B. TECH. IT 5TH SEM SOS(E & T), GGU, BILASPUR C G

27-09-2021

Elective-I Software Engineering(IT05TPE11)

By-Mrs. Akanksha Gupta

Unit-5 Software Testing Contents

- Software Testing Fundamental
- Black Box & White Box Testing
- Basic Path Testing
- A strategic issue
- Types of Testing-Unit Testing, Integration Testing, Validation Testing, System Testing
- Software Metric, Software evaluation
- Software maintenance & reliability

Software Testing Fundamental

- Testing Objective-
  - 1. Testing is a process of executing a program with the intent of finding an error.
  - 2. A good test case is one that has a high probability of finding an as-yet undiscovered error.
  - 3. A successful test is one that uncovers an as-yet-undiscovered error.

Testing Principles

- All tests should be traceable to customer requirements.
- Tests should be planned long before testing begins.
- The Pareto principle applies to software testing.
- Testing should begin “in the small” and progress toward testing “in the large.”

Testing Principles

- Exhaustive testing is not possible.
- To be most effective, testing should be conducted by an independent third party

Testability

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

Attributes of “Good” Test

- A good test has a high probability of finding an error.
- A good test is not redundant.
- A good test should be “best of breed”.
- A good test should be neither too simple nor too complex.

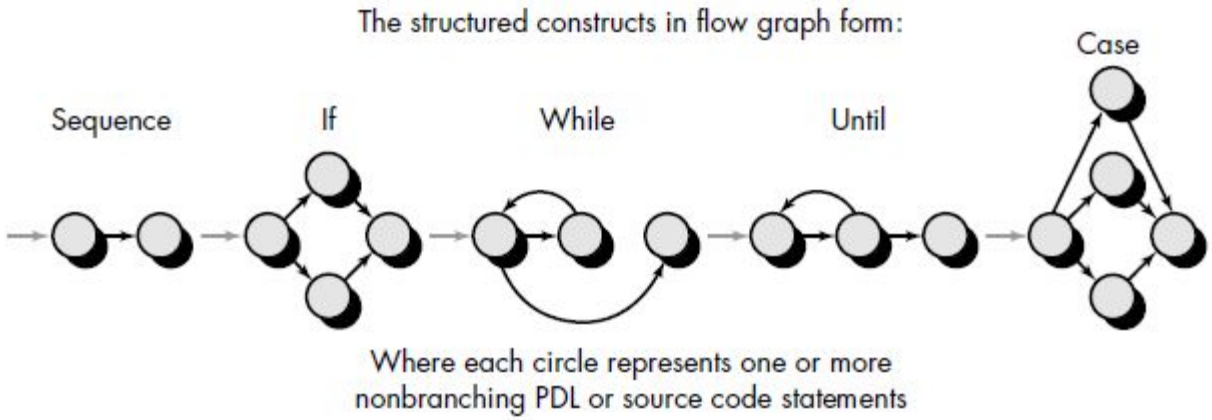
White Box Testing

- Using white-box testing methods, the software engineer can derive test cases that
  - (1) guarantee that all independent paths within a module have been exercised at least once,
  - (2) exercise all logical decisions on their true and false sides,
  - (3) execute all loops at their boundaries and within their operational bounds, and
  - (4) exercise internal data structures to ensure their validity.

Basic Path Testing

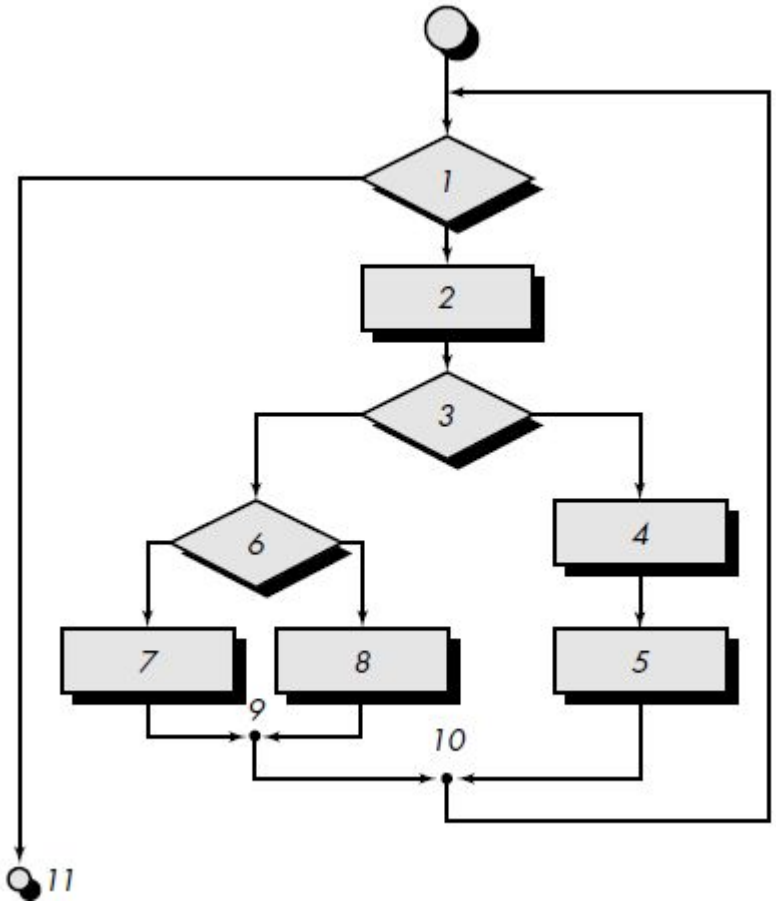
- *Basis path testing is a white-box testing technique*
- The basis path method enables the test case designer to derive a logical complexity measure of a procedural design and use this measure as a guide for defining a basis set of execution paths. Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time during testing.

