Static testing

- Static testing is a software testing method that involves examination of the program's code and its associated documentation but does not require the program be executed. Dynamic testing, the other main category of software testing methods, involves interaction with the program while it runs. The two methods are frequently used together to try to ensure the functionality of a program.

- It is primarily checking of the code and/or manually reviewing the code or document to find errors
- This type of testing can be used by the developer who wrote the code,
- . Code reviews, inspections and Software walkthroughs are also used.

- 1. **Code review** is systematic examination (often known as peer review) of computer source code. It is intended to find and fixmistakes overlooked in the initial development phase, improving both the overall quality of software and the developers' skills.
- 2. An inspection is one of the most common sorts of review practices found in software projects. The goal of the inspection is for all of the inspectors to reach consensus on a work product and approve it for use in the project.[inspection is a formal testing where the application is checked against requirements]
- 3. **walk-through** is a form of software peer review "in which a designer or programmer leads members of the development team and other interested parties through a software product, and the participants ask questions and make comments about possible errors, violation of development standards, and other problems

- static testing involves reviewing requirements and specifications. This is done with an eye toward completeness or appropriateness for the task at hand. This is the **verification** portion of Verification and Validation.
- Even static testing can be automated. A static testing test suite consists of programs to be analyzed by an interpreter or a compiler that asserts the programs syntactic validity.
- Bugs discovered at this stage of development are less expensive to fix than later in the development cycle
- The people involved in static testing are application developers and testers.

- **Which Test finds which Error?**
- **Possible errorCan be best found byExample**Syntax errors
- **Compiler, Lint**Missing semicolons, Values defined but not initalized or used, order of evaluation disregarded.Data errors
- **Software inspection**, module testsOverflow of variables at calculation, usage of inappropriate data types, values not initialized, values loaded with wrong data or loaded at a wrong point in time, lifetime of pointers. Algorithm and logical errors
- **Software inspection**, module testsWrong program flow, use of wrong formulas and calculations.Interface errors
- **Software inspection**, module tests, component tests. Overlapping ranges, range violation (min. and max. values not observed or limited), unexpected inputs, wrong sequence of input parameters.Operating system errors, architecture and design errors
- **Design inspection**, integration tests Disturbances by OS interruptions or hardware interrupts, timing problems, lifetime and duration problems. Integration errors
- **Integration tests**, system tests Resource problems (runtime, stack, registers, memory, etc.)System errors
- **System tests Wrong** system behavior, specification errors

Functional testing

- **Functional testing** is a quality assurance (QA) process[1] and a type of black box testing that bases its test cases on the specifications of the software component under test. Functions are tested by feeding them input and examining the output, and internal program structure is rarely considered (not like in white-box testing).[2] Functional testing usually describes *what*the system does.

- Functional testing differs from system testing in that functional testing "*verifies* a program by checking it against ... design document(s) or specification(s)", while system testing "*validate[s]* a program by checking it against the published user or system requirements"

- **Functional testing typically involves six steps**[*citation needed*]
- The identification of functions that the software is expected to perform
- The creation of input data based on the function's specifications
- The determination of output based on the function's specifications
- The execution of the test case
- The comparison of actual and expected outputs
- To check whether the application works as per the customer need.

Regression testing

- **Regression testing** is a type of software testing that seeks to uncover new software bugs, or *regressions*, in existingfunctional and non-functional areas of a system after changes such as enhancements, patches or configuration changes, have been made to them.
- The intent of regression testing is to ensure that changes such as those mentioned above have not introduced new faults.[1]One of the main reasons for regression testing is to determine whether a change in one part of the software affects other parts of the software.[2]
- Common methods of regression testing include rerunning previously completed tests and checking whether program behavior has changed and whether previously fixed faults have re-emerged. Regression testing can be performed to test a system efficiently by systematically selecting the appropriate minimum set of tests needed to adequately cover a particular change.
- Contrast with non-regression testing (usually validation-test for a new issue), which aims to verify whether, after introducing or updating a given software application, the change has had the intended effect.

Volume testing in software testing?

- It is a type of non-functional testing.
- Volume testing refers to testing a software application or the product with a certain amount of data. E.g., if we want to volume test our application with a specific database size, we need to expand our database to that size and then test the application's performance on it.
- The purpose of **volume testing** is to determine system performance with increasing volumes of data in the database.

Stress testing in software testing?

