SOFTWARE TESTING

"Testing can only show the presence of errors, not their absence" …

Dijkstra et al .

What is Software Testing?

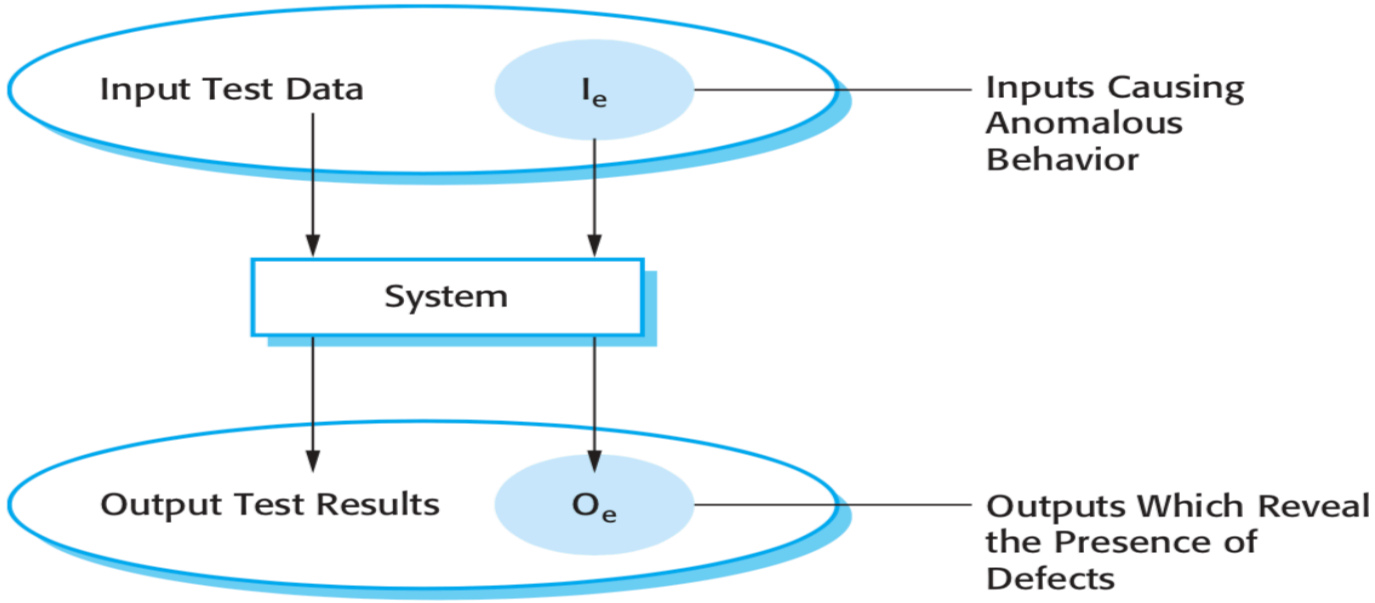
- **Testing** is intended to show that a program does what it is intended to do and to discover program defects before it is put into use.

- The testing process has two distinct **goals**:
  1. Demonstrate to the developer and the customer that the software meets its requirements.

  2. To discover situations in which the behavior of the software is incorrect, undesirable. → (Defect Testing). In order words, to ensure that the software system is Defect free by removing the undesirable faults.

What is Software Testing?

- **Defect testing**
  - Priority in defect testing is to find those inputs in the set I_e because these reveal problems with the system.



Basic Definitions

- **Failure**
  - There is a deviation of the observed behavior of a program or a system from its specification.
- **Fault/defect**
  - An incorrect step, process or data definition.

Example: for any integer n, square (n) = n*n.

Implementation	Test cases	

int square (int x){
return x * 2;
}
Fault
	Square (2) = 4	**Correct**

	Square (3) = 6	**Failure**

What is main focus on Software Testing?

- **Correctness** of software with respect to requirements or intent
- **Performance** of software under various conditions
- **Robustness** of software, its ability to handle erroneous input and unanticipated conditions
- **Installation** and other facets of a software release

What is Software Testing?

here

- In simple terms, Software Testing means **Verification of Application Under Test (AUT).** It can be either done manually or using automated tools.

Why is Software Testing Important?

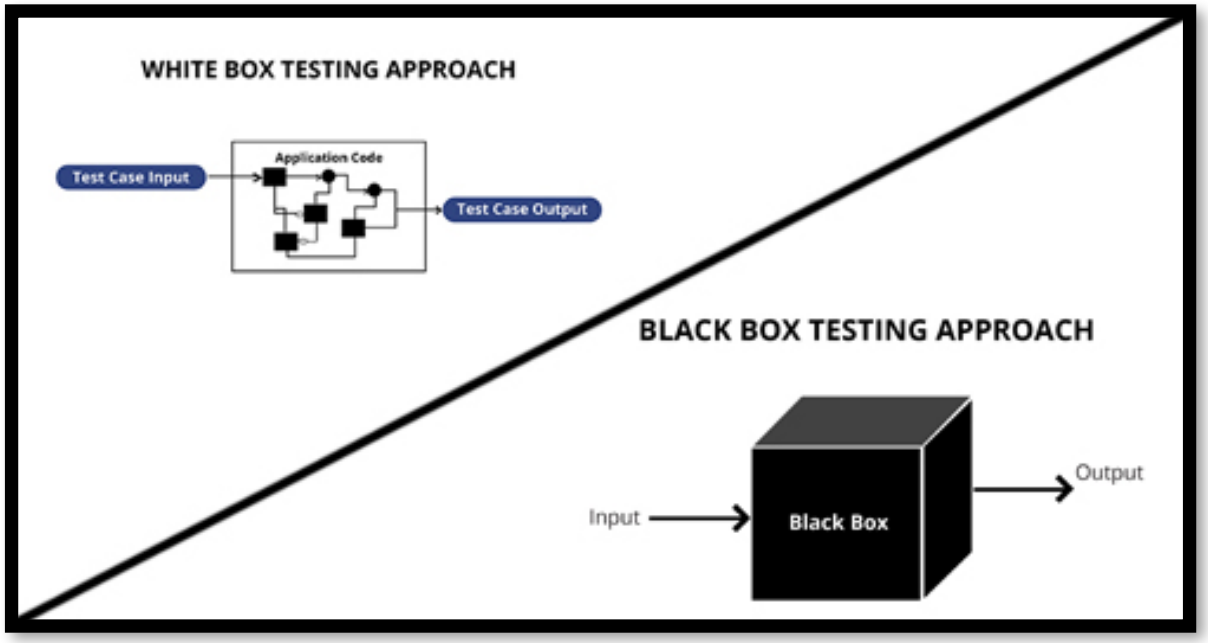
- Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss, and history is full of such examples.
- Software testing helps to identify errors, gaps or missing requirements in contrary to the actual requirements.

Verification and Validation (V&V)

- Testing is part of a broader process of software **verification** and **validation** (V & V).
- Verification
  - The aim of is to check that the software meets its stated functional and non-functional requirements.
  - *Are we building the product right?*
- Validation
  - Is a more general process, with the aim of ensuring that the software meets the customer's expectations.
  - *Are we building the right product ?*

Types of Testing

- Black-box Testing
- White-Box Testing



BLACK BOX TESTING

- **BLACK BOX TESTING** is defined as a testing technique in which functionality/features of the Application Under Test (AUT) is tested without looking at the **internal code structure**, **implementation details** and **knowledge of internal paths** of the software.
- This type of testing is based entirely on **software requirements and specifications.**
- In BlackBox Testing we just focus on inputs and output of the software system without bothering about internal knowledge of the software program.

How is Black Box Testing done?

Here are the generic steps followed to carry out any type of Black Box Testing.

- Initially, the **requirements and specifications** of the system are examined.
- Tester **chooses valid inputs** (positive test scenario) to check whether SUT processes them correctly. Also, some **invalid inputs** (negative test scenario) are chosen to verify that the SUT is able to detect them.
- Tester determines **expected outputs** for all those inputs.
- Software tester **constructs test cases** with the selected inputs.
- The test cases are **executed.**
- Software tester **compares** the **actual outputs** with the **expected outputs**.
- **Defects** if any are **fixed** and **re-tested**

Types of Black Box Testing

There are many types of Black Box Testing but the following are the prominent ones -

- **Functional testing** - This black box testing type is related to the functional requirements of a system; it is done by software testers.
- **Non-functional testing** - This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.
- **Regression testing** - Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

Black-Box Testing

- **Advantages**
  - Test case selection is done before the implementation of a program.
  - Help in getting the design and coding correct with respect to the specification.

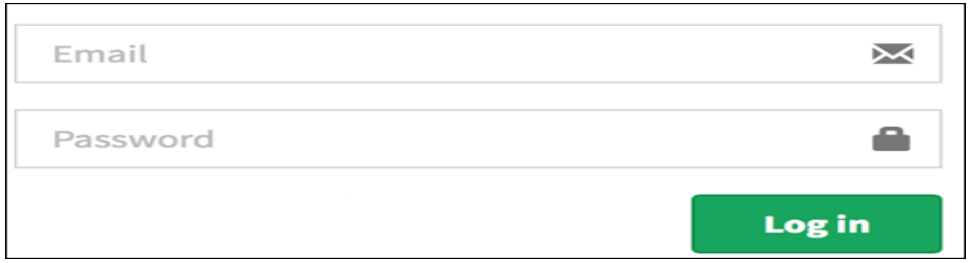
Black-Box Testing Techniques

Following are the prominent Test Strategy amongst the many used in Black box Testing

- **Decision Table Testing**: A decision table puts causes and their effects in a matrix. There is a unique combination in each column.
- **Equivalence Class Testing:** It is used to minimize the number of possible test cases to an optimum level while maintains reasonable test coverage.
- **Boundary Value Testing:** Boundary value testing is focused on the values at boundaries. This technique determines whether a certain range of values are acceptable by the system or not. It is very useful in reducing the number of test cases. It is most suitable for the systems where an input is within certain ranges.

Example 1: How to make Decision Base Table for Login Screen

The condition is simple if the user provides correct username and password the user will be redirected to the homepage. If any of the input is wrong, an error message will be displayed.



- Case 1 – Username and password both were wrong. The user is shown an error message.
- Case 2 – Username was correct, but the password was wrong. The user is shown an error message.
- Case 3 – Username was wrong, but the password was correct. The user is shown an error message.
- Case 4 – Username and password both were correct, and the user navigated to homepage

Conditions	Rule 1	Rule 2	Rule 3	Rule 4
Username (T/F)	F	T	F	T
Password (T/F)	F	F	T	T
Output (E/H)	E	E	E	H

Legend:

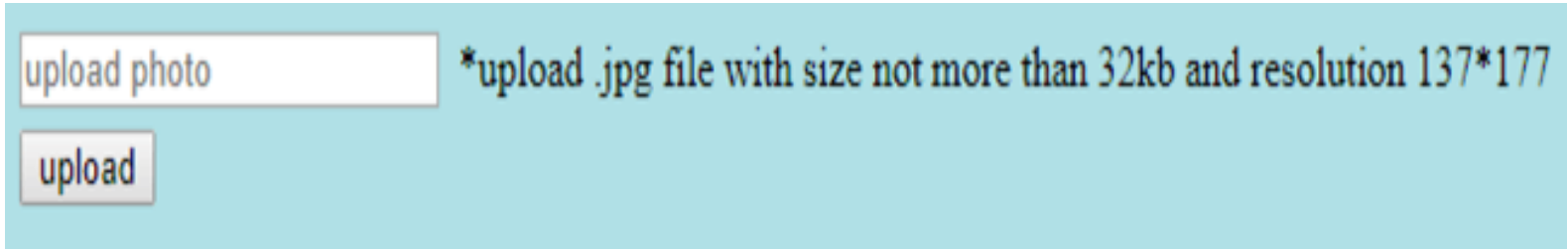
- T – Correct username/password
- F – Wrong username/password
- E – Error message is displayed
- H – Home screen is displayed

Example 2: How to make Decision Table for Upload Screen

Now consider a dialogue box which will ask the user to upload photo with certain conditions like –

- You can upload only '.jpg' format image
- file size less than 32kb
- resolution 137*177.

- If any of the conditions fails the system will throw corresponding error message stating the issue and if all conditions are met photo will be updated successfully



Example 2: How to make Decision Table for Upload Screen

Conditions	Case 1	Case 2	Case 3	Case 4	Case 5	Case 6	Case 7	Case 8
Format	.jpg	.jpg	.jpg	.jpg	Not .jpg	Not .jpg	Not .jpg	Not .jpg
Size	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb	Less than 32kb	Less than 32kb	>= 32kb	>= 32kb
resolution	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177	137*177	Not 137*177
Output	Photo uploaded	Error message resolution mismatch	Error message size mismatch	Error message size and resolution mismatch	Error message for format mismatch	Error message format and resolution mismatch	Error message for format and size mismatch	Error message for format, size, and resolution mismatch

Process for decision table testing

1. Analyze the given test inputs or requirements and list out the various conditions in the decision table.
2. Calculate the number of possible combinations (Rules)
3. Fill columns of the decision table with all possible combinations (Rules)
4. Find out cases where the values assumed by a variable are immaterial for a given combination Fill the same by "don't care" symbol (X).
5. For each of the combination of values, find out the action or expected result.
6. Create at least one Test case for each rule. If the rules are binary, a single test for each combination is probably sufficient. Else if a condition is a range of values, consider testing at both the low and high end of range.

Why Decision Table Testing is Important?

- **Decision Table Testing is Important** because it helps to test different combinations of conditions and provide better test coverage for complex business logic. When testing the behavior of a large set of inputs where system behaviour differs with each set of input, decision table testing provides good coverage and the representation is simple so it is easy to interpret and use.

Review Question: Decision Table Testing

A program to compute medical insurance has the following specification: If the driver is male and older than 35 years, offer a discount of 10%. Using the decision table testing technique, how many test cases should we have for the above rule?

Select one:

- a. 4
- b. 1
- c. 2
- d. 8

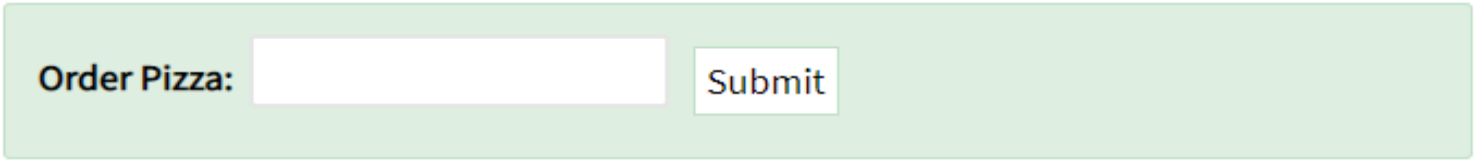
Use Decision Table Testing show all the test cases and the result.

What is Equivalent Class Partitioning?

- Equivalent Class Partitioning is a black box technique (code is not visible to tester) which can be applied to all levels of testing like unit, integration, system, etc. In this technique, you divide the set of test condition into a partition that can be considered the same.
- It divides the input data of software into different equivalence data classes.
- You can apply this technique, where there is a range in the input field.

Example 1: Equivalent Class Partitioning

- Let's consider the behavior of Order Pizza Text Box Below
- Pizza values 1 to 10 is considered valid. A success message is shown.
- While values greater than 10 are considered invalid for order and an error message will appear, **"Only 1 to 10 Pizzas can be ordered"**

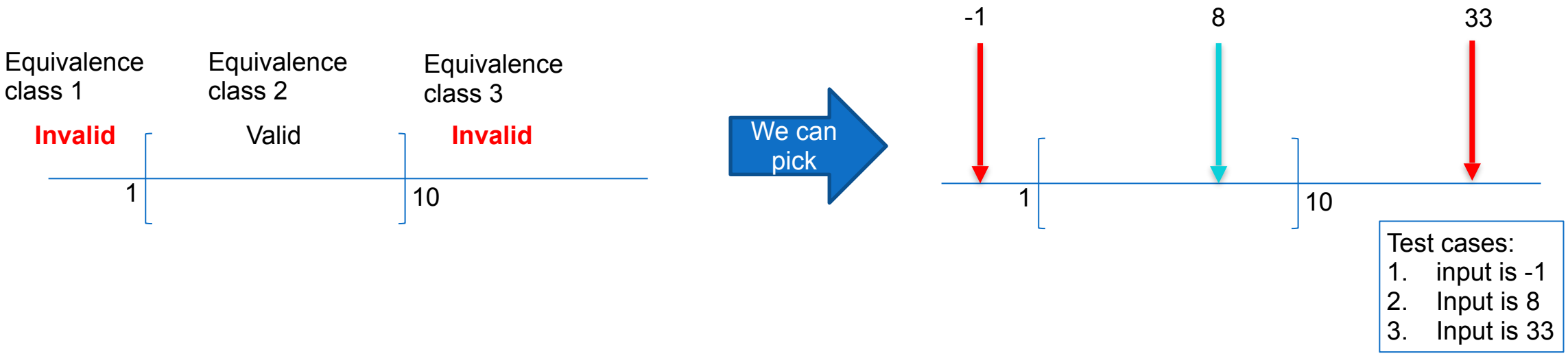


Here is the test condition

- Any Number greater than 10 entered in the Order Pizza field(let say 11) is considered invalid.
- Any Number less than 1 is considered invalid.
- Numbers 1 to 10 are considered valid

Example 1: Equivalent Class Partitioning

- We cannot test all the possible values because the number of test cases will be unlimited. To address this problem, we use equivalence classes partitioning hypothesis where we divide the possible values into intervals as shown below where the system behavior can be considered the same.
- Then we pick only one value from each partition for testing. The hypothesis behind this technique is **that if one condition/value in a partition passes all others will also pass**. Likewise, **if one condition in a partition fails, all other conditions in that partition will fail**.



Review Question

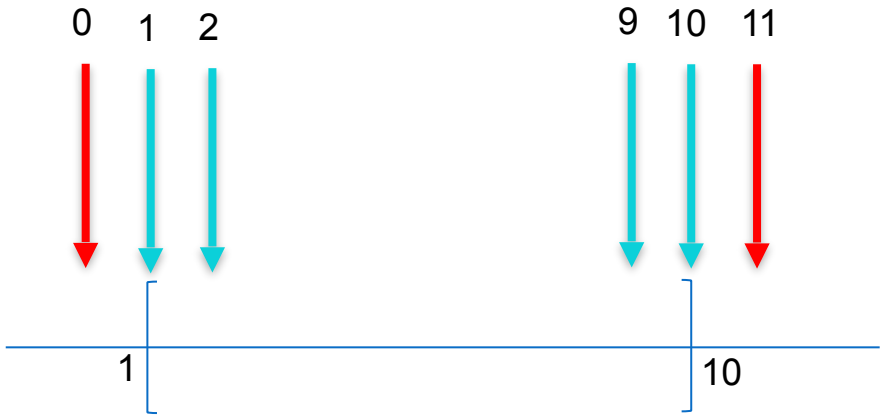
Which set of test cases derive from the following problem **if we apply equivalence classes partitioning**: Our application expects the user to enter his age. Teenagers (ages from 12 to 16) are given a 10% discount ?

a. Testcases : inputs { 9, 12, 16 }
b. Testcases : inputs { 0, 9, 14, 45 }
c. Testcases : inputs { -1, 9, 12, 16, 21 }
d. Testcases : inputs { 3, 14, 58 }

What is Boundary Testing?

- The basic idea in boundary value testing is to select boundary values for the given valid interval as follows:
  - Valid test cases:
    - Minimum valid value
    - Just above the minimum
    - Just below the maximum
    - Maximum valid value
  - Invalid test cases :
    - Just below the minimum valid value
    - Just above the maximum valid value

Example 1: Boundary Value Testing



In our earlier example instead of checking, one value for each partition you will check the values at the partitions like 0, 1, 2, 9, 10, 11. As you may observe, you test values at **both valid and invalid boundaries**. Boundary Value Analysis is also called **range checking**.

Equivalence partitioning and boundary value analysis(BVA) are closely related and can be used together at all levels of testing.

Review Question

The number of items in an order of a stock control system can range between 100 and 9999 inclusive. What are the test cases that should be derived from this problem **if we are applying boundary value testing?**

a. 99, 100, 101, 400, 9999, 10000
b. 50,, 99, 100, 9999, 10000, 10500
c. 100, 101, 9999, 10000
d. 99, 100, 101, 9998, 9999, 10000

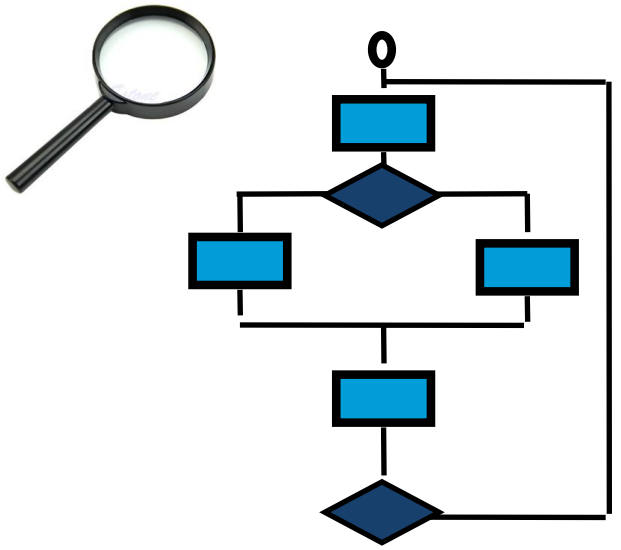
Review Question

To test an input field that accepts an integer that represents a day. What are the test cases that should be derived from this problem if we are applying boundary value testing?

a. 0, 1, 31, 32
b. -1, 0, 1, 30, 31, 32
c. 0, 1, 2, 30, 31, 32
d. 0, 1, 2, 29, 30, 31

White-Box Testing

- Test cases are derived from the internal design specification or actual code for the program.



- **Advantages**
  - Tests the internal details of the code
  - Checks all paths that a program can execute

- **Limitations**
  - Wait until after designing and coding the program under test in order to select test cases

Basis Path Testing

A White Box Method for Designing Test Cases

- Basis Path testing is a structural testing **method that involves using the source code** of a program in order to **find every possible executable path**.
- ~~It helps to determine all faults lying within a piece of code.~~
- This method is designed to execute all or selected path through a computer program.

Basis Path Testing
A White Box Method for Designing Test Cases

- Basis path testing **involves execution of all possible blocks** in a program and achieves **maximum path coverage with the least number of test cases.**

- The analysis of the code can be used to find **how many test cases are needed.**

Basis Path Testing

Why Basis Path Testing?

- Any software program includes, multiple entry and exit points. Testing each of these points is challenging as well as time-consuming.
- In order to reduce the redundant tests and to achieve maximum test coverage, basis path testing is used.

Basis path testing: CYCLOMATIC COMPLEXITY

- **CYCLOMATIC COMPLEXITY** is a **software metric** used to measure the **paths complexity of a program**.
- It is a quantitative **measure of independent paths** in the source code of the program.
- Independent path is defined as a path that has at least one edge which has not been traversed before in any other paths.
- Cyclomatic complexity can be calculated with respect to functions, modules, methods or classes within a program.
- It is based on a control flow representation of the program. Control flow depicts a program as a graph which consists of Nodes and Edges.

Steps for Basis Path testing

The following steps should be followed for computing Cyclomatic complexity and test cases design.

Step 1: Draw a control graph with nodes and edges from the source code (to determine different program paths)

Step 2: Calculate Cyclomatic Complexity (metric to determine the number of independent paths)

$V(G) = E - N + 2$ or also $V(G) = P + 1$

(n: nb of nodes, e: nb of edges, p: number of decision nodes)

Step 3: Identify basis set of paths

Step 4: Generate Test Cases to execute all the paths.

Source Code

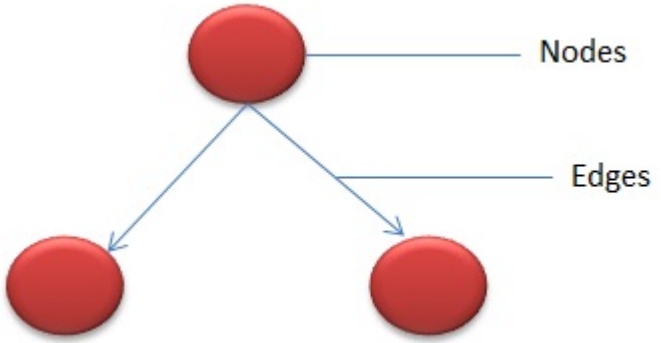
```
min = A[0]; I = 1;


while (I < N) {
  if (A[I] < min)
        min = A[I];
     I = I + 1;
}
print min
```

Flow Graph Notations

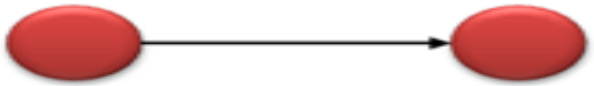
- In flow graph, each circle called a flow graph node represents **one** or **more procedural** statements.
- A sequence of **process** boxes and a decision diamond boxes can map into a **single node**.
- The arrows on the flow graph are called **edges or links**. They represent flow of control and are same as **flowchart arrows.**

1-Flow Graph Notation for a Program

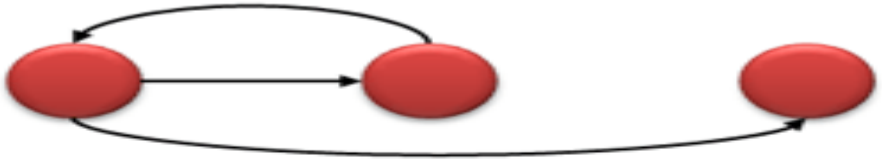
Flow Graph notation for a program defines several nodes connected through the edges.



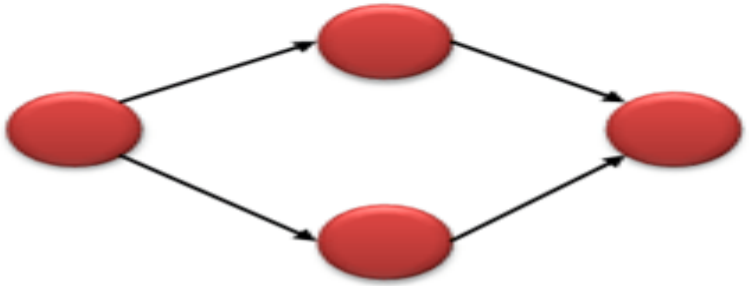
Sequence



While



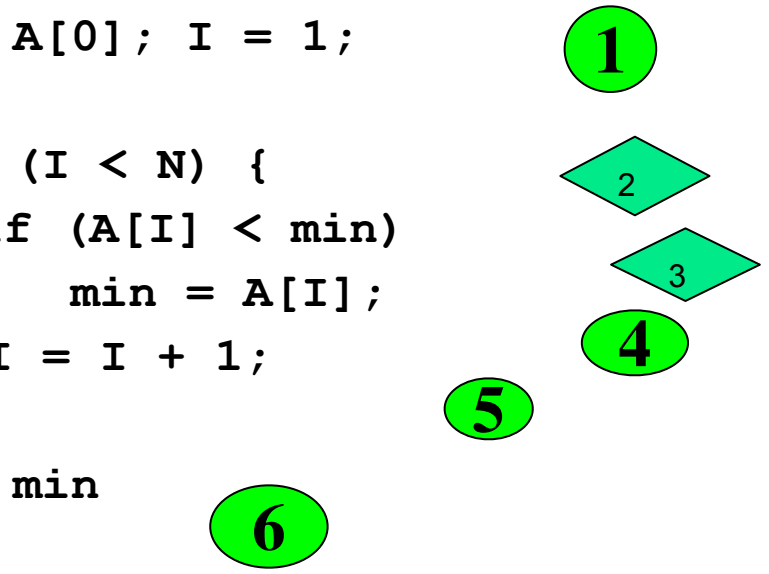
If-then-else

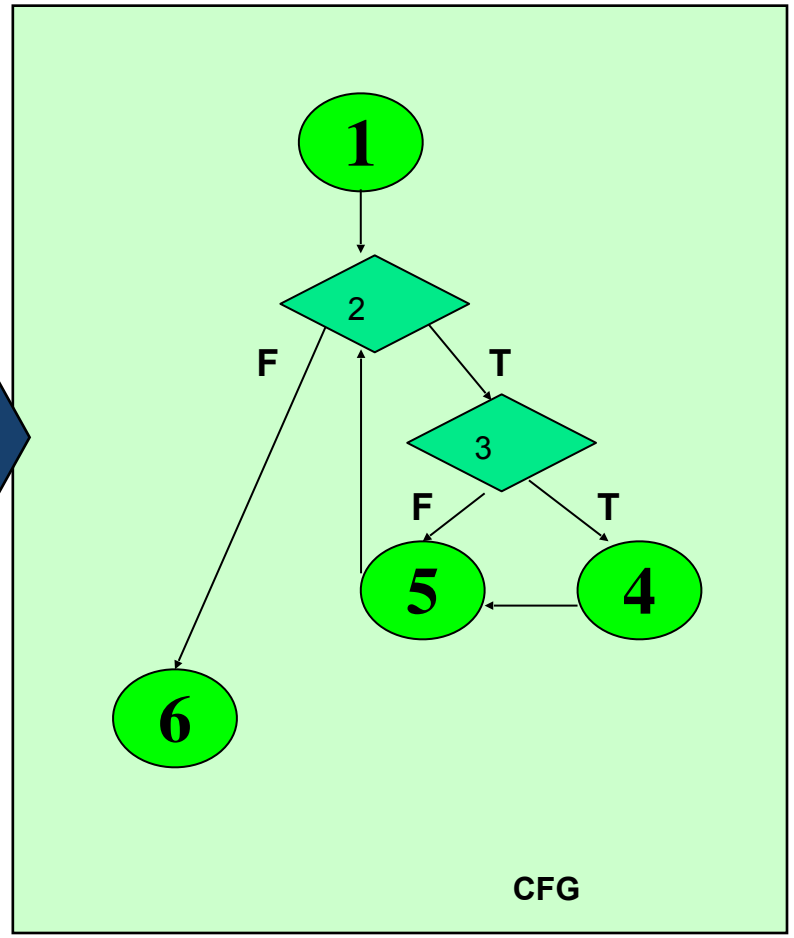


Until

Basis Path Testing: Step 1

```
min = A[0]; I = 1;

while (I < N) {
     if (A[I] < min)
         min = A[I];
      I = I + 1;
}
print min
```

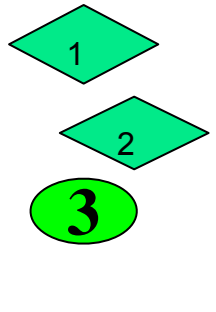




Basis Path Testing: Step 1

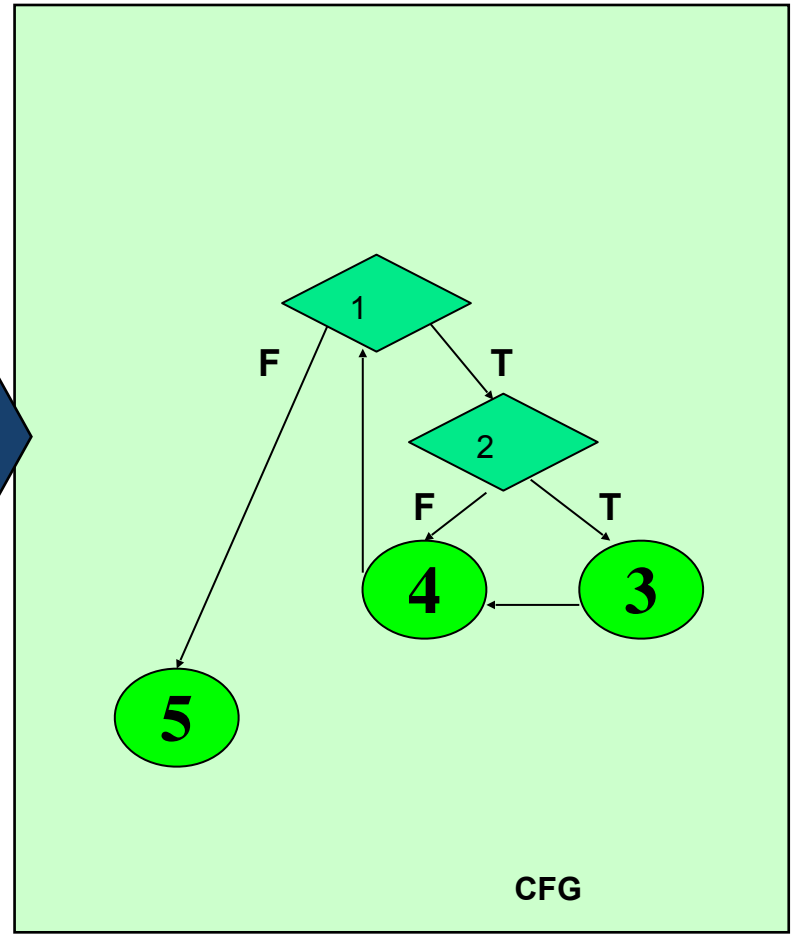
If we combine node 1 and 2 in the previous example, Still the answer is same.

```
min = A[0]; I = 1;

while (I < N) {
     if (A[I] < min)
         min = A[I];
     I = I + 1;
}
print min
```



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

V(G) = P+1
V(G) = 2+1 =3



How to Calculate Cyclomatic Complexity

Mathematical Representation:

- Mathematically, it is the number of independent paths through the graph diagram. The Code complexity of the program can be defined using the formula –
- **V(G) = E - N + 2**

  Where,

  E - Number of edges

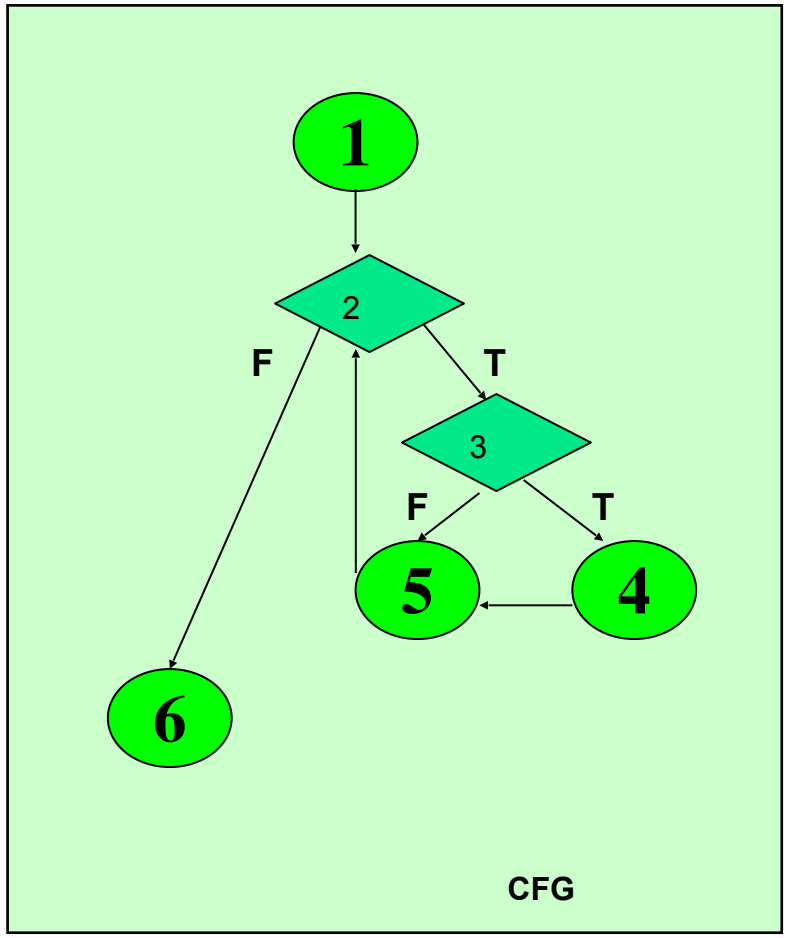
  N - Number of Nodes

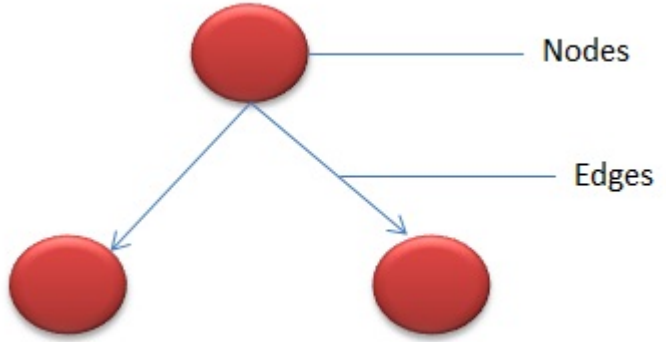
V (G) = P + 1

Where

P = Number of condition nodes (node that contains condition)

Basis Path Testing: Step 2

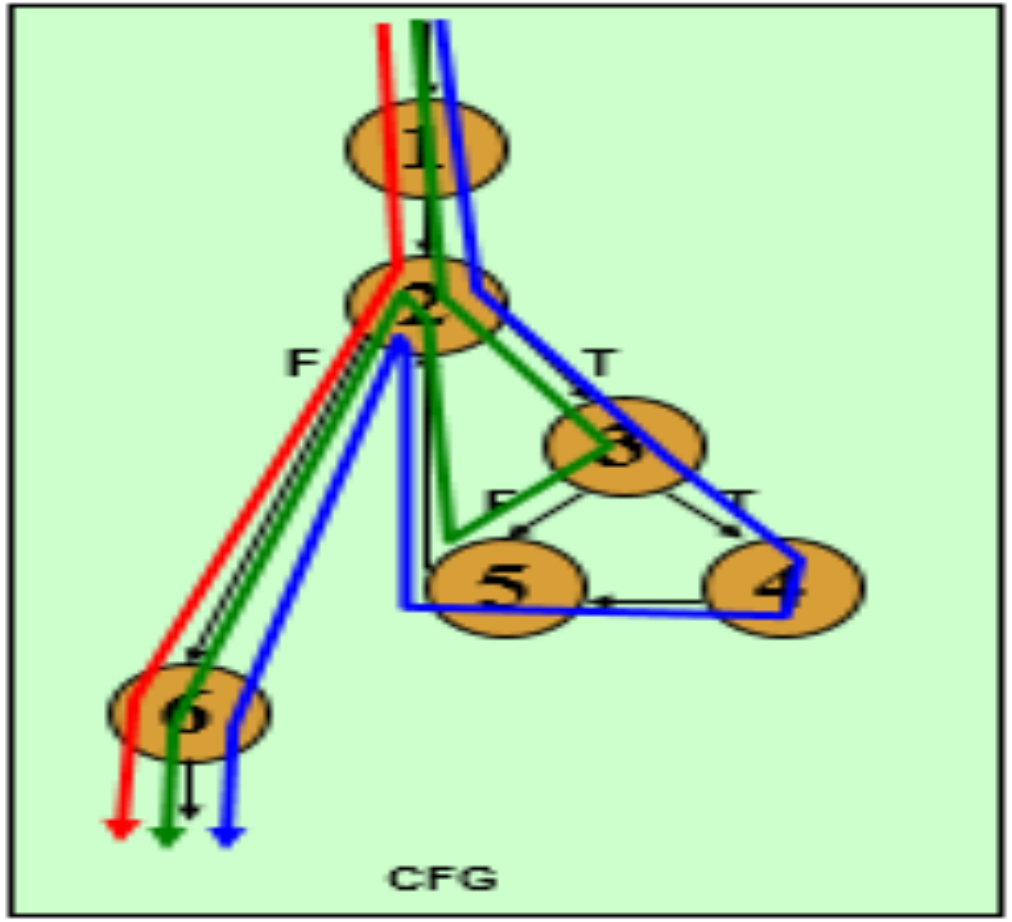




Cyclomatic Complexity
= E - N + 2
= 7 - 6 + 2 = 3

Cyclomatic Complexity
= Number of Predicates
+ 1
= 2 + 1 = 3

Basis Path Testing: Step 3



- Cyclomatic complexity = 3.
- Need at most **3** independent paths to cover the Control Flow Graph.
- In this example:
  - [1 – 2 – 6]
  - [1 – 2 – 3 – 5 – 2 – 6]
  - [1 – 2 – 3 – 4 – 5 – 2 – 6]

Basis Path Testing: Step 4

- Prepare a test case for each independent path.
- In this example:
  - [1 – 2 – 6]: N=0, A[]
  - [1 – 2 – 3 – 5 – 2 – 6]: N=2, A[5,7]
  - [1 – 2 – 3 – 4 – 5 – 2 – 6]: N=3, A[5,3,4]

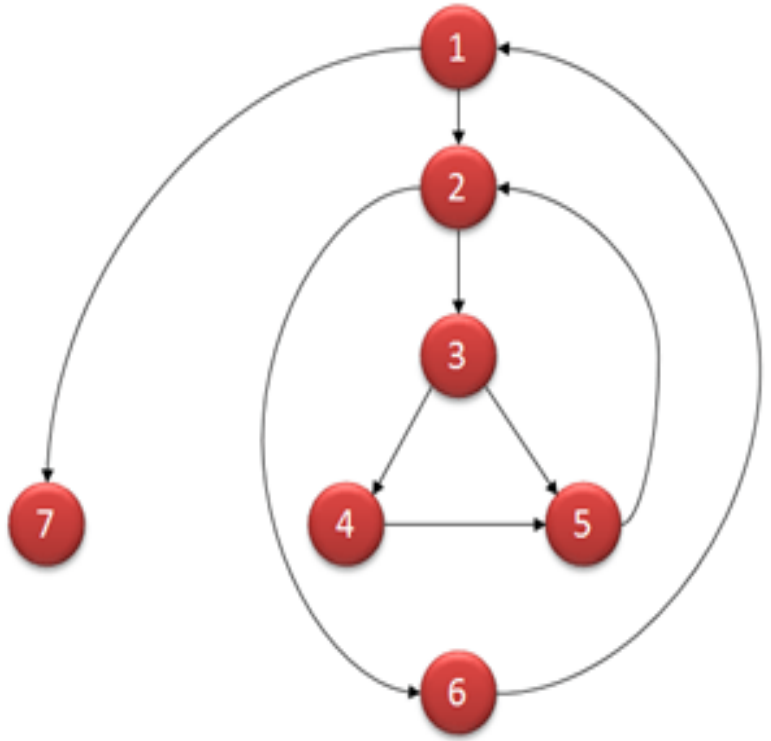
Sorting the array in descending order

```
i = 0;
n=4; //N-Number of nodes present in the graph

while (i<n-1) do
j = i + 1;

while (j<n) do

if A[i]<A[j] then
swap(A[i], A[j]);

end do;
i=i+1;

end do;
```

exit

1 2 3 4 5 6 7

Flow graph for this program will be



Sorting the array in descending order

Computing mathematically,

V(G) = 9 - 7 + 2 = 4

V(G) = 3 + 1 = 4 (Condition nodes are 1,2 and 3 nodes)

Basis Set - A set of possible execution path of a program

1, 7

1, 2, 6, 1, 7

1, 2, 3, 4, 5, 2, 6, 1, 7

1, 2, 3, 5, 2, 6, 1, 7

How this metric is useful for software testing?

- Basis Path testing is one of White box technique and it guarantees to execute each statement at least once during testing.
- It checks each "linearly independent path" of the program, which **means that the number of test cases will be equal to the cyclomatic complexity of the program.**

Complexity number and Corresponding meaning of v (G):

Complexity Number	Meaning
1-10	Structured and well written code High Testability Cost and Effort is low
10-20	Complex Code Medium Testability Cost and effort is Medium
20-40	Very complex Code Low Testability Cost and Effort are high
>40	Not at all testable Very high Cost and Effort

Uses of Cyclomatic Complexity

- Helps developers and testers to determine independent path executions
- Developers can assure that all the paths have been tested at least once
- Helps to focus more on the uncovered paths
- Improve code coverage in testing
- Evaluate the risk associated with the application or program
- Using these metrics early in the cycle reduces risks of project failure

Comparison

Black Box Testing

- The main focus of black box testing is on the validation of functional requirements.
- Black box testing gives abstraction from code and focuses on testing effort on the software system behavior.
- Black box testing facilitates testing communication amongst modules

White Box Testing

- White Box Testing validates internal structure and working of your software code
- To conduct White Box Testing, knowledge of underlying programming language is essential.
- White box testing does not facilitate testing communication amongst modules

Levels of Testing

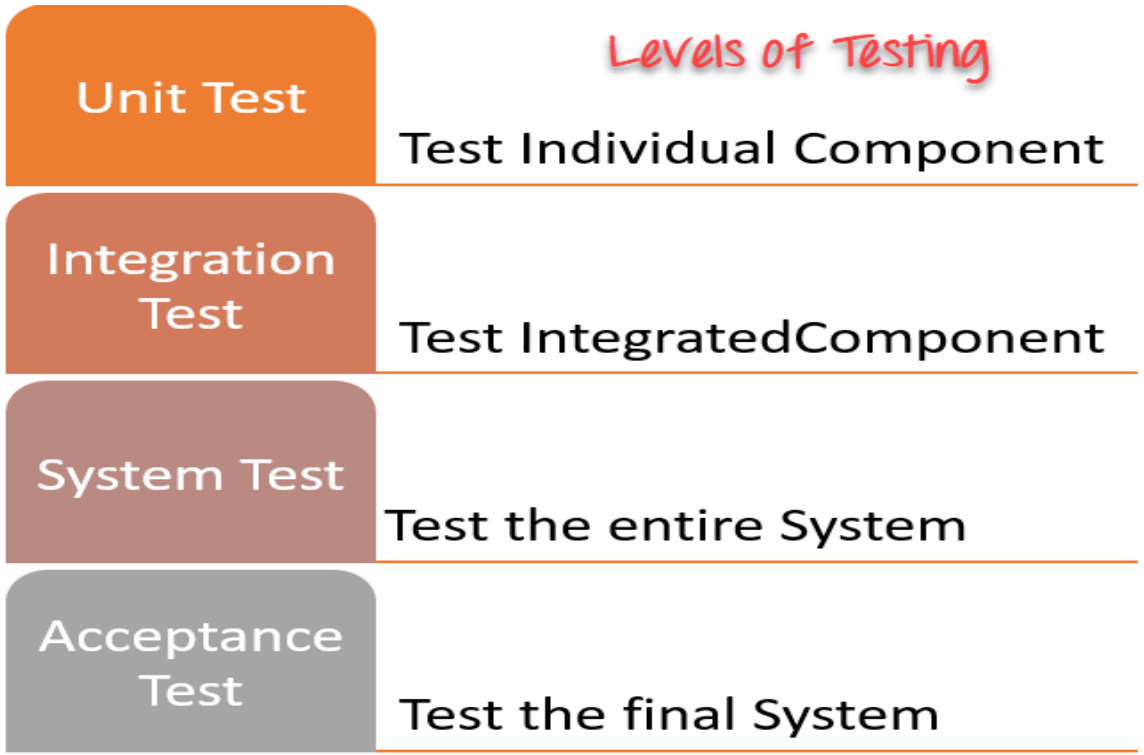
here

There are different levels of testing. These levels are executed by different persons and at different moments.

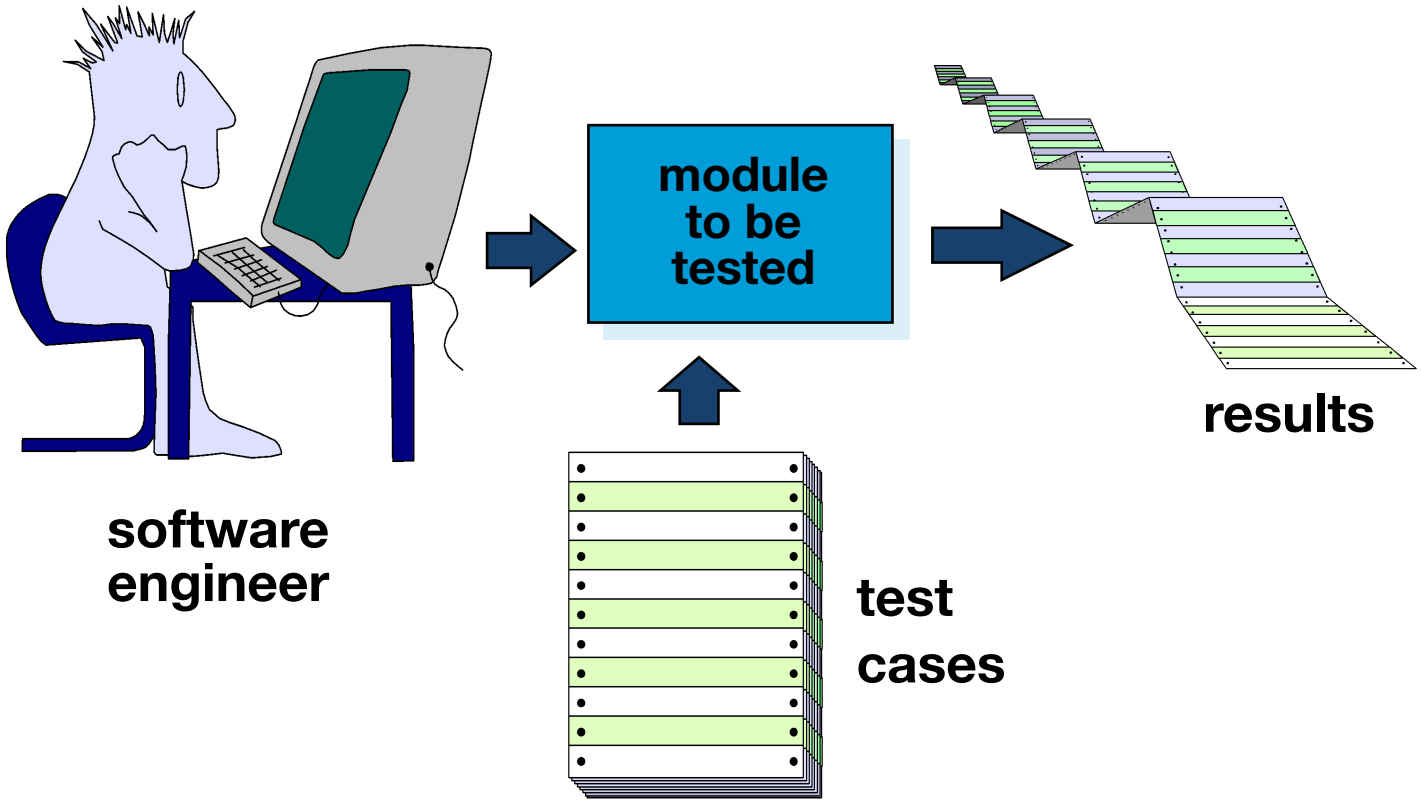
- Unit testing
- Integration testing
- System testing
- Acceptance testing:

Other Types of Testing:

- Regression testing



Unit Testing



UNIT TESTING

- **Unit Testing** is a type of software testing where individual units or components of a software are tested. The purpose is to validate that each unit of the software code performs as expected.
- A Unit is a smallest testable portion of system or application (individual function, method, procedure) which can be compiled and executed.
- The aim is to **test** each part of the software by separating it. It checks that component are fulfilling functionalities and verify its correctness. Usually this kind of testing is performed by developers.

UNIT TESTING

Why Unit Testing?

- Sometimes software developers attempt to save time by doing minimal unit testing. This is a myth because skipping on unit testing leads to higher Defect fixing costs during System Testing, Integration Testing and even Beta Testing after the application is completed.
- Proper Unit tests help to fix bugs early in the development cycle and save both time and money in the end.
- It helps the developers understand the code base and enables them to make changes quickly.
- Good unit tests serve as project documentation.
- Unit tests help with code re-use. Migrate both your code **and** your tests to your new project.

Unit Testing
Code Review

Code walkthrough

1. It's a type of **Semi Formal Review.**
2. **Author is Presenter.**
3. **Lead by Author** only.
4. **Reviewers are not aware** of the subject/topic.

Code inspection

It's totally a **Formal Review.**
Author is **not presenter**. Someone else is giving the presentation.
Lead by **Moderator.**
Reviewers are **aware** & **well prepared** for the subject/topic.

Unit Testing

The Best Team Size for Inspections

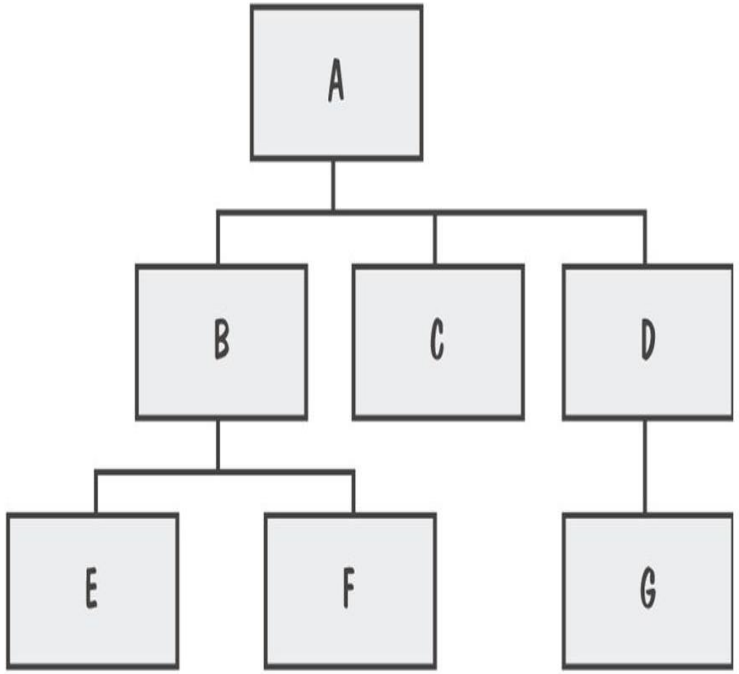
- The preparation level, not the team size, determines inspection effectiveness
- The team's effectiveness and efficiency depend on their familiarity with their product

Integration Testing

- Involves building a system from its components and
  - testing it for problems that arise from component interactions.

To simplify error localisation, systems should be incrementally integrated.

- System viewed as a hierarchy of components.
- In this testing phase, different software modules are combined and tested as a group to make sure that integrated system is ready.
- A typical software project consists of multiple software modules coded by different programmers. Integration Testing focuses on checking data flow from one module to other. This kind of testing is performed by **testers**.



Integration Testing

- The purpose of this level of testing is to expose defects in the interaction between these software modules when they are integrated.
- Integration Testing focuses on checking data communication amongst these modules.

Why do Integration Testing?

- A Module, in general, is designed by an individual software developer whose understanding and programming logic may differ from other programmers. **Integration Testing** becomes necessary to verify the **software modules work in unity**
- At the time of module development, there are wide **chances of change in requirements by the clients**. These new requirements may not be unit tested and hence system integration Testing becomes necessary.
- **Interfaces of the software modules** with the database could be erroneous
- **External Hardware interfaces**, if any, could be erroneous
- Inadequate exception handling could cause issues.

Example of Integration Test Case

Sample Integration Test Cases for the following scenario:

- Application has 3 modules say 'Login Page', 'Mailbox' and 'Delete emails' and each of them is integrated logically.
- The software tester does not concentrate much on the Login Page testing as it's already been done in Unit Testing. But he/she checks how Login Page is linked to the Mail Box Page.
- Similarly check how Mail Box Page is integrated with the Delete Mails Module.

Example of Integration Test Case

Test Case ID	Test Case Objective	Test Case Description	Expected Result
1	Check the interface link between the Login and Mailbox module	Enter login credentials and click on the Login button	To be directed to the Mail Box
2	Check the interface link between the Mailbox and Delete Mails Module	From Mailbox select the email and click a delete button	Selected email should appear in the Deleted/Trash folder

Types of Integration Testing

- **Big-Bang Approach**
- **Incremental Approach:** which is further divided into the following
  - Top Down Approach
  - Bottom Up Approach

Approaches, Strategies, Methodologies of Integration Testing

Big Bang Approach:

- Here all component are **integrated together** at **once** and then **tested**.
- **Advantages:**
- Convenient for small systems.
- **Disadvantages:**
- Fault Localization is difficult.

Example Fault localization
BUG: Concatenation (meaning two words together likethis)
SOLUTION: Insert a space
TEAM: Localization engineer

- Given the sheer number of interfaces that need to be tested in this approach, some interfaces link to be tested could be missed easily.
- Since all modules are tested at once, high-risk critical modules are not isolated and tested on priority.

Approaches, Strategies, Methodologies of Integration Testing

- **Incremental Approach**
  - In this approach, testing is done by joining two or more modules that are ***logically related***.
  - Then the other related modules are added and tested for the proper functioning.
  - The process continues until all of the modules are joined and tested successfully.
- Incremental Approach is carried out by two different methods:
  - Bottom Up
  - Top Down

Integration Testing Approach: Incremental Approach

- **What is Stub and Driver?**
- Incremental Approach is carried out by using dummy programs called **Stubs and Drivers**. Stubs and Drivers do not implement the entire programming logic of the software module but just simulate data communication with the calling module.
- **Stub**: a special-purpose program to simulate the activity of the missing component. Is called by the Module under Test.
- **Driver**: A routine that calls a particular component and passes a test case to it. Calls the Module to be tested.
- Stubs and drivers both are dummy modules and are only created for testing purposes.

What is a Stub?

Example:

Consider a computer program that queries a database to obtain the min price of all products that are ”cars” stored in the database.

In this example, the query is slow and consumes a large number of system resources.

Secondly, tests may include values outside those currently in the database. The method (or call) used to perform this is *get_min()*.

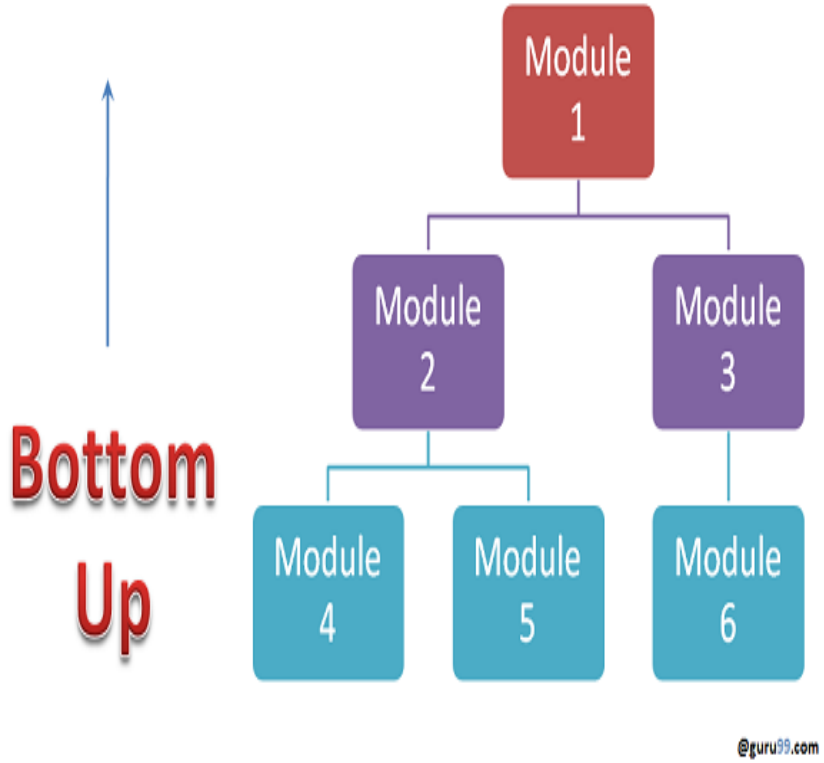
For testing purposes, the source code in ***get_min()*** can be temporarily **replaced with a simple** statement that returns a specific value. This would be a **test stub**.

What is a Driver?

- On the other hand, **Drivers** are the "**calling" programs**. Drivers are used in **bottom up testing** approach. Drivers are **dummy code**, which is used when the **sub modules are ready** but the **main module is still not ready.**

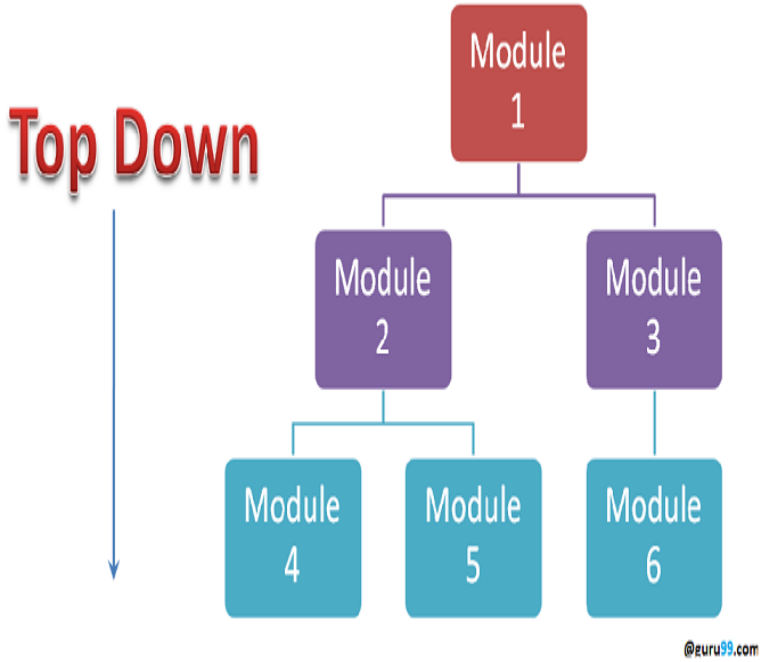
Integration Testing Approach: Incremental Approach

- **Bottom-up Integration**
- In the bottom-up strategy, each module at lower levels is tested with higher modules until all modules are tested. It takes help of Drivers for testing
- **Advantages:**
- Fault localization is easier.
- No time is wasted waiting for all modules to be developed unlike Big-bang approach
- **Disadvantages:**
- Critical modules (at the top level of software architecture) which control the flow of application are tested last and may be prone to defects.
- An early prototype is not possible



Integration Testing Approach: Incremental Approach

- **Top-down Integration:**
- In Top to down approach, testing takes place from top to down following the control flow of the software system.
- Takes help of stubs for testing.
- **Advantages:**
- Fault Localization is easier than big-bang approach.
- Possibility to obtain an early prototype.
- Critical Modules are tested on priority; major design flaws could b found and fixed first.
- **Disadvantages:**
- Needs many Stubs.
- Modules at a lower level are tested inadequately.



System Testing

here

- **SYSTEM TESTING** is a level of testing that validates the complete and fully integrated software product.
- The purpose of a system test is to evaluate the **end-to-end system specification.**
- System testing is done by a **professional testing agent** on the **completed software product** before it is introduced to the market.
- Usually, the software is only one element of a larger computer-based system. Ultimately, the **software** is **interfaced** with other **software**/hardware **systems**.
- System Testing is actually a **series of different tests** whose sole purpose is to exercise the **full computer-based system**.

System Testing

- System **Functional** Test
  - Test entire system against the functional requirements.

- System **Performance** Test
  - Test the non-functional requirements of the system. For example,
  - Response times, load testing etc.

Non-functional software testing

- **NON-FUNCTIONAL Software testing** is defined as a type of Software testing to check non-functional aspects (performance, usability, reliability, etc) of a software application.
- An example of non-functional test is to check how many users can simultaneously use a software.
- Non-functional testing is equally important as functional testing and affects client satisfaction.

Non-functional software testing (2)

- The purpose of **Performance Testing** is not to find functional defects but to eliminate performance bottlenecks in the software or device.

- Performance Testing checks the speed, response time, reliability, resource usage, scalability of a software program under their expected workload.
  - Speed - Determines whether the application responds quickly
  - Scalability - Determines maximum user load the software application can handle.
  - Stability - Determines if the application is stable under varying loads

- Performance Testing is popularly called "Perf Testing".

Non-functional software testing (3)

Security: Test how a system is protected from deliberate attacks from internal and external sources.

Reliability: Test if the system continuously performs the specified functions without failure.

Survivability (Resiliency): Test if the software system continues to function and recovers itself in case of system failure.

Examples:

- When an application is receiving data from the network, unplug the connecting cable. After some time, plug the cable back in and analyze the application's ability to continue receiving data from the point at which the network connection was broken.
- Restart the system while a browser has a definite number of sessions open and check whether the browser is able to recover all of them or not

Availability: Test the degree to which a user can depend on the system during its operation. It involves exercising the system with a heavy number of users and measuring the performance parameters to verify whether the system can support the anticipated load (Stress tests).

Non-functional software testing (4)

Usability: Test the ease with which the user can learn, operate, prepare inputs and outputs through interaction with a system.

Scalability: Test if software application can expand its processing capacity to meet an increase in demand by rapidly and gradually increasing the load on the application until it reaches a threshold/limit.

What is Acceptance Testing?

- Acceptance testing is beta testing of the product done by the actual end users.
- Here, only black-box Testing techniques are used.
- Acceptance testing is a test conducted to find if the requirements of a specification or contract are met as per its delivery.
- Acceptance testing is basically done by the user or customer. However, other stakeholders can be involved in this process.

What is Regression Testing?

When new software is released then

- the need to test new functionality is obvious. Equally important is the need to re-run old tests that the application previously passed, to ensure that new software does not re-introduce old defects or create new side effects: **called regressions**.
- While regression testing is an apparently simple concept, it can be quite challenging in practice due to limited resources such as time, personnel, and test systems.