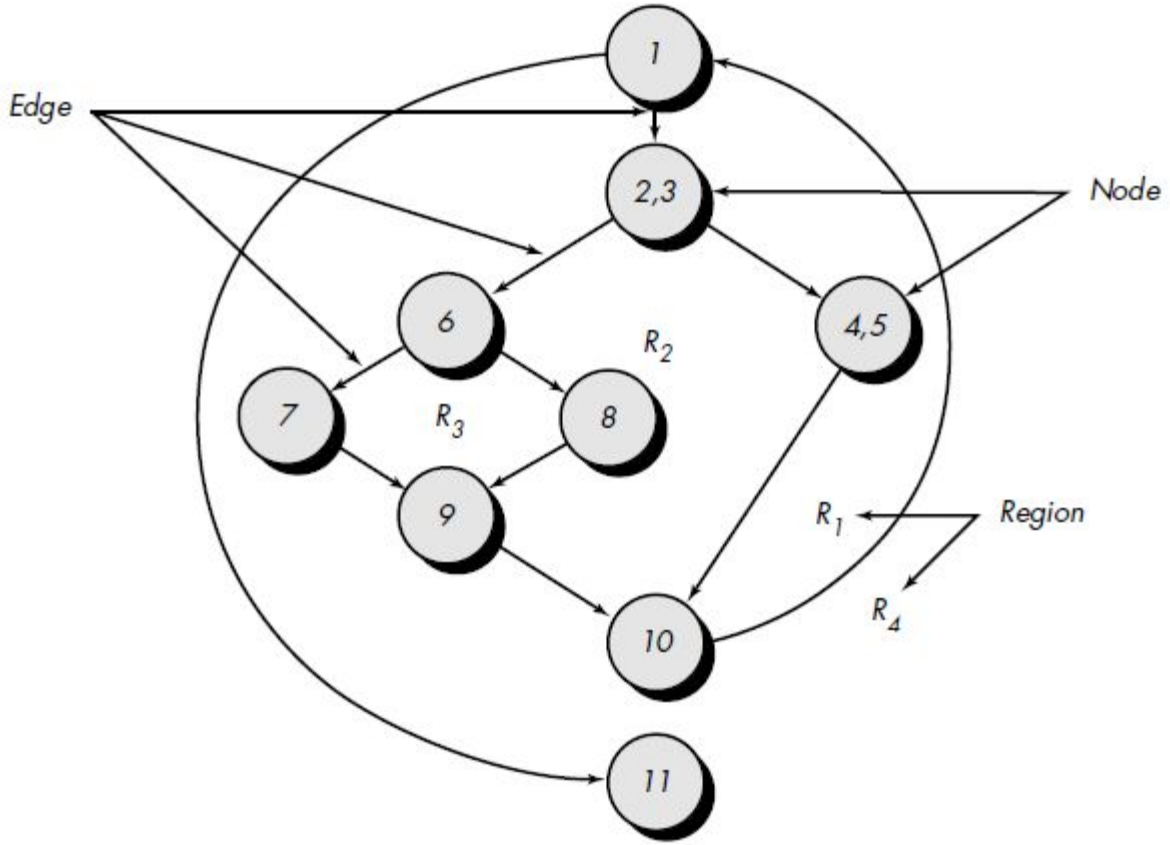
Flow Graph Notation

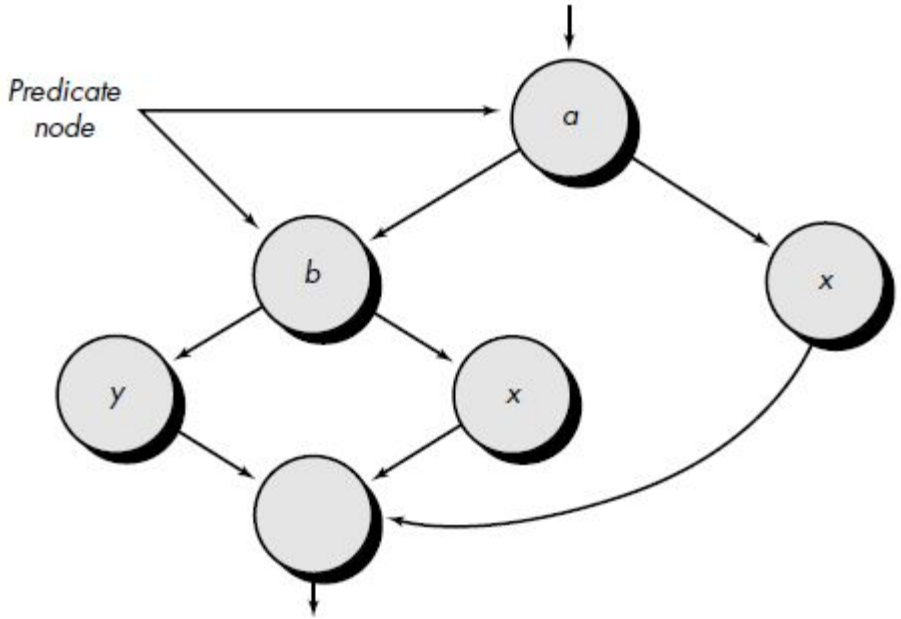


Cyclomatic Complexity

- Cyclomatic complexity defines the number of independent paths in the basis set of a program and provides us with an upper bound for the number of tests that must be conducted to ensure that all statements have been executed at least once.
- An *independent path is any path through the program that introduces at least one* new set of processing statements or a new condition.


1
2
3
6
4
7
8
5
9
10
11


1
Edge
2,3
Node
6
4,5
R_2
7
R_3
8
9
R_1
Region
R_4
10
11


Predicate
node
a
b
x
y
x

- 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 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. 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 computed in one of three ways:
- 1. The number of regions of the flow graph correspond 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, 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.*

- Referring once more to the flow graph in Figure 17.2B, the cyclomatic complexity can be computed using each of the algorithms just noted:

1. The flow graph has four regions.

2. *V(G) = 11 edges 9 nodes + 2 = 4.*

3. *V(G) = 3 predicate nodes + 1 = 4.*

- Therefore, the cyclomatic complexity of the flow graph in Figure is 4.

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.

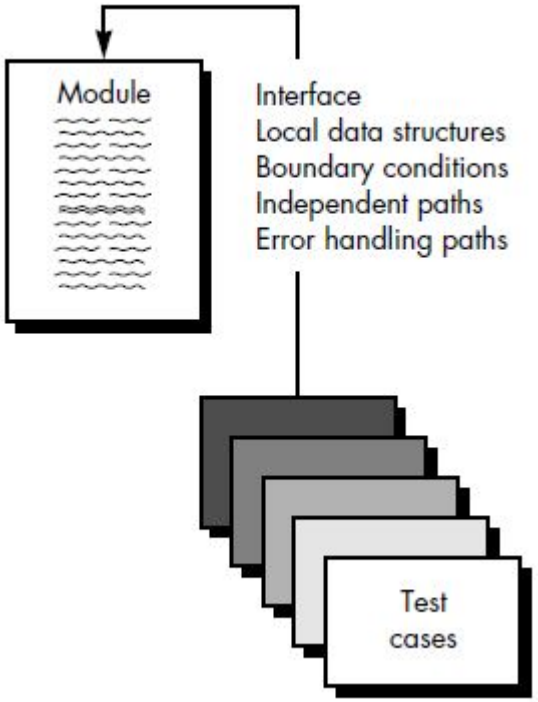
STRATEGIC ISSUES

- Use effective formal technical reviews as a filter prior to testing.
- Conduct formal technical reviews to assess the test strategy and test cases themselves.
- Develop a continuous improvement approach for the testing process.

Unit Testing

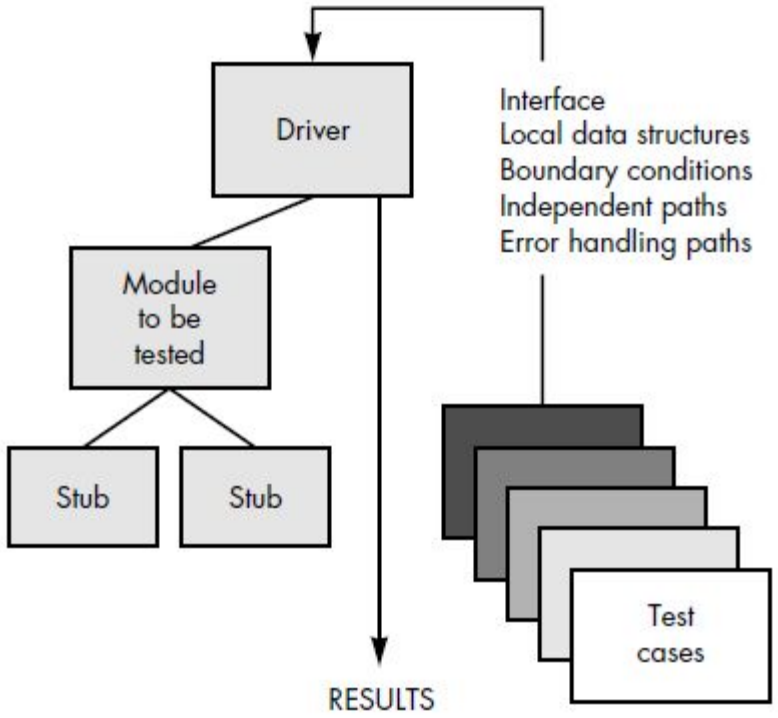
- Unit testing focuses verification effort on the smallest unit of software design—the software component or module.

Unit Testing Consideration



- Among the potential errors that should be tested when error handling is evaluated are -

  1. Error description is unintelligible.
  2. Error noted does not correspond to error encountered.
  3. Error condition causes system intervention prior to error handling.
  4. Exception-condition processing is incorrect.
  5. Error description does not provide enough information to assist in the location of the cause of the error.