- It is a type of non-functional testing.
- It involves testing beyond normal operational capacity, often to a breaking point, in order to observe the results.
- It is a form of software testing that is used to determine the stability of a given system.
- It put greater emphasis on robustness, availability, and error handling under a heavy load, rather than on what would be considered correct behavior under normal circumstances.
- The goals of such tests may be to ensure the software does not crash in conditions of insufficient computational resources (such as memory or disk space).

What is Alpha testing?

- Alpha testing is one of the most common software testing strategy used in software development. Its specially used by product development organizations.
- This **test takes place at the developer's site**. Developers observe the users and note problems.
- Alpha testing is testing of an application when development is about to complete. Minor design changes can still be made as a result of alpha testing.

- wo phases:In the **first phase** of alpha testing, the software is tested by in-house developers. They use either debugger software, or hardware-assisted debuggers. The goal is to catch bugs quickly.
- In the **second phase** of alpha testing, the software is handed over to the software

What is Beta testing?

- It is also known as field testing. It takes place at **customer's site**. It sends the system to users who install it and use it under real-world working conditions.
- A beta test is the second phase of software testing in which a sampling of the intended audience tries the product out. (Beta is the second letter of the Greek alphabet.) Originally, the term *alpha test* meant the first phase of testing in a software development process. The first phase includes unit testing, component testing, and system testing. Beta testing can be considered "pre-release testing.
- The goal of beta testing is to place your application in the hands of real users outside of your own engineering team to discover any flaws or issues from the user's perspective that you would not want to have in your final, released version of the application.

Test plan

- A **test plan** is a document detailing a systematic approach to testing a system such as a machine or software. The plan typically contains a detailed understanding of the eventual workflow
- A test plan documents the strategy that will be used to verify and ensure that a product or system meets its design specifications and other requirements

- Depending on the product and the responsibility of the organization to which the test plan applies, a test plan may include a strategy for one or more of the following:
- *Design Verification or Compliance test* - to be performed during the development or approval stages of the product, typically on a small sample of units.
- *Manufacturing or Production test* - to be performed during preparation or assembly of the product in an ongoing manner for purposes of performance verification and quality control.
- *Acceptance or Commissioning test* - to be performed at the time of delivery or installation of the product.
- *Service and Repair test* - to be performed as required over the service life of the product.
- *Regression test* - to be performed on an existing operational product, to verify that existing functionality didn't get broken when other aspects of the environment are changed
- Test plan document formats can be as varied as the products and organizations to which they apply

Test Specification

- **Test Specification** – It is a detailed summary of what scenarios will be tested, how they will be tested, how often they will be tested, and so on and so forth, for a given feature. Trying to include all Editor Features or all Window Management Features into one Test Specification would make it too large to effectively read.
- However, a Test Plan is a collection of all test specifications for a given area. The Test Plan contains a high-level overview of what is tested for the given feature area.

- **Test Specification Items**Each test specification should contain the following items:

  Case No.: The test case number should be a three digit identifer of the following form: c.s.t, where: c- is the chapter number, s- is the section number, and t- is the test case number.

  Title: is the title of the test.

  ProgName: is the program name containing the test.

  Author: is the person who wrote the test specification.

  Date: is the date of the last revision to the test case.
  Background: (Objectives, Assumptions, References, Success Criteria): Describes in words how to conduct the test.

  Expected Error(s): Describes any errors expected

  Reference(s): Lists reference documententation used to design the specification.

  Data: (Tx Data, Predicted Rx Data): Describes the data flows between the Implementation Under Test (IUT) and the test engine.

  Script: (Pseudo Code for Coding Tests): Pseudo code (or real code) used to conduct the test.

- **Example Test SpecificationCase No.** 7.6.3 **Title:** Invalid Sequence Number (TC)
  ProgName: UTEP221 **Author:** B.C.G. **Date:** 07/06/2000
  Background: (Objectives, Assumptions, References, Success Criteria)
  Validate that the IUT will reject a normal flow PIU with a transmissionheader that has an invalid sequence number.
  Expected Sense Code: $2001, Sequence Number Error
  Reference - SNA Format and Protocols Appendix G/p. 380
  Data: (Tx Data, Predicted Rx Data)
  IUT
  <-------- DATA FIS, OIC, DR1 SNF=20
  <-------- DATA LIS, SNF=20
  --------> -RSP $2001
  Script: (Pseudo Code for Coding Tests)
  SEND_PIU FIS, OIC, DR1, DRI SNF=20
  SEND_PIU LIS, SNF=20
  R_RSP $2001
-

- **Analysis and Design:**
- **Test analysis and Test Design** has the following major tasks:
  i. To review the **test basis.** (The test basis is the information we need in order to start the test analysis and create our own test cases. Basically it's a documentation on which test cases are based, such as requirements, design specifications, product risk analysis, architecture and interfaces. We can use the test basis documents to understand what the system should do once built.)
  ii. To identify test conditions.
  iii. To design the tests.
  iv. To evaluate testability of the requirements and system.
  v. To design the test environment set-up and identify and required infrastructure and tools.

- **Test execution** has the following major task:
  i. To execute test suites and individual test cases following the test procedures.
  ii. To re-execute the tests that previously failed in order to confirm a fix. This is known as **confirmation testing or re-testing.**
  iii. To log the outcome of the test execution and record the identities and versions of the software under tests. The **test log** is used for the audit trial. (A test log is nothing but, what are the test cases that we executed, in what order we executed, who executed that test cases and what is the status of the test case (pass/fail). These descriptions are documented and called as test log.).
  iv. To Compare actual results with expected results.
  v. Where there are differences between actual and expected results, it report discrepancies as Incidents

Defect logging and tracking

- Defect logging, a process of finding defects in the application under test or product by testing or recording feedback from customers and making new versions of the product that fix the defects or the clients feedback.
- Defect tracking is an important process in software engineering as Complex and business critical systems have hundreds of defects. One of the challenging factors is Managing, evaluating and prioritizing these defects. The number of defects gets multiplied over a period of time and to effectively manage them, defect tracking system is used to make the job easier.

Equivalence partitioning in Software testing?

- Equivalence partitioning (EP) is a specification-based or black-box technique.
- It can be applied at any level of testing and is often a good technique to use first.
- The idea behind this technique is to divide (i.e. to partition) a set of test conditions into groups or sets that can be considered the same (i.e. the system should handle them equivalently), hence 'equivalence partitioning'. **Equivalence partitions** are also known as equivalence classes – the two terms mean exactly the same thing.
- In equivalence-partitioning technique we need to test only one condition from each partition. This is because we are assuming that all the conditions in one partition will be treated in the same way by the software. If one condition in a partition works, we assume all of the conditions in that partition will work, and so there is little point in testing any of these others. Similarly, if one of the conditions in a partition does not work, then we assume that none of the conditions in that partition will work so again there is little point in testing any more in that partition.

Boundary value analysis

- **Boundary value analysis** is a software testing technique in which tests are designed to include representatives of boundary values. The idea comes from the boundary. Given that we have a set of test vectors to test the system, a topology can be defined on that set. Those inputs which belong to the same equivalence class as defined by the equivalence partitioning theory would constitute thebasis. Given that the basis sets are neighbors, there would exist a boundary between them. The test vectors on either side of the boundary are called boundary values. In practice this would require that the test vectors can be ordered, and that the individual parameters follows some kind of order

- **Example:2**
- A store in city offers different discounts depending on the purchases made by the individual. In order to test the software that calculates the discounts, we can identify the ranges of purchase values that earn the different discounts. For example, if a purchase is in the range of $1 up to $50 has no discounts, a purchase over $50 and up to $200 has a 5% discount, and purchases of $201 and up to $500 have a 10% discounts, and purchases of $501 and above have a 15% discounts.
- We can identify 4 valid equivalence partitions and 1 invalid partition as shown below.
- Invalid PartitionValid Partition(No Discounts)Valid Partition(5%)Valid Partition(10%)Valid Partition(15%)$0.01$1-$50$51-$200$201-$500$501-AboveFrom this table we can identify the boundary values of each partition. We assume that two decimal digits are allowed.
- Boundary values for Invalid partition: 0.00
  Boundary values for valid partition(No Discounts): 1, 50
  Boundary values for valid partition(5% Discount): 51, 200
  Boundary values for valid partition(10% Discount): 201,500
  Boundary values for valid partition(15% Discount): 501, Max allowed number in the software application

Robustness testing

- **Robustness testing** is any quality assurance methodology focused on testing the robustness of software. Robustness testing has also been used to describe the process of verifying the robustness (i.e. correctness) of test cases in a test process.

