Software Engineering

Software Testing Techniques

Software Testing Techniques

Programmers attempt to built s/w from an abstract concept to a tangible product.

Engineers creates a series of test cases that are intended to “demolish” the s/w that has been built.

Testing is one step in s/w process that could be viewed as destructive rather than constructive

Testing objectives

- Testing is process of executing a program with the intent of finding an error
- A good test case is one that has a high probability of finding as an yet undiscovered error
- A successful test is one that uncovers an as yet undiscovered error

testability

- Software testability is simply how easily a computer program can be tested
- Since testing is difficult , it pays to know what can be done to streamline it
- Sometimes programmers are willing to do things that will help the testing process and a check list of possible design point features
- There are certain metrics that could be used to measure testability in most of aspects ,sometime testability is used to mean how adequately a particular set of test will cover the product

Characteristics that lead to testable software

- Operability: “the better it works , the more efficiently it can be tested”
  - The system has few bugs(bugs adds analysis and reporting overhead to the test process)
  - No bugs block execution of tests
  - The product evolves in functional stages(allows simultaneous development and testing)

- Observability :”what you see is what you test”
  - Distinct output is generated for each input
  - System states and variables are visible or queriable during execution
  - Past system states and variables are visible or queriable e.g.(transaction log)
  - All factor affecting output are visible
  - Incorrect output is easily identified
  - Internal errors are automatically detected through self testing mechanism
  - Internal errors are automatically reported
  - Source code is accessible

- Controllability : “The better we can control the software , the more testing can be automated and optimized”
    - All possible output can be generated through some combination of input
    - All code is executable through some combination of input
    - Software and hardware states and variables can be controlled directly by the test engineer
    - Input and output formats are consistent and structured
    - Test can be conveniently specified , automated and reproduced

- Decomposability : “ By controlling the scope of testing , we can more quickly isolate problems and perform smarter retesting”
  - The software system is built from independent modules
  - Software modules can be tested independently

- Simplicity : “The less there is to test more quickly we can test it”
  - Functional simplicity e.g. the feature set is the minimum necessary to meet requirements
  - Structural simplicity e.g. architecture is modularized to limit the propagation of faults
  - Code simplicity e.g. a coding standard is adopted for ease of inspection and maintenance

- Stability : “the fewer the changes , the fewer the disruptions to testing”
  - Changes to software are infrequent
  - Changes to software are controlled
  - Changes to software do not invalidate existing tests
  - The software recover well from failures

- Understandability : “The more information we have , the smarter we will test”
  - Design is well understood
  - Dependencies between internal , external and shared components are well understood.
  - Changes to design are communicated
  - Technical documentation is instantly accessible
  - Technical documentation is well organized
  - Technical documentation is specified and detailed
  - Technical documentation is accurate

Attributes of a “good” test

- A good test has a high probability of finding an error
  - The tester must understand the software and how it might fail
- A good test is not redundant
  - Testing time is limited; one test should not serve the same purpose as another test
- A good test should be “best of breed”
  - Tests that have the highest likelihood of uncovering a whole class of errors should be used
- A good test should be neither too simple nor too complex
  - Each test should be executed separately; combining a series of tests could cause side effects and mask certain errors

Test case design

- There is only one rule in designing test case cover all features but do not make too many test cases
- The highest likelihood of finding the most errors with a minimum amount of time and effort

Two Unit Testing Techniques

- Black-box testing
  - Knowing the specified function that a product has been designed to perform, test to see if that function is fully operational and error free
  - Includes tests that are conducted at the software interface
  - Not concerned with internal logical structure of the software
- White-box testing
  - Knowing the internal workings of a product, test that all internal operations are performed according to specifications and all internal components have been exercised
  - Involves tests that concentrate on close examination of procedural detail
  - Logical paths through the software are tested
  - Test cases exercise specific sets of conditions and loops

White-box Testing

- Uses the control structure part of component-level design to derive the test cases
- These test cases
  - Guarantee that 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

- Logical errors and incorrect assumptions are inversely proportional to the probability that a program path will be executed
- We often believe that a logical path is not likely to be executed when, in fact , it may be executed on a regular basis
- Typographical errors are random

Basic path testing

- White-box testing technique proposed by Tom McCabe
- Enables the test case designer to derive a logical complexity measure of a procedural design
- Uses this measure as a guide for defining a basis set of execution paths
- Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing