Testing Tactics-WHAT IS IT?

Once source code has been generated software must be tested to uncover and correct as many errors as possible before delivering it to customer. Here software testing enters into picture. Techniques are applied to:

- Exercise the internal logic and interfaces of every software component
- Exercise the input and output domains of the program to uncover errors in program function, behavior and performance.
WHY IS IT IMPORTANCE?

- Reviews and other SQA activities can and do uncover errors, but they are not sufficient.
- The program is executed, the customer tests it!
- To execute the program before it gets to the customer with the specific intent of finding and removing all errors.
- To find the highest possible number of errors, tests must be conducted systematically and test cases must be designed using disciplined techniques.
What are the steps?

For conventional applications, software is tested from two different perspectives:

- Internal program logic is exercised using “white box” test case design techniques.
- Software requirements are exercised using “black box” test case design techniques.
- For object-oriented applications, “testing” begins prior to the existence of source code, but once code has been generated, a series of tests are designed to exercise operations with a class and examine whether errors exist as one class collaborates with others.
- Classes are integrated to form a subsystem, use-based testing, along with fault-based approaches, is applied to fully exercise collaborating classes.
Software Testing:-

- Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.
Who Tests the Software?

**developer**
Understands the system
but, will test "gently"
and, is driven by "delivery"

**independent tester**
Must learn about the system,
but, will attempt to break it
and, is driven by quality
WHAT IS A GOOD TEST

- **A good test has a high probability of finding error** - To achieve this goal, the tester must understand the software and attempt to develop a mental picture of how the software might fail. Ideally, the classes of failure are probed.

- **A good test is not redundant** - Testing time and resources are limited. There is no point in
conducting a test that has the same purpose as another test. Every test should have a different purpose (even if it is subtly different).

- **A good test should be “best of breed”** - In a group of tests that have a similar intent, time and resource limitations may mitigate toward the execution of only a subset of these tests. In such cases, the test that has the highest likelihood of uncovering a whole class of errors should be used.

- **A good test should be neither too simple nor too complex** - Although it is sometimes possible to
TESTING

Any engineered product (and most other things) can be tested in one of two ways:

- Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function. This is known as **BLACK BOX TESTING**.
Knowing the internal workings of a product, tests can be conducted to ensure that “all gears mesh”; that is internal operations are performed according to specifications, and all internal components have been adequately exercised. This is known as **WHITE BOX TESTING**.
BLACK BOX TESTING

- The tests are conducted at the software-interface.
- It examines some of the fundamental aspect of a system with little regard for the internal logical structure of the software.
WHITE BOX TESTING

- It is also known as glass-box testing.
- Uses the control structure described as part of component-level design to derive test cases.
- Using white-box testing methods, the software engineer can derive test cases that
  1. Guarantee that all independent paths within a module have been exercised at least once,
  2. Exercise all logical decisions on their true and false sides,
3. Execute all loops at their boundaries and within their operational bounds, and
4. Exercise internal data structures to ensure their

At first glance it would seem that very thorough white-box testing would lead to 100 percent correct programs. All we need to do is identify all logical paths, develop test cases to exercise them, and evaluate results, that is, generate test cases to exercise program logic exhaustively. Unfortunately, exhaustive testing presents certain logistical
Importance of Software Testing Tactics/Strategy

• Testing is an integral part in software development. It is broadly deployed in every phase in the software development cycle. Typically, more than 50% percent of the development time is spent in testing. Each and every tactics of testing are much important, we can see the importance of testing tactics in following points:-
To improve quality

- Quality means the conformance to the specified design requirement. Being correct, the minimum requirement of quality, means performing as required under specified circumstances. Debugging, a narrow view of software testing, is performed heavily to find out design defects by the programmer.
To improve quality

- The imperfection of human nature makes it almost impossible to make a moderately complex program correct the first time. Finding the problems and get them fixed, is the purpose of debugging in the programming phase.
Software reliability has important relations with many aspects of software, including the structure, and the amount of testing it has been subjected to. Based on an operational profile (an estimate of the relative frequency of use of various inputs to the program, testing can serve as a statistical sampling method to gain failure data for reliability estimation.
Testing plays an important role to avoid these situations

- Testing also avoid these situation to be occur….,
  - Cost of fixing the bug will be more if it is found in later stage than it is found earlier.
  - Quality can be ensured by testing only. In the competitive market, only Quality product can exist for long time.

