Software Testing

Unit 4

Red: indicates important

Contents

- Black box testing
- White box testing
- Testing activities
- Levels of testing
- Debugging
- Testing tools

Software testing

- It's a process of operating a system or component under specified conditions, observing or recording the results, and making an evaluation of some aspect of the system or component.
- Its executing a program to assure:
  - Correctness of software with respect to requirements
  - Performance of software.
  - Robustness of software.
  - Usability of software under various condition.
  - Reliability, availability, survivability
  - Installability of software

- **Validation:** it's the process of evaluating a system or component during or at the end of development process to determine if it satisfies the requirements of the system or **"are we building the correct system"?**
- **Verification:** it's the process of evaluating a system or component at the end of a phase to determine if it satisfies the conditions imposed at the start of that phase or **"are we building the system correctly"?**

Black box testing

- A software testing method in which the functionality is tested without looking at the internal code structure.
- It focuses over the input taken and output to be produced.
- Test cases are derived from functional specification of the system.
- Input domain is SRS.
- Validates functional requirements of the system.
- Enables to find errors like incorrect or missing functions, performance errors, interface errors.

Advantages: Tester does not require any knowledge on the language, platform etc..

Test cases can be derived before design or coding of program.

Good at testing for missing functionalities.

Disadvantage: cannot identify additional functions or features that have been added to the code.

Black box testing strategies

1. Equivalence partitioning

- Aka equivalence classes partitioning.
- Inputs to the software or system are divided into groups that are expected to exhibit similar behavior. These groups are called equivalence classes.
- Can be applied at any level of testing.
- Equivalence classes can be found for both valid and invalid data.
- Aim is to minimize number of test cases.

Steps to choose equivalence classes

1. Identify the equivalence classes
2. Choose test cases

Identify the equivalence classes

If an input condition specifies range of values, then identify 1 valid equivalence class and 2 invalid equivalence class.

For example, if an input condition takes values between 0 and 10,

Then there are 3 equivalence classes:

Valid: value between 0-10

2 Invalid: value<0 and value >10

Choose test cases

We generate test case on the boundaries of partitions and close to the midpoint of the partition.

Triangle problem (equivalence classes)

- Problem: This program reads three integer values each representing the length of sides of a triangle. The program prints a message telling if it's a scalene, isosceles, equilateral, right-angled triangle. (each integer must be positive)

Solution: Identify valid and invalid data

Valid: x>o and y>o and z>o

Invalid : x<o and y>o and z>o

x>o and y<o and z>o

x>o and y>o and z<o

Equivalence classes	Test inputs	Expected outputs
ECscalene	f(3,5,7),…g	"Scalene"
ECequilateral	f(5,5,5),…g f(5,5,5),…g	"Equilateral"
ECright	f(3,4,5),..	"Right"
ECisosceles	f(2,3,3),…g f(2,3,3),…g	"Isosceles"
Ecno_triangle	f(1,1,3),…	"not a triangle"
ECinvalid1	f(-1,2,3),(0,1,3),…	"Error value"
ECinvalid2	F(1,-2,3),(1,-2,3),….	"Error value"
ECinvalid3	F(1,2,-3),(1,2,-3)	"Error value"

Boundary Value Analysis

- Can be applied at all levels of testing.
- Testing technique test cases are designed to check the boundaries of the input domain. By doing this, more probability of finding errors. We have both valid and invalid boundaries.
- Test case selection:
  1. If an input condition specifies a range of values, then construct valid test cases for the ends of the range, and invalid test cases for input just beyond the end of the range.
  2. If an input condition specifies a number of values, construct test cases for the minimum and maximum values; and one beneath and beyond these values.
  3. If an output condition specifies a range of values, then construct valid test cases for the ends of the output range, and invalid input test cases for inputs just beyond the ends of the output range.
  4. If an output condition specifies a number of values, construct test cases for the minimum and maximum values; and one beneath and beyond these values.
  5. If the input or output of a program is an ordered set, focus attention on the first and last element of the set.

