ABSTRACT

This paper aims to impart reasonably detailed information regarding the components that make up Dynamic Analysis Testing and how to carefully select the most suitable testing methodology to meet deadlines without compromising on the scope and quality of testing performed.

INTRODUCTION

Dynamic Analysis is what is generally considered as ``testing``, i.e. it involves running the system. ``The analysis of the behavior of a software system before, during and after its execution in an artificial or real application environment characterizes dynamic analysis''.

Software Testing comprises of multiple stages, beginning with functional specifications analysis leading to breakdown of independently testable features that help in determining the best testing methodology. As time and budget constraints plague the testing phase, accurately determining the ideal testing technique(s) can be paramount in delivering a product that thoroughly meets the stringent specification requirements.

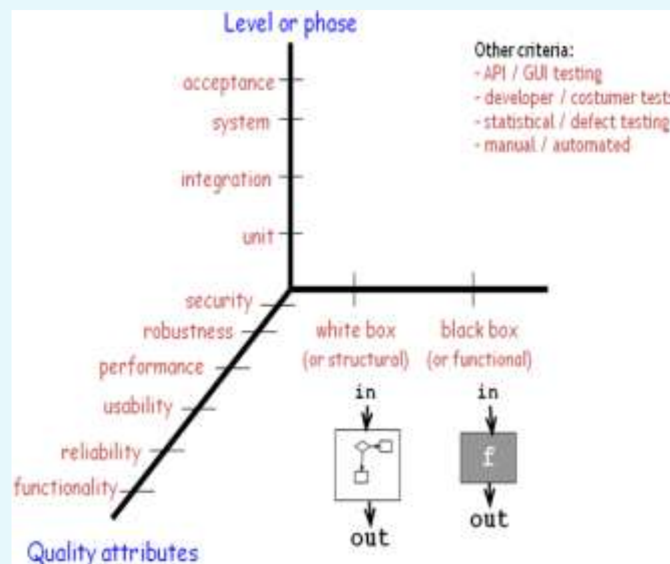

Figure 1

With various testing strategies and techniques available, it can sometimes become daunting for a quality assurance or test engineer, to determine which testing technique to adopt in order to thoroughly test a software module. Moreover, the scope of the software project also plays a major role in making that important decision, in addition to the detail available in functional specifications. Incorrect identification of testing technique will lead to valuable time being lost, as deadlines cannot be adhered to, leading to a ripple effect resulting in delays to be introduced at each of the dependant subsequent SDLC stages.

For example;

"How can a QA engineer determine how many values to test such that considerable pool of potential faults has been accounted for"?

It may be impossible to test all values given the time constraints, therefore knowledge of test techniques which minimize time and effort allocation whilst maximizing the test results (within minimum test cases) becomes vital or for example; "

How many test cycles should be executed before a large percentage of potential faults have been covered to ensure a reasonably bug free software has been developed"?

As you read further, various dynamic analysis testing techniques will be discussed in detail, to enable you to better understand and perform thorough software testing.

© 2011 Kualitatem (Pvt) Ltd. ALL RIGHTS RESERVED

Characteristics of Dynamic Analysis Testing

Dynamic analysis is the testing and evaluation of a program by executing data in real-time. The objective is to find errors in a program while it is running, rather than by repeatedly examining the code offline.

By debugging a program in all the scenarios for which it is designed, dynamic analysis eliminates the need to artificially create situations likely to produce errors. Advantages include:

1. Reducing the cost of testing and maintenance
2. Identifying and eliminating unnecessary program components
3. Ensuring that the program being tested is compatible with other programs.
4. Dynamic analysis can collect exactly the information needed to solve a problem
   - Procedure specialization: parameter values
   - Dynamic program slicing: flow dependences
   - Race conditions: message sends
5. Scales very well
6. Can be language independent!
   - Record information at interfaces

Program + Input = Behavior

Input + behavior as a guide to the program which reflects that input is very sensitive and one should be careful while its selection.

Dynamic analysis techniques involve the running of the program formally under controlled circumstances and with specific results expected. It shows whether a system functions as expected in the states under examination or not.

Among the most important dynamic analysis techniques are path and branch testing. During dynamic analysis path testing all possible logical paths of a program are executed. The major quality attribute measured by path testing is program complexity. Branch testing requires that tests be constructed in a way that every branch in a program is traversed at least once. Issues reported while running these branches lead to the probability of defects occurring at a later stage.

Today there are a number of dynamic analyzers that are used during the software development process. The most important tools are presented:

Type of Dynamic Analyzer	Functionality of Tool
Test Coverage Analysis	Tests to which extent the code can be checked by glass box techniques
Tracing	Follows all paths used during program execution and provides e.g. values for all variables etc.
Tuning	Measures resources used during program execution
Simulator	Simulates parts of systems, if e.g. the actual code or hardware are not available
Assertion Checking	Tests whether certain conditions are given in complex logical constructs

© 2011 Kualitatem (Pvt) Ltd. ALL RIGHTS RESERVED

Choosing the "Right" Testing Approach

The answer to the question on "What testing approach to apply to a specific module or an entire program" lies in in-depth knowledge of available testing techniques and being able to utilize the most suitable technique given the nature of the program under development.

