Volume 2, Issue 10, October 2012 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Software Testing Techniques

Shivkumar Hasmukhrai Trivedi
[B.com, M.Sc I.T (Information Technology), P.G.D.B.M –Pursuing]
Senior System Administrator, S.E.C.C [Socio Economic and Cast Census]
Central Govt. Project – Bhavnagar [Gujarat – India],

ABSTRACT: Software testing provides a means to reduce errors, cut maintenance and overall software costs. Numerous software development and testing methodologies, tools, and techniques have emerged over the last few decades promising to enhance software quality. While it can be argued that there has been some improvement it is apparent that many of the techniques and tools are isolated to a specific lifecycle phase or functional area. One of the major problems within software testing area is how to get a suitable set of cases to test a software system. This set should assure maximum effectiveness with the least possible number of test cases. There are now numerous testing techniques available for generating test cases.

KEYWORDS: Software Testing, Level of Testing, Testing Technique, Testing Process

I. 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. Modern software systems must be extremely reliable and correct. Automatic methods for ensuring software correctness range from static techniques, such as (software) model checking or static analysis, to dynamic techniques, such as testing. All these techniques have strengths and weaknesses: model checking (with abstraction) is automatic, exhaustive, but may suffer from scalability issues. Static analysis, on the other hand, scales to very large programs but may give too many spurious warnings, while testing alone may miss important errors, since it is inherently incomplete.

IEEE STANDARDS: Institute of Electrical and Electronics Engineers designed an entire set of standards for software and to be followed by the testers. **i.e.** Standard Glossary of Software Engineering Terminology, Standard for Software Quality Assurance Plan,Standard for Software Configuration Management Plan

→ **What is Software Testing?**

Software testing is more than just error detection; Testing software is operating the software under controlled conditions, to (1) verify that it behaves "as specified"; (2) to detect errors, and (3) to validate that what has been specified is what the user actually wanted.

1. Verification is the checking or testing of items, including software, for conformance and consistency by evaluating the results against pre-specified requirements. [*Verification: Are we building the system right?*]
2. Error Detection: Testing should intentionally attempt to make things go wrong to determine if things happen when they shouldn't or things don't happen when they should.
3. Validation looks at the system correctness – i.e. is the process of checking that what has been specified is what the user actually wanted. [*Validation: Are we building the right system?*]

The definition of testing according to the ANSI/IEEE 1059 standard is that testing is the process of analysing 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.

The purpose of testing is verification, validation and error detection in order to find problems – and the purpose of finding those problems is to get them fixed.

Most Common Software problems: Inadequate software performance, Data searches that yields incorrect results. Incorrect data edits & ineffective data edits, Incorrect coding / implementation of business rules, Incorrect calculation, Incorrect data edits and ineffective data edits, Incorrect processing of data relationship, Incorrect or inadequate interfaces with other systems, Inadequate performance and security controls, Incorrect file handling, Inadequate support of business needs, Unreliable results or performance, Confusing or misleading data, Software usability by end users & Obsolete Software, Inconsistent processing.

→ **Terminology:**

- **Mistake** – A human action that produces an incorrect result.
- **Fault [or Defect]** – An incorrect step, process, or data definition in a program.

- **Failure** – The inability of a system or component to perform its Required function within the specified performance requirement.
- **Error** – The difference between a computed, observed, or Measured value or condition and the true, specified, or theoretically correct value or condition.
- **Specification** – A document that specifies in a complete, precise, Verifiable manner, the requirements, design, behaviour, or other Characteristic of a system or component, and often the Procedures for determining whether these provisions have been Satisfied. We observe errors, which can often be associated with failures. But the ultimate cause of the fault is often very hard to find.

II. OBJECTIVES

- A good test case is one that has a probability of finding an as yet undiscovered error.
- A good test is not redundant.
- A successful test is one that uncovers a yet undiscovered error.
- A good test should be "best of breed".
- A good test should neither be too simple nor too complex.
- To check if the system does what it is expected to do.
- To check if the system is "Fit for purpose".
- To check if the system meets the requirements and be executed successfully in the Intended environment.
- Executing a program with the intent of finding an *error*.

Testing Type	Specification	General Scope	Opacity	Who does it?
Unit	Low-level design: actual code	Classes	White box	Programmer
Integration	Low-level design: High level design	Multiple Classes	White box; black box	Programmer
Function	High level design	Whole Product	Black box	Independent Tester
System	Requirement Analysis	Whole Product in Environment	Black box	Independent Tester
Acceptance	Requirement Analysis	Whole Product in Environment	Black box	Customer
Beta	Ad hoc	Whole Product in Environment	Black box	Customer
Regression	Changed documentation; High level design	Any of the above	Black box; white box	Programmer of Independent Tester

III. SOFTWARE TESTING LIFECYCLE – PHASES

1. **Requirements study**
   - Testing Cycle starts with the study of client's requirements.
   - Understanding of the requirements is very essential for testing the product.
2. **Test Case Design and Development**
   - Component Identification
   - Test Specification Design
   - Test Specification Review
3. **Test Execution**
   - Code Review
   - Test execution and evaluation
   - Performance and simulation

4. **Test Closure**
   - Test summary report
   - Project De-brief
   - Project Documentation
5. **Test Process Analysis**
   - Analysis done on the reports and improving the application's performance by implementing new technology and additional features.