Testing will be necessary even if it is not possible to do 100% testing for an application.
Testing plays an important role to avoid these situations

• One more important reason for doing testing is user/production environment will be completely different from development environment

• For example, a webpage developer may be using FireFox as browser for doing his webpage development. But the user may be using different browser such as Internet Explorer, Safari, Chrome and Opera.
Testing plays an important role to avoid these situations

- The web page appearing good in FireFox may not appear good in other browsers (particularly IE). So ultimately, user will not be happy even if the developer puts more efforts to develop the webpage. As we know that Users satisfaction is more important for growth of any business, testing becomes more important. So we can assume/treat the Testers as the representatives of the Users
Basis path testing

• Basis path testing is a white-box technique.

• Path testing is an approach to testing where you ensure that every path through a program has been executed at least once.

• Aim is to derive a logical complexity measure of a procedural design and use this as a guide for defining a basic set of execution paths.
1: Flow Graph Notation

- Notation for representing control flow
- On a flow graph:
Flow Graph Notation

- Arrows called *edges* represent flow of control
- Circles called *nodes* represent one or more actions.
- Areas bounded by edges and nodes called *regions*.
- A *predicate node* is a node containing a condition
- Any procedural design can be translated into a flow graph.
- Note that compound Boolean expressions at tests generate at least two predicate node and additional arcs.
Flow Graph Notation

- Each node in a flow graph represents a line in the program with an executable statement. By tracing the flow, therefore, you can see that the independent paths through the binary search flow graph are:
  - 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 14
  - 1, 2, 3, 4, 5, 14
  - 1, 2, 3, 4, 5, 6, 7, 11, 12, 5, …
  - 1, 2, 3, 4, 6, 7, 2, 11, 13, 5, …
Flow Graph Notation

• If all of these paths are executed we can be sure that every statement in the method has been executed at least once and that every branch has been exercised for true and false conditions. The number of tests that you need to ensure that all paths through the program are exercised is the same as the cyclomatic complexity of the code fragment that is being tested.
Flow graph for a binary search routine
2. Independent program paths

• An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.

• Cyclomatic complexity is a software metric that provides a quantitative measure of the logical complexity of a program.
2. Independent program paths

- Mathematically, the cyclomatic complexity of a structured program is defined with reference to the control flow graph of the program, a directed graph containing the basic blocks of the program, with an edge between two basic blocks if control may pass from the first to the second. The complexity $M$ is then defined as:
2. Independent program paths

- \( M = E - N + 2P \)
- where
  - \( E \) = the number of edges of the graph
  - \( N \) = the number of nodes of the graph
  - \( P \) = the number of connected components
2. Independent program paths

The same function as above, shown as a *strongly connected* control flow graph, for calculation via the alternative method. For this graph, \( E = 10, N = 8 \) and \( P = 1 \), so the cyclomatic complexity of the program is \( 10 - 8 + 1 = 3 \).
3. Deriving test cases

• As a procedural design the basis path testing is given as a series of steps.

• 1. Using the design or code as a foundation, draw a corresponding flow graph.

• 2. Determine the cyclomatic complexity of the resultant flow graph.
3. Deriving test cases

- 3. Determine a basis set of linearly independent paths.

- 4. Prepare test cases that will force execution of each path in the basis set.
4. **Graph Matrices**

- A graph matrix is a square matrix whose size is equal to the number of nodes on the flow graph.
- Each row and column corresponds to an identified node, and matrix entries correspond to connections between nodes.
2: Condition testing

- Condition testing is a test case design method that exercises the logical conditions contained in a program module.
- A simple condition is a Boolean variable or a relational expression, possibly preceded with one NOT operator, it takes form.
- E1<Relational-Operator>E2
Condition testing

• Where $E_1$ and $E_2$ are arithmetic expressions and $<\text{Relational-Operator}>$ is one of the following:
  • $<,\leq,=,\neq,\geq,$

• A compound condition is composed of two or more simple conditions, boolean operators and parentheses.
3: Loop testing

• Loop Testing is a white box testing technique that focuses exclusively on the validity of loop constructs.

