Different Software Testing Strategies and Techniques

P. B. Selvapriya

Abstract— In this paper main testing strategies and techniques are precisely described. General classification is outlined: two testing methods – black box testing and white box testing, and their often used techniques:

Black Box techniques: Equivalent Partitioning, Cause-Effect Graphing Techniques, Boundary Value Analysis and Comparison Testing.

White Box techniques: Loop Testing, Basis Path Testing, and Control Structure Testing.

Keywords- Boundary Value Analysis and Comparison Testing.

I. INTRODUCTION

Definition and The Goal Of Testing

PROCESS of creating a program consists of the following phases: 1. defining a problem; 2. designing a program; 3. constructing a program; 4. analyzing performances of a program, and 5. final arranging of a product. According to this division, software testing comes under the third phase, and means checking if a program for specified inputs gives correctly and expected results.

Software testing (Figure 1) is an important component of software quality assurance. Many software organizations are spending nearly 40% of their resources (both economical and HR) on testing. For life-critical software (e.g., flight control) testing can be highly expensive.

Because of that, many studies about **risk analysis** have been made. There are a many definitions of **software testing**, but one can shortly define that as: A **process of executing a program with the goal of finding errors**.

So, testing means that one inspects behavior of a program on a finite set of **test cases** (a set of the inputs given, execution with preconditions, and expected outcomes for a particular constrains, such as to proceed a particular program path or to verify compliance with a specific requirement, for which valued inputs always exist.

Practically, the whole set of test cases is considered as infinite, hence theoretically there are too many test cases even for the simplest programs. In this case, testing could require months and months to execute.

The problem is to select the most proper and appropriate set of test cases. In practice, various techniques are used for that, and some of them are correlated with risk analysis, while others with test engineering expertise.

Testing is an activity performed for evaluating software quality and for improving it. Hence, the goal of testing is systematical detection of different classes of errors (**error** can be defined as a human action that produces an incorrect result) in a minimum amount of time and with a minimum amount of effort.

Manuscript received December, 2013.

P.B.Selvapriya, Computer Science and Engineering, N.S.N. College of Engineering and Technology, Karur, Tamilnadu, India.

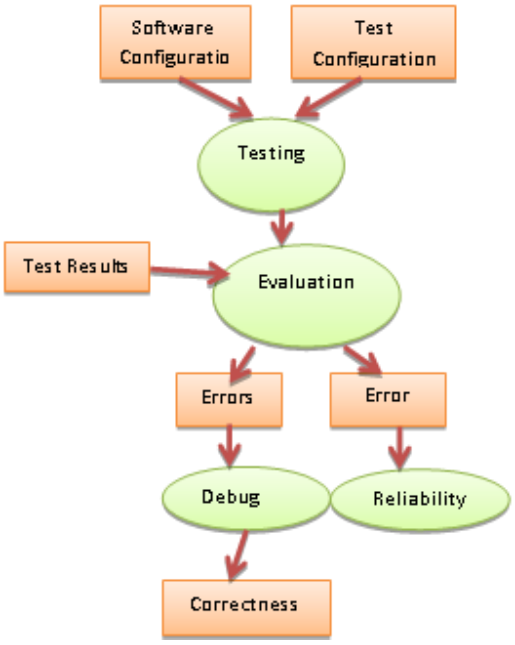


Figure 1: Test Information Flow

- Good test cases - have a good chance of finding an yet undiscovered error; and
- Successful test cases - uncovers a new error.

Anyway, a good test case is one which validate whether code implementation follows intended design, to validate implemented security functionality, and to uncover exploitable vulnerabilities.

II. TESTING METHODS AND TYPES

Test cases are developed using various test techniques to achieve more effective testing. Using 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 be selected and test techniques to design testing conditions in a systematic way. If one combines all sorts of previously existing test techniques, we will obtain better results rather if one uses just one test technique.

Software can be tested in two ways or two different methods:

1. Black box testing and
2. White box testing.

White box testing is highly effective in detecting and resolving problems, because bugs (**bug** or **fault** is a manifestation of an error in a software) can often be found before they cause trouble. We can define this method as testing software with the knowledge of the internal structure and coding inside the program. White box testing is also called glass box testing, clear box testing or clear box analysis.

It is a strategy for software **debugging** (it is the process of locating and fixing bugs in computer program code or the engineering of a hardware device) in which the tester has excellent knowledge of how the program's various components interact.

