Software Testing for Finding Errors

Kusuma Kumari B.M
Department of Computer Science, University College of Science, Tumkur University, Tumkur, Karnataka
Kusuma.kuku@gmail.com

Abstract:
Software testing is an activity which is aimed for evaluating an attribute or capability of a program and ensures that it meets the required result. It analyzes the software for finding bugs. Software testing is not just used for finding and fixing of bugs but it also ensures that the system is working according to the specifications. Software testing is a series of process which is designed to make sure that the computer code does what it was designed to do. In this paper I have described different software testing levels and methods.

Keywords: software testing, levels of testing, white box testing, black box testing, grey scale testing

1. Introduction
Software testing is as old as the hills in the history of digital computers. The testing of software is an important means of assessing the software to determine its quality. Since testing typically consumes 40~50% of development efforts, and consumes more effort for systems that require higher levels of reliability, it is a significant part of the software engineering. With the development of Fourth generation languages (4GL), which speeds up the implementation process, the proportion of time devoted to testing increased. As the amount of maintenance and upgrade of existing systems grow, significant amount of testing will also be needed to verify systems after changes are made [1]. Despite advances in formal methods and verification techniques, a system still needs to be tested before it is used. Testing remains the truly effective means to assure the quality of a software system of non-trivial complexity [2], as well as one of the most intricate and least understood areas in software engineering [3]. Testing, an important research area within computer science is likely to become even more important in the future.

Software testing is one of the “verification and validation,” or V&V, software practices. Some other V&V practices, such as inspections and pair programming. Verification (the first V) is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase [4]. Verification activities include testing and reviews. Validation is the process of evaluating a system or component during or at the end of the development process to determine
whether it satisfies specified requirements [4]. At the end of development validation (the second V) activities are used to evaluate whether the features that have been built into the software satisfy the customer requirements and are traceable to customer requirements. Boehm [5] has informally defined verification and validation as follows:

2. Software Testing

Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not. This activity results in the actual, expected and difference between their results. In simple words testing is executing a system in order to identify any gaps, errors or missing requirements in contrary to the actual desire or requirements. According to ANSI/IEEE 1059 standard, Testing can be defined as a process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

It depends on the process and the associated stakeholders of the project(s). In the IT industry, large companies have a team with responsibilities to evaluate the developed software in the context of the given requirements. Moreover, developers also conduct testing which is called Unit Testing. In most cases, following professionals are involved in testing of a system within their respective capacities:

- Software Tester
- Software Developer
- Project Lead/Manager
- End User

Different companies have difference designations for people who test the software on the basis of their experience and knowledge such as Software Tester, Software Quality Assurance Engineer, and QA Analyst etc.

It is not possible to test the software at any time during its cycle. The next two sections state when testing should be started and when to end it during the SDLC.

An early start to testing reduces the cost, time to rework and error free software that is delivered to the client. However in Software Development Life Cycle (SDLC) testing can be started from the Requirements Gathering phase and lasts till the deployment of the software. However it also depends on the development model that is being used. For example in Water fall model formal testing is conducted in the Testing phase, but in incremental model, testing is performed at the end of every increment/iteration and at the end the whole application is tested.
Testing is done in different forms at every phase of SDLC like during Requirement gathering phase, the analysis and verifications of requirements are also considered testing. Reviewing the design in the design phase with intent to improve the design is also considered as testing. Testing performed by a developer on completion of the code is also categorized as Unit type of testing.

Unlike when to start testing it is difficult to determine when to stop testing, as testing is a never ending process and no one can say that any software is 100% tested. Following are the aspects which should be considered to stop the testing:

- Testing Deadlines.
- Completion of test case execution.
- Completion of Functional and code coverage to a certain point.
- Bug rate falls below a certain level and no high priority bugs are identified.
- Management decision.

3. Levels of Testing

Tests are frequently grouped by where they are added in the software development process, or by the level of specificity of the test. The main levels during the development process as defined by the SWEBOK guide are unit-, integration-, and system testing that are distinguished by the test target without implying a specific process model [6]. Other test levels are classified by the testing objective [6].

3.1 Unit testing

Unit testing, also known as component 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 [7].

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 assure that the building blocks the software uses work independently of each other.

