UNIT IV
TESTING

Taxonomy of software testing – levels – test activities – types of s/w test – black box testing – testing boundary conditions – structural testing – test coverage criteria based on data flow mechanisms – regression testing – testing in the large. S/W testing strategies – strategic approach and issues - unit testing – integration testing – validation testing – system testing and debugging.

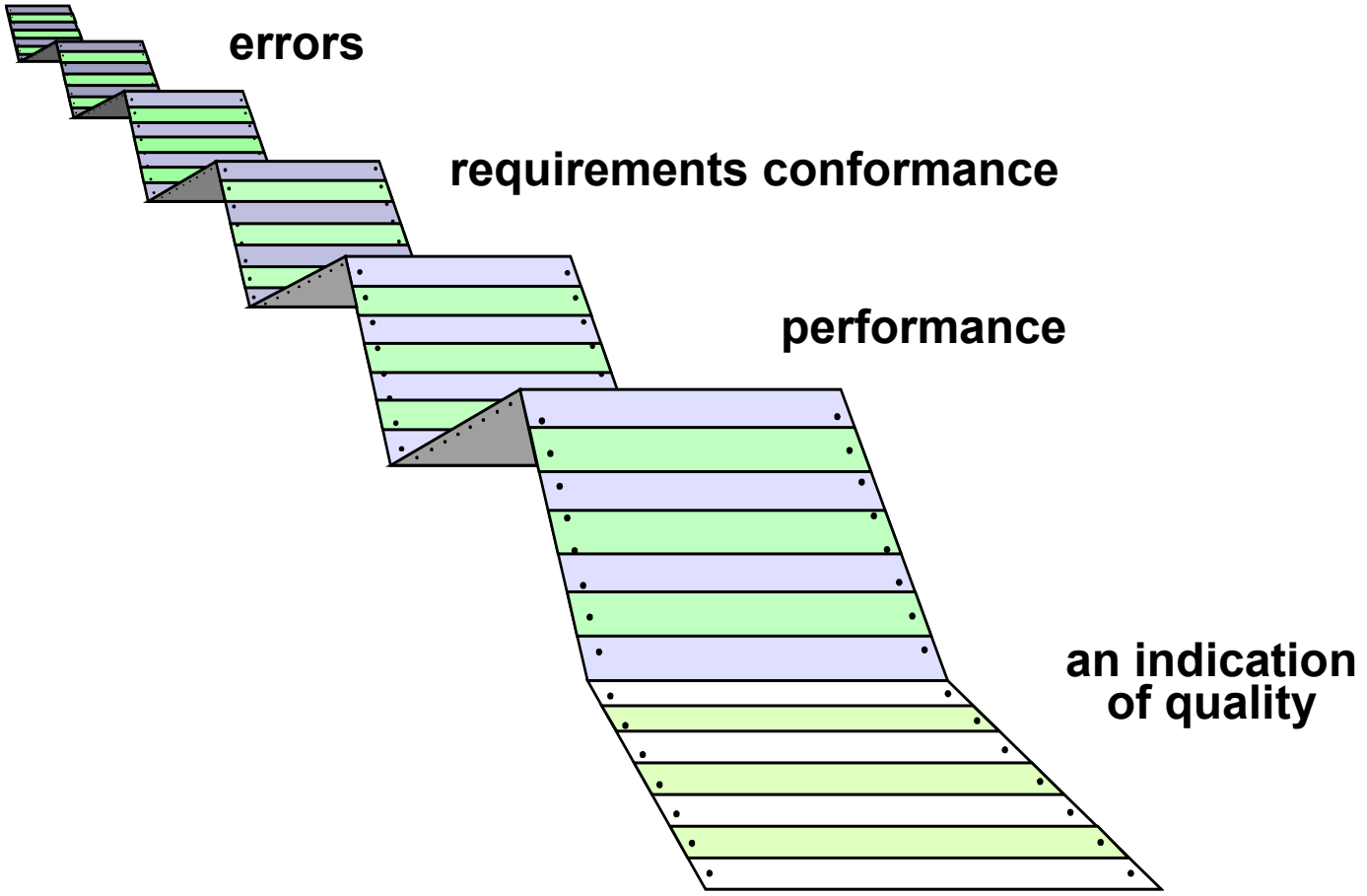
Software Testing

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

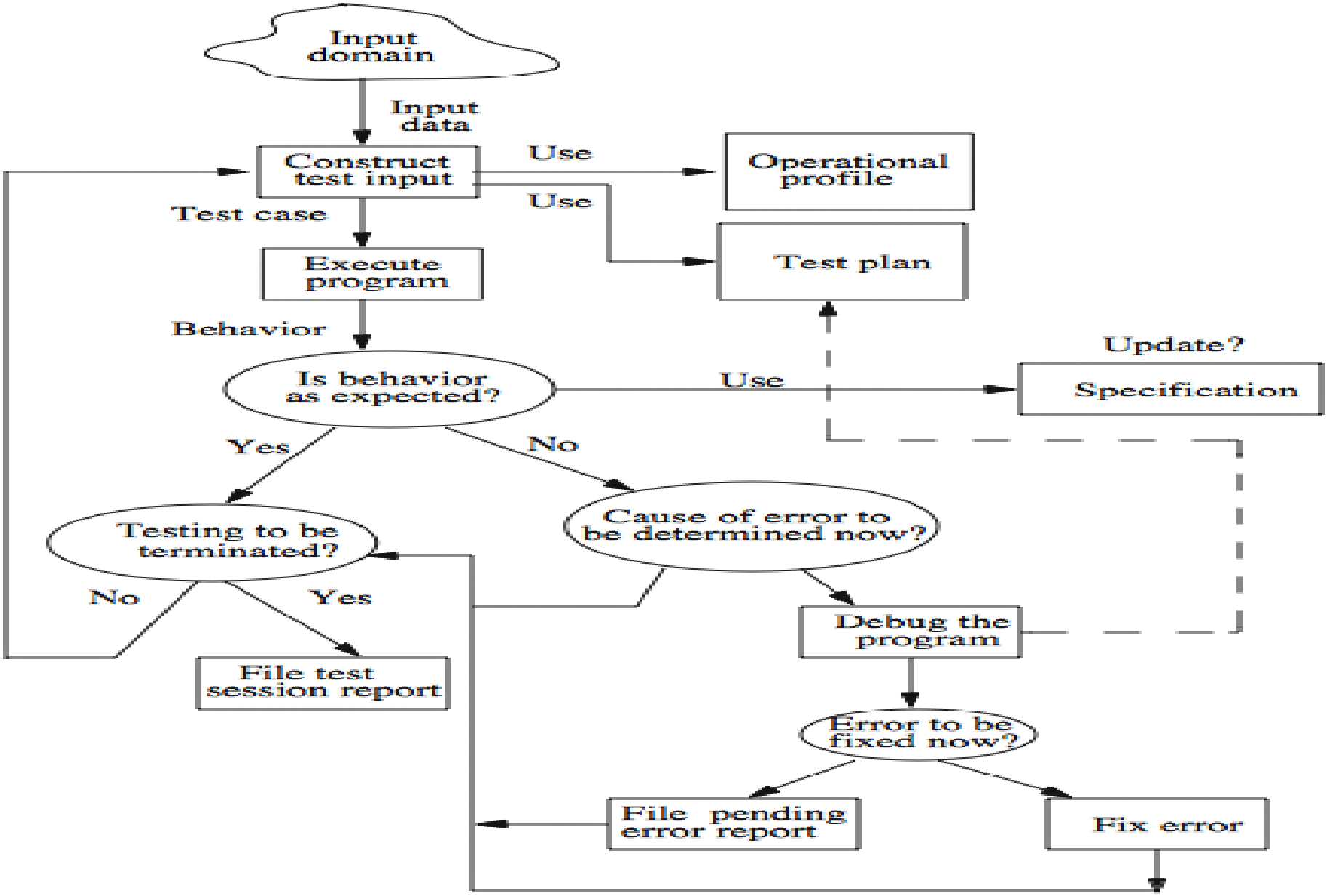
- Testing is the process of determining if a program has any errors.

- When testing reveals an error, the process used to determine the cause of this error and to remove it, is known as debugging.

What Testing Shows



A test/debug cycle



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

Taxonomy of software testing

- Classified by purpose, software testing can be divided into:
  - Correctness testing(Block, White)
  - Performance testing(Bandwidth requirements, CPU cycles, disk space, disk access operations, and memory usage)
  - Reliability testing (block box)
  - Security testing (security violations, and validating the effectiveness of security measures).

Cont..

- Classified by life-cycle phase, software testing can be classified into the following categories:
  - Requirements phase testing
  - Design phase testing
  - Program phase testing
  - Evaluating test results
  - Installation phase testing
  - Acceptance testing
  - Maintenance testing.

Cont..

- By scope, software testing can be categorized as follows:
  - Unit testing,
  - Component testing,
  - Integration testing,
  - System testing.

Some testing tools:

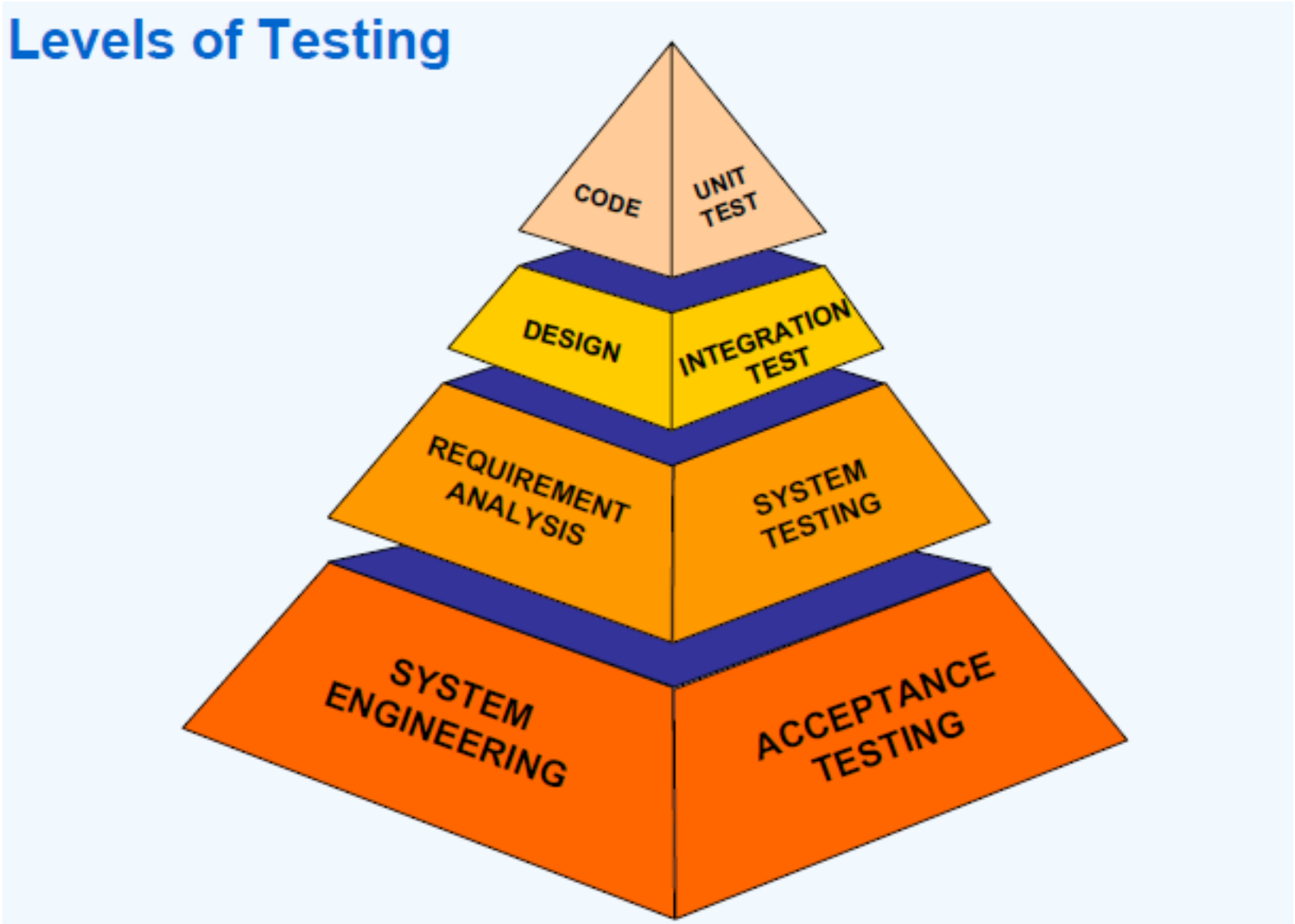
- GUI Testing:
  - Eggplant
  - Marathon
  - Pounder
- Performance or load testing:
  - eLoadExpert
  - DBMonster
  - JMeter
  - Dieseltest
  - WAPT
  - LoadRunner
  - Grinder
- Regression testing:
  - WinRunner
  - TerstTube
  - Echelon

Test Activities

Various testing activities are:

- ***Test Planning:*** Indicates the scope, approach, resources and the schedule of testing activity.
- ***Test Case design:*** The goal of test case design is to create a set of tests that are effective in testing. Good test cases have a high probability of detecting undiscovered errors.
- ***Test Procedure:*** Identifies all the steps required to operate the system and implement the test design.
- ***Test Execution:*** The test data is derived through various test cases in order to obtain the test result. It is the exercising of test procedures. It moves through all the levels of software testing.
- ***Data Collection:*** The test results are collected and verified.
- ***Test Report:*** The above test activities are performed on the software model and the maximum number of errors are uncovered.

Levels of Testing



Types of S/W Test

There are six different types of testing they are

1. White-Box Testing
2. Black-Box Testing
3. Unit Testing
4. Integration Testing
5. Validation Testing
6. System Testing

1.White-Box Testing

- *White-box testing* of software is predicated on close examination of procedural detail.
- Logical paths through the software are tested by providing test cases that exercise specific sets of conditions and/or loops.
- The "status of the program" may be examined at various points.
- White-box testing, sometimes called *glass-box testing,* is a test case design method that uses the control structure of the procedural design to derive test cases.

Cont..

The major focus of White-Box Testing is on structural testing, statements, paths, branches, control flows, data flows, conditions and loops.

Using this method, SE 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
4. Exercise internal data structures to ensure their validity.

Basis path testing

- *Basis path testing* is a white-box testing technique
- To derive a logical complexity measure of a procedural design.
- Test cases derived to exercise the basis set are guaranteed to execute every statement in the program at least one time.

Methods:

1. Flow graph notation
2. Independent program paths or Cyclomatic complexity
3. Deriving test cases
4. Graph Matrices

Flow Graph Notation

- Start with simple notation for the representation of *control flow (called flow graph).* It represent logical control flow.

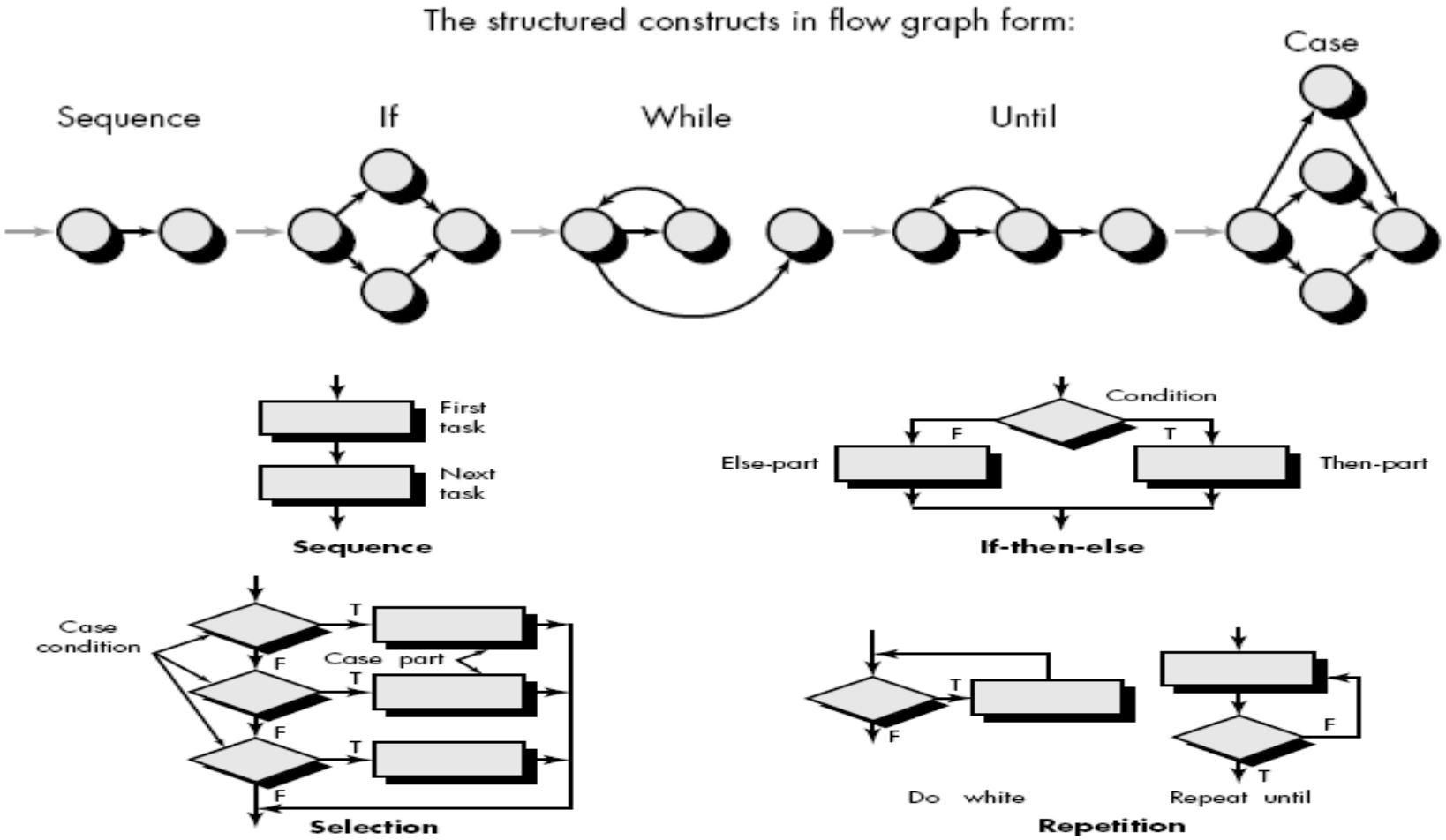
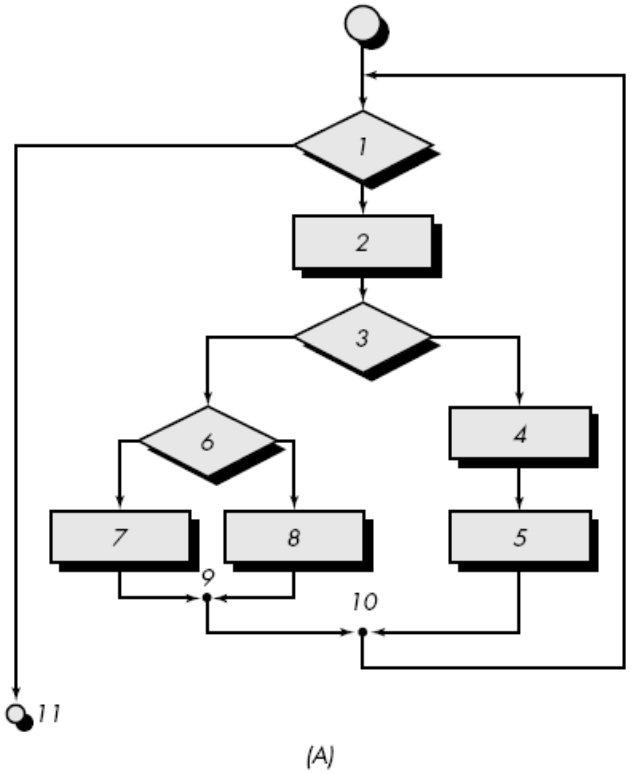
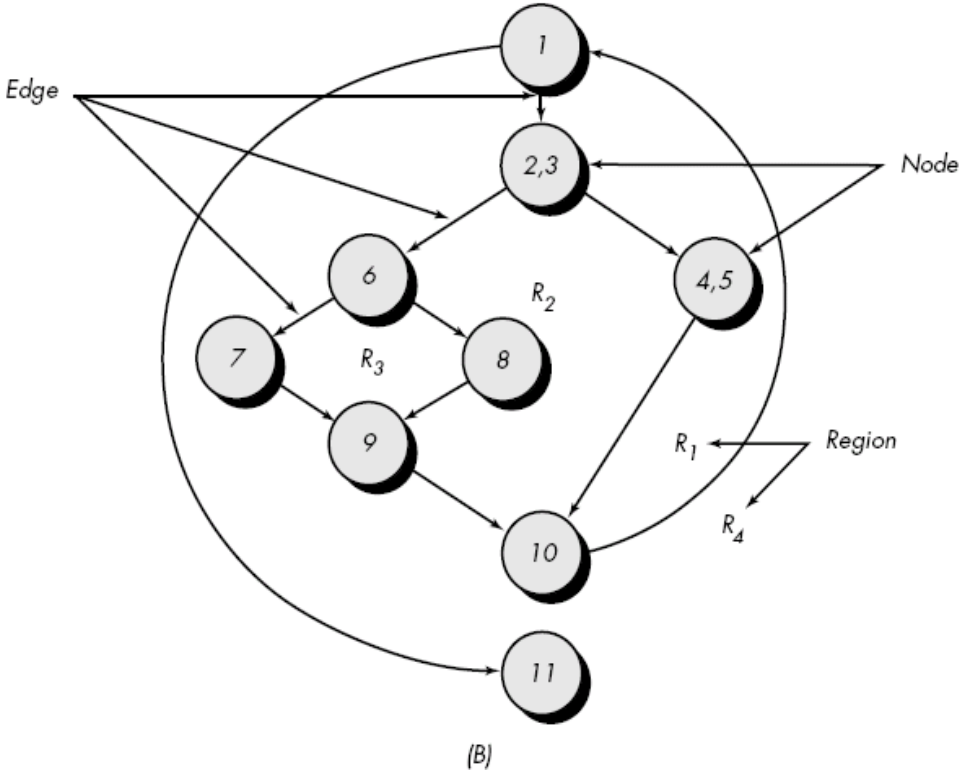


Fig. A represent program control structure and fig. B maps the flowchart into a corresponding flow graph.