This method can be widely used for Web services applications.

Besides, in white box testing is considered as a **security testing** (the process to determine that an information system protects data and maintains functionality as intended) method that can be used to reduce errors.

Black box testing is testing software based on output requirements and without any knowledge of internal structure or coding in the program. In another words, a black box is any device whose workings are not understood by or accessible to its user. In data mining, a black box is an algorithm that doesn't provide an explanation of how it works.

A black box is a dedicated hardware device in film–making. It is equipment that is specifically used for a particular function, but in the financial world, it is a computerized trading system that doesn't make its rules easily available. The main characteristics are described and comparison between white box testing and black box testing are follows.

2.1. Black Box Testing Versus White Box

Testing

Black Box Testing:

- Performing the tests which exercise all functional requirements of a program;
- Finding the following errors:

1. Incorrect or missing functions;
2. Interface errors;
3. Errors in data structures or external database access;
4. Performance errors;
5. Initialization and termination errors.

- Advantages of this method:

- The number of test cases are minimized to achieve reasonable testing;
- The test cases can show absence or presence of classes of errors.

White Box Testing:

- Considering the internal logical arrangement of software;
- The test cases exercise certain sets of conditions and loops;
- Advantages of this method:

- All independent paths in a module will be used at least once;
- All logical decisions will be used;
- All loops at their boundaries will be executed;
- Internal data structures will be used to maintain their validity.

III. GENERAL CLASSIFICATION OF TEST TECHNIQUES

In this paper, the most important test techniques are shortly described, as it is shown in Figure 2.

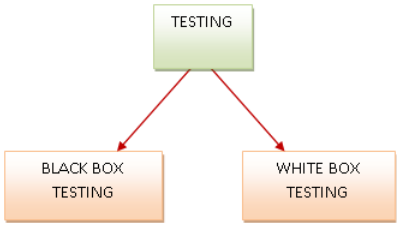


Figure 2: General Classification of Test Techniques

3.1. Equivalence Partitioning

Summary: equivalence class

This technique divides the input domain of a program onto equivalence classes.

Equivalence classes – set of valid or invalid states for input conditions, and can be defined in the following way:

1. An input condition specifies a range → one valid and two invalid equivalence classes are defined.
2. An input condition needs a specific value → one valid and two invalid equivalence classes are defined;
3. An input condition specifies a member of a set→ one valid and one invalid equivalence class are defined.

An input condition is Boolean→ one valid and one invalid equivalence class are defined.

Using this technique, we can get test cases which identify the classes of errors.

3.2. Boundary Value Analysis

Summary: complement equivalence partitioning

This technique is similar to Equivalence Partitioning, except that for creating the test cases beside input domain use output domain.

We can form the test cases in the following way:

1. An input condition specifies a range bounded by values a and→ b test conditions;

If internal program data structures have indicated boundaries, produce test cases to exercise that data structure at its boundary.

3.3. Cause-Effect Graphing Techniques

Summary: translate

One uses this technique when one wants to translate a policy or procedure specified in a natural language into software's language.

This technique means:

Input conditions and actions are listed for a module ⇒ an identifier is allocated for each one of them ⇒ cause-effect graph is created

- this graph is changed into a decision table
- The rules of this table are modified to test cases.

3.4. Comparison Testing

Summary: independent versions of an application

In situations when reliability of software is critical, redundant software is produced. In that case one uses this technique.

This technique means:

Software engineering teams produce independent versions of an application→ each version can be tested with the same test data → so the same output can be ensured.

Residual black box test techniques are executing on the separate versions.

3.5. Fuzz Testing

Summary: random input

Fuzz testing is often called negative testing, fuzzing or robustness testing. It is developed by Barton Miller at the University of Wisconsin in 1989. This technique feeds random input to application. The main characteristic of fuzz testing, according to the are:

- the input is random;
- the reliability criteria: if the application hangs or crashes eventually the test is failed;

- fuzz testing can be automated to a high degree.

A tool called fuzz tester which indicates causes of founded vulnerability, works best for problems that can cause a program to crash such as buffer overflow, cross -site scripting, denial of service attacks, format bug and SQL injection. Fuzzing is less effective for spyware, some viruses, worms, Trojans, and key loggers. However, fuzzers are most effective when are used together with extensive black box testing techniques.