Driver
Interface
Local data structures
Boundary conditions
Independent paths
Error handling paths
Module to be tested
Stub
Stub
Test cases
RESULTS

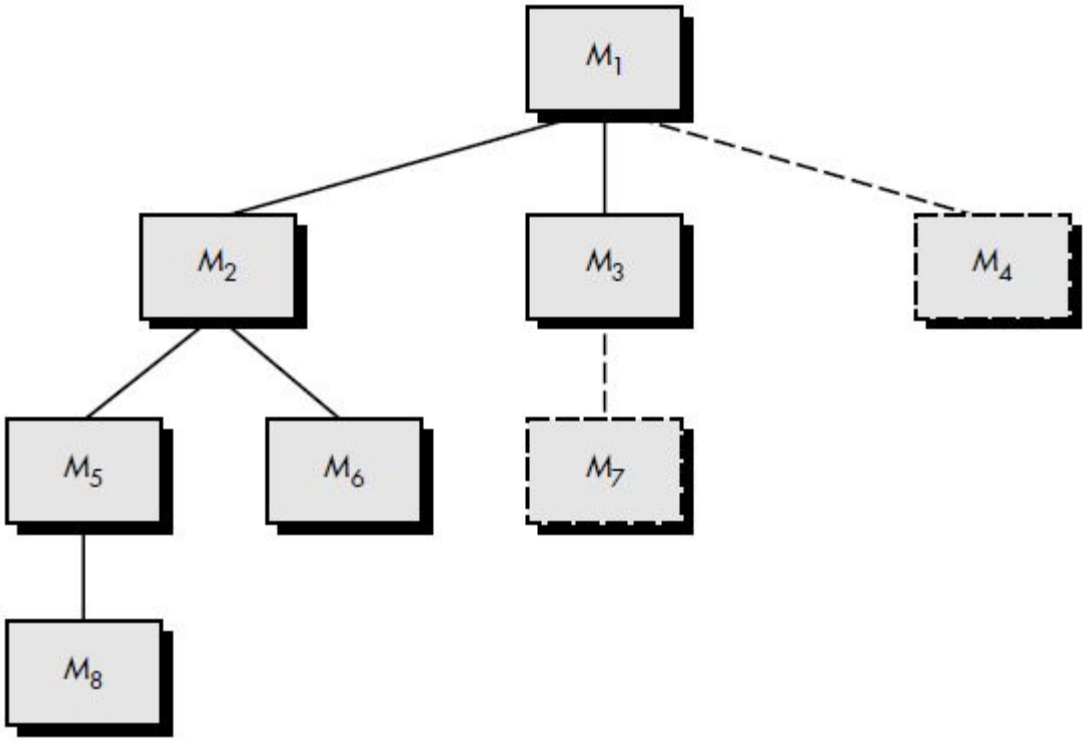
Integration Testing

- Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing.
- The objective is to take unit tested components and build a program structure that has been dictated by design.