BVA for Triangle Problem

- Problem: To classify a triangle. It takes three positive integers as input. Input will be greater than 0 and less than or equal to100. The three integers represent the lengths of the sides of a triangle. Output of this is a message whether the triangle is scalene, isosceles, equilateral or right angled.
- Given the sides A,B,C.
- Boundary conditions:
  - Scalene: sum of two sides must be greater than the third side.(A+B>C, B+C>A, A+C>B)
  - Isosceles: two sides of the triangle must be equal.(A=B, A=C,B=C)
  - Equilateral: All sides are equal. (A=B=C)
  - Right angled: $A^2+B^2=C^2$

Test Cases

Test Case	A	B	C	Expected Output
1	50	50	1	Isosceles
2	50	50	2	Isosceles
3	50	50	50	Equilateral
4	50	50	99	Isosceles
5	50	50	100	Not a triangle
6	50	1	50	Isosceles
7	99	50	100	Scalene
8	50	50	50	Equilateral
9	3	4	5	Right-angled
10	2	3	4	Not a triangle

White box testing

- Aka glass box testing
- White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.
- In this test cases are derived from the actual code or internal design specification.
- Advantage: it tests the internal details of the code and tries to check all the paths that a program can execute to determine if a problem occurs. It can test additional functions that has been implemented but not specified.
- Disadvantage: we must wait for the completion of the design and coding of the programs of functions under test in order to select test cases.
- Complete knowledge about the internal structure of the source code is required.

Features of white box testing

- Exercise all logical decisions on their true ad false sides.
- Execute all loops at their boundaries and within their operational bounds.
- Exercise internal data structures to assure their validity.

White box testing strategies

1. Coverage based testing

- Aim is to cover the program with test cases that satisfy some fixed coverage criteria.
- Common coverage criteria are:
  - Statement coverage or node coverage: every statement of the program should be exercised at least once.
  - Branch coverage or decision coverage: Every possible alternative in a branch or decision of the program should be exercised at least once.
  - Decision/Condition coverage: Each condition is made to evaluate to both true and false and each branch is made to evaluate to both true and false.
  - Multiple condition coverage: all possible combinations of condition outcomes within each branch should be exercised at least once.
  - Path coverage: Every execution path of the program should be exercised at least once.
- Example in text: page 59,60

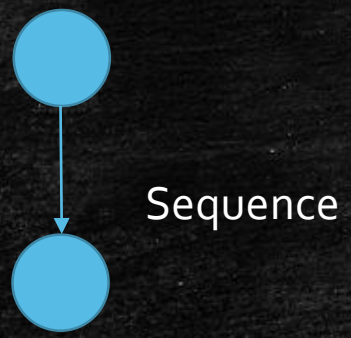
Cyclomatic complexity

- Software metric to measure the complexity of a program.
- Used to find the number of independent paths through a program.
- Can be found out from CFG of a program as V(G)=E-N+2 where N is the number of nodes, E is the number of edges and p is the number of predicate nodes(nodes those which split eg.. If)
- Properties:
  - V(G) is the maximum number of independent paths in graph G.
  - Inserting and deleting functional statements will not effect V(G).
  - G has only one path if V(G)=1.

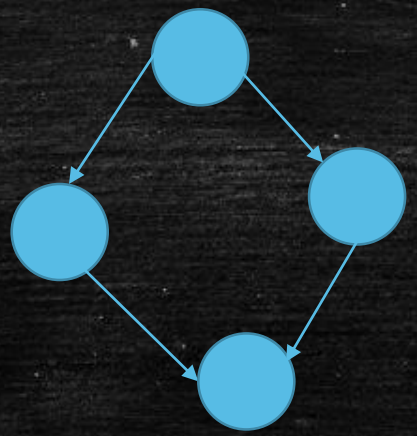
2. Control flow graph

- Graphical representation of a program.
- Describes the sequence in which instructions of a program get executed.
- Also describes the flow of control in a program.