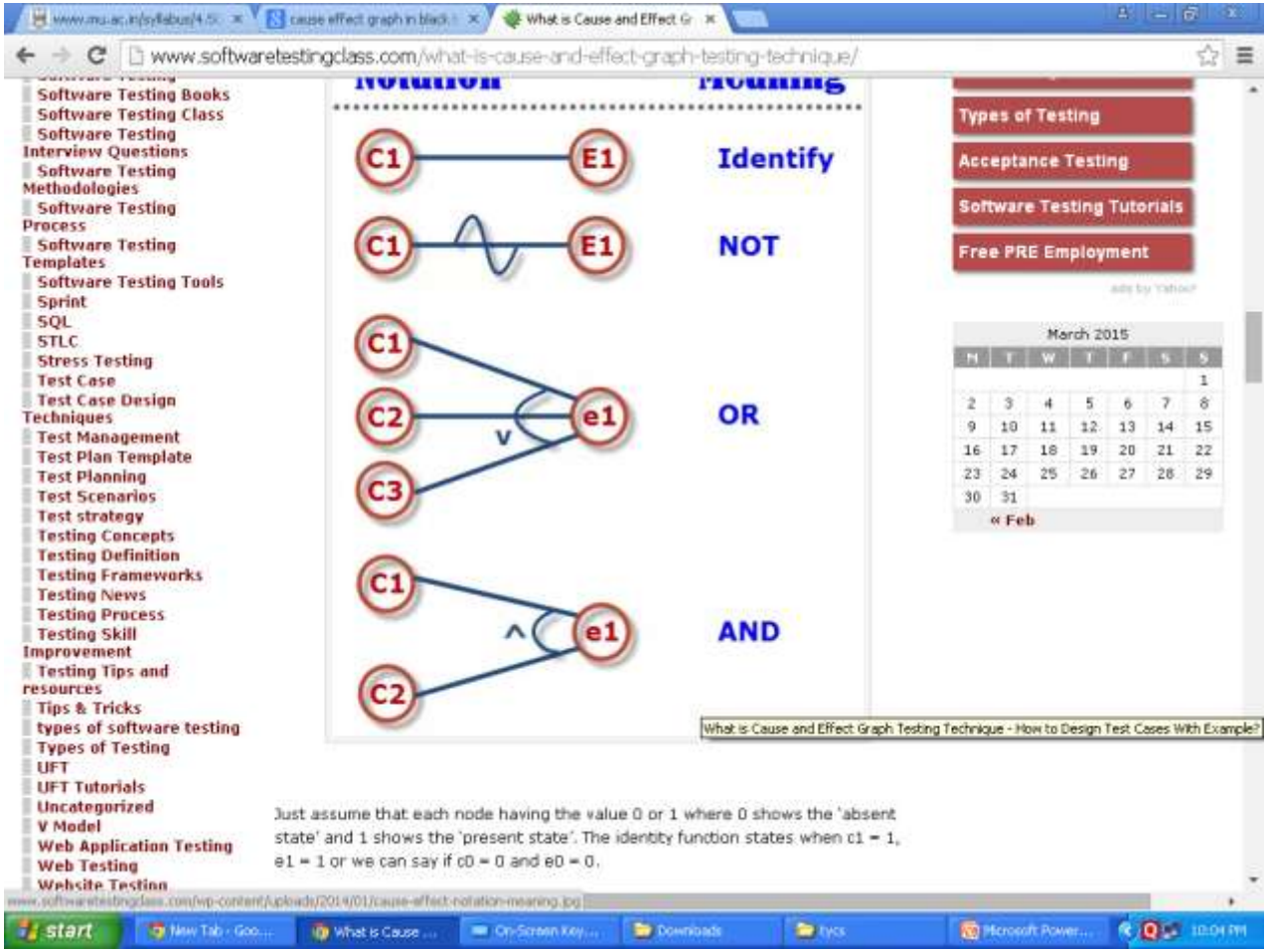
Cause-Effect Graph

- Cause-Effect Graph graphically shows the connection between a given outcome and all issues that manipulate the outcome. Cause Effect Graph is a black box testing technique -

- The Cause-Effect Diagram can be used under these Circumstances:
- To determine the current problem so that right decision can be taken very fast.
- To narrate the connections of the system with the factors affecting a particular process or effect.
- To recognize the probable root causes, the cause for a exact effect, problem, or outcome.