Unit testing is a software development process that involves 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 lifecycle. Rather than replace traditional QA focuses, it augments it. Unit testing aims to eliminate construction errors before code is promoted to QA; this strategy is intended to increase the quality of the resulting software as well as the efficiency of the overall development and QA process.
3.2 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 [8].

3.3 System testing
System Testing is a level of the software testing process where a complete, integrated system/software is tested. The purpose of this test is to evaluate the system’s compliance with the specified requirements.

3.4 Acceptance testing
Acceptance Testing is a level of the software testing process where a system is tested for acceptability. The purpose of this test is to evaluate the system’s compliance with the business requirements and assess whether it is acceptable for delivery.

4. Software Testing Methods
Test cases are developed using various test techniques to achieve more effective testing. By this, software completeness is provided and conditions of testing which get the greatest probability of finding errors are chosen. So, testers do not guess which test cases to chose, and test techniques enable them to design testing conditions in a systematic way. Also, if one combines all sorts of existing test techniques, one will obtain better results rather if one uses just one test technique. There are different methods which can be use for Software testing. This section briefly describes those methods.

4.1 Black Box Testing
The technique of testing without having any knowledge of the interior workings of the application is Black Box testing. The tester is oblivious to the system architecture and does not have access to the source code. Typically, when performing a black box test, a tester will interact with the system's user interface by providing inputs and examining outputs without knowing how and where the inputs are worked upon.

<table>
<thead>
<tr>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Well suited and efficient for large code segments. Code Access not required.</td>
<td>1. Limited Coverage since only a selected number of test scenarios are actually performed.</td>
</tr>
</tbody>
</table>
2. Clearly separates user's perspective from the developer's perspective through visibly defined roles.
3. Large numbers of moderately skilled testers can test the application with no knowledge of implementation, programming language or operating systems.

2. Inefficient testing, due to the fact that the tester only has limited knowledge about an application.
3. Blind Coverage, since the tester cannot target specific code segments or error prone areas. The test cases are difficult to design.

4.2 White box testing
White box testing is the detailed investigation of internal logic and structure of the code. White box testing is also called glass testing or open box testing. In order to perform white box testing on an application, the tester needs to possess knowledge of the internal working of the code. The tester needs to have a look inside the source code and find out which unit/chunk of the code is behaving inappropriately.

**Advantages**
- 1. As the tester has knowledge of the source code, it becomes very easy to find out which type of data can help in testing the application effectively.
- 2. It helps in optimizing the code. Extra lines of code can be removed which can bring in hidden defects.
- 3. Due to the tester's knowledge about the code, maximum coverage is attained during test scenario writing.

**Disadvantages**
- 1. Due to the fact that a skilled tester is needed to perform white box testing, the costs are increased.
- 2. Sometimes it is impossible to look into every nook and corner to find out hidden errors that may create problems as many paths will go untested.
- 3. It is difficult to maintain white box testing as the use of specialized tools like code analyzers and debugging tools are required.

4.3 Grey Box Testing
Grey Box testing is a technique to test the application with limited knowledge of the internal workings of an application. In software testing, the term the more you know the better carries a lot of weight when testing an application.

Mastering the domain of a system always gives the tester an edge over someone with limited domain knowledge. Unlike black box testing, where the tester only tests the application's user interface, in grey box testing, the tester has access to design documents and the database. Having this knowledge, the tester is able to better prepare test data and test scenarios when making the test plan.
Advantages |
---|
Offers combined benefits of black box and white box testing wherever possible.

Disadvantages |
---|
Since the access to source code is not available, the ability to go over the code and test coverage is limited.

Grey box testers don't rely on the source code; instead they rely on interface definition and functional specifications.
The tests can be redundant if the software designer has already run a test case.
Based on the limited information available, a grey box tester can design excellent test scenarios especially around communication protocols and data type handling.
The test is done from the point of view of the user and not the designer.

Testing every possible input stream is unrealistic because it would take an unreasonable amount of time; therefore, many program paths will go untested.

5. Conclusion
Software testing is an important technique for the improvement and measurement of a software system quality. But it is really not possible to find out all the errors in the program. So, the fundamental question arises, which strategy we would adopt to test. Software testing, depending on the testing method employed, can be implemented at any time in the software development process. Traditionally most of the test effort occurs after the requirements have been defined and the coding process has been completed, but in the Agile approaches most of the test effort is on-going. As such, the methodology of the test is governed by the chosen software development methodology. In my paper, I have described different types of levels and methods.

References:


