UNIT 5
Software Coding & Testing

Prepared by :

Dr. Pooja M Bhatt

MBIT,CVM University

Index

- Coding Standard and Coding Guidelines
- Code Review
- Software Documentation
- Testing Strategies
- Testing fundamentals
- Testing Conventional Approach
- Testing Object Oriented Approach
- Testing Web and Mobile Applications
- Testing Tools(Win runner,Load Runner)

Coding Standards and Coding guidelines

- The goal of coding is to implement the design in the best possible manner. Coding affects both testing and maintenance. The coding should be done in such a manner that the instead of getting the job of programmer simplified the task of testing and maintenance phase should get simplified.
- Most software development organizations formulate their own coding standards that suit them most, and require their engineers to follow these standards strictly.

Coding Standards and Coding guidelines

- **The objective of using specific coding style are:**

1.It produces uniform codes.

2. It is very easy for the developer to understand the code.

3. It is very good programming practice.

4. Newly join members in the development team can easily cope up with progress of the development process.

Coding Standards and Coding guidelines

- There are some commonly used programming practice to avoid the common errors.
- **The following are some representative coding guidelines**
- Do not use a coding style that is too clever or too difficult to understand.
- Do not use an identifier for multiple purposes .
- The code should be well-documented.
- The length of any function or procedure should be short, it should not be too lengthy.
- Avoid using goto statements.
- Avoid using complex conventions.
- Use user defined data type for increase in code readability. like enumerated data type

Coding Standards and Coding guidelines

- Avoid obscure side effects.If some part of code is changed randomly then it will cause some side effect. For example if number of parameters passed to the function is changed then it will be difficult to understand the purpose of that function.
- Use meaningful name for the specific purpose.
- Avoid deep nesting. For example

Too much nesting	Avoid nesting

While (condition)
{
if condition then statement
else if condition then statement
else if condition then statement
else if condition then statement
}
	While (condition)
	{
	if condition then statement
	if condition then statement
	if condition then statement
	}

Coding standards

- A coding standard lists **several rules** to be **followed** such as, the **way variables** are to be **named**, the **way** the **code** is to **be laid out**, **error** return **conventions**, etc.
- The following are some representative coding standards
- **1. Naming conventions**
- Package name and variable names should be in lower case.
- Variable name must not begin with numbers
- The type name should be noun and it should start with capital letter.
- Constants must be in upper case (like PI,SIZE).
- Method name must be given in lower case.
- The variables with large scope must have long name. for example count_total, sum.
- The variables with short scope must have short name. for example i, j.

- **2.Files**
- Readers should have idea about file by its name .
- Example in java programming file should have extension as java. class and file name should be same.
- **3.Commenting/Layout**
- For enhance the readability of code and to explain logic of program comments(single line or multi line) should be used.
- **4.Statements**
- Guideline for executable statement are as follow:
- Declare some related variables on same line and unrelated variables on another line.
- Avoid complex conditional expressions. Make use of temporary variables instead.
- Make use of only loop control within the for loop.
- Class variable should never be declared public.
- Avoid do while statement.

Coding Standards and Coding guidelines

Advantages of Coding Standards :

1.Coding standard brings uniform appearance in system implementation.

2. The code becomes readable and hence can be understood easily.

3.The coding standard helps in adopting good programming practices.

Software Faults

- Quite inevitable
- Many reasons

1. Software systems with large number of states
2. Complex formulas, activities, algorithms
3. Customer is often unclear of needs
4. Size of software
5. Number of people involved

Types of Faults

Algorithmic	**Logic is wrong Code reviews**
Syntax	Wrong syntax; type of Compiler
Computation/ Precision	Not enough accuracy
Documentation	Misleading documentation
Stress/Overload	Maximum load violated
Capacity/Boundary	Boundary cases are usually special cases
Timing/Coordination	Synchronization issues Very hard to replicate
Throughput/Performance	System performs below expectations
Recovery	System restarted from abnormal state
Hardware & related software	Compatibility issues

Code Review

- Code Review is carried out **after the module is successfully compiled** and all the syntax errors have been eliminated.
- Code Reviews are extremely **cost-effective strategies** for **reduction in coding errors** and to produce high quality code.
- Types of review:

1. Code walk through
2. Code inspection

Types of review:

- **1. Code walk through**
- Code walk through is an **informal code analysis** technique.
- The main **objectives** of the walk through are **to discover** the **algorithmic** and **logical errors** in the **code**.
- A **few members** of the development **team** are given the **code** few days before the walk through meeting to read and understand code.
- Each **member selects some test cases** and **simulates execution** of the code **by hand**
- The members **note down their findings** to **discuss** these **in a walk through meeting** where the coder of the module is present.

Types of review:

- **2. Code inspection**
- The **aim** of **Code Inspection** is to **discover** some **common types** of **errors** caused **due to improper programming**.
- In other words, during Code Inspection **the code is examined** for the **presence** of certain **kinds of errors**.
- For instance, consider the classical error of writing a procedure that modifies a parameter while the calling routine calls that procedure with a constant actual parameter.
- It is more likely that **such an error will be discovered by looking for these kinds of mistakes in the code**.
- In addition, **commitment to coding standards** is also **checked**.

Few classical programming errors

- loops with no termination
- Improper storage allocation and deallocation
- Use of uninitialized variables
- Jumps into loops
- Incompatible assignments
- Array indices out of bounds
- Mismatches between actual and formal parameter in procedure calls
- Use of incorrect logical operators or incorrect precedence among operators
- Improper modification of loop variables

Software Documentation

- When various kinds of software products are developed, various kinds of **documents** are also developed as part of any software engineering process e.g.
- Users' manual,
- Software requirements specification (SRS) documents,
- Design documents,
- Test documents,
- Installation manual, etc
- Different **types of software documents** can broadly be classified into the following:
- 1. Internal documentation
- 2. External documentation

Types of Software Documentation

1.Internal Documentation

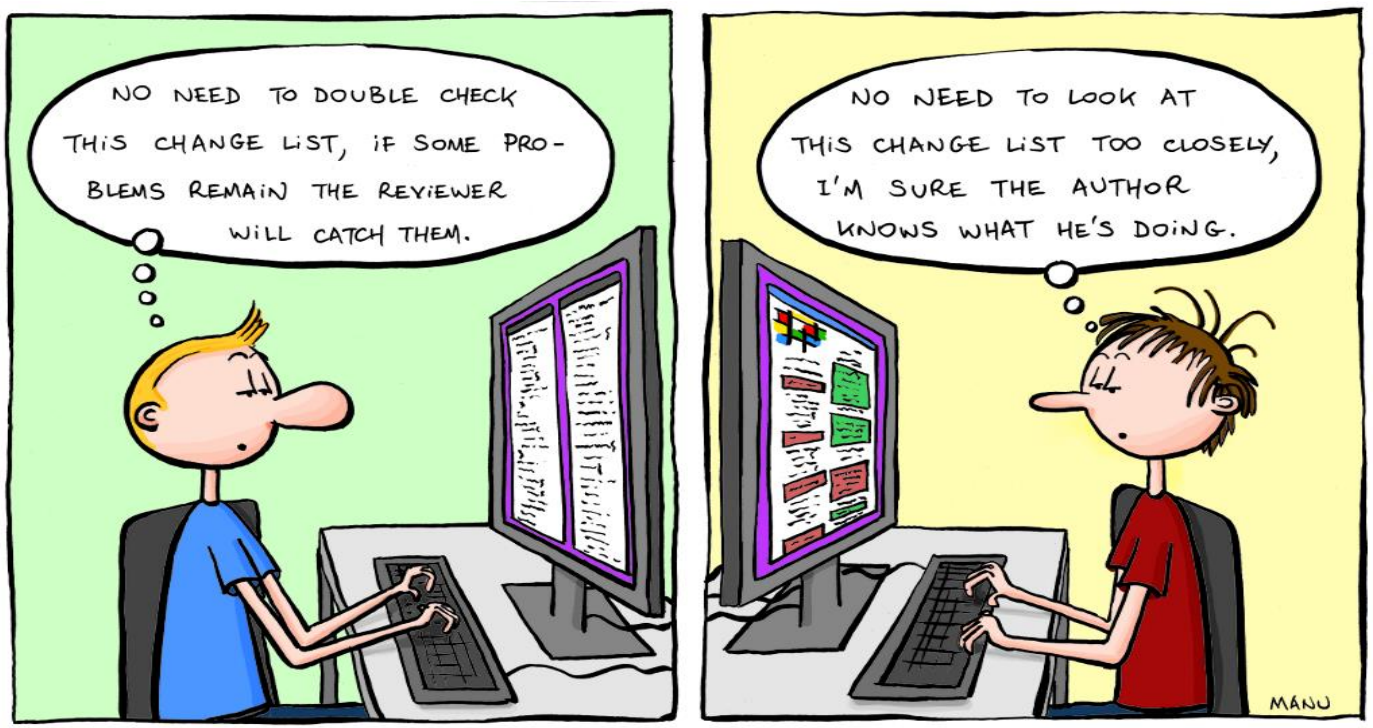
- It is the **code perception features** provided as part of the source code.
- It is provided through appropriate **module headers and comments** embedded in the source code.
- It is also provided through the useful **variable names, module and function headers, code indentation, code structuring, use of enumerated types and constant identifiers, use of user-defined data types**, etc.
- Even when code is carefully commented, **meaningful variable names** are still more helpful in understanding a piece of code.
- Good organizations ensure good internal documentation by appropriately formulating their coding standards and guidelines.

2. External Documentation

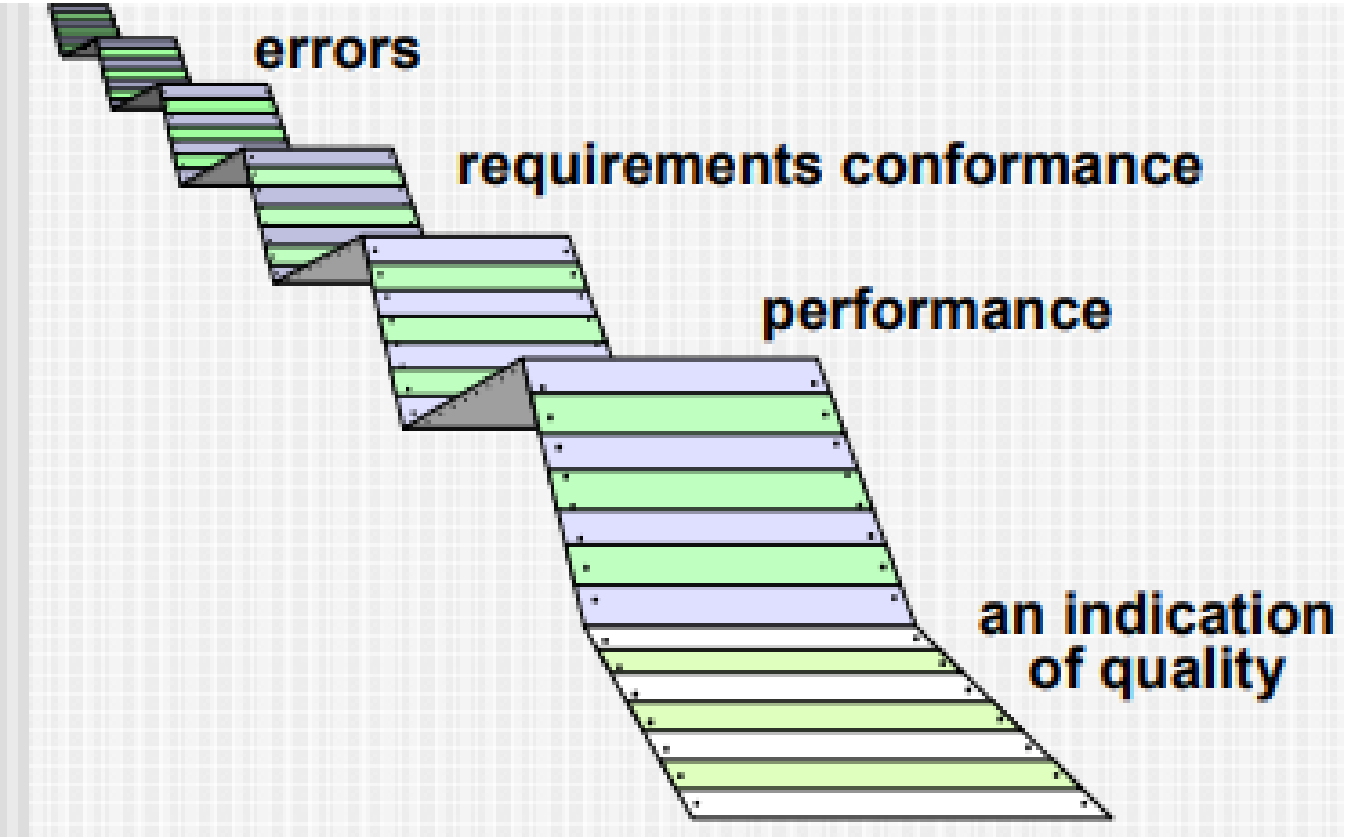
- It is provided through various types of **supporting documents**
- such as users' manual
- software requirements specification document
- design document
- test documents, etc.
- A systematic software development style ensures that all these documents are **produced in an orderly fashion**.

Software Testing

- Testing is the process of exercising a program with the specific intent of finding errors prior to delivery to the end user.



What Testing Shows



Strategic approach

- To perform effective testing, you should conduct effective technical reviews. By doing this, many errors will be eliminated before testing commences.
- Testing begins at the component level and works "outward" toward the integration of the entire computer based system.
- Different testing techniques are appropriate for different software engineering approaches and at different points in time.
- **Testing** is conducted by the developer of the software and (for large projects) an independent test group.
- **Testing** and **debugging** are different activities, but debugging must be accommodated in any testing strategy.

V & V

- **Verification** refers to the set of tasks that ensure that software correctly implements a specific function.
- **Validation** refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements. Boehm [Boe81] states this another way:
- **Verification**: "Are we building the product right?" **Validation**: "Are we building the right product?"

Who Tests the Software?

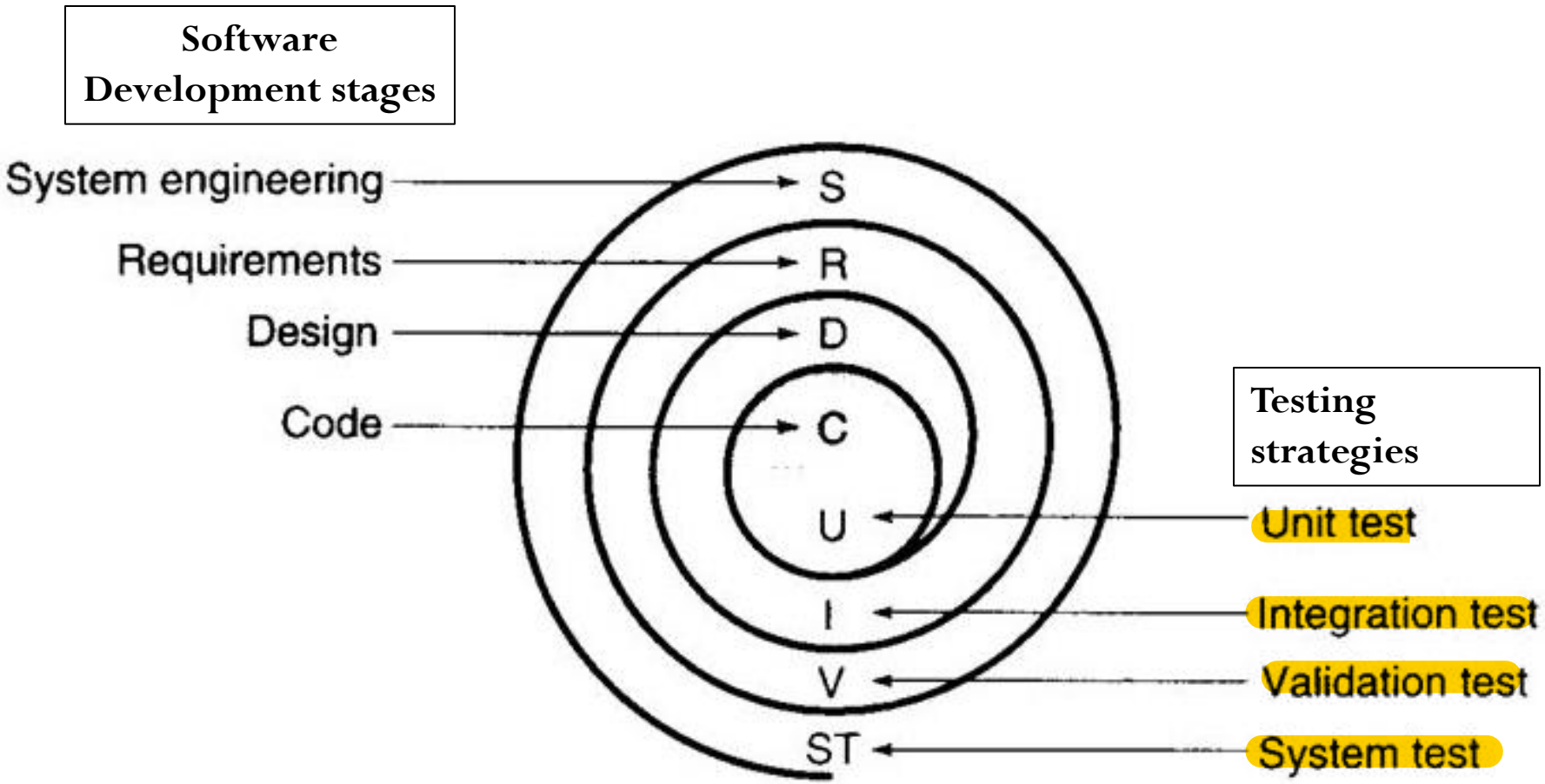
developer

Understands the system
but, will test "gently"
and, is driven by "delivery"

independent tester

Must learn about the system,
but, will attempt to break it
and, is driven by quality

Testing Strategies



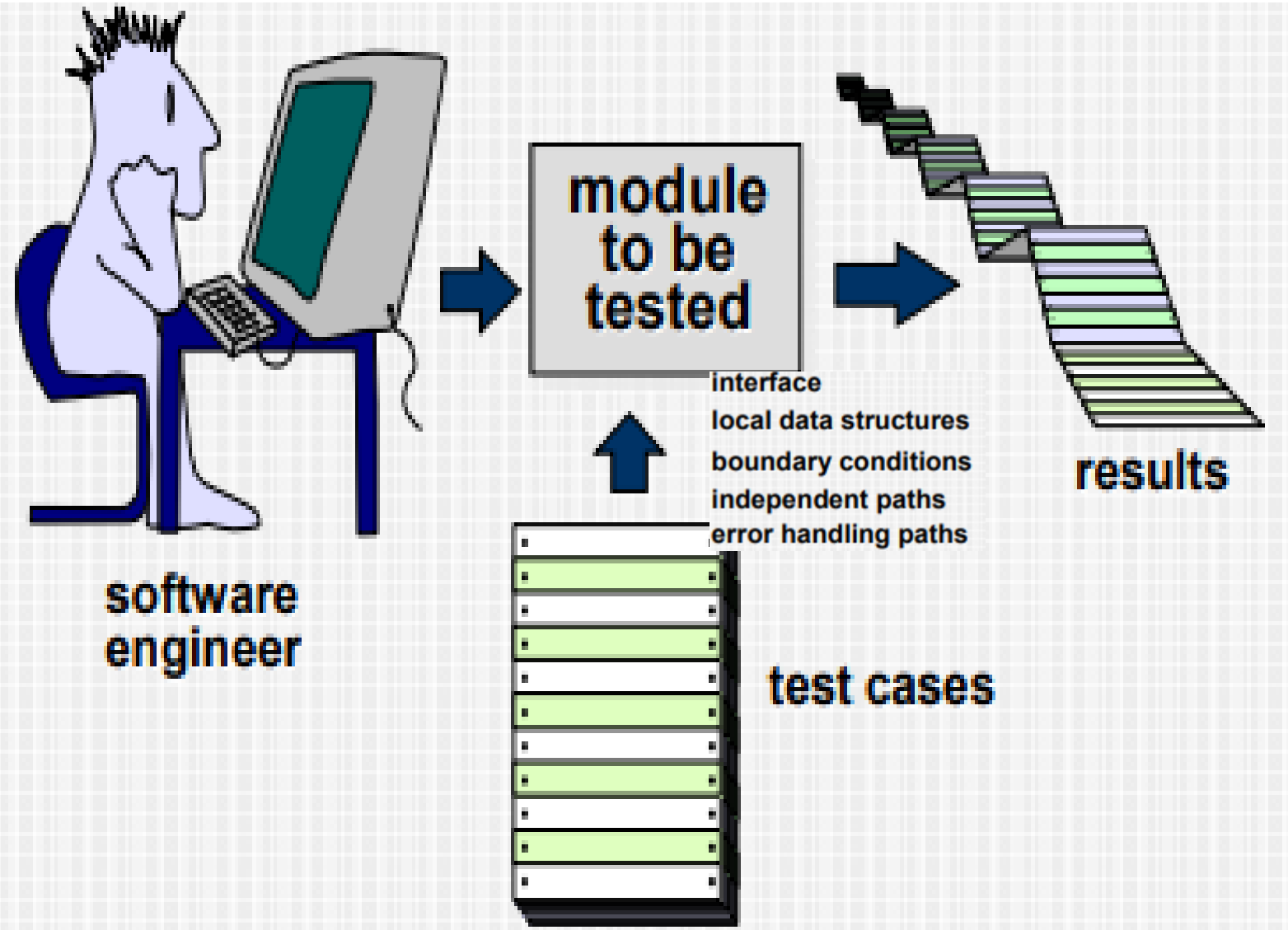
Testing Strategy

- We begin by ‘testing-in-the-small’ and move toward ‘testing-in-the-large’
- **For conventional software:**
- The module (component) is our initial focus
- Integration of modules follows
- **For OO software :**
- our focus when “testing in the small” changes from an individual module (the conventional view) to an OO class that encompasses attributes and operations and implies communication and collaboration

Strategic Issues

- Specify product requirements in a quantifiable manner long before testing commences.
- State testing objectives explicitly.
- Understand the users of the software and develop a profile for each user category.
- Develop a testing plan that emphasizes "rapid cycle testing."
- Build "robust" software that is designed to test itself .
- Use effective technical reviews as a filter prior to testing.
- Conduct technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

(A)Unit Testing

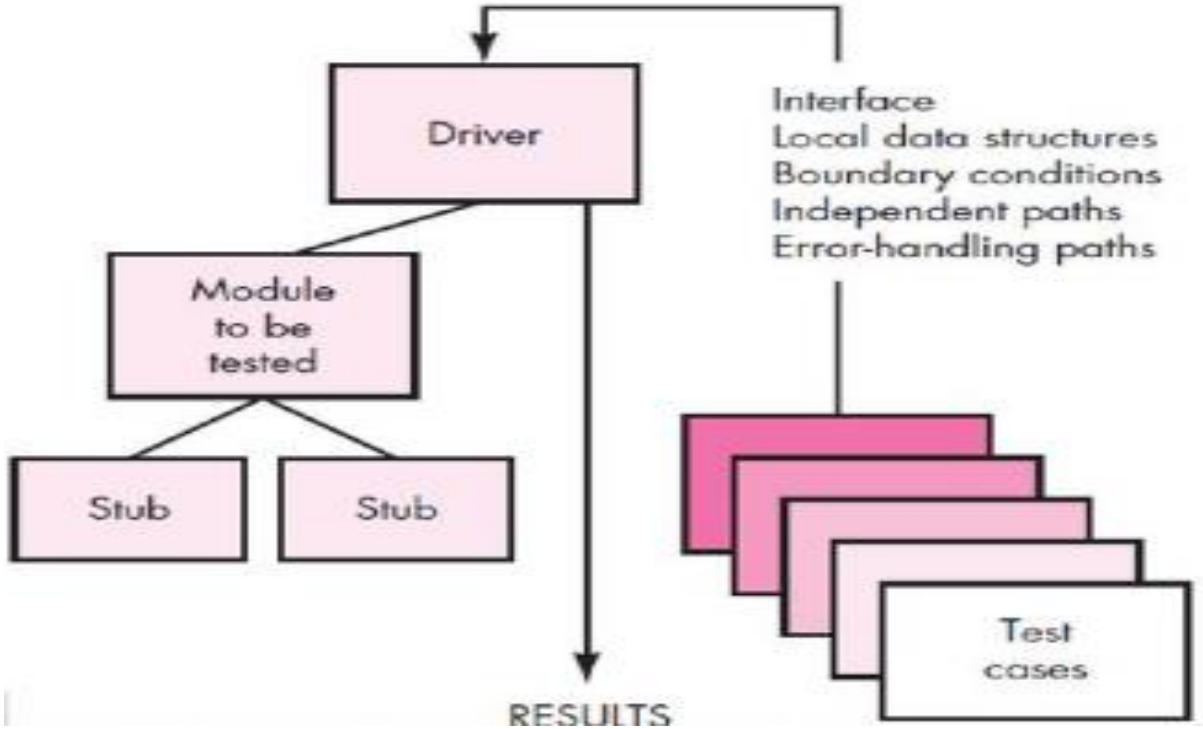


Unit Testing

- In old traditional testing we first constructed whole system then tested. But, we may get result as buggy software.
- Unit testing focuses verification effort on the smallest unit of software design – the software component or module.
- **Testing is apply on internal processing logic and data structure.** This type of testing can be conducted in parallel for multiple components.
- **The module interface** is tested to ensure that information properly flows into and out of the program unit under test.
- **Local data structures** are examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm's execution.
- **All independent paths** through the control structure are exercised to ensure that all statements in a module have been executed at least once.

- **Boundary conditions** are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.
- And finally, all **error-handling paths** are tested.

Unit Test Environment



Unit Test Environment

- Design unit tests before coding begins or after source code has been generated.
- Component-testing (Unit Testing) may be done in isolation from rest of the system
- In such case the missing software is replaced by Stubs and Drivers and simulate the interface between the software components in a simple manner.
- Let's take an example to understand it in a better way.
- Suppose there is an application consisting of three modules say, module A, module B & module C.
- Developer has design in such a way that module B depends on module A & module C depends on module B
- The developer has developed the module B and now wanted to test it.
- But the module A and module C has not been developed yet.
- In that case to test the module B completely we can replace the module **A** by Driver and module C by stub.

Driver:

- Main program.
- It accepts test case data
- Passes such data to the component
- Accepts the test data and prints the relevant results.
- Used in Bottom up approach
- Lowest modules are tested first.
- Simulates the higher level of components
- Dummy program for Higher level component

Stub(dummy subprogram):

- Subprogram.
- It uses the module interfaces and perform the minimal data manipulation if required.
- Prints verification of entry
- Returns control to the module undergoing testing
- Used in Top down approach
- Top most module is tested first
- Simulates the lower level of components
- Dummy program of lower level components

(B)Integration Testing

- A group of dependent components are tested together to ensure their integration unit's quality.
- The objective is to take unit tested components and build a program structure that has been directed by software design.
- Focus of integration testing is to uncover errors in:
- 1)Design and construction of software architecture
- 2)Integrated functions or operations at subsystem level
- 3)Resource integration and/or environmental integration

Integration Testing

- Integration testing carried out using two approaches.

1.non incremental approach:

- Big bang approach

2 incremental approach :

i)Top down testing

ii)Bottom up testing

iii)Regression testing

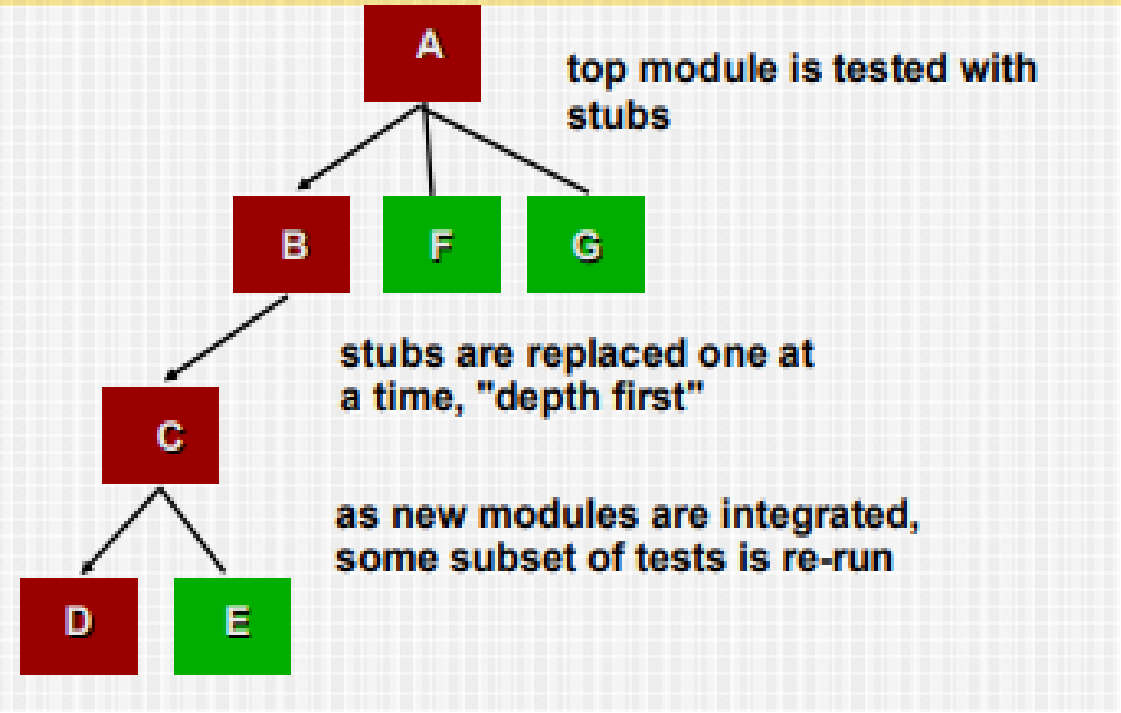
iv)Smoke testing

1.non incremental approach: Big bang approach

- All companies are combined in advance.
- The entire program is tested as a whole.
- A set of errors is tested as a whole.
- correction is difficult as isolation of causes is complicated by the size of entire program.
- Once errors are corrected new ones appear.
- This process continues infinitely.
- **Advantage:** Simple
- **Disadvantages:** Hard to debug,not easy to validate test results, after testing it is impossible to form an integrated system.

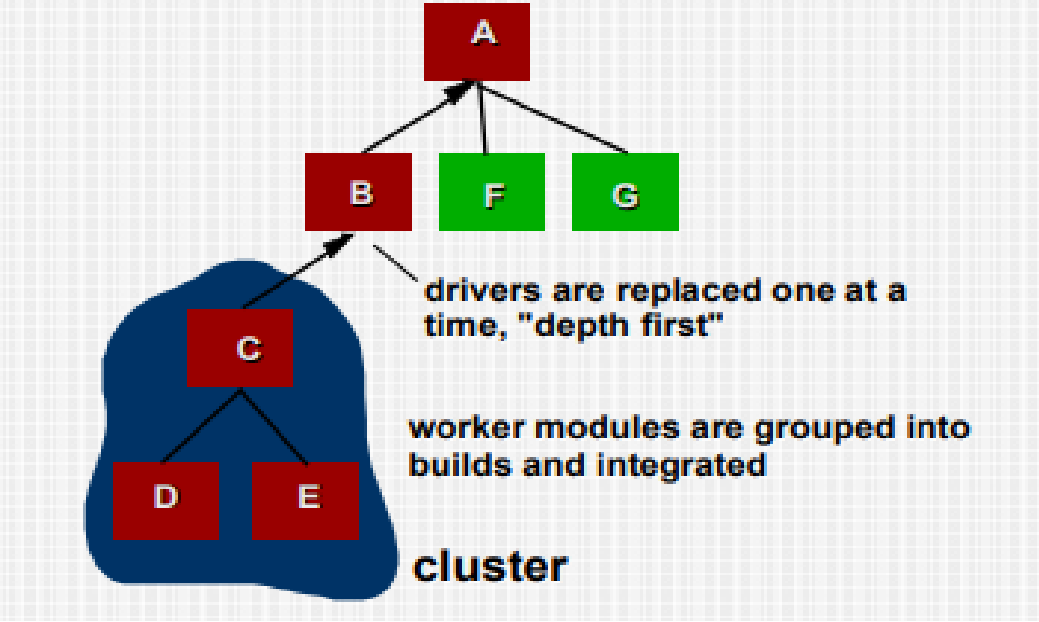
2 incremental approach :
i)Top down testing

- **Testing take place** from **top** to **bottom**
- **High level** modules are **tested first** and then low-level modules and **finally integrated** the **low level modules** to high level to ensure the system is working as intended
- **Stubs** are used as a temporary module, if a module is not ready for integration testing



2 incremental approach : ii)Bottom Up testing

- **Testing take place** from **bottom** to **up.**
- **Lowest level** modules are **tested first** and then high-level modules and finally integrated the high level modules to low level to ensure the system is working as intended.
- **Drivers** are used as a temporary module, if a module is not ready for integration testing.



2 incremental approach :
iii) Regression testing

- Regression testing is the re-execution of some subset of tests that have already been conducted to ensure that changes have not propagated unintended side effects.
- Whenever software is corrected, some aspect of the software configuration (the program, its documentation, or the data that support it) is changed.
- Regression testing helps to ensure that changes (due to testing or for other reasons) do not introduce unintended behaviour or additional errors.
- Regression testing may be conducted manually, by reexecuting a subset of all test cases or using automated capture/playback tools.
- **When to use?:**new functionalities are added to the application, change requirement, performance issue and environment change

2 incremental approach : iv)Smoke testing

- A common approach for creating "daily builds" for product software
- This test is performed after each Build Release
- Smoke testing verifies – Build Stability
- This testing is performed by "Tester" or "Developer"

This testing is executed for

- Integration Testing
- System Testing
- Acceptance Testing

What to Test?

- All **major and critical functionalities** of the application is tested.
- It **does not go into depth** to test each functionalities.
- This does not includes detailed testing for the build.

(C)Validation Testing

- The **process** of evaluating software to **determine** whether it **satisfies specified business requirements** (client's need).
- It **provides** final **assurance** that **software meets** all informational, functional, behavioral, and performance **requirements**
- When **custom software** is build for one customer, a **series of acceptance tests** are conducted to validate all requirements
- It is **conducted** by **end user** rather then software engineers
- If **software** is developed as **a product** to be **used** by **many customers**, it is impractical to perform formal acceptance tests with each one
- Most software product builders use a process called **alpha** and **beta testing** to **uncover errors** that only the end user seems able to find

Validation testing: Acceptance Testing

- It is a **level of** the software **testing** where a **system is tested for acceptability**.
- It ensures that the software works correctly in the user work environment.
- It is a formal testing conducted to **determine** whether or not a **system satisfies the acceptance criteria** with respect to user needs, requirements, and business processes.
- It enables the customer to determine, whether or not to accept the system.
- The acceptance testing can be conducted over a period of weeks or months.

Acceptance Testing – Alpha & Beta Test

- **Alpha Test**
- The alpha test is conducted by customer under the supervision of developer.
- The testing is performed at developer's site.
- The software is used in natural setting in presence of developer.
- The alpha tests are conducted in a controlled environment.
- **Beta Test**
- The beta test is **conducted** by customer without the developer being present.
- The test is performed at customer 's site.
- As there is no presence of developer during testing, it is not controlled by developer.
- The end user records the problems and report them to developer.
- The developer then makes appropriate modification.

Alpha Testing	Beta Testing
Alpha testing performed by Testers who are usually internal employees of the organization	Beta testing is performed by Clients or End Users who are not employees of the organization
Alpha Testing performed at developer's site	Beta testing is performed at a client location or end user of the product
Reliability and Security Testing are not performed in-depth Alpha Testing	Reliability, Security, Robustness are checked during Beta Testing
Alpha testing involves both the white box and black box techniques	Beta Testing typically uses Black Box Testing
Alpha testing requires a lab environment or testing environment	Beta testing doesn't require any lab environment or testing environment. The software is made available to the public and is said to be real time environment
Long execution cycle may be required for Alpha testing	Only a few weeks of execution are required for Beta testing
Critical issues or fixes can be addressed by developers immediately in Alpha testing	Most of the issues or feedback is collected from Beta testing will be implemented in future versions of the product
Alpha testing is to ensure the quality of the product before moving to Beta testing	Beta testing also concentrates on the quality of the product, but gathers users input on the product and ensures that the product is ready for real time users.

(D)System Testing

- In system testing the software and other system elements are tested.
- To test computer software, you spiral out in a clockwise direction along streamlines that increase the scope of testing with each turn.
- System testing verifies that all elements mesh properly and overall system function/performance is achieved.
- System testing is actually a series of different tests whose primary purpose is to fully exercise the computer-based system.

Types of System Testing

- Recovery Testing
- Security Testing
- Stress Testing
- Performance Testing
- Deployment Testing

Types of System Testing

- Recovery Testing:This testing determines whether operations can be continued after a disaster or after the integrity of the system has been lost.
- The best example of this supposes we are downloading one file and suddenly connection goes off. After resuming connection our downloading starts at where we left. It does not start from starting again.
- This used where continuity of the operations is essential.
- If recovery is automatic (performed by the system itself)
  - Re-initialization, check pointing mechanisms, data recovery, and restart are evaluated for correctness.
- If recovery requires human intervention
  - **The mean-time-to-repair (MTTR) is evaluated to determine whether it is within acceptable limits.**

Types of System Testing

- **Security Testing:**
- It is used to verify software's protection mechanisms, that protect it from improper penetration (access) or data alteration.
- During this test, the tester plays the role of the individual who desires to penetrate the system.
- **The goal is to make penetration attempt more costly than the value of the information that will be obtained.**
- **Stress Testing:**
- **Determines breakpoint of system to establish maximum service level.**
- It executes a system in a manner that demands resources in abnormal quantity, frequency or volume.
- A variation of stress testing is a technique called sensitivity testing.
- It is a testing in which it is tried to uncover data from a large class of valid data that may cause instability or improper processing.

Types of System Testing

- **Performance Testing:**
- It is designed to test the run-time performance of software.
- Here resource utilization such as **CPU load, throughput, response time, memory usage can be measured**.
- It occurs throughout all steps in the testing process.
- Even at the unit testing level, the performance of an individual module may be tested.
- **Beta testing is useful for performance testing**.
- For big system (banking) involving many users connecting to servers performance testing is very difficult.

Software testing fundamentals

- **Testability**: Its enables individual to design test case with ease.
- **Operability**: If software works properly then the testing can be conducted more efficiently.
- **Observability**: Ability to see what is being tested. Distinct outputs should be generated for each input values.
- **Controllability**: If we control software properly then the testing can be automated and optimized very well.
- **Decomposability**: Controlling the scope of testing means finding the cause of the problem quickly and once it is done, smarter retesting can be done.
- **Simplicity**: some coding standard can be used, minimum necessity of function requirements should satisfy, use of partition in architecture to stop error propagation.
- **Stability**: If less changes and modifications are there, then the disruption to testing will be minimized. No frequent changes are allowed ,recovers from all types of failure.
- **Understandibility**: The design of program is well understood by the user.

Testing conventional approach

- There two general approaches for the software testing
- 1)Black box Testing 2) White box Testing
- **1)Black box Testing**
- It is also known as behavioral testing.
- It focuses on the functional requirements of the software.
- It is tested weather the input is accepted properly and output is correctly produced.
- Its not alternative of white box testing.
- It is a complementary method that detects a different class of errors compared to white-box testing methods.

Black box testing detect following types of errors:

- The performance errors/ behavior errors.
- Missing or incorrectly defined functions.
- Data structures incorrectly defined and external data base access errors.
- Errors occurred during initialization and termination.
- Errors occurred during interface.

Types of Black box testing:

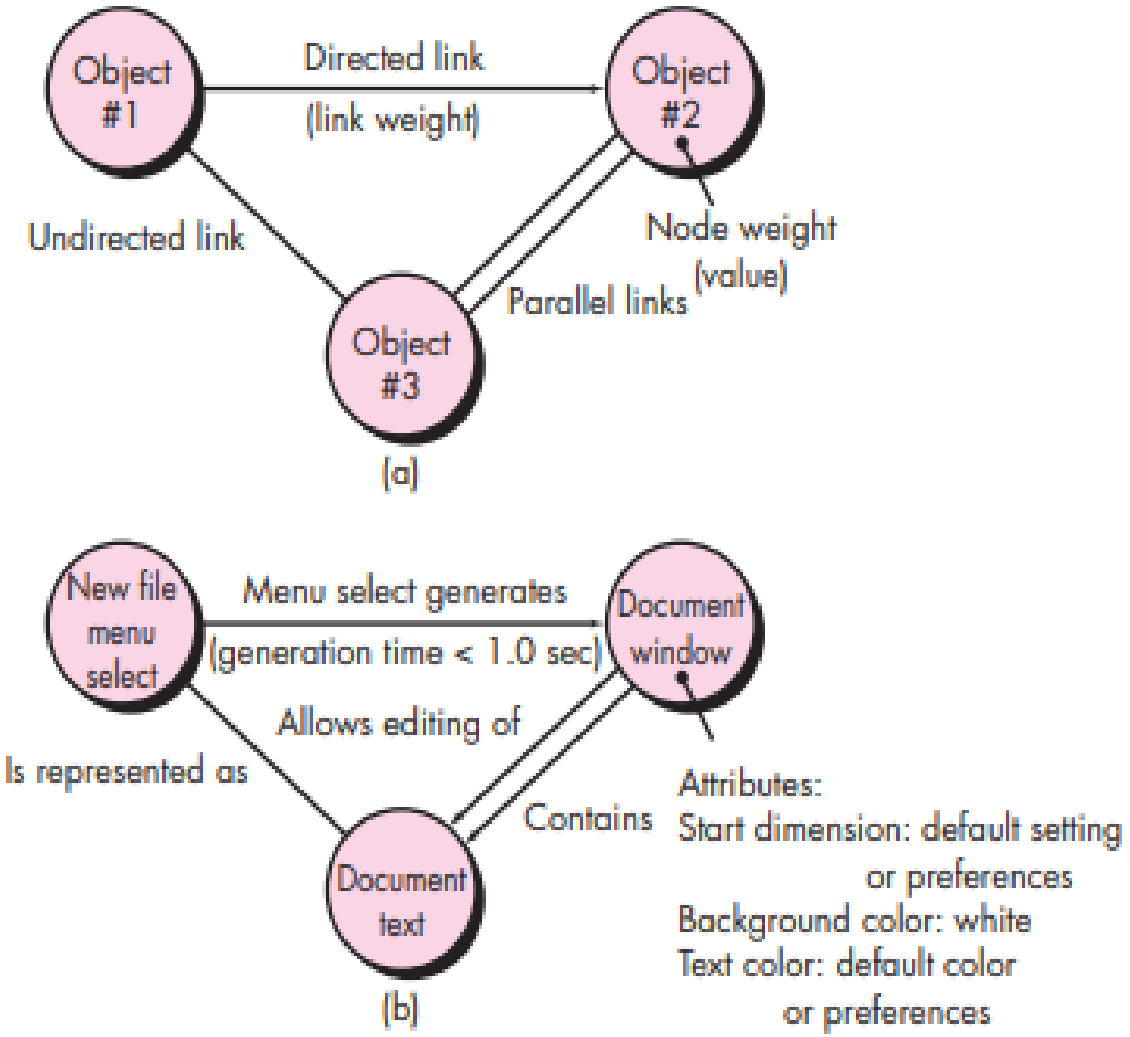
- 1.Graph based testing method
- 2.Equvivalence partitioning
- 3.Boundry Value Analysis
- 4.Orthogonal Array Testing

Black box Testing
1.Graph based testing method

- The first step in black-box testing is to understand the objects that are modelled in software and the relationships that connect these objects.
- Once this has been accomplished, the next step is to define a series of tests that verify "all objects have the expected relationship to one another.
- Stated in another way, software testing begins by creating a graph of important objects and their relationships and then devising a series of tests that will cover the graph so that each object and relationship is exercised and errors are uncovered.

1.Graph based testing method (continue..)

- How to create a graph—a collection of nodes that represent objects, links that represent the relationships between objects, node weights that describe the properties of a node (e.g., a specific data value or state behaviour), and link weights that describe some characteristic of a link.
- In figure a, nodes are represented as circles connected by links that take a number of different forms. A directed link (represented by an arrow) indicates that a relationship moves in only one direction.
- A bidirectional link, also called a symmetric link, implies that the relationship applies in both directions. Parallel links are used when a number of different relationships are established between graph nodes.



(a) Graph notation; (b) simple example

- As a simple example, consider a portion of a graph for a word-processing application (Figure b) where
- Object #1 newFile (menu selection)
- Object #2 documentWindow
- Object #3 documentText
- Referring to the figure, a menu select on new File generates a document window.
- The node weight of document Window provides a list of the window attributes that are to be expected when the window is generated.
- The link weight indicates that the window must be generated in less than 1.0 second.
- An undirected link establishes a symmetric relationship between the newFile menu selection and documentText, and parallel links indicate relationships between documentWindow and documentText.

Black box Testing
2.Equvivalence partitioning

- Input data for a program unit usually falls into a number of partitions, e.g. all negative integers, zero, all positive numbers
- Each partition of input data makes the program behave in a similar way
- Two test cases based on members from the same partition is likely to reveal the same bugs
- By identifying and testing *one member of each partition* we gain *'good'* coverage with *'small'* number of test cases.
- An equivalence class represents a set of valid or invalid states for input conditions. Typically, an input condition is either a specific numeric value, a range of values, a set of related values, or a Boolean condition.

- Equivalence classes may be defined according to the following guidelines:
- 1. If an input condition specifies a range, one valid and two invalid equivalence classes are defined.
- 2. If an input condition requires a specific value, one valid and two invalid equivalence classes are defined.
- 3. If an input condition specifies a member of a set, one valid and one invalid equivalence class are defined.
- 4. If an input condition is Boolean, one valid and one invalid class are defined.
- By applying the guidelines for the derivation of equivalence classes, test cases for each input domain data item can be developed and executed. T

2.Equvivalence partitioning (continue..)

- Example: **for binary search** the **following partitions** exist
  - **Inputs** that *conform* to *pre-conditions*
  - **Inputs** where the *precondition* is *false*
  - **Inputs** where the *key element* is *a member* of the *array*
  - **Inputs** where the *key element* is *not a member* of the *array*
- **Pick specific conditions** of the array
  - The array has a single value
  - Array length is even
  - Array length is odd

Example - Equivalence Partitioning

- Example: Assume that we have to test field which accepts SPI (Semester Performance Index) as input (SPI range is 0 to 10)

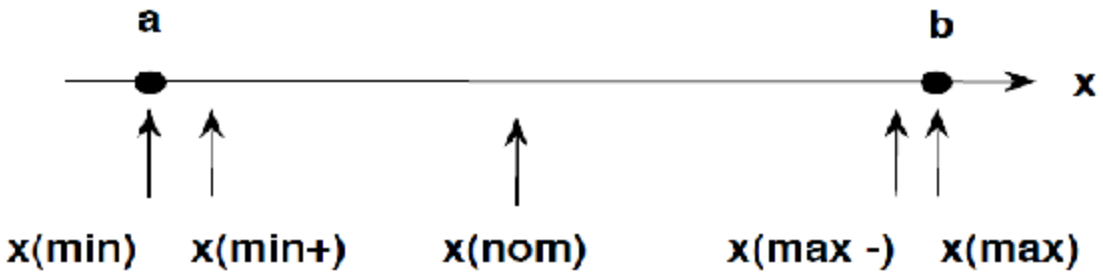
SPI [ ] * Accepts value 0 to 10

Equivalence Partitioning		
Invalid	**Valid**	**Invalid**
<=-1	0 to 10	>=11

Valid Class: 0 – 10, pick any one input test data from 0 to 10
Invalid Class 1: <=-1, pick any one input test data less than or equal to -1
Invalid Class 2: >=11, pick any one input test data greater than or equal to 11

Black box Testing
3.Boundry Value Analysis

- Boundary testing is the process of testing between extreme ends or boundaries between partitions of the input values.
- So these extreme ends like Start- End, Lower- Upper, Maximum-Minimum, Just Inside-Just Outside values are called boundary values and the testing is called "boundary testing".
- The basic idea in boundary value testing is to select input variable values at their:
- Minimum
- Just above the minimum
- A nominal value
- Just below the maximum
- Maximum



Suppose system asks for "a number between **100** and **999 inclusive**"
The **boundaries** are **100** and **999**
We therefore **test for values**

99	100	101

Lower boundary

999	999	1000

Upper boundary

- **BVA – Advantages**
- The BVA is **easy to use and remember** because of the uniformity of identified tests and the automated nature of this technique.
- One can **easily control the expenses** made on the testing by controlling the number of identified test cases.
- BVA is the **best approach** in cases where the **functionality** of a software is based on numerous variables representing physical quantities.
- The technique **is best at user input troubles** in the software.
- The **procedure and guidelines are crystal clear** and easy when it comes to determining the test cases through BVA.
- The **test cases** generated through BVA are **very small**.
- **BVA - Disadvantages**
- This technique **sometimes fails** to test **all the potential input** values. And so, the **results are unsure**.
- The **dependencies** with **BVA are not tested between two inputs**.
- This technique **doesn't fit** well when it comes to **Boolean Variables**.
- It **only works** well with **independent variables** that depict quantity.

Black box Testing
4.Orthogonal Array Testing

- There are many applications in which the input domain is relatively limited.
- That is, the number of input parameters is small and the values that each of the parameters may take are clearly bounded.
- When these numbers are very small (e.g., three input parameters taking on three discrete values each), it is possible to consider every input permutation and exhaustively test the input domain.
- However, as the number of input values grows and the number of discrete values for each data item increases, exhaustive testing becomes impractical or impossible.
- **Orthogonal array testing can be applied to problems in which the input domain is relatively small but too large to accommodate exhaustive testing.**
- The orthogonal array testing method is particularly useful in finding region faults—an error category associated with faulty logic within a software component.

- To illustrate the difference between orthogonal array testing and more conventional "one input item at a time" approaches, consider a system that has three input items, X, Y, and Z. Each of these input items has three discrete values associated with it. There are $3^3 = 27$ possible test cases.

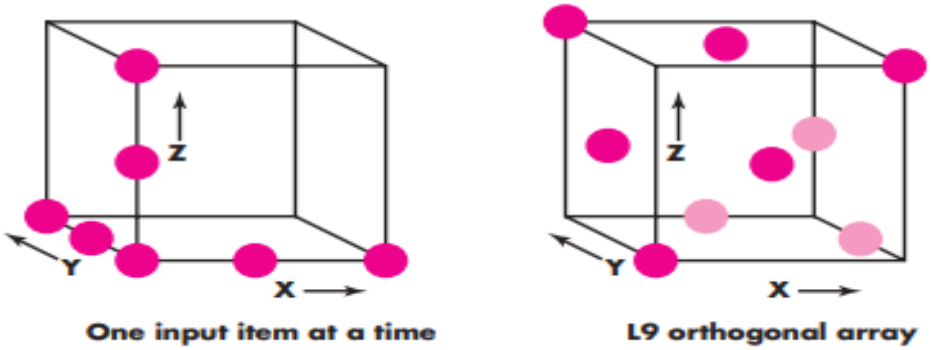


Figure c. Geometric view of test cases

- To illustrate the use of the L9 orthogonal array, consider the send function for a fax application.
- Four parameters, P1, P2, P3, and P4, are passed to the send function. Each takes on three discrete values.
- For example, P1 takes on values:
- P1= 1, send it now
- P1 = 2, send it one hour later
- P1= 3, send it after midnight
- P2, P3, and P4 would also take on values of 1, 2, and 3, signifying other send functions.
- If a "one input item at a time" testing strategy were chosen, the following sequence of tests (P1, P2, P3, P4) would be specified: (1, 1, 1, 1), (2, 1, 1, 1), (3, 1, 1, 1), (1, 2, 1, 1), (1, 3, 1, 1), (1, 1, 2, 1), (1, 1, 3, 1), (1, 1, 1, 2), and (1, 1, 1, 3).

- Such test cases are useful only when one is certain that these test parameters do not interact.
- They can detect logic faults where a single parameter value makes the software malfunction. These faults are called **single mode faults.**
- This method cannot detect logic faults that cause malfunction when two or more parameters simultaneously take certain values; that is, it cannot detect any interactions. Thus its ability to detect faults is limited.
- Given the relatively small number of input parameters and discrete values, exhaustive testing is possible.
- The number of tests required is $3^4 = 81$, large but manageable. All faults associated with data item permutation would be found, but the effort required is relatively high.

- The orthogonal array testing approach enables you to provide good test coverage with far fewer test cases than the exhaustive strategy.
- An L9 orthogonal array for the fax send function is illustrated in Figure d.

Test case	Test parameters			
	P1	P2	P3	P4
1	1	1	1	1
2	1	2	2	2
3	1	3	3	3
4	2	1	2	3
5	2	2	3	1
6	2	3	1	2
7	3	1	3	2
8	3	2	1	3
9	3	3	2	1

Figure d. L9 orthogonal array for fax function

- Phadke [Pha97] assesses the result of tests using the L9 orthogonal array in the following manner:
- **Detect and isolate all single mode faults.** A single mode fault is a consistent problem with any level of any single parameter.
- For example, if all test cases of factor P1 1 cause an error condition, it is a single mode failure. In this example tests 1, 2 and 3 [Figure d] will show errors. By analyzing the information about which tests show errors, one can identify which parameter values cause the fault. In this example, by noting that tests 1, 2, and 3 cause an error, one can isolate [logical processing associated with "send it now" (P1 1)] as the source of the error. Such an isolation of fault is important to fix the fault.
- **Detect all double mode faults.** If there exists a consistent problem when specific levels of two parameters occur together, it is called a double mode fault. Indeed, a double mode fault is an indication of pairwise incompatibility or harmful interactions between two test parameters.
- **Multimode faults.** Orthogonal arrays [of the type shown] can assure the detection of only single and double mode faults. However, many multimode faults are also detected by these tests.

- **2) White box Testing:**
- It is also known as glass box testing.
- The procedural details are closely examined in this testing.
- It test internals of software which make sure that they operate according to specifications and designs.
- Major focus of this testing is on internal structure, logic paths, control flow, data flows, internal data structures , conditions, loops etc.

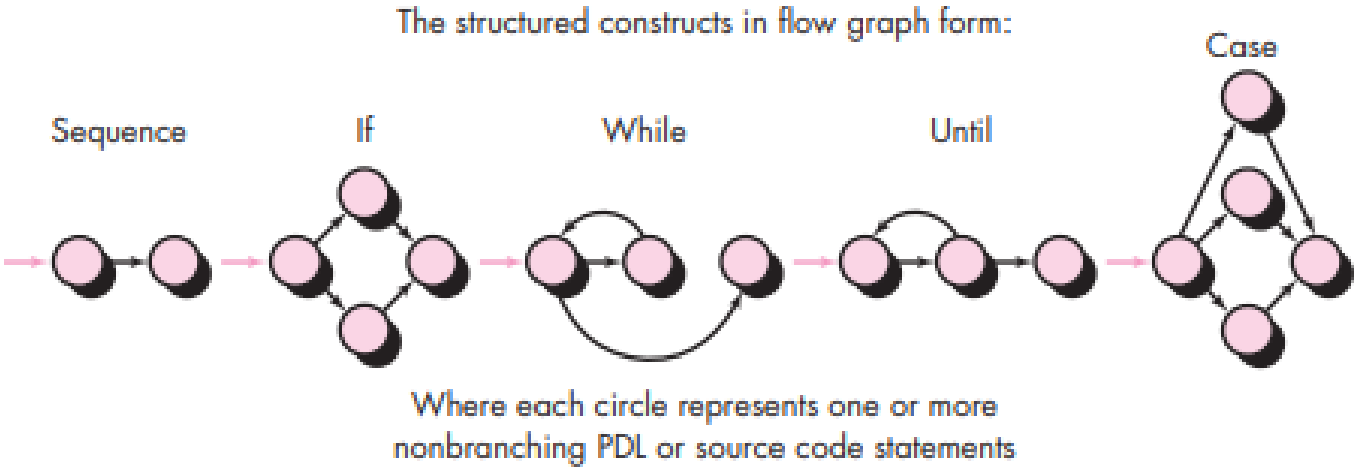
- **By using White box testing ,the developer can derive test cases that has the following characteristics:**
- It Guarantees the traversal of all independent paths in a module at least once. It refers the concept of tree traversal.
- Evaluate all logical decisions and find weather they are true or false.
- Evaluate all the loops to check their boundaries and operational bounds.
- Evaluate all internal data structures to confirm their validity.
- **Types of White box testing:**
- 1)Basis path testing
- 2) Control Structure testing

White box testing
1)Basis path testing

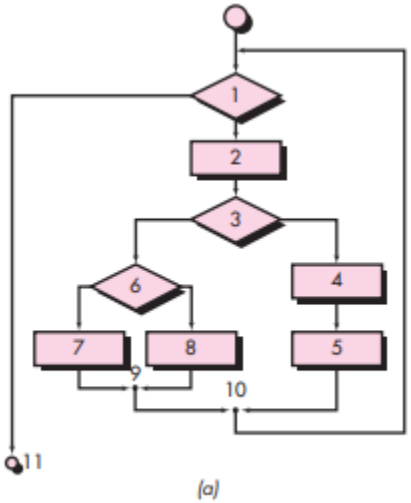
- Basis path testing is a white-box testing technique first proposed by Tom McCabe [McC76].
- The basis path method enables the test-case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths.
- Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.
- 1) Flow Graph Notation
- 2) Independent Program Paths
- 3) Deriving Test Cases
- 4) Graph Matrices

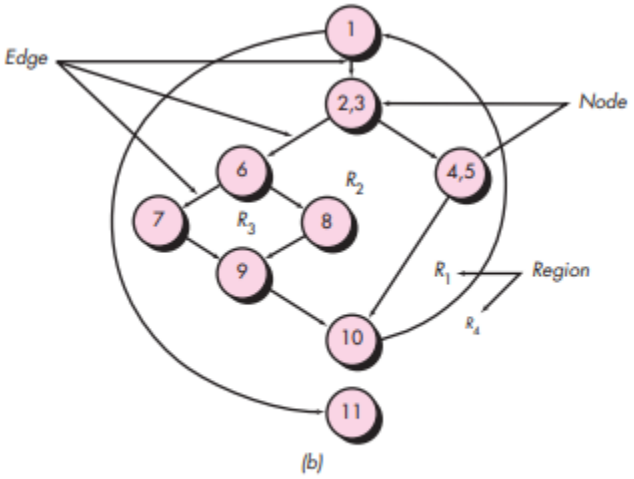
White box testing
1)Basis path testing (continue...)

- 1) Flow Graph Notation:
- Before we consider the basis path method, a simple notation for the representation of control flow, called a flow graph (or program graph). Following are flow graph notations.



1)Basis path testing (continue...)
Example of Flow Graph Notation





flow chart of program

flow graph.

- each circle, called a flow graph **node**, represents one or more procedural statements.
- A sequence of process boxes and a decision diamond can map into a single node.
- The arrows on the flow graph, called **edges** or **links**, represent flow of control and are analogous to flowchart arrows.
- An edge must terminate at a node, even if the node does not represent any procedural statements (e.g., see the flow graph symbol for the if-then-else construct).
- Areas bounded by edges and nodes are called **regions**. When counting regions, we include the area outside the graph as a region.

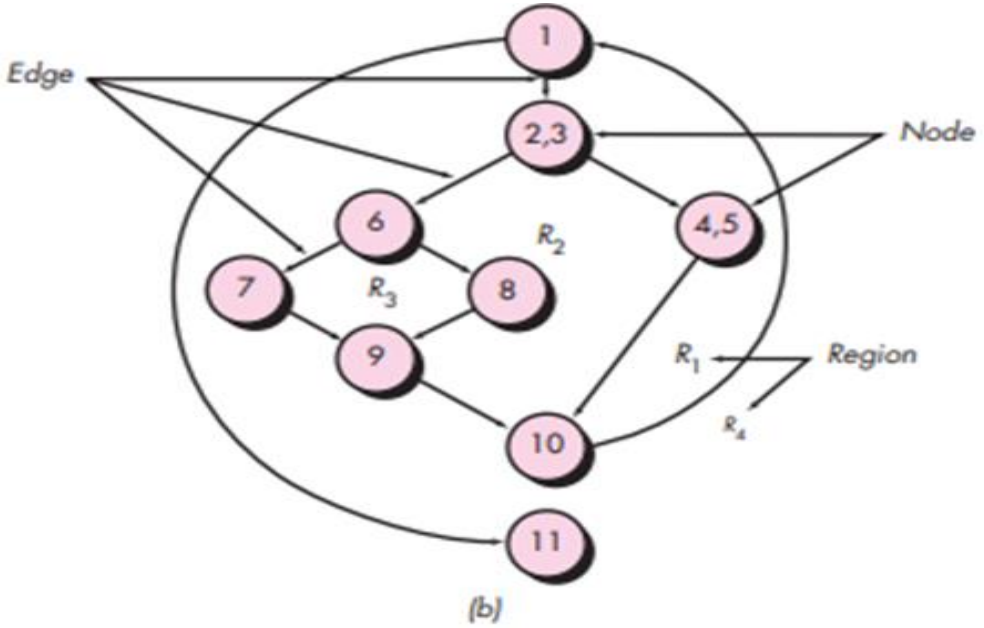
1)Basis path testing (continue...)

- 2) Independent Program Paths:
- An independent path is any path through the program that introduces at least one new set of processing statements or a new condition.
- When stated in terms of a flow graph, an independent path must move along at least one edge that has not been traversed before the path is defined.
- For example, a set of independent paths for the flow graph illustrated in Figure b is
- Path 1: 1-11
- Path 2: 1-2-3-4-5-10-1-11
- Path 3: 1-2-3-6-8-9-10-1-11
- Path 4: 1-2-3-6-7-9-10-1-11
- Note that each new path introduces a new edge. Paths 1 through 4 constitute a basis set for the flow graph.
- The path 1-2-3-4-5-10-1-2-3-6-8-9-10-1-11 is not considered to be an independent path because it is simply a combination of already specified paths and does not traverse any new edges.

- **Cyclomatic complexity** is a software metric that provides a quantitative measure of the logical complexity of a program.
- When used in the context of the basis path testing method, the value computed for cyclomatic complexity defines the number of independent paths in the basis set of a program and provides you with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.
- Complexity is computed in one of three ways:
- 1. The number of regions of the flow graph corresponds to the cyclomatic complexity.
- 2. Cyclomatic complexity V(G) for a flow graph G is defined as V(G) =E - N +2 where E is the number of flow graph edges and N is the number of flow graph nodes.
- 3. Cyclomatic complexity V(G) for a flow graph G is also defined as V(G) =P +1 where P is the number of predicate nodes contained in the flow graph G

Example

- 1.The flow graph has four regions.
- 2.V(G) =11 edges -9 nodes+2 = 4.
- 3.V(G) =3 predicate nodes +1= 4.



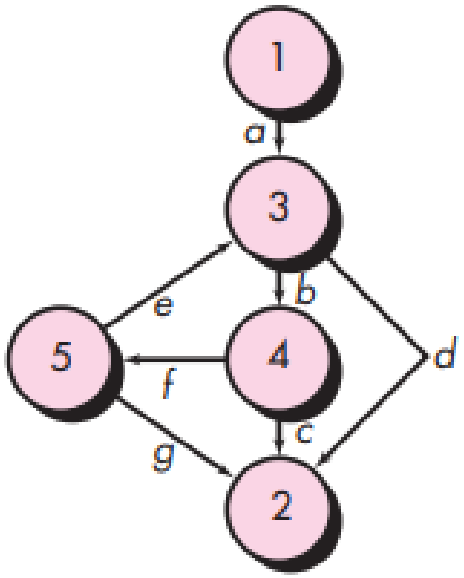
1)Basis path testing (continue...)

- 3) Deriving Test Cases
- The basis path testing method can be applied to a procedural design or to source code.
- The following steps can be applied to derive the basis set:
- 1. Using the design or code as a foundation, draw a corresponding flow graph.
- 2. Determine the cyclomatic complexity of the resultant flow graph.
- 3. Determine a basis set of linearly independent paths.
- 4. Prepare test cases that will force execution of each path in the basis set.

1)Basis path testing (continue...)

- 4) Graph Matrices
- The procedure for deriving the flow graph and even determining a set of basis paths is amenable to mechanization.
- A data structure, called a graph matrix, can be quite useful for developing a software tool that assists in basis path testing.
- A graph matrix is a square matrix whose size (i.e., number of rows and columns) 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 (an edge) between nodes.
- A simple example of a flow graph and its corresponding graph matrix is shown in Figure as shown in next slide.

1)Basis path testing (continue...)



Flow graph

Node \ Connected to node	1	2	3	4	5
1			a		
2					
3		d		b	
4		c			f
5		g	e		

Graph matrix

•Referring to the figure, each node on the flow graph is identified by numbers, while each edge is identified by letters.

•A letter entry is made in the matrix to correspond to a connection between two nodes. For example, node 3 is connected to node 4 by edge b.

1)Basis path testing (continue...)

- The graph matrix is nothing more than a tabular representation of a flow graph.
- However, by adding a link weight to each matrix entry, the graph matrix can become a powerful tool for evaluating program control structure during testing.
- The link weight provides additional information about control flow. In its simplest form, the link weight is 1 (a connection exists) or 0 (a connection does not exist). But link weights can be assigned other, more interesting properties:
- The probability that a link (edge) will be execute.
- The processing time expended during traversal of a link.
- The memory required during traversal of a link.
- The resources required during traversal of a link.

White box testing
2) Control Structure testing

- The basis path testing technique is one of a number of techniques for control structure testing.
- Although basis path testing is simple and highly effective, it is not sufficient in itself.
- The control structure testing broaden testing coverage and improve the quality of white-box testing.
- 1) Condition Testing
- 2) Data Flow Testing
- 3) Loop Testing

2) Control Structure testing(continue...)

1)Condition Testing

- It is a test-case design method that exercises the logical conditions contained in a program module.
- A simple condition is a Boolean variable or a relational expression, possibly preceded with one NOT (¬) operator.
- A relational expression takes the form

E1<Relational Operator> E2 (operators:=,!=,<,> ,And ,or,not)

- A condition without relational expressions is referred to as a Boolean expression.
- The condition testing method focuses on testing each condition in the program to ensure that it does not contain errors.

2) Control Structure testing(continue...)

- 2) Data Flow Testing
- The data flow testing method selects test paths of a program according to the locations of definitions and uses of variables in the program.
- To illustrate the data flow testing approach, assume that each statement in a program is assigned a unique statement number and that each function does not modify its parameters or global variables.
- For a statement with S as its statement number,
- DEF(S) ={X | statement S contains a definition of X}
- USE(S')= {X | statement S contains a use of X}

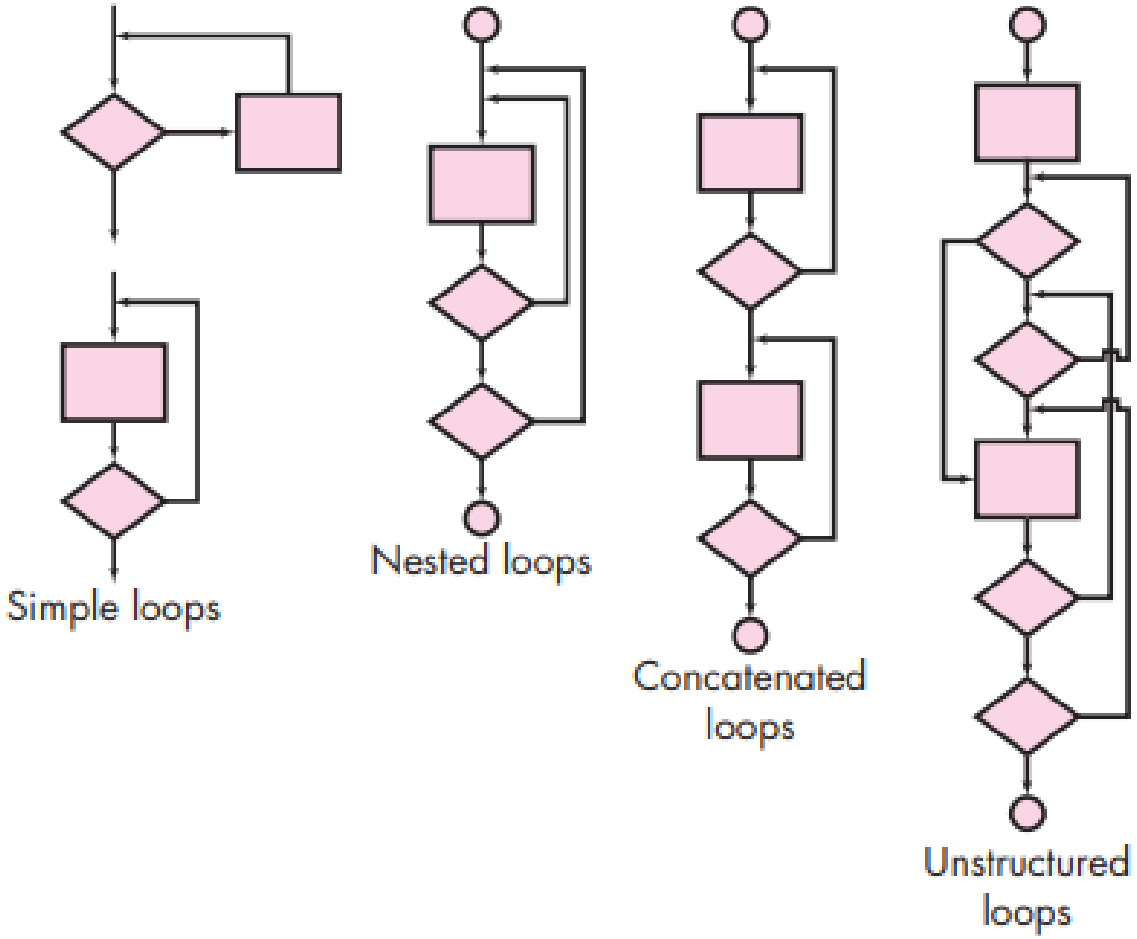
2) Control Structure testing(continue...)

- If statement S is an if or loop statement, its DEF set is empty and its USE set is based on the condition of statement S.
- The definition of variable X at statement S is said to be live at statement S' if there exists a path from statement S to statement.
- A definition-use (DU) chain of variable X is of the form [X, S, S'], where S and S' are statement numbers, X is in DEF(S) and USE(S'), and the definition of X in statement S is live at statement S'.S' that contains no other definition of X.
- One simple data flow testing strategy is to require that every DU chain be covered at least once.

2) Control Structure testing(continue...)

- 3) Loop Testing
- Loops are the cornerstone for the vast majority of all algorithms implemented in software. And yet, we often pay them little heed while conducting software tests.
- Loop testing is a white-box testing technique that focuses exclusively on the validity of loop constructs.
- 4 classes of loops:Simple loops, concatenated loops, nested loops, and unstructured loops .
- **i) Simple loops.** The following set of tests can be applied to simple loops, where n is the maximum number of allowable passes through the loop.
- 1. Skip the loop entirely.
- 2. Only one pass through the loop.
- 3. Two passes through the loop.
- 4. m passes through the loop where m < n.
- 5. n- 1, n, n +1 passes through the loop.

2) Control Structure testing(continue...)



- **ii) Nested loops:** If we were to extend the test approach for simple loops to nested loops, the number of possible tests would grow geometrically as the level of nesting increases. This would result in an impractical number of tests.
- suggests an approach that will help to reduce the number of tests: 1. Start at the innermost loop. Set all other loops to minimum values.
- 2. Conduct simple loop tests for the innermost loop while holding the outer loops at their minimum iteration parameter (e.g., loop counter) values. Add other tests for out-of-range or excluded values.
- 3. Work outward, conducting tests for the next loop, but keeping all other outer loops at minimum values and other nested loops to "typical" values.
- 4. Continue until all loops have been tested.

- **iii)Concatenated loops:**
- Concatenated loops can be tested using the approach defined for simple loops, if each of the loops is independent of the other.
- However, if two loops are concatenated and the loop counter for loop 1 is used as the initial value for loop 2, then the loops are not independent.
- **iv)Unstructured loops:**
- Whenever possible, this class of loops should be redesigned to reflect the use of the structured programming constructs.