- Benefits of making cause-
- Effect Diagram It finds out the areas where data is collected for additional study.
- It motivates team contribution and uses the team data of the process.
- Uses synchronize and easy to read format to diagram cause-and-effect relationships.
- Point out probable reasons of difference in a process. It enhances facts of the procedure by helping everyone to learn more about the factors at work and how they relate.
- It assists us to decide the root reasons of a problem or quality using a structured approach.

- **Steps to proceed on Cause-Effect Diagram:**
- **Firstly:** Recognize and describe the input conditions (causes) and actions (effect)
- **Secondly:** Build up a cause-effect graph
- **Third:** Convert cause-effect graph into a decision table
- **Fourth:** Convert decision table rules to test cases. Each column of the decision table represents a test case
-
- - See more at: http://www.softwaretestingclass.com/what-is-cause-and-effect-graph-testing-technique/#sthash.S904W31h.dpuf


www.softwaretestingclass.com/what-is-cause-and-effect-graph-testing-technique/
Notation
Meaning
C1
E1
Identify
C1
E1
NOT
C1
C2
C3
v
e1
OR
C1
C2
^
e1
AND
Types of Testing
Acceptance Testing
Software Testing Tutorials
Free PRE Employment
March 2015
« Feb
What is Cause and Effect Graph Testing Technique - How to Design Test Cases With Example?
Just assume that each node having the value 0 or 1 where 0 shows the 'absent state' and 1 shows the 'present state'. The identity function states when c1 = 1, e1 = 1 or we can say if c0 = 0 and e0 = 0.

What is Statement coverage?

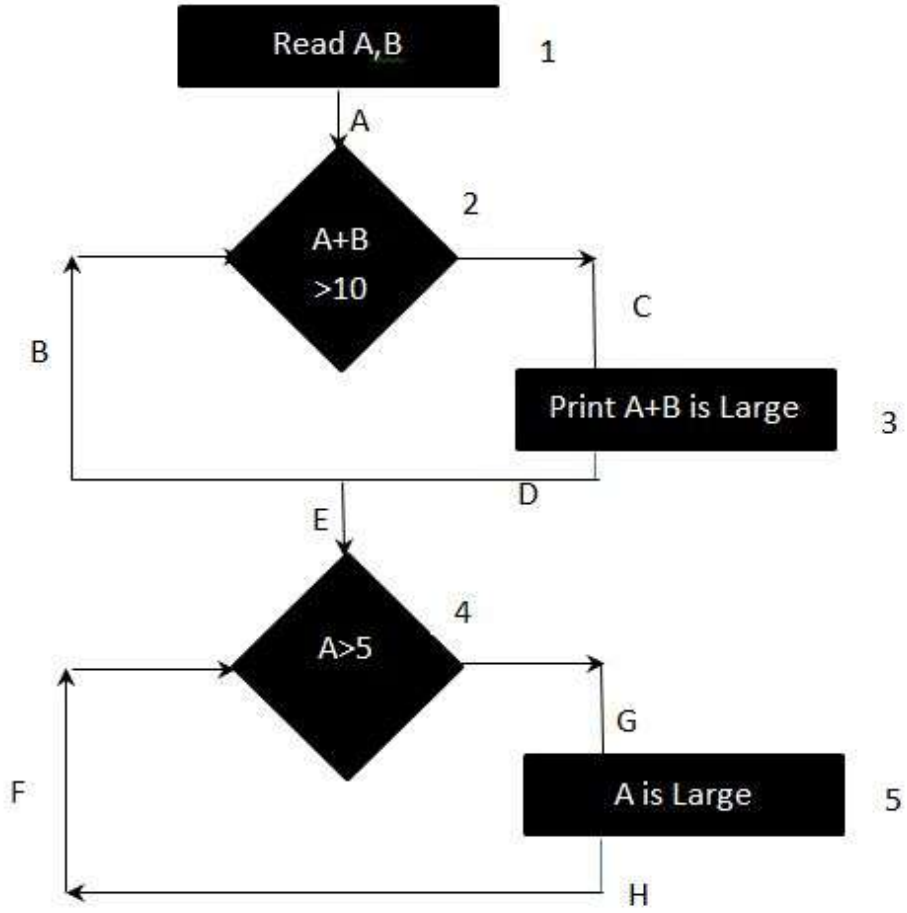
- The statement coverage is also known as line coverage or segment coverage.
- The statement coverage **covers only the true conditions.**
- Through statement coverage we can identify the statements executed and where the code is not executed because of blockage.
- In this process each and every line of code needs to be checked and executed

- **Advantage of statement coverage:**
- It verifies what the written code is expected to do and not to do
- It measures the quality of code written
- It checks the flow of different paths in the program and it also ensure that whether those path are tested or not.