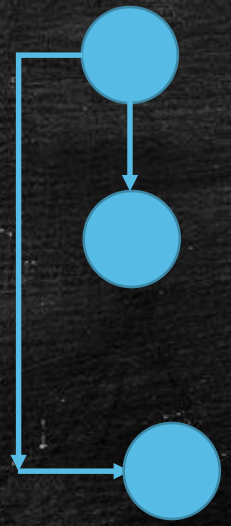
Different constructs in CFG:



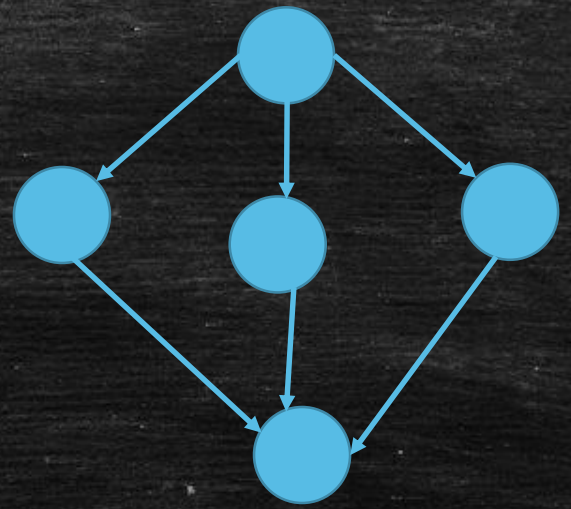
Sequence



If-else

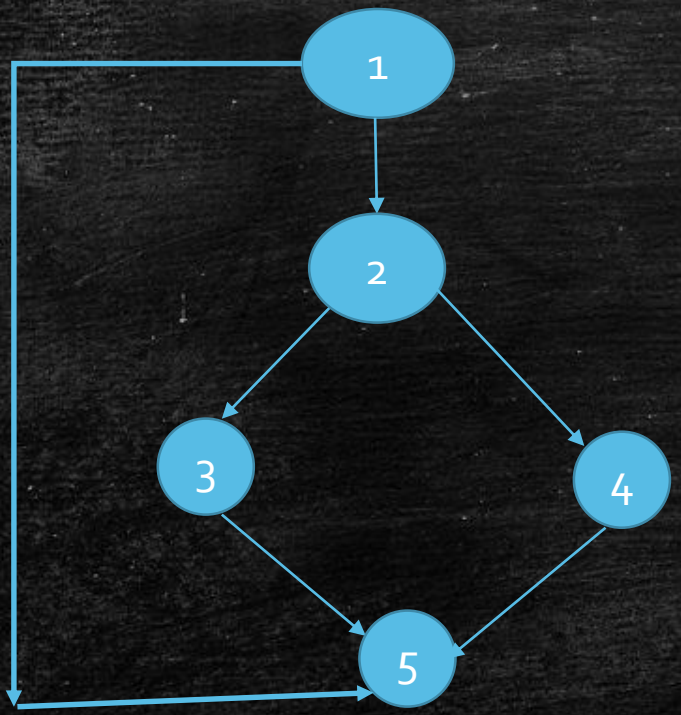


While loop



Switch

- Example:



V(G)=E-N+2
V(G)=6-5+2
V(G)=3

3. Mutation Testing

- Powerful method for finding errors in software program.
- Multiple copies of programs are made, each copy is altered and each altered copy is called a mutant.
- Mutants are executed with test data to determine whether the test data are capable of detecting the change.
- Mutant that is detected is called “killed”.
- Goal is to find a set of test cases that are able to kill groups of mutant programs.
- Mutants are produced by applying mutant operators(operator is a grammatical rule that changes a single expression into another expression)
- Its essential that all mutants are killed by the test cases or shown to be equivalent to the original expression.

- possibilities on executing a mutated program:

1. The results of the program were affected by the code change and the test suite detects it.
2. The results of the program are not changed and the test suite does not detect the mutation. Such mutants are known as "equivalent mutant".
3. The results of the program are different, but the test suite does not detect it because it does not have the right test case.

- Ratio of killed mutants to all the mutants is a number that is less than 1.