Testing
Object Oriented Applications

Unit Testing in the OO Context

- The concept of the unit testing changes in object-oriented software
- **Encapsulation drives** the **definition** of **classes** and **objects**
  - Means, each class and each instance of a class (object) packages attributes (data) and the operations (methods or services) that manipulate these data
  - **Rather than testing an individual module**, the smallest **testable unit** is the **encapsulated class**
- **Unlike** unit testing of **conventional software**,
  - which focuses on the algorithmic detail of a module and the data that flows across the module interface,
  - **class testing** for OO software is **driven by** the **operations encapsulated** by the class and the **state behavior** of the class

Integration Testing in the OO Context

- Object-oriented software does not have a hierarchical control structure.
- conventional top-down and bottom-up integration strategies have little meaning.
- There are two different strategies for integration testing of OO systems.
- **Thread-based testing**
- integrates the set of classes required to respond to one input or event for the system
- Each thread is integrated and tested individually
- Regression testing is applied to ensure that no side effects occur

Integration Testing in the OO Context

- **Use-based testing**
- begins the construction of the system by testing those classes (called independent classes) that use very few (if any) of server classes.
- After the independent classes are tested, the next layer of classes, called dependent classes, that use the independent classes are tested.
- **Cluster testing** is one step in the integration testing of OO software.
- Here, a **cluster of collaborating classes** is exercised by designing test cases that attempt to uncover.

Validation Testing in an OO Context

- At the validation or system level, the details of class connections disappear.
- Like conventional validation, the validation of OO software focuses on user-visible actions and user-recognizable outputs from the system.
- To assist in the derivation of validation tests, the tester should draw upon use cases that are part of the requirements model
- Conventional black-box testing methods can be used to drive validation tests.

Testing Web Applications

Testing Web Applications

- WebApp testing is a collection of related activities with a single goal to uncover errors in WebApp content, function, usability, navigability, performance, capacity, and security
- To accomplish this, a testing strategy that encompasses both reviews and executable testing is applied.
- **Content** is **evaluated** at both a **syntactic** and **semantic** level.
- At the **syntactic** level **spelling**, **punctuation**, and **grammar** are assessed for text-based documents.
- At a **semantic** level **correctness of information** presented, **Consistency** across the entire content object and related objects, and **lack of ambiguity** are all assessed.
- **Function** is tested to uncover errors that indicate **lack of conformance** to **customer requirements**
- **Structure** is **assessed** to ensure that **it properly delivers** WebApp **content**
- **Usability** is **tested** to ensure that **each category of user** is **supported** by the interface and can learn and apply all required navigation.

- **Navigability** is **tested** to ensure that **all navigation syntax** and **semantics** are **exercised** to uncover any **navigation errors**
  - Ex., *dead links*, *improper links,* and *erroneous links*
- **Performance** is **tested** under a variety of **operating conditions**, **configurations** and **loading**
  - to **ensure** that the **system** is **responsive** to **user interaction** and handles extreme loading
- **Compatibility** is tested by **executing** the WebApp in a **variety** of different **host configurations** on both the client and server sides
- **Interoperability** is tested to ensure that the WebApp **properly interfaces** with other applications and/or databases
- **Security** is tested by assessing potential vulnerabilities

Content Testing

- Errors in WebApp content can be :
- 1) as trivial as minor typographical errors or
- 2)as significant as incorrect information, improper organization, or violation of intellectual property laws.
- Content testing attempts to uncover these and many other problems before the user encounters them.
- Content testing combines both reviews and the generation of executable test cases.
- Reviews are applied to uncover semantic errors in content.
- Executable testing is used to uncover content errors that can be traced to dynamically derived content that is driven by data acquired from one or more databases.

User Interface Testing

- Verification and validation of a WebApp user interface occurs at three distinct points
- **During requirements analysis:**
- the interface model is reviewed to ensure that it conforms to stakeholder requirements
- **During design :**
- the interface design model is reviewed to ensure that generic quality criteria established for all user interfaces have been achieved
- **During testing:**
- the focus shifts to the execution of application-specific aspects of user interaction as they are manifested by interface syntax and semantics.
- In addition, testing provides a final assessment of usability

Component-Level Testing

- Component-level testing (function testing), focuses on a set of tests that attempt to uncover errors in WebApp functions.
- Each WebApp function is a software component (implemented in one of a variety of programming languages)
- WebApp function can be tested using black-box (and in some cases, white-box) techniques.
- Component-level test cases are often driven by forms-level input.
- Once forms data are defined, the user selects a button or other control mechanism to initiate execution.

Navigation Testing

- The job of **navigation testing** is to ensure that
  - the **mechanisms** that allow the WebApp user **to travel through** the WebApp are **all functional** and,
  - to **validate** that each **Navigation Semantic Unit (NSU)** can be achieved by the appropriate user category
- **Navigation mechanisms** should be **tested** are
  - Navigation links,
  - Redirects,
  - Bookmarks,
  - Frames and framesets,
  - Site maps,
  - Internal search engines.

Configuration Testing

- Configuration variability and instability are important factors that make WebApp testing a challenge.
- Hardware, operating system(s), browsers, storage capacity, network communication speeds, and a variety of other client-side factors are difficult to predict for each user.
- One user's impression of the WebApp and the manner in which he/she interacts with it can differ significantly.
- Configuration testing is to test a set of probable client-side and server-side configurations
  - to ensure that the user experience will be the same on all of them and,
  - to isolate errors that may be specific to a particular configuration

Security Testing

- Security tests are designed to probe
  - vulnerabilities of the client-side environment,
  - the network communications that occur as data are passed from client to server and back again, and
  - the server-side environment.
- Each of these domains can be attacked, and it is the job of the security tester to uncover weaknesses
  - that can be exploited by those with the intent to do so.

Performance Testing

- Performance testing is used to uncover
- performance problems that can result from lack of server-side resources,
- inappropriate network bandwidth,
- inadequate database capabilities,
- faulty or weak operating system capabilities,
- poorly designed WebApp functionality, and
- other hardware or software issues that can lead to degraded client-server performance

Testing tools(Load runner, Win Runner)

- **LOADRUNNER** is a Performance Testing tool which was pioneered by Mercury in 1999.
- It is used to test applications, measuring system behaviour and performance under load.
- LoadRunner can simulate thousands of users concurrently using application software, recording and later analyzing the performance of key components of the application.
- LoadRunner simulates user activity by generating messages between application components or by simulating interactions with the user interface such as keypresses or mouse movements.
- The messages and interactions to be generated are stored in scripts. LoadRunner can generate the scripts by recording them, such as logging HTTP requests between a client web browser and an application's web server.

- The key components of LoadRunner are:
- **Load Generator** generates the load against the application by following scripts
- **VuGen** (Virtual User Generator) for generating and editing scripts
- **Controller** controls, launches and sequences instances of Load Generator - specifying which script to use, for how long etc. During runs the Controller receives real-time monitoring data and displays status.
- **Agent process** manages connection between Controller and Load Generator instances.
- **Analysis** assembles logs from various load generators and formats reports for visualization of run result data and monitoring data.

- **Advantages:**
- No need to install it on the server under test. It uses native monitors. For Ex: perfmon for windows or rstatd daemon for Unix
- Uses ANSI C as the default programming language[1] and other languages like Java and VB.
- Excellent monitoring and analysis interface where you can see reports in easy to understand colored charts and graphics.
- Supports most of the protocols[2].
- Makes correlation[3] much easier. We will dig into correlation through a series of posts later.
- Nice GUI generated script through a one click recording, of course you would need to modify the script according to your needs.
- Excellent tutorials, exhaustive documentation and active tool support from HP.
- **Disadvantages:**
- The only disadvantage I can think is the prohibitive cost associated with the tool but that can also be compensated in the long run when you start getting a good ROI from the tool.
- Programming/Scripting language is used to represent the captured protocol data and manipulate the data for play-back.
- Protocol is simply a language that your client uses to communicate with the system.
- Correlation is a way to substitute values in dynamic data to enable successful playback.

- WIN RUNNER:
- **WinRunner** software was an automated functional GUI testing tool that allowed a user to record and play back user interface (UI) interactions as *test scripts*.
- As a functional test suite, it worked with HP QuickTest Professional and supported enterprise quality assurance.
- It captured, verified and replayed user interactions automatically, in order to identify defects and determine whether business processes worked as designed.
- The software implemented a proprietary Test Script Language (TSL) that allowed customization and parameterization of user input.
- HP WinRunner was originally written by Mercury Interactive.

- **Advantages of WinRunner**
- The Winrunner was initially a creation of the mercury interactive but later was taken under the supervision of HP (Hewett Packard).
- - With winrunner, the quick creation of the sophisticated automated tests has been made possible.
- - With the aid of winrunner you have the provision of automating the whole software testing process right from the test development to the execution part. - The tests that we develop or create using the winrunner are not only automated but they are also by far adaptable as well as reusable.
- - The test scripts that they consist prove to be quite challenging to the software system or application which is under testing.
- - The winrunner also provides you with the convenience of running all the tests you created in an overnight just before the release date of your software product.
- - Such an overnight test run helps you to detect any last moment defects and also ensures that your software product is of superior quality.
- ***Testing with winrunner reduces the following problems dramatically:***
- Reduces time consumption and tediousness.
- Reduces the amount of investment in human resources.
- Roots out most of the serious bugs.
- Winrunner is a functional testing tool that was designed with the aim of solving quite a large number of problems and it does assists the process of software testing in quite a realistic manner.
- With winrunner you get an opportunity for testing your software systems and applications on quite a large scale or we can say company wide scale.
- The user functions are verified as well as recorded in an automated way which makes sure the proper and cooperative working of the business processes.

- **Limitations ofWinRunner**
- - Winrunner because of its limitations has now been superseded by the HP quick test pro.
- - The winrunner makes use of the source of the program to create test cases and moreover the coding of the test cases is done in test script language (TSL) which is a proprietary language.
- - One of the disadvantages of the TSL is that very few resources are available for TSL programming.
- -The testers need immense training to be able to implement winrunner properly.
- - Another disadvantage is the GUI map, which can be many a time quite difficult to understand as well as implement.
- ***The winrunner does not extends support for the following applications:***
- Few web based applications
- NET applications as well.
- XML
- Multimedia applications
- Sap apps
- Oracle apps

References

- Win runner: https://www.youtube.com/watch?v=gclvv8H85Ug
- Load runner:
- https://www.guru99.com/introduction-to-hp-loadrunner-and-its-archtecture.html
-

Assignment 6

Sr. No.	Questions
1	What is software testing? Explain different techniques of testing.
2	Differentiate Alpha and Beta testing.
3	Describe coding standards.
4	Explain White Box (Glass Box) testing. Explain any one technique to carry out each testing.
5	Explain Black Box testing. Explain any one technique to carry out each testing.
6	Explain: Unit testing ,cyclomatic complexity and Load testing.
7	How unit testing strategy works on a software module? What errors are commonly found during unit testing?
8	What is Cyclomatic Complexity? Define steps to find cyclomatic complexity using flow graph.
9	Compare White box and Black box testing.
10	Explain boundary value analysis (BVA).