Software Quality Assurance, Part 2

Charles Carter

November 2, 2017

Table of Contents

Software Testabilty

Software Testabilty

- Operability
- Observability
- Controllability
- Decomposability
- Simplicity
- Stability
- Understandability

Operability

The better it works, the more efficiently it can be tested.

Observability

What you see is what you test.

Controllability

The better you can control the software, the more the testing can be automated and optimized.

Decomposability

By controlling the scope of the program, the more quickly we can isolate problems and perform smarter testing.

Simplicity

The less there is to test, the quicker we can test it.

- Functional simplicity — the feature set is the minimum necessary to meet all requirements
- Structural simplicity — the architecture is modularized to inhibit the propagation of faults
- Code simplicity — the use of coding standards promoting inspection and maintenance

Stability

The fewer the changes, the fewer disruptions to testing.

Understandability

The more information we have, the smarter we can test.

Test Characteristics

Test Characteristics

- Error finding
- Non-redundancy
- Test best
- Goldilocks principle

Error finding

A good test is one that has the highest probability of finding an error.

Non-redundancy

A good test is not redundant.

Test best

A good test should be “best of breed.”

Goldilocks principle

A good test should not be too simple. A good test should not be too complex. A good tesst should be just right.

Black box testing

Black box testing . . .

. . . test against the requirements. A tester exhaustively tests all the requirements, looking only to see if the software meets eachg requirement. Knowledge of the internals of the software is not know. In fact, black box testers cannot have any knowledge of the internals of the software.

Purpose of black box testing

Black box testing is not an alternative to white box testing, but a complement to it. Black box testing uncovers different classes of errors that white box testing.

Questions answered

Black box testing can ask the following kinds of questions:

- How is functional validity tested?
- How is system behavior and performance tested?
- What classes of input make good test cases?
- Is the system particularly sensitive to certain input values?
- How are the boundary valued isolated?
- What data amounts and data concurrency are tolerated?
- What effect will certain combinations of data have on the system?

Classes of errors

Black box testing can uncover the following types of errors:

- Incorrect or missing functions
- Interface errors, both external and internal
- Data errors, both in external database connectivity and internal data structures
- Behavior or performance defects
- Initialization and termination errors

Black box testing methods

Black box testing can uncover the following types of errors:

- graph based testing — How do the relationships between objects effect the software?
- transaction flow modeling — How do discrete transactions flow through the system?
- finite state modeling — What is the representational state of each of the nodes in the system?
- data flow modeling — Viewing the nodes as data objects and the edges as transformations, are the transformations valid?
- timing modeling — Given that transitions have "weight," do the sequential links have acceptable performance?

Equivalence partitioning

Divide the input domain into classes of data from which test cases can be derived

1. ranges of data — One valid and two invalid sequences are defined.
2. specific values — One valid and two invalid values are defined.
3. members of a set — One valid and two invalid subsets are defined.
4. Boolean values — One TRUE class and one FALSE class are defined.

Boundary value analysis

More software defects occur at the edges (extreme values) than at the center (typical values).

1. ranges of data — Test values at, just below, and just above, the acceptable range boundaries.
2. input values — Test the minimum an maximum number of values, and values greater and lesser than those allowed.
3. output values — Test values that output the minimum and maximum values, and values greater and lesser than those allowed.
4. data structures — Test minimum and maximum number of values, and values greater and lesser than those allowed.

White box testing

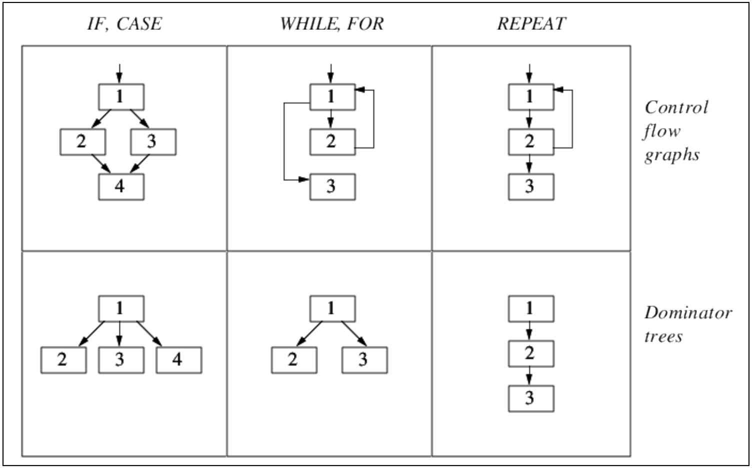
This brings us to the next topic: basis paths. In this case, the tester must have intimate knowledge of the internals of the software.

Basis Paths

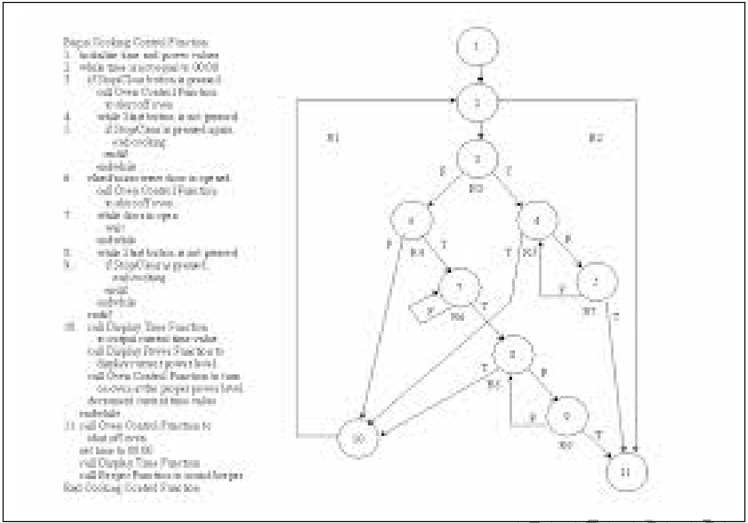
Control flow graphs

A control flow graph is a representation, using graph notation, of all paths that might be traversed through a program during its execution.

CFG types


IF, CASE
WHILE, FOR
REPEAT
Control flow graphs
Dominator trees
1
2
3
4

CFG example



```
Begin Cooking Control Function
1.  Initialize time and power values
2.  while time is not equal to 00:00
3.     if Stop/Clear button is pressed
          call Oven Control Function
             to shut off oven
4.        while Start button is not pressed
5.           if Stop/Clear is pressed again
                end cooking
             endif
          endwhile
6.     elseif microwave door is opened
          call Oven Control Function
             to shut off oven
7.        while door is open
             wait
          endwhile
8.        while Start button is not pressed
9.           if Stop/Clear is pressed
                end cooking
             endif
          endwhile
       endif
10.    call Display Time Function
          to output current time value
       call Display Power Function to
          display current power level
       call Oven Control Function to turn
          on oven at the proper power level
       decrement current time value
    endwhile
11. call Oven Control Function to
       shut off oven
    set time to 00:00
    call Display Time Function
    call Beeper Function to sound beeper
End Cooking Control Function
```

Independent program paths

An independent path through a program introduces at least one new edge.

Exhaustive testing

You have a program with 100 lines of source code that contains 2 nested loops, each executing 20 times. If you can test each case in one millisecond, how long will it take to exhaustively test the program?

Exhaustive testing

You have a program with 100 lines of source code that contains 2 nested loops, each executing 20 times. If you can test each case in one millisecond, how long will it take to exhaustively test the program? **3,170 years!**