UNIT IV Testing Strategies and Metrics for Process and Products

Testing Strategies

Testing Strategies:

- **Purpose:** Uncover errors introduced during design and construction of software.
- **Effort:** Testing often consumes more project effort than other software engineering activities, accounting for about 40% of the total project cost.
- **Development:** Developed collaboratively by the project manager, software engineers, and testing specialists.
- **Process:** Testing is the process of executing a program with the intention of finding errors.
- **Testing Strategy Components:**
    - Test planning
    - Test case design
    - Test execution
    - Resultant data collection and analysis

Validation:

- **Definition:** Refers to a set of activities ensuring that the software is traceable to customer requirements.
- **Scope:** Encompasses a wide array of Software Quality Assurance (SQA) activities.

A Strategic Approach to Software Testing

1. **Planned and Systematic:**
    - A set of activities that can be planned in advance and conducted systematically.
2. **Characteristics of Testing Strategy:**
    - **Usage of Formal Technical Reviews (FTR):** Incorporate formal technical reviews as part of the testing process.

    - **Begins at Component Level and Covers Entire System:** Start testing at the component level and extend the coverage to the entire system.
    - **Different Techniques at Different Points:** Utilize various testing techniques at different stages of the development process.
    - **Conducted by Developer and Test Group:** Involves both developers and a dedicated testing group.
    - **Includes Debugging:** Testing strategy should encompass debugging activities.
3. **Verification and Validation:**
    - **Verification:** Ensures that the software correctly implements a specific function. It answers the question, "Are we building the product right?"
    - **Validation:** Ensures that the software built is traceable to customer requirements. It addresses the question, "Are we building the right product?"
4. **Testing as an Element of Verification and Validation:**
    - **Role:** Testing is one element of both verification and validation.
5. **Testing Participants:**
    - **Performed by Software Developer and Independent Testing Group:** Testing can be carried out by both the software developer and an independent testing group.
6. **Testing vs. Debugging:**
    - **Difference:** Testing and debugging are distinct activities, with debugging following the testing phase.
7. **Test Levels:**
    - **Low-Level Tests:** Verify small code segments.
    - **High-Level Tests:** Validate major system functions against customer requirements.

Test Strategies for Conventional Software

- **Spiral Representation of Testing:**
    - Four Levels:
        1. Unit Testing
        2. Integration Testing
        3. Validation Testing

4. System Testing

1. **Unit Testing:**
    - Focuses on individual units of software in the source code.
    - Uses testing techniques to ensure complete coverage and maximum error detection.
    - Emphasizes internal processing logic and data structures.
    - Boundary testing is crucial.
    - Test cases can be designed before or after coding.

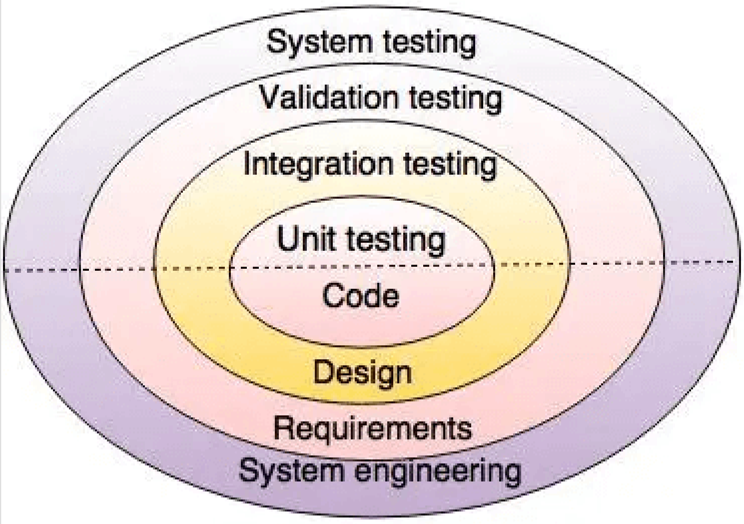


Fig. - Testing Strategy

2. **Integration Testing:**
    - Focuses on design and construction of software architecture.
    - Addresses issues related to verification and program construction.
    - Uncover errors associated with interfacing.
    - Two approaches: Top-down integration and Bottom-up integration.
    - A combined approach called Sandwich strategy is also an option.
3. **Validation Testing:**
    - Validates requirements against the software constructed.
    - High-order tests ensuring that software meets functional, behavioral, and performance requirements.
    - Criteria include:
        - Validation Test Criteria
        - Configuration Review
        - Alpha and Beta Testing
    - Alpha testing at the developer's site, Beta testing at end-user sites.

4. **System Testing:**
    - Tests software and other system elements as a whole.
    - Involves combining software with hardware, people, and databases.
    - Types of tests include:
        - Recovery testing
        - Security testing
        - Stress testing
        - Performance testing

- **Testing Tactics:**
    - Goal: Find errors.
    - A good test is one with a high probability of finding errors.
    - Tests should not be redundant, and they should be appropriately complex.
- **Two Major Categories of Software Testing:**
    - **Black Box Testing:** Examines fundamental aspects of a system, ensuring each function of the product is operational.
    - **White Box Testing:** Examines internal operations and procedural details of a system.

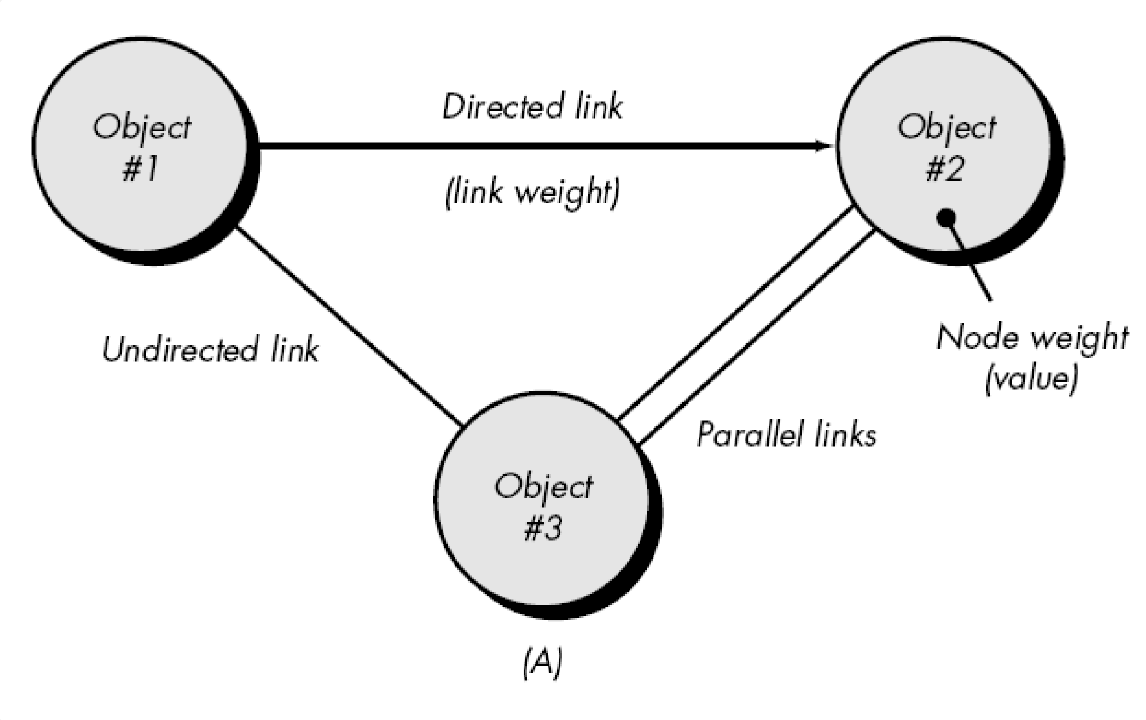
Black-box and White-box Testing

Black Box Testing:

- **Definition:** Also known as behavioral testing, it focuses on the functional requirements of software.
- **Objectives:** Fully exercises all functional requirements, identifies incorrect or missing functions, interface errors, and database errors.
- **Approach:** Treats the system as a black box, studying its input and observing the corresponding output. It is not concerned with the internal workings.
- **Methods:**
    1. **Graph-Based Testing Method:**

        - Begins by creating a graph of important objects and their relationships.
        - Series of tests devised to cover the graph, ensuring each object

and relationship is exercised.
- Uncover errors through graph coverage.



2. **Equivalence Partitioning:**

   - Divides the input domain into classes of data.
   - Derives test cases from these classes to uncover errors.
   - Reduces the number of test cases.
   - Based on equivalence classes representing valid or invalid states for input conditions.

   Example: Input consists of 1 to 10, classes are n<1, 1<=n<=10, n>10.
3. **Boundary Value Analysis:**

   - Selects input lying at the edge or boundary of equivalence classes.
   - Exercises boundary values to uncover errors at the input domain boundaries.