IV. LEVEL OF TESTING

1. **Unit testing**
   - The most 'micro' scale of testing.
   - Tests done on particular functions or code modules.
   - Requires knowledge of the internal program design and code.
   - Done by Programmers (not by testers).

Objectives	To test the function of a program or unit of code such as a program or module,To test internal logic,To verify internal design,To test path & conditions coverage,To test exception conditions & error handling,To test the function of a program or unit of code such as a program or module,To test internal logic,To verify internal design,To test path & conditions coverage,To test exception conditions & error handling
When	After modules are coded
Who	Developer
Input	Internal Application Design,Master Test Plan,Unit Test Plan
Output	Unit Test Report
Methods	White Box testing techniques,Test Coverage techniques
Tools	Debug,Re-structure,Code Analyzers,Path/statement coverage tools

2. **Incremental Integration Testing**
   - Continuous testing of an application as and when a new functionality is added.
   - Application's functionality aspects are required to be independent enough to work separately before completion of development.
   - Done by programmers or testers.

Integration Testing
- Involves building a system from its components and testing it for problems that arise from component interactions.
- **Top-down integration:**
  - Develop the skeleton of the system and populate it with components.
- **Bottom-up integration**
  - Integrate infrastructure components then add functional components.
- To simplify error localization, systems should be incrementally integrated.

Objectives	To technically verify proper interfacing between modules, and within sub-systems
When	After modules are unit tested
Who	Developers
Input	Internal & External Application Design ,Master Test Plan ,Integration Test Plan
Output	Integration Test report
Methods	White and Black Box techniques ,Problem / Configuration Management
Tools	Debug ,Re-structure ,Code Analyzers

3. **Functional Testing**
   - Black box type testing geared to functional requirements of an application.
   - Done by testers.

4. System Testing

Objectives	To verify that the system components perform control functions,To perform inter-system test,To demonstrate that the system performs both functionally and operationally as specified,To perform appropriate types of tests relating to Transaction Flow, Installation, Reliability, Regression etc.
When	After Integration Testing
Who	Development Team and Users
Input	Detailed Requirements & External Application Design,Master Test Plan,System Test Plan
Output	System Test Report
Methods	Problem / Configuration Management
Tools	Recommended set of tools

5. Acceptance Testing

Objectives	To verify that the system meets the user requirements
When	After System Testing
Who	User / End User
Input	Business Needs & Detailed Requirements,Master Test Plan,User Acceptance Test Plan
Output	User Acceptance Test report
Methods	Black Box techniques,Problem / Configuration Management
Tools	Compare, Keystroke capture & Playback, Regression testing

6. Beta Testing

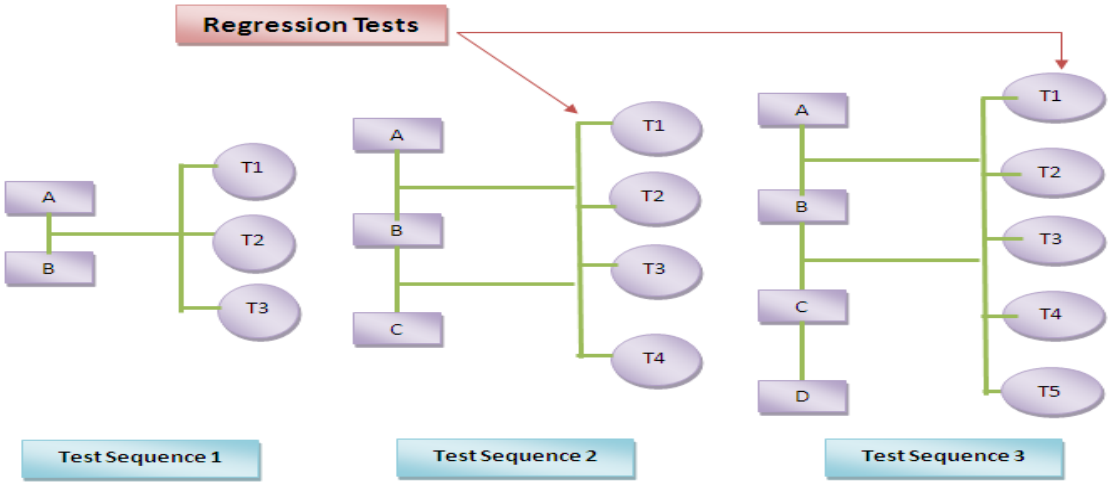
- Relied on too heavily by large vendors, like Microsoft.
- Allow early adopters easy access to a new product on the condition that they report errors to vendor. A good way to stress test a new system.

7. End-to-end Testing

- Similar to system testing; involves testing of a complete application environment in a situation that mimics real-world use.

8. Regression Testing

- Re-testing after fixes or modifications of the software or its environment.



9. Sanity Testing

- Initial effort to determine if a new software version is performing well enough to accept it for a major testing effort.

10. Load Testing

- Testing an application under heavy loads.
- **I.e.** Testing of a web site under a range of loads to determine, when the system response time degraded or fails.

11. Stress Testing

- Testing under unusually heavy loads, heavy repetition of certain actions or inputs, input of large numerical values, large complex queries to a database etc.
- Term often used interchangeably with 'load' and 'performance' testing.

12. Performance Testing
- Testing how well an application complies to performance requirements.

13. Install/uninstall Testing:
- Testing of full, partial or upgrade install/uninstall process.

14. Recovery Testing
- Testing how well a system recovers from crashes, HW failures or other problems.

15. Compatibility Testing
- Testing how well software performs in a particular HW/SW/OS/NW environment.

16. Loop Testing
- This white box technique focuses on the validity of loop constructs. four different classes of loops can be defined (1).simple loops (2).nested loops(3).concatenated loops(4).Unstructured loops

17. Recovery Test
- Confirms that the system recovers from expected or unexpected events without loss of data or functionality
- **I.e.** Shortage of disk space and unexpected loss of communication Power out conditions

V. TEST PLAN

A]. Purpose of preparing a Test Plan
- Validate the acceptability of a software product.
- Help the people outside the test group to understand 'why' and 'how' of product validation.
- A Test Plan should be
  - Thorough enough (Overall coverage of test to be conducted)
  - Useful and understandable by the people inside and outside the test group.

B]. Scope
- The areas to be tested by the QA team.
- Specify the areas which are out of scope (screens, database, mainframe processes etc).

C]. Test Approach
- Details on how the testing is to be performed.
- Any specific strategy is to be followed for testing (including configuration management).

VI. TESTING METHODOLOGIES AND TYPES

Here, I have considered the two testing methodology and types that is mention above:

1. Black box testing
2. White box testing

1. BLACK BOX TESTING
- No knowledge of internal design or code required.
- Tests are based on requirements and functionality

1.1 Black Box - Testing Technique
- Incorrect or missing functions
- Interface errors
- Errors in data structures or external database access
- Performance errors
- Initialization and termination errors

1.2. Black box / Functional Testing
- Based on requirements and functionality
- Not based on any knowledge of internal design or code
- Covers all combined parts of a system
- Tests are data driven

2. WHITE BOX TESTING
- Knowledge of the internal program design and code required.
- Tests are based on coverage of code statements, branches, paths, conditions.

2.1 White Box - Testing Technique
- All independent paths within a module have been exercised at least once
- Exercise all logical decisions on their *true* and *false* sides

- Execute all loops at their boundaries and within their operational bounds
- Exercise internal data structures to ensure their validity

2.2 White box Testing / Structural Testing

- Based on knowledge of internal logic of an application's code
- Based on coverage of code statements, branches, paths, conditions
- Tests are logic driven

2.3 Other White Box Techniques

- Statement Coverage – execute all statements at least once
- Example :

```
A + B
If (A = 3) Then
        B = X + Y
End-If

While (A > 0) Do
        Read (X)
        A = A - 1
End-While-Do
```

- Decision Coverage – execute each decision direction at least once
- Example:

```
If A < 10 or A > 20 Then
        B = X + Y
End -if
```

- Condition Coverage – execute each decision with all possible outcomes at least once
- Example:

```
A = X
If (A > 3) or (A < B) Then
        B = X + Y
End-If-Then

While (A > 0) and (Not EOF) Do
        Read (X)
        A = A – 1
End-While-Do
```

VII. CONCLUSION

- Testing can show the presence of faults in a system; it cannot prove there are no remaining faults.
- Component developers are responsible for component testing; system testing is the responsibility of a separate team.
- Integration testing is testing increments of the system; release testing involves testing a system to be released to a customer.
- Use experience and guidelines to design test cases in defect testing.
- Interface testing is designed to discover defects in the interfaces of composite components.
- Equivalence partitioning is a way of discovering test cases - all cases in a partition should behave in the same way.
- Structural analysis relies on analysing a program and deriving tests from this analysis.
- Test automation reduces testing costs by supporting the test process with a range of software tools.

VIII. REFERENCES

[1]. Lessons Learned in Software Testing, by C. Kaner, J. Bach, and B. Pettichord
[2]. Testing Computer Software, by C. Kaner, J. Falk, and H. Nguyen
[3]. Effective Software Testing, by E. Dustin
[4]. Software testing, by Ron Patton
[5]. Software engineering, by Roger Pressman
[6]. http://people.engr.ncsu.edu/txie/testingresearchsurvey.htm
[7]. http://www.engpaper.com
[8]. http://www.people.engr.ncsu.edu/txie/testingresearchsurvey.htm
[9]. www.cs.cmu.edu/luluo/Courses/17939Report.pdf
[10]. www.findwhitepapers.com
[11]. www.scribd.com