Example

```
Int a,b,c ;
if( a>b && a > c)
{
System.out.println (“ The greatest number is “ +a);
}
else if ( b>c)
{
System.out.println (“ The greatest number is” +b);
}
```

```
else
{
System.out.println(“ The greatest number is “ +c);
}
```

- List of mutants for the above program:

Mutant	Test Case	Input	Original output	Mutant output	Test result	Verdict
M1: if (c> a && c<b)	1	<1,2,3>	3	Program terminates	Pass	Killed
	2	<1,1,1>	1	Program terminates	Pass	Killed
M2: If(a==b && a>c)	1	<1,2,3>	3	3	Fail	Killable
	2	<1,1,1>	1	Program terminates	Pass	Killed
M3: if(a>b ||a>c)	1	<1,2,3>	3	3	Fail	Killable
	2	<1,1,1>	1	Program terminates	Pass	Killed
M4: : if(a>b&&&a>c) System.out.println("the greatest no is +c")	1	<1,2,3>	3	3	Fail	Killable
	2	<1,1,1>	1	Program terminates	Pass	Killed
M5: if(b>=a && b>c)	1	<1,2,3>	3	3	Fail	Killable
	2	<1,1,1>	1	Program terminates	Pass	Killed

- Three types of mutants:
  1. Killed or dead
  2. Live
  3. Equivalent
- Score associated with a test case:

Score=((no. of killed mutants)/(total mutants – equivalent mutants)) * 100

Score for the previous example:

Mutation Score= 100 * D/(N-E)

= 100 *5/(5-0)

=100

Testing Activities

- Testing:
  - Requirements analysis: Testing should begin in the requirements phase of SDLC
  - Design analysis: in the design phase, testers work with developers to identify the design aspects that are testable and under what parameters the testers should work on.
  - Test planning: Test plan and test strategy are prepared.
  - Test development: Test procedures, test scenarios, test cases are developed.
  - Test execution: testers execute the software based on the plans and tests and reports any errors to the development team.
  - Test Reporting: once testing is completed, testers finally prepare a report on test efforts and whether or not the software is ready for release or not.
  - Defects are retested to check whether they are resolved or not.

Levels of testing

- Software undergoes three levels of testing:
  1. Unit testing
  2. Integration testing
  3. System testing
- Unit testing:
  - Software testing method used to verify that a particular segment of code is working properly.
  - Tests cases are written for all functions or methods.
  - Done by developers.
  - Goal is to isolate each part of program and show that individual parts are correct.
  - Tests only the functionality of the units.
  - Provides a strict contract that the piece of code must satisfy.
  - Finds problems early in the development cycle.

- Integration testing:
  - Software testing method in which individual modules are combined and tested together.
  - Done after unit testing and before system testing.
  - Takes the modules that are checked by unit testing as input and outputs an integrated system that is ready for system testing.
  - Verifies functional, performance and reliability requirements.

- System testing:
  - Software testing method in which the system as a whole is tested against the SRS.
  - Input is: software components that has passed integration testing.
  - Includes both functional and non-functional testing.
  - Usually a black box testing.
  - Acceptance testing is done by customer to check whether the software meets all requirements or not. This may continue for a few weeks to several months.

Debugging

- It's the process of identifying the cause for defective behavior of a software and addressing that problem.
- 2 outcomes of debugging:
  1. The cause will be found and necessary action will be taken.
  2. The cause will not be found.
- Characteristics of bugs:
  1. Symptom and the cause may be geographically remote.
  2. Symptom may disappear when another error is corrected.
  3. Symptom may be caused by non-errors(round-off inaccuracies)
  4. Symptom may be caused by human error that is not easily traced.
  5. Symptom may be due to timing problems.
  6. It may be difficult to accurately produce input conditions.

7. Symptom may be intermittent.
8. Symptom may be due to causes that are distributed across a number of tasks running on different processors.