3.6. Model-based testing

Model-based testing is automatic generation of efficient test procedures/vectors using models of system requirements and specified functionality.

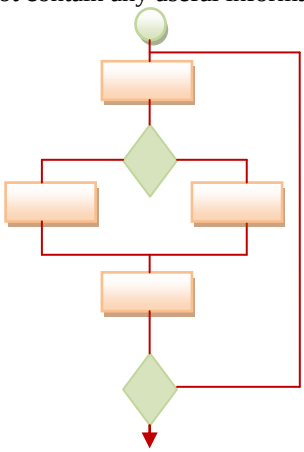
These test cases are referred as the abstract test suite and for their selection the following techniques have been used:

- generation by theorem proving;
- generation by constraint logic programming;
- generation by model checking;
- generation by symbolic execution;
- generation by using an event-flow model;
- generation by using an Markov chains model.

3.7 Basis Path Testing

Summary: basis set, independent path, flow graph, cyclomatic complexity, graph matrix, link weight Forobtaining the basis set and for presentation control flow in the program, one uses **flow graphs** (Figure 3). Main components of that graphs are:

- Node – it represents one or more procedural statements. Node which contains a condition is called predicate node.
- Edges between nodes – represent flow of control. Each node must be bounded by at least one edge, even if it does not contain any useful information.



Region – an area or coverage space bounded by nodes and edges

Figure 3: Flow Graph

Cyclomatic Complexity is software metric. The value evaluated for cyclomatic complexity defines the number of independent paths in the basis set of a program.

For the given graph G, cyclomatic complexity V(G) is equal to:

1. The number of regions in the flow graph;
2. **V(G) = E - N + 2**, where E denotes the number of edges and N denotes the number of nodes.

V(G) = P + 1, where P is the number of predicate nodes.

So, the core of this technique is: one draws the flow graph according to the design or code like the basis ⇒ one determines its cyclomatic complexity; cyclomatic complexity can be determined without a flow graph → in that case one computes the number of conditional statements in the code ⇒ after that, one determines a basis set of the linearly independent paths; the predicate nodes are useful when necessary paths must be determined ⇒ at the end, one prepares test cases by which each path in the basis set will be executed. Each test case will be executed and compared to the expected results.

Example1. (Cyclomatic complexity)

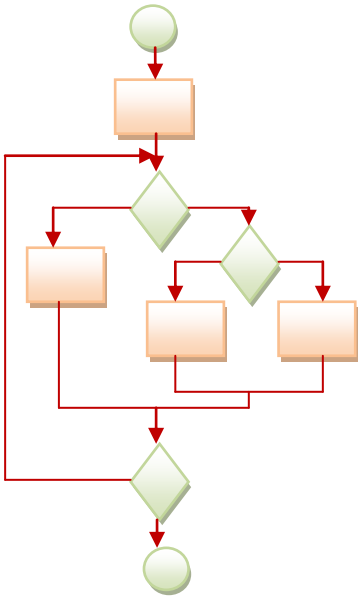


Figure 4

The cyclomatic complexity for the upper graph (figure 4) is:

- V(G) = the number of predicate nodes +1 = 3+1 =4, or
- V(G)= the number of simple decisions +1 = 4.

Well, V(G) = 4, so there are four independent paths:

The path 1: 1, 2, 3, 6, 7, 8 The path 2: 1, 2, 3, 5, 7, 8; The path 3: 1, 2, 4, 7, 8;

The path 4: 1,2,4,7,2,4,…,7,8.

Now, test cases should be designed to execute these paths.

3.8. Loop Testing

There are four types of loops

- Simple loops
- Concatenated loops
- Nested loops
- Unstructured loops

3.9. Control Structure Testing

Two main components of classification of Control Structure Testing are described below.

3.9.1. Condition Testing

By this technique, each logical condition in a program is tested. A **relational expression** takes a form:

E_1 < relational - operator >E_2, where E1 and E2 are arithmetic expressions and relational operator is one of the following: <, =, ≠, ≤, >, or ≥.

A **simple condition** is a relational expression or a boolean variable, possibly with one NOT operator.

A **compound condition** is composed of two or more simple conditions, Boolean operators, and parentheses.

3.9.2. Data Flow Testing

By this technique, one can choose test paths of a program based on the locations of definitions and uses of variables in a program.

Unique **statement number** is allocated for each statement in a program. For statement with S as its statement number, one can define:

DEF(S) = {X| statement S contains a definition of X}

USE(S) = {X| statement S contains a use of X}.