Decision Coverage Testing?

- What is Decision Coverage Testing?
- Decision coverage or Branch coverage is a testing method, which aims to ensure that each one of the possible branch from each decision point is executed at least once and thereby ensuring that all reachable code is executed.
- That is, every decision is taken each way, true and false. It helps in validating all the branches in the code making sure that no branch leads to abnormal behavior of the application.

- Example:
- Read A Read B IF A+B > 10 THEN Print "A+B is Large" ENDIF If A > 5 THEN Print "A Large" ENDIFThe above logic can be represented by a flowchart as:
- Result :
- To calculate Branch Coverage, one has to find out the minimum number of paths which will ensure that all the edges are covered. In this case there is no single path which will ensure coverage of all the edges at once. The aim is to cover all possible true/false decisions. (1) 1A-2C-3D-E-4G-5H (2) 1A-2B-E-4F


Read A,B
1
A
A+B
>10
2
B
C
Print A+B is Large
3
D
E
A>5
4
F
G
A is Large
5
H

Graph Matrices

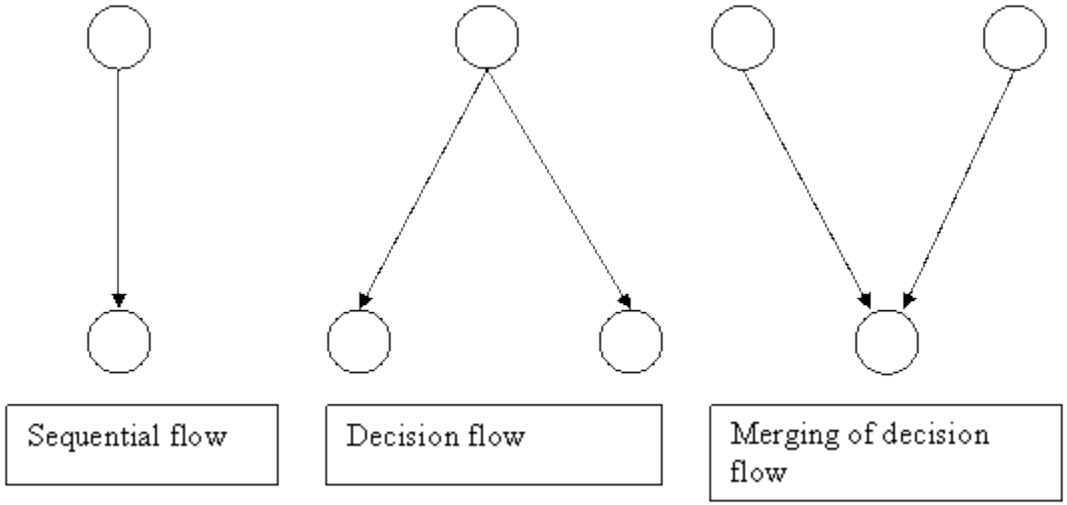
- **4.**
  The procedure for deriving the flow graph and even determining a set of basis paths is amenable to mechanization. To develop a software tool that assists in basis path testing, a data structure, called a graph matrix can be quite useful.
  A Graph Matrix is a square matrix whose size is equal 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