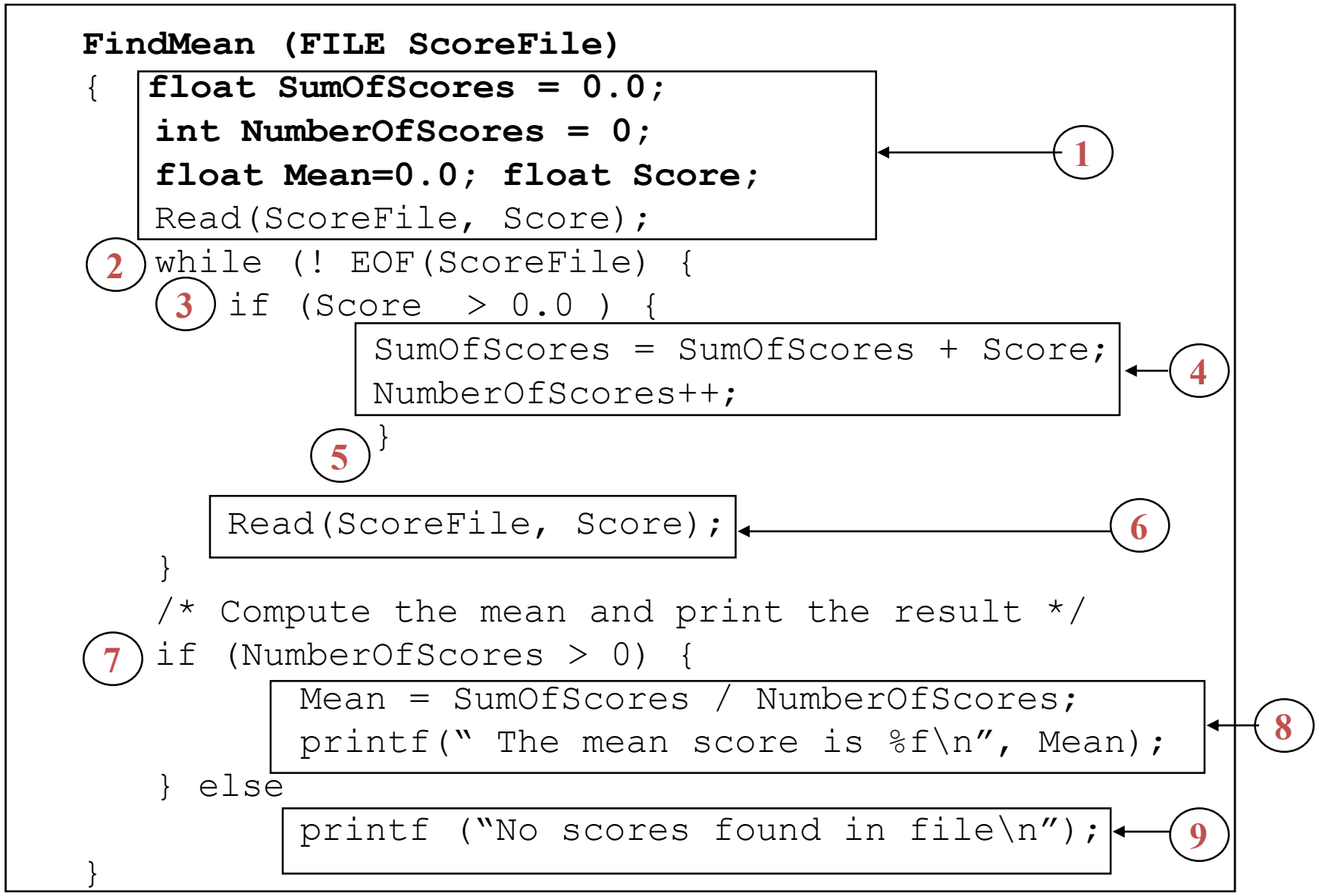
(A)