Other factors like time constraints, depth of testing along with scope as well as budget available for testing and resource availability will significantly determine the most viable testing strategy.

Furthermore it is also vital that all testing teams are synchronized with respect to the testing process applicable during the testing phase of the project.

"Dynamic testing is about cure". A range of approaches adopted to test functionality of the program are discussed as:

Boundary Value Test Technique aims to identify errors which normally occur at the boundary/extreme of the input value. The extreme could be slightly below, or slightly above the selected input value.

This is evident mostly in programs written in languages which fail to provide the software engineer with sufficient control whilst coding so that adequate checks can be implemented to handle such errors.

With all testing techniques, boundary value analysis has its inherent limitations. For example, it is effective with physical variables having fixed boundaries, in comparison to logical variables i.e., color ranges or product categories.

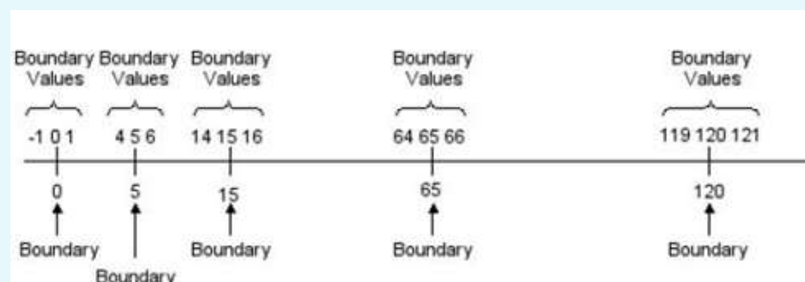

Figure 2

Boundary value analysis helps in distinguishing between logical variables i.e., types of products and physical variables i.e., temperature ranges. However, because it is simple to implement, boundary value analysis test cases can be easily automated, thereby saving time during testing.

Robustness Testing is a branch of boundary value analysis, and works on the concept of extending the 5 boundary values identified in boundary value analysis. This enables the software to be tested in terms of how well it can handle exceptions i.e., values that should not be accepted, and respond by displaying a corresponding error message as a result of run-time error.

Usually in a programming language where the coder does not have the ability to explicitly define unacceptable values, a range maybe provided for simplicity or ease of coding, to handle a range of unacceptable values. On the contrary, in languages where the coder has the ability to use explicit logic, it will provide the coder the opportunity to define explicit code to handle values which fall outside the range.

Worst Case Testing is ideal for applications with physical inputs with variable values leading to an output, and the results of an incorrect output being extremely high in terms of cost. As discussed earlier where boundary values were identified which a variable should be tested with, in the case of more than one variable, we will have **(number of boundary values) * n** test cases, where n is the number of variables being tested.

© 2011 Kualitatem (Pvt) Ltd. ALL RIGHTS RESERVED

As we've observed that boundary value analysis has been used on input variables, it can also be used on the output range such as error messages. Not only that, boundary value analysis can also be used on internal variables i.e., programming loop control variables, pointers etc.

Equivalence Class Testing is a domain based testing holds a common approach is to divide the input domain into Equivalence Classes such that the program can reasonably be expected to behave the same for any points within a class. It has two categories. The *Strong Normal EC Testing* cover the all combinations of equivalence classes for the domain of all variable i.e. multiple fault assumption. The *Weak Normal EC Testing* cover equivalence classes for the domain of each variable i.e. single fault assumption.

How is an equivalence class formed? To answer that question, you need to be able, to identify clear groups within a domain. However, at this stage it is important to remember, that an equivalence class is determined by some set of rules or conditions. Taking this one step further, it is important to note that in equivalence class testing, when one member of a class is tested, and catches a bug, so will the others. If one doesn't catch a bug, neither will the rest. Therefore, all members of an equivalence class behave in the same way.

Decision Table-based Testing is primarily used to summarize the functional requirements along with the expected outcomes under specific conditions into a table.

Moreover, this is implemented when testing complex business rules by assigning Boolean states of True (T) or False (F) to inputs, conditions and outputs in tabular format with rules represented, in columns, and actions (obtained from functional requirements) and conditions in rows.

Pair-wise Testing is used when the number of variables and their combinations is too large. It helps in identifying a minimal subset which enables testing for all single- and double-mode defects.

ENDING NOTES

There is no definite testing solution for any program under development. Knowledge and understanding of available dynamic analysis testing approaches enables the test engineer to make an informed decision on which approach is likely to maximum resource and time utilization, whilst providing near 100% test coverage.

This leads to software having undergone rigorous testing, within budget and time frames without compromising on quality. It is not only imperative to develop high quality software, but also one that meets customer expectations and performs as per requirement specifications, and delivers expected results.

REFERENCES

1. http://www.win.tue.nl/~mousavi/testing/2.pdf
2. http://users.encs.concordia.ca/~dssouli/Chap4-TT.pdf

Copyright in whole and in part of this document "Dynamic Analysis Testing" belongs to Kualitatem. Kualitatem is a testing outsourcing company which provides seamless services and solutions for software testing and QA processes. For more information, please visit www.kualitatem.com.

© 2011 Kualitatem (Pvt) Ltd. ALL RIGHTS RESERVED