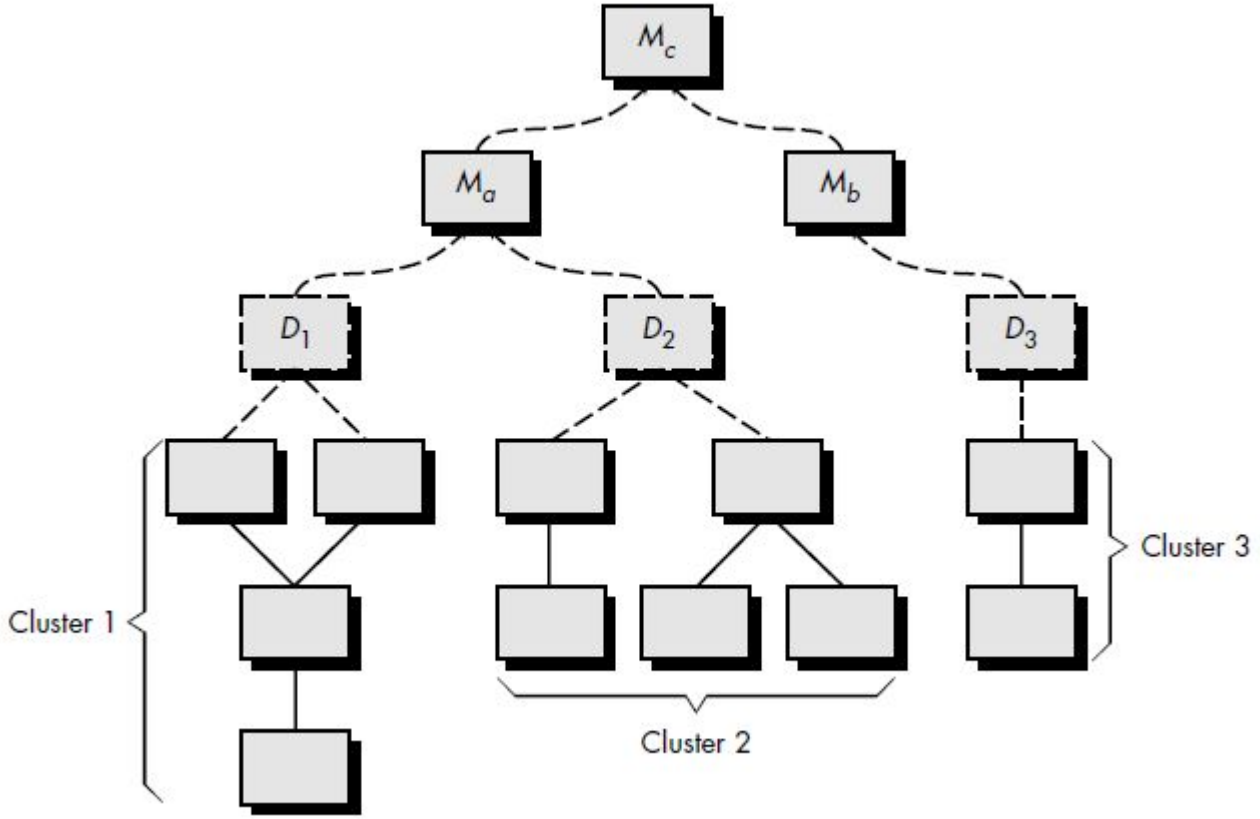
Types of Integration Testing

- Top Down Testing
- Bottom Up Testing
- Smoke Testing
- Regression Testing
- Sandwich Testing

Top Down Integration



Bottom Up Integration



Regression Testing

- A representative sample of tests that will exercise all software functions.
- Additional tests that focus on software functions that are likely to be affected

  by the change.
- Tests that focus on the software components that have been changed.

Smoke Testing

- Software components that have been translated into code are integrated into a "build." A build includes all data files, libraries, reusable modules, and engineered components that are required to implement one or more product functions.
- A series of tests is designed to expose errors that will keep the build from properly performing its function. The intent should be to uncover "show stopper" errors that have the highest likelihood of throwing the software project behind schedule.

Smoke Testing

- The build is integrated with other builds and the entire product (in its current form) is smoke tested daily.
- The integration approach may be top down or bottom up.

Validation Testing

- Validation Test Criteria
- Configuration Review
- Alpha and Beta Testing

Validation Test Criteria

- After each validation test case has been conducted, one of two possible conditions exist:

(1) The function or performance characteristics conform to specification and are accepted or

(2) a deviation from specification is uncovered and a *deficiency list is created.*

Configuration Reviews

- The intent of the review is to ensure that all elements of the software configuration have been properly developed, are cataloged, and have the necessary detail to bolster the support phase of the software life cycle. The configuration review, sometimes called an *audit*.