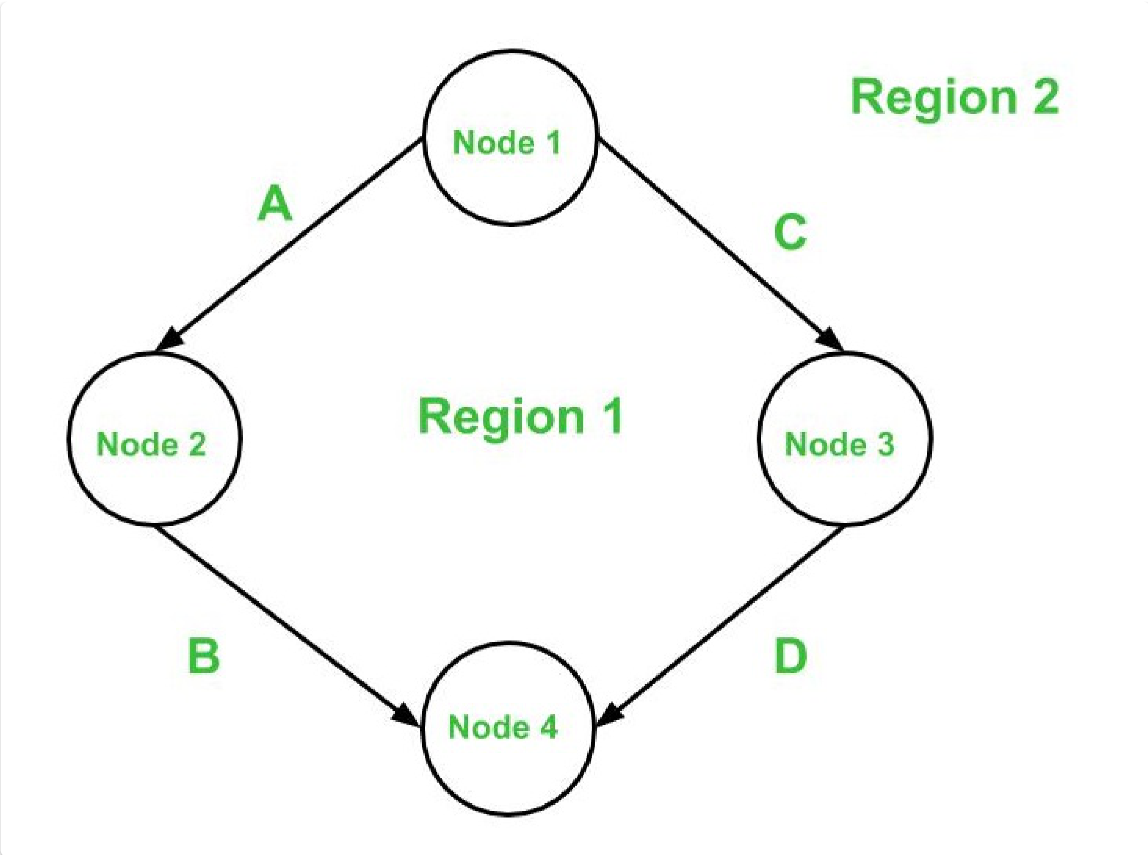
   Example: If 0.0<=x<=1.0, test cases include (0.0, 1.0) for valid input and (-0.1, 1.1) for invalid input.
4. **Orthogonal Array Testing:**
   - Applied to problems with a relatively small input domain but too large for exhaustive testing.
   - Reduces the number of test cases.

- Example: For three inputs A, B, C, each having three values, exhaustive testing requires 27 test cases, while orthogonal testing reduces it to 9.

White Box Testing:

- Also Known As: Glass box testing.
- Characteristics:
  - Uses the control structure to derive test cases.
  - Involves knowing the internal workings of a program.
  - Guarantees the execution of all independent paths at least once.
  - Exercises all logical decisions on their true and false sides.
  - Executes all loops.
  - Exercises all data structures for their validity.
- White Box Testing Techniques:
  1. Basis Path Testing:

     - Proposed by Tom McCabe.
     - Defines a basic set of execution paths based on the logical complexity of a procedural design.
     - Guarantees the execution of every statement in the program at least once.
     - Steps:
       1. Draw the flow graph from the program's flow chart.
       2. Calculate the cyclomatic complexity of the resultant flow graph.
       3. Prepare test cases that force the execution of each path.



2. Control Structure Testing:
   - Broadens testing coverage and improves quality.
   - Methods:

     a) Condition Testing:

     - Exercises logical conditions in a program module.
     - Focuses on testing each condition to ensure it does not contain errors.
     - Types of errors include operator errors, variable errors, and arithmetic expression errors.

     b) Data Flow Testing:

     - Selects test paths based on the locations of variable definitions and uses in a program.
     - Aims to ensure that variable definitions and subsequent uses are tested.
     - Constructs a definition-use graph from the program's control flow.

     c) Loop Testing:
     - Focuses on the validity of loop constructs.

- Four categories: Simple loops, Nested loops, Concatenated loops, Unstructured loops.
- Testing of simple loops involves scenarios like skipping the loop, one pass, two passes, m passes (where m > N), N-1, N, N+1 passes.

Validation Testing

Validation testing is a crucial phase in the software development lifecycle, irrespective of whether it involves conventional software, object-oriented software, or web applications. The testing strategy remains consistent across these types of software. When a software requirements specification is in place, it outlines the validation criteria forming the basis for the validation-testing approach.

Key Components of Validation Testing:

1. **Test Plan:**
   - Outlines the classes of tests to be conducted.
   - Defines the scope and objectives of the testing phase.
2. **Test Procedure:**
   - Defines specific test cases to ensure:
     - All functional requirements are satisfied.
     - Behavioral characteristics are achieved.
     - Content is accurate and properly presented.
     - Performance requirements are met.
     - Documentation is correct.
     - Usability and other requirements are fulfilled (e.g., transportability, compatibility, error recovery, maintainability).
3. **Validation Test Case Execution:**
   - After each validation test case, one of two conditions exists:
     1. The function or performance characteristic is accepted.
     2. A deviation from the specification is found, and a deficiency list is created.
4. **Configuration Review (Audit):**
   - An essential element of the validation process.

- Ensures that all elements of the software configuration have been properly developed and cataloged.

Alpha and Beta Testing:

1. **Alpha Testing:**
    - Conducted at the developer's site by a group of representative users.
    - Software is used in a natural setting, recording errors and usage problems.
    - Conducted in a controlled environment.
    - Intended to uncover errors that end-users may not identify.
2. **Beta Testing:**
    - Conducted at one or more end-user sites.
    - Developer generally does not present during alpha testing.
    - A "live" application of the software in a real-world environment.
    - End-users record all encountered problems and report them to the developer.

Customer Acceptance Testing:

- **Purpose:**
    - Typically performed when custom software is delivered to a customer under contract.
    - The customer conducts specific tests to uncover errors before accepting the software.

System Testing

System testing is a comprehensive phase that consists of various tests, each serving a specific purpose. The primary goal is to fully exercise the computer-based system, ensuring that all integrated elements function as allocated.

Types of System Testing:

1. **Recovery Testing:**
    - **Objective:** Verify that the system can recover from faults and resume processing with minimal or no downtime.
    - **Requirements:** The system must be fault-tolerant, and faults should not cause a complete system function failure.

- Evaluation: If recovery requires human intervention, the Mean Time To Repair (MTTR) is assessed to determine acceptability.

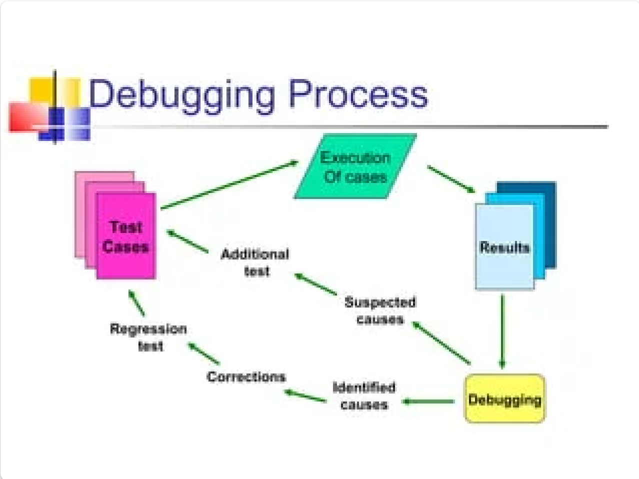
2. Security Testing:
    - Objective: Verify that protection mechanisms within the system will prevent improper or illegal penetration.
    - Scope: Particularly critical for systems managing sensitive information.
    - Evaluation: Ensures that the system is resilient against unauthorized access and security breaches.
3. Stress Testing:
    - Objective: Confront the system with abnormal situations by demanding resources in abnormal quantity, frequency, or volume.
    - Examples:
        - Increased input data rates by an order of magnitude.
        - Execution of test cases requiring maximum memory.
    - Purpose: Evaluate system performance under extreme conditions.
4. Performance Testing:
    - Objective: Test the run-time performance of software within the integrated system context.
    - Scope: Particularly critical for real-time and embedded systems.
    - Continuous Process: Performance testing occurs throughout all steps in the testing process.
5. Deployment Testing:
    - Objective: Ensure that the software executes on various platforms and under different operating system environments.
    - Scope: Exercises the software in each environment where it will operate.
    - Focus Areas:
        - Examines all installation procedures.
        - Tests specialized installation software used by customers.
        - Ensures documentation is suitable for introducing the software to end-users.

The Art of Debugging

The Art of Debugging:

- Introduction:

- Debugging occurs as a consequence of successful testing, resulting in the removal of errors.
- An art that involves finding and correcting the causes of software errors.



- Debugging Outcomes:
  - Cause will be found and corrected.
  - Cause will not be found.
- Characteristics of Bugs:
  - Symptom and cause can be in different locations.
  - Symptoms may be caused by human error or timing problems.
- Human Trait:
  - Debugging is an innate human trait; some individuals are naturally adept at it.
- Debugging Strategies:
  - Objective: Find and correct the cause of a software error through systematic evaluation, intuition, and luck.
  - Three strategies:
    1. Brute Force Method:
       - Common but least efficient method.
       - Applied when other methods fail.
       - Involves memory dumps, run-time traces, and extensive use of output statements.
       - Can lead to a waste of time and effort.
    2. Back Tracking:
       - Common debugging approach, useful for small programs.
       - Traces the source code backward from the site where the symptom is uncovered.

    - Challenging for large programs with numerous lines of code.
3. **Cause Elimination:**
    - Based on binary partitioning.
    - Organizes data related to error occurrences to isolate potential causes.
    - Develops a "cause hypothesis" and conducts tests to prove or disprove it.
    - Creates a list of all possible causes and systematically eliminates them.

- **Automated Debugging:**
    - Supplements manual approaches with debugging tools.
    - Provides semi-automated support, including debugging compilers, dynamic debugging aids, test case generators, mapping tools, etc.

Metrics for Process and Products

Metrics for Testing:

- **n1:** The number of distinct operators that appear in a program.
- **n2:** The number of distinct operands that appear in a program.
- **N1:** The total number of operator occurrences.
- **N2:** The total number of operand occurrences.

Program Level and Effort:

- **PL =** 1 / [(n1 / 2) x (N2 / n2)]
- **e =** V / PL

Metrics for Maintenance:

- **Mt:** The number of modules in the current release.
- **Fc:** The number of modules in the current release that have been changed.
- **Fa:** The number of modules in the current release that have been added.
- **Fd:** The number of modules from the preceding release that were deleted in the current release.

Software Maturity Index (SMI):

- SMI = (Mt - (Fc + Fa + Fd)) / Mt

Software Measurement

- Categorization:
  1. Direct Measure:
     - *Software Process:* Includes cost and effort.
     - *Software Product:* Includes lines of code, execution speed, memory size, defects per reporting time period.
  2. Indirect Measure:
     - Examines the quality of the software product itself (e.g., functionality, complexity, efficiency, reliability, and maintainability).
- Reasons for Measurement:
  - Gain a baseline for future assessments.
  - Determine status with respect to the plan.
  - Predict size, cost, and duration estimates.
  - Improve product quality and process.
- Metrics in Software Measurement:
  - Size-Oriented Metrics:
    - Concerned with the measurement of software.
    - Includes LOC, effort, cost, PP document, errors, defects, and people.
  - Function-Oriented Metrics:
    - Measures functionality derived by the application.
    - Widely used metric: Function Point (independent of programming language).
  - Object-Oriented Metrics:
    - Relevant for object-oriented programming.
    - Based on the number of scenarios, key classes, support classes, average support classes per key class, and subsystems.
  - Web-Based Application Metrics:
    - Measure:
      1. Number of static pages (NSP)
      2. Number of dynamic pages (NDP)
      3. Customization (C) = NSP / (NSP + NDP) (C should approach 1).

Metrics for Software Quality

- **Correctness:** Defects per KLOC (thousands of lines of code).
- **Maintainability:** Mean-time to change (MTTC).
- **Integrity:** Sigma[1 - (threat * (1 - security))].
  - *Threat:* Probability of a specific attack within a given time.
  - *Security:* Probability of repelling a specific attack.
- **Usability:** Ease of use.
- **Defect Removal Efficiency (DRE):** DRE = E / (E + D) (E: errors found before delivery, D: defects reported after delivery; ideal DRE is 1).