularly important when conducting integration testing between two modules of code written by two different developers, where only the interfaces are exposed for the test.

Unit Testing

Unit testing refers to tests that verify the functionality of a specific section of code, usually at the function level. In an object-oriented environment, this is usually at the class level, and the minimal unit tests include the constructors and destructors.

These types of tests are usually written by developers as they work on code (white-box style), to ensure that the specific function is working as expected. One function might have multiple tests, to catch corner cases or other branches in the code. Unit testing alone cannot verify the functionality of a piece of software, but rather is used to ensure that the building blocks of the software work independently from each other.

Unit testing is a software development process that involves a synchronized application of a broad spectrum of defect prevention and detection strategies in order to reduce software development risks, time, and costs. It is performed by the software developer or engineer during the construction phase of the software development life cycle. Unit testing aims to eliminate construction errors before code is promoted to additional testing; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development process.

System Testing

System testing tests a completely integrated system to verify that the system meets its requirements.[50][obsolete source] For example, a system test might involve testing a login interface, then creating and editing an entry, plus sending or printing results, followed by summary processing or deletion (or archiving) of entries, then logoff.

Integration testing

Integration testing is any type of software testing that seeks to verify the interfaces between components against a software design. Software components may be integrated in an iterative way or all together ("big bang"). Normally the former is considered a better practice since it allows interface issues to be located more quickly and fixed.

Integration testing works to expose defects in the interfaces and interaction between integrated components (modules). Progressively larger groups of tested software components corresponding to elements of the architectural design are integrated and tested until the software works as a system.

Integration tests usually involves a lot of code, and produce traces that are larger than those produced by unit tests. This has an impact on the ease of localizing the fault when an integration test fails. To overcome this issue, it has been proposed to automatically cut the large tests in smaller pieces to improve fault localization.

Acceptance Testing

Acceptance testing can mean one of two things:

- A smoke test is used as a build acceptance test prior to further testing, e.g., before integration or regression.
- Acceptance testing performed by the customer, often in their lab environment on their own hardware, is known as user acceptance testing (UAT). Acceptance testing may be performed as part of the hand-off process between any two phases of development.

Alpha testing

Alpha testing is simulated or actual operational testing by potential users/customers or an independent test team at the developers' site. Alpha testing is often employed for off-the-shelf software as a form of internal acceptance testing before the software goes to beta testing.

Beta testing

Beta testing comes after alpha testing and can be considered a form of external user acceptance testing. Versions of the software, known as beta versions, are released to a limited audience outside of the programming team known as beta testers. The software is released to groups of people so that further testing can ensure the product has few faults or bugs. Beta versions can be made available to the open public to increase the feedback field to a maximal number of future users and to deliver value earlier, for an extended or even indefinite period of time (perpetual beta).

Usability testing

Usability testing is to check if the user interface is easy to use and understand. It is concerned mainly with the use of the application. This is not a kind of testing that can be automated; actual human users are needed, being monitored by skilled UI designers.

Security testing

Security testing is essential for software that processes confidential data to prevent system intrusion by hackers.

The International Organization for Standardization (ISO) defines this as a "type of testing conducted to evaluate the degree to which a test item, and associated data and information, are protected so that unauthorised persons or systems cannot use, read or modify them, and authorized persons or systems are not denied access to them."

Testing Tools

Program testing and fault detection can be aided significantly by testing tools and debuggers. Testing/debug tools include features such as:

Program monitors, permitting full or partial monitoring of program code, including:
Instruction set simulator, permitting complete instruction level monitoring and trace facilities
Hypervisor, permitting complete control of the execution of program code including:-
Program animation, permitting step-by-step execution and conditional breakpoint at source level or in machine code
Code coverage reports
Formatted dump or symbolic debugging, tools allowing inspection of program variables on error or at chosen points
Automated functional Graphical User Interface (GUI) testing tools are used to repeat system-level tests through the GUI
Benchmarks, allowing run-time performance comparisons to be made
Performance analysis (or profiling tools) that can help to highlight hot spots and resource usage
Some of these features may be incorporated into a single composite tool or an Integrated Development Environment (IDE).