Cyclomatic complexity

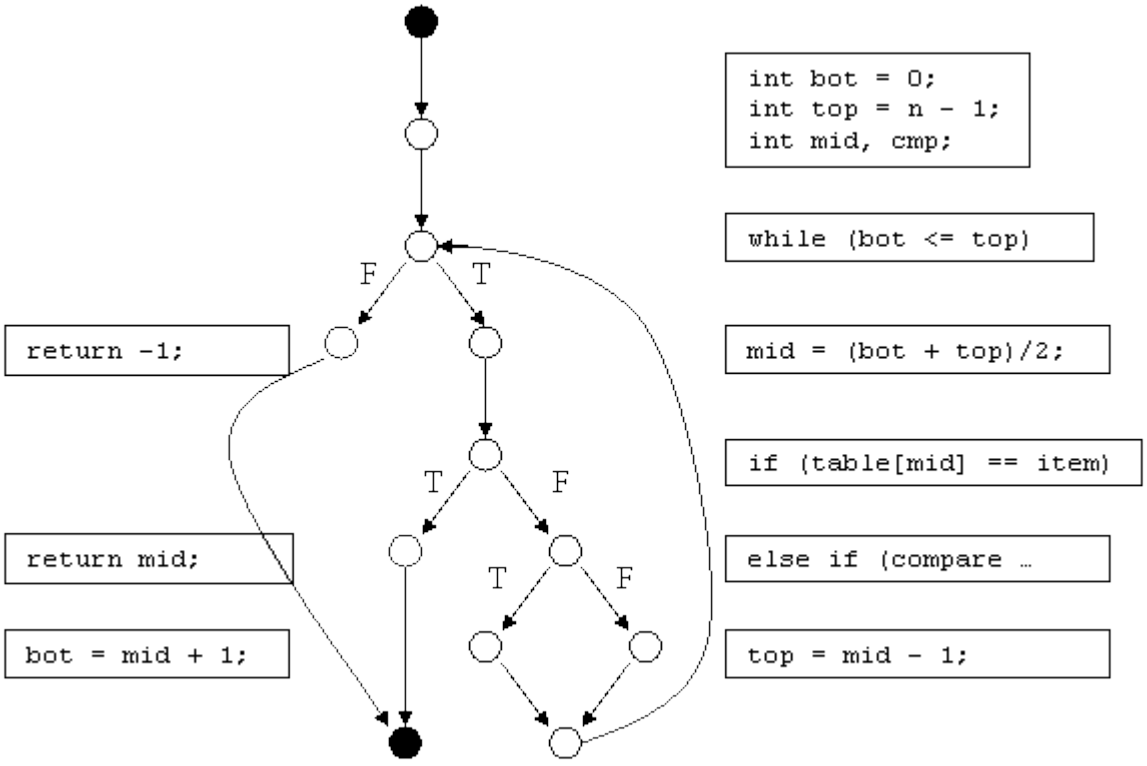
- **McCabe's Cyclomatic complexity**
-
- Cyclomatic complexity introduced by thomas McCabe in 1976. It simply measures the amount of decision logic in the program module.Cyclomatic complexity gives the minimum number of paths that can generate all possible paths through the module.Cyclomatic complexity is often referred to as McCabe's complexity.McCabe's complexity used to define minimum number of test cases required for a module and it is used during software development lifecycle to quantify maintainability and testability.Cyclomatic complexity is defined as
-

- CC = E - N + P
-
- Where
-
- E = the number of edges of the graph
-
- N = the number of nodes of the graph
-
- P = the number of connected components
-
-

- In case of connected graph
-
- CC = E - N + 2
-
-
- In simplified way it can be defined as
-
- CC = D +1
-
- Where
-
- D = the number of decision points in the graph
-


Sequential flow
Decision flow
Merging of decision flow

- control flow graph or CFG is a graphical representation of program module. Control flow graph is a directed graph in which node represent program's region and edges represent flow from one program region to other.


int bot = 0;
int top = n - 1;
int mid, cmp;
while (bot <= top)
F
T
return -1;
mid = (bot + top)/2;
T
F
if (table[mid] == item)
return mid;
T
F
else if (compare …
bot = mid + 1;
top = mid - 1;

- In this example
-
- CC = number of edges – number of nodes + 2
-
- CC = 14 – 12 + 2
-
- = 4
-
- Now CC from simplified formula
-
- CC = number of decision point + 1
-
- = 3 + 1
-
- = 4

Mutation testing

- Mutation testing is a kind of testing in which, the application is tested for the code that was modified after fixing a particular bug/defect. It also helps in finding out which code and which strategy of coding can help in developing the functionality effectively. You can refer the Mutation testing Pdf training tutorials after reading this basic introduction.

Mutation testing

- Mutation testing is used to test the quality of your test suite. This is done by mutating certain statements in your source code and checking if your test code is able to find the errors. However, mutation testing is very expensive to run, especially on very large applications. There is a mutation testing tool, Jester, which can be used to run mutation tests on Java code. Jester looks at specific areas of your source code, for example: forcing a path through an if statement, changing constant values, and changing Boolean values