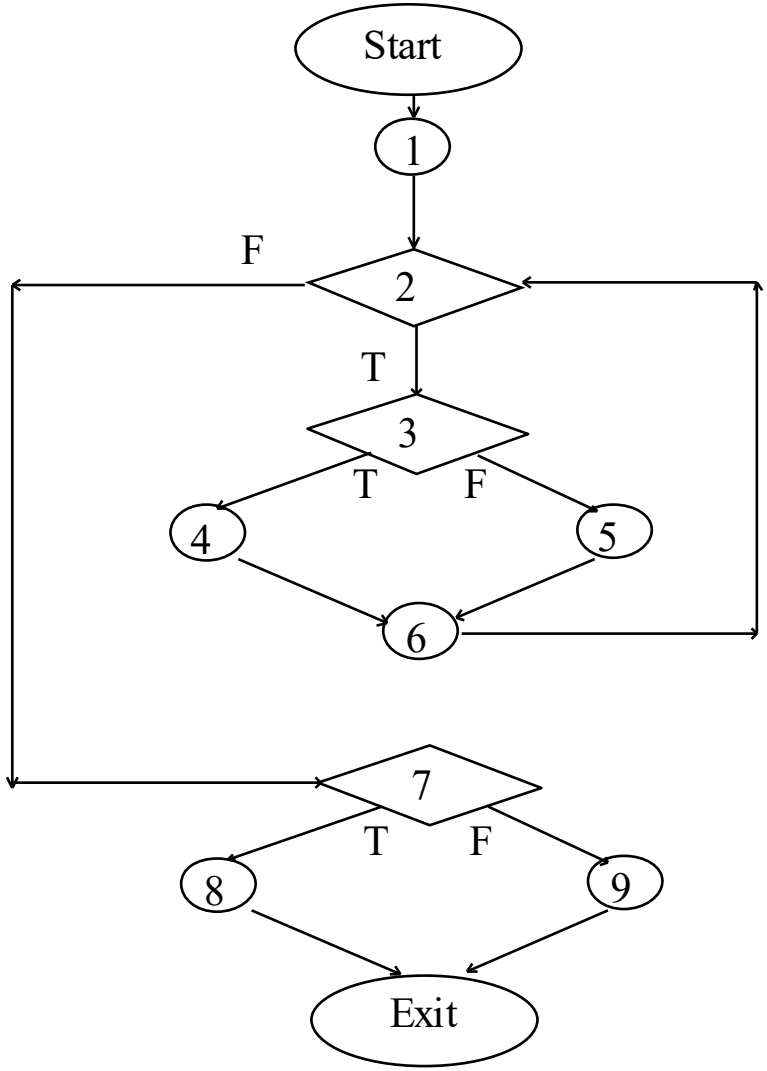
(B)

In fig. B each circle, called flow graph node, represent one or more procedural statement.

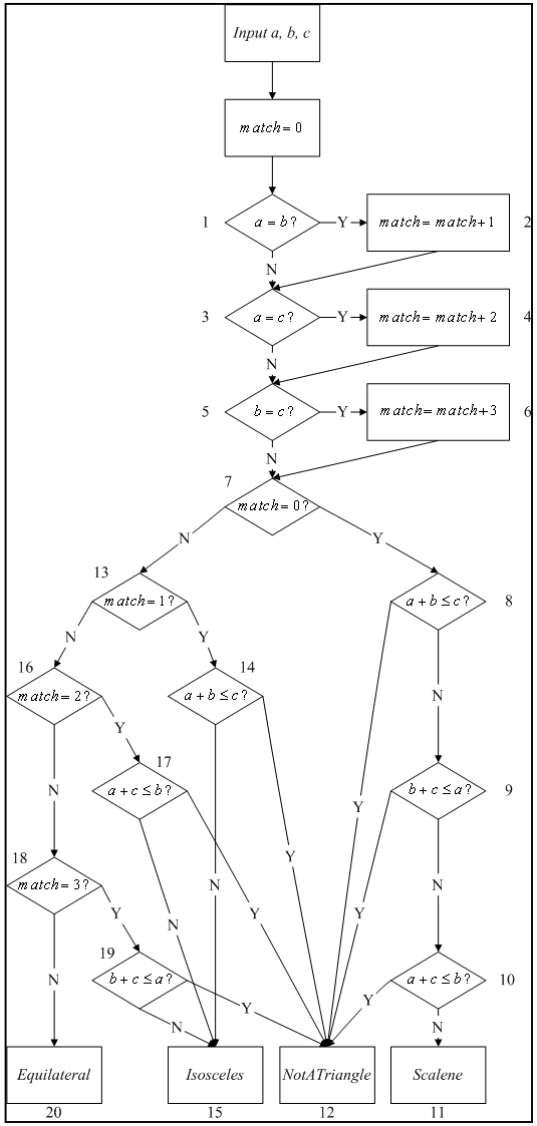
White-box Testing: Determining the Basic Blocks

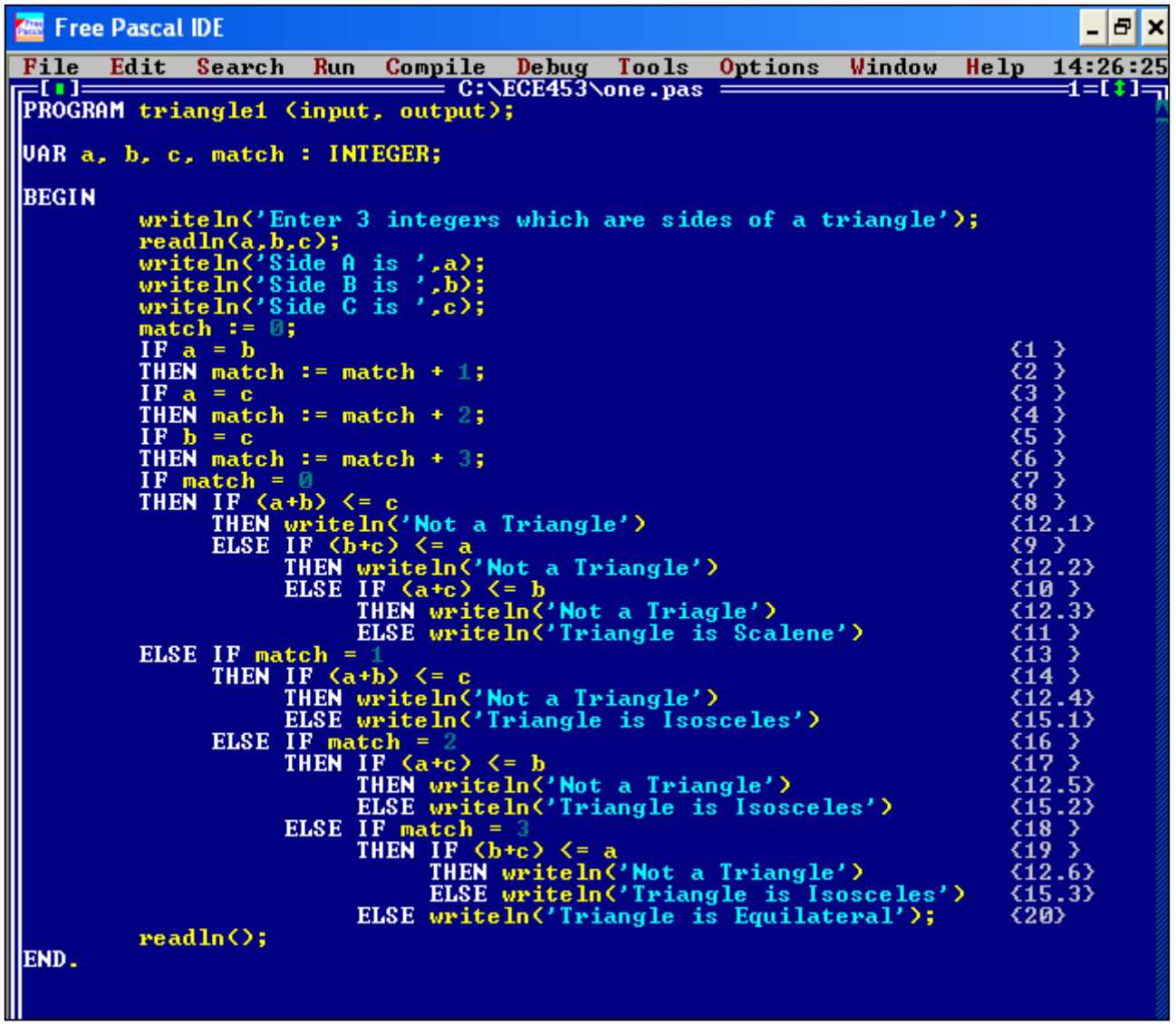


Constructing the Logic Flow Diagram



Program Graph - Example





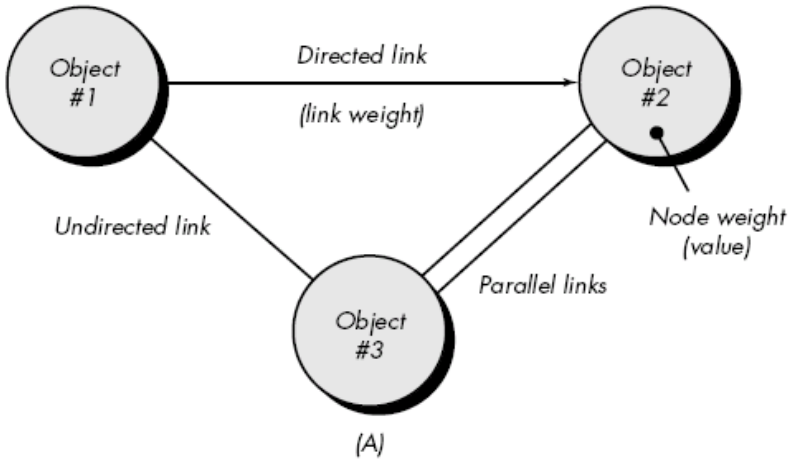
2.Black box testing

- Also called *behavioral testing,* focuses on the functional requirements of the software.
- It enables the software engineer to derive sets of input conditions that will fully exercise all functional requirements for a program.
- Black-box testing is not an alternative to white-box techniques but it is complementary approach.
- Black-box testing attempts to find errors in the following categories:
  - Incorrect or missing functions,
  - Interface errors,
  - Errors in data structures or external data base access.
  - Behavior or performance errors,
  - Initialization and termination errors.

- Black-box testing purposely ignored control structure, attention is focused on the information domain. Tests are designed to answer the following questions:
  - **How is functional validity tested?**
  - **How is system behavior and performance tested?**
  - **What classes of input will make good test cases?**
- By applying black-box techniques, we derive a set of test cases that satisfy the following criteria
  - Test cases that reduce the number of additional test cases that must be designed to achieve reasonable testing (i.e minimize effort and time)
  - Test cases that tell us something about the presence or absence of classes of errors
- Black box testing methods
  - Graph-Based Testing Methods
  - Equivalence partitioning
  - Boundary value analysis (BVA)

Graph-Based Testing Methods

- To understand the objects that are modeled in software and the relationships that connect these objects.
- Next step is to define a series of tests that verify "all objects have the expected relationship to one another.
- Begin by creating graph –
  - a collection of *nodes* that represent objects
  - *links* that represent the relationships between objects
  - *node weights* that describe the properties of a node
  - *link weights* that describe some characteristic of a link.



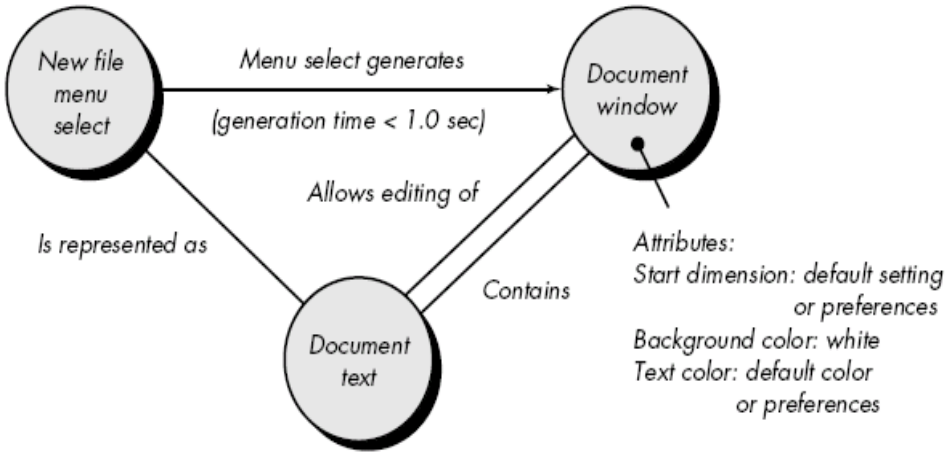
Cont..

- 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.

- *Object #1* = **new file menu select**
- *Object #2* = **document window**
- *Object #3* = **document text**

Referring to example figure, a menu select on **new file** generates a **document window.**

- The link weight indicates that the window must be generated in less than 1.0 second.
- The node weight of **document window** provides a list of the window attributes that are to be expected when the window is generated.
- An undirected link establishes a symmetric relationship between the **new file menu select** and **document text,**
- parallel links indicate relationships between **document window** and **document text**



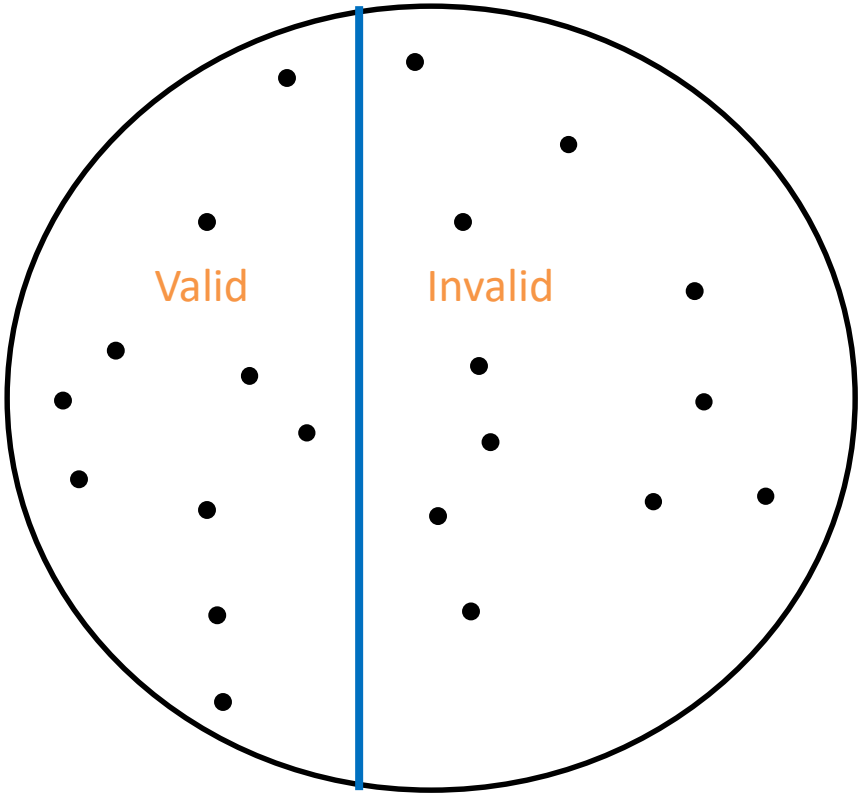
(B)

Equivalence Partitioning

- *Equivalence partitioning* is a black-box testing method that divides the input domain of a program into classes of data from which test cases can be derived.
- Test case design for equivalence partitioning is based on an evaluation of *equivalence classes* for an input condition.
- 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.

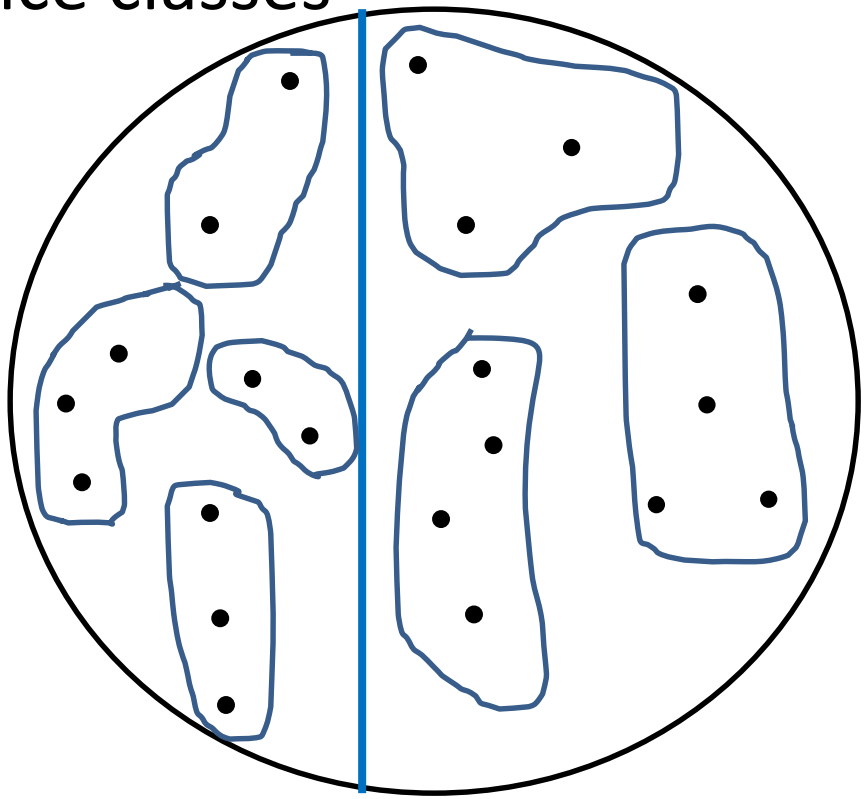
Equivalence Partitioning

- First-level partitioning: Valid vs. Invalid test cases