The definition of a variable X at statement S is live at statement S' if there exists a path from statement S to S' which does not contain any condition of X.

A definition-use chain (or DU chain) of variable X is of the type [X, S, S'] where S and S' are statement numbers, X is in DEF(S), USE(S'), and the definition of X in statement S is live at statement S.

V. CONCLUSION

Software testing is an important component or phase in development lifecycle of a project. Another important component is **software quality control** (**SQC**). SQC means control the quality of software engineering products, which is conducting using tests of the software system. These tests can be: unit tests (this testing checks each coded module for the presence of bugs), integration tests (connects sets of previously tested modules to ensure that the sets behave as well as they did as independently tested modules), or system tests (checks that the entire software system embedded in its actual hardware environment behaves according to the requirements document).

SQC is different from **software quality assurance** (**SQA**), which means control the software engineering processes and methods that are used to ensure quality. Control conduct by inspecting quality management system. One or more standards can be used for that. It is usually ISO 9000. SQA relates to the whole software development process, which includes the following events: coding, software design, code reviews, source code control, change management, release management and configuration management.

Finally we can come to conclusion that SQC is a control of products, and SQA is a control of processes.

Huge losses can be avoided if timely testing and discovering bugs in initial phases of testing are conducting. Therefore software testing is greatly important, and test techniques too are important.

There is considerable controversy between software testing writers and consultants about what is important in software testing and what constitutes responsible software testing.

So, some of the major controversies include:

- **What constitutes responsible software testing?** – Members of the "context-driven" school of testing believe that the "best practices" of software testing don't exist, but that testing is a collection of skills which enable testers to choose or improve test practices proper for each unique situation.
- **Traditional vs. Agile** – Agile testing is popular in commercial circles and military software providers. Some researchers think that testers should work under conditions of uncertainly and constant change, but others think that they should aim to at process "maturity".
- **Scripted vs. Exploratory** – Some researchers believe that tests should be created at time when they are executed, but others believe that they should be designed beforehand.
- **Manual vs. automated** – Propagators of agile development recommend complete automation of all test cases. Others believe that test automation is pretty expensive.
- **Software design vs. software implementation** – The question is: Should testing be carried out only at the end or throughout the whole process?
- **Who observes the watchmen** – Any form of observation is an interaction, so the act of testing can affect an object of testing.

REFERENCES

1. http://www.his.sunderland.ac.uk/~cs0mel/comm83wk5.doc, February 08, 2009.
2. Stacey, D. A., "Software Testing Techniques"
3. Guide to the Software Engineering Body of Knowledge, Swebok – A project of the IEEE Computer Society Professional Practices Committee, 2004.
4. "Software Engineering: A Practitioner's Approach, 6/e; Chapter 14: Software Testing Techniques", R.S. Pressman & Associates, Inc., 2005.
5. Myers, Glenford J., IBM Systems Research Institute, Lecturer in Computer Science, Polytechnic Institute of New York, "The Art of Software Testing", Copyright 1979. by John Wiley & Sons, Inc.
6. Wikipedia, The Free Encyclopedia, http://en.wikipedia.org/wiki/
7. http://www2.umassd.edu/CISW3/coursepages/pages/CI S311/outline.html
8. Wei–Tek, Tsai, "Risk – based testing", Arizona State University, Tempe, AZ 85287
9. Redmill, Felix, "Theory and Practice of Risk-based Testing", Software Testing, Verification and Reliability, Vol. 15, No. 1, March 2005.
10. http://www.testingstandards.co.uk/living_glossary.htm# Testing, February 08, 2009.
11. http://www.pcmag.com/encyclopedia_term/0,2542,t=b lack+box+testing&i=38733,00.asp, February 08, 2009.
12. http://www.pcmag.com/encyclopedia_term/0,254 2,t=gray+box+testing&i=57517,00.asp, February 08, 2009.
13. "Software Testing Methods and Techniques"-

AUTHORS PROFILE

P.B.Selvapriya, received her B.E. (Computer Science and Engineering) degree from Avinashilingam University, Coimbatore in 2008 and M.E. (Computer Science and Engineering) degree from Anna University, Chennai in 2012. She is presently working as Assistant Professor in the department of Computer Science and Engineering, N.S.N College of Engineering and Technology, Karur.