• Four classes of loops can be defined: Simple loops, Concatenated loops, nested loops, and unstructured loops.
1. Simple Loops

- The following sets of tests can be applied to simple loops, where ‘n’ is the maximum number of allowable passes through the loop.
  1. Skip the loop entirely.
  2. Only one pass through the loop.
  3. Two passes through the loop.
  4. ‘m’ passes through the loop where m
  5. n-1, n, n+1 passes through the loop.
Diagrams

Simple  Nested  Concatenate  Unstructured
2. Nested Loops

- If we extend the test approach from simple loops to nested loops, the number of possible tests would grow geometrically as the level of nesting increases.

  1. Start at the innermost loop. Set all other loops to minimum values.

  2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter values. Add other tests for out-of-range or exclude values.
2. Nested Loops

3. Work outward, conducting tests for the next loop, but keep all other outer loops at minimum values and other nested loops to “typical” values.

4. Continue until all loops have been tested.
3. Concatenated Loops

Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other. However, if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then the loops are not independent.
4. **Unstructured Loops**

- Whenever possible, this class of loops should be redesigned to reflect the use of the structured programming constructs.
Boundary value analysis

• Boundary value analysis is a test case design technique that complements equivalence partitioning.
• Rather than selecting any element of equivalence class, BVA leads to the selection of test cases at the “edges” of the class.
• BVA leads to the selection of the test cases that exercise bounding values.
Test conduct for client server system

- Application function test
- Server test
- Database test
- Transaction test
- Network communication test
Application function test

- The application is tested in stand alone fashion

Server test

- The coordination and data management function of the server are tested

Database test

- The accuracy and integrity of data stored by the server is tested
Transaction test
  ◦ Tests focus on the correctness of processing and also an performance issues

Network communication tests
  ◦ These tests verify that communications among the nodes of the network occurs correctly and that message passing, transactions, and related network traffic occur without error.
Effective strategy in testing real-time system

• The intimate relationship that exists between real-time software and its hardware environment can also cause testing problems.

• Software tests must consider the impact of hardware faults on software processing. Such faults can be extremely difficult to simulate realistically.
Four step strategy can be proposed:

- **Task testing** – The first step in the testing of real-time software is to test each task independently. That is, conventional tests are designed and executed for each task. Each task is executed independently during these tests. Task testing uncovers errors in logic and function, but not timing or behavior.

- **Behavioral testing** – Using system models created with automated tools, it is possible to simulate the behavior of a real-time system and examine its behavior as a consequence of external events. These analysis activities can serve as the basis for the design of test cases that are conducted when the real-time software has been built.

- **Intertask testing** – Errors in individual tasks and in system behavior have been isolated, testing shifts to time-related errors. Asynchronous tasks that are known to communicate with one another are tested with different data rates and processing load to determine if intertask synchronization errors will occur. In addition, tasks that communicate via a message queue or data store are tested to uncover errors in the sizing of these data storage areas.
• **System testing** – Software and hardware are integrated and a full range of system tests are conducted in an attempt to uncover errors at the software / hardware interface. Most real-time systems process interrupts. Therefore, testing the handling of these Boolean events is essential. Using the state diagram and the control specification, the tester develops a list of all possible interrupts and the processing that occurs as a consequence of the interrupts.

• Tests are then designed to assess the following system characteristics:
  - Are interrupt priorities properly assigned and properly handled?
  - Is processing for each interrupt handled correctly?
  - Does the performance (e.g., processing time) of each interrupt-handling procedure conform to requiements?
  - Does a high volume of interrupts arriving at critical times create problems in function or performance?