Alpha Testing

- The *alpha test is conducted at the developer's site by a customer. The software is* used in a natural setting with the developer "looking over the shoulder" of the user and recording errors and usage problems. Alpha tests are conducted in a controlled environment.

Beta Testing

- The *beta test is conducted at one or more customer sites by the end-user of the* software. Unlike alpha testing, the developer is generally not present. Therefore, the beta test is a "live" application of the software in an environment that cannot be controlled by the developer. The customer records all problems (real or imagined) that are encountered during beta testing and reports these to the developer at regular intervals.

System Testing

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

Recovery Testing

- *Recovery testing is a system test that forces the software to fail in a variety of ways* and verifies that recovery is properly performed.
- If recovery is automatic (performed by the system itself), reinitialization, 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.

Stress Testing

- special tests may be designed that generate ten interrupts per second, when one or two is the average rate,
- input data rates may be increased by an order of magnitude to determine how input functions will respond,
- test cases that require maximum memory or other resources are executed,
- test cases that may cause thrashing in a virtual operating system are designed,
- test cases that may cause excessive hunting for disk-resident data are created. Essentially, the tester attempts to break the program.

Security Testing

- During security testing, the tester plays the role(s) of the individual who desires to penetrate the system. Anything goes! The tester may attempt to acquire passwords through external clerical means; may attack the system with custom software designed to breakdown any defenses that have been constructed; may overwhelm the system, thereby denying service to others; may purposely cause system errors, hoping to penetrate during recovery; may browse through insecure data, hoping to find the key to system entry.

Performance Testing

- *Performance testing* is designed to test the run-time performance of software within the context of an integrated system.
- Performance testing occurs throughout all steps in the testing process.
- Even at the unit level, the performance of an individual module may be assessed as white-box tests are conducted.

Software Reliability

- *Software reliability is defined in statistical* terms as "the probability of failure-free operation of a computer program in a specified environment for a specified time“
- If we consider a computer-based system, a simple measure of reliability is *meantime- between-failure (MTBF), where*

$$MTBF = MTTF + MTTR$$

- The acronyms MTTF and MTTR are mean-time-to-failure and mean-time-to-repair, respectively.

Software Availability

- *Software availability is the probability that a program is operating according to requirements*

  at a given point in time and is defined as

  Availability = [MTTF/(MTTF + MTTR)] * 100%

- The MTBF reliability measure is equally sensitive to MTTF and MTTR.
- The availability measure is somewhat more sensitive to MTTR, an indirect measure of the maintainability of software.

Software Safety & Reliability

- *Software safety is a software quality assurance activity that focuses on the identification* and assessment of potential hazards that may affect software negatively and cause an entire system to fail. If hazards can be identified early in the software engineering process, software design features can be specified that will either eliminate or control potential hazards.

Software Maintenance

- Much of the software we depend on today is on average 10 to 15 years old. Even when these programs were created using the best design and coding techniques known at the time [and most were not], they were created when program size and storage space were principle concerns.
- They were then migrated to new platforms, adjusted for changes in
- machine and operating system technology and enhanced to meet new user needs—all without enough regard to overall architecture.
- The result is the poorly designed structures, poor coding, poor logic, and poor documentation of the software systems we are now called on to keep running . . .

Types of Maintenance

- Corrective Maintenance
- Perfective Maintenance
- Adaptive Maintenance
- Preventive Maintenance

Note: These all are covered in Unit 1.

Software Metric

- % of Testing effort(K) =???(Pressmen)

Software Evaluation

Black Box Testing

- Why we measure?
- Project Metrics
- Software Measurement
- Metrics—

1.Size Oriented Metrics

2. Function Oriented Metrics

Mettrics for Software Quality

--Correctness

--Maintainabililty

--Integrity

--Usability

- Defect Removal Efficiency(DRE)=E/(E+D)
- Graph Matrics: Square Metric

Thank You

Any Doubt in Unit 5?????