Life cycle of a debugging task:

- Steps involved in debugging:

1. Defect Identification/Confirmation

- Problem is identified in a system and defect report is created.
- Defect is assigned to a software engineer
- Engineer analyzes and performs the following actions:
  i. What is the expected behavior of the system?
  ii. What is the actual behavior?

iii. Is this really a defect in the system?

iv. Can the defect be reproduced?

2. Defect analysis

- In this step, the root cause of the problem is identified. This is the most challenging step in debugging. Debugging is done by a debugging tool.

3. Defect Resolution

- Once the root cause is identified, the defect is resolved by making the appropriate change.

Debugging approaches

- Three categories of debugging approaches:

1. Brute force: most popular but least successful approach. This method is used when all else fails. All memory dumps are taken, run-time traces are invoked, and the program is loaded with WRITE statements.

2. Backtracking: common debugging method and successful in small programs. This process starts from the site where a particular symptom gets detected, from there on backward tracing is done across the entire source code till the error is found.

3. Cause elimination: list of all possible causes is developed and tests are conducted to eliminate each.

Testing tools

- Different categories of tools used for testing:
- Data Acquisition: Tools that acquire data to be used for testing.
- Static Measurement: tools that analyze source code without executing test cases.
- Dynamic Measurement: Tools that execute source code during execution.
- Simulation: Tools that simulate functions of hardware or other externals.
- Test management: Tools that assist in planning, development and control of testing.
- Cross-functional tools: Tools that cross the bounds of preceding categories.
- Commercial software testing tools: AQtest, csUnit, Sahi, Rational Test Real Time Unit Testing

Testing tools

- Commercial software testing tools: AQtest, csUnit, Sahi, Rational Test Real Time Unit Testing

Rational Test Real Time Unit Testing

- Kind of Tool: It automates C,C++ software component testing.
- Organization: IBM Rational Software
- Description: It performs black-box/functional testing. Its integrated with native development environments (Unix and Windows) as well as a large variety of cross-development environments.
- Platforms: Its available for most development and target systems including Windows and Unix.

AQtest

- Kind of Tool: Automated support for functional, unit and regression testing.
- Organization: AutomatedQA Corp

- Description: It automates and manages functional tests, unit tests and regression tests for applications written with VC++, VB, Delphi, C++Builder, Java or VS.NET. It also supports white box testing, down to private properties or methods. External tests can be recorded in three scripting languages (VBScript, Jscript, DelphiScript). Aqtest as an OLE server, unit-test drivers can also run it directly from application code. Aqtest is easily extended through plug-ins using the complete IDL libraries supplied. Plugins currently support Win32 API calls, direct ADO access, direct BDE access, etc..
- Platform: Windows 95, 98, NT or 2000

csUnit

- Kind of Tool: "Complete Solution Unit Testing" for Microsoft .NET
- Organization: csUnit.org
- Description: It's a unit testing framework for Microsoft .NET. It focuses on test driven development using .NET languages like C#, Visual Basic .NET
- Platform: Microsoft Windows

Sahi

- Description: It's an automation and testing tool for web applications. It offers the facility to record and playback scripts. Developed in Java and JavaScript, it uses simple JavaScript to execute events on the browser. It supports HTTP and HTTPS. It runs as a proxy server and the browser needs to use Sahi server as its proxy. It then injects JavaScript so that it can access elements in the webpage. This feature makes Sahi independent of the website/ web application.
- Platforms: OS independent, needs atleast JDK 1.4

Difference between black box and white box testing

Black box testing	White box testing
It focuses only over the input taken and the output produced.	It focuses over the internal development of software.
Does not require indepth knowledge in language and technologies.	Requires indepth knowledge in language and technologies.
Input domain is SRS.	Input domain is design and externally accessible program variables.
Validates functional requirements.	Validates internal structure and working of software code.
This type of testing is carried out by testers.	this type of testing is carried out by software developers.
Testing is applicable on higher levels of testing like System Testing, Acceptance testing.	Testing is applicable on lower level of testing like Unit Testing, Integration testing.
Black box testing is the software testing method which is used to test the software without knowing the internal structure of code or program.	White box testing is the software testing method in which internal structure is being known to tester who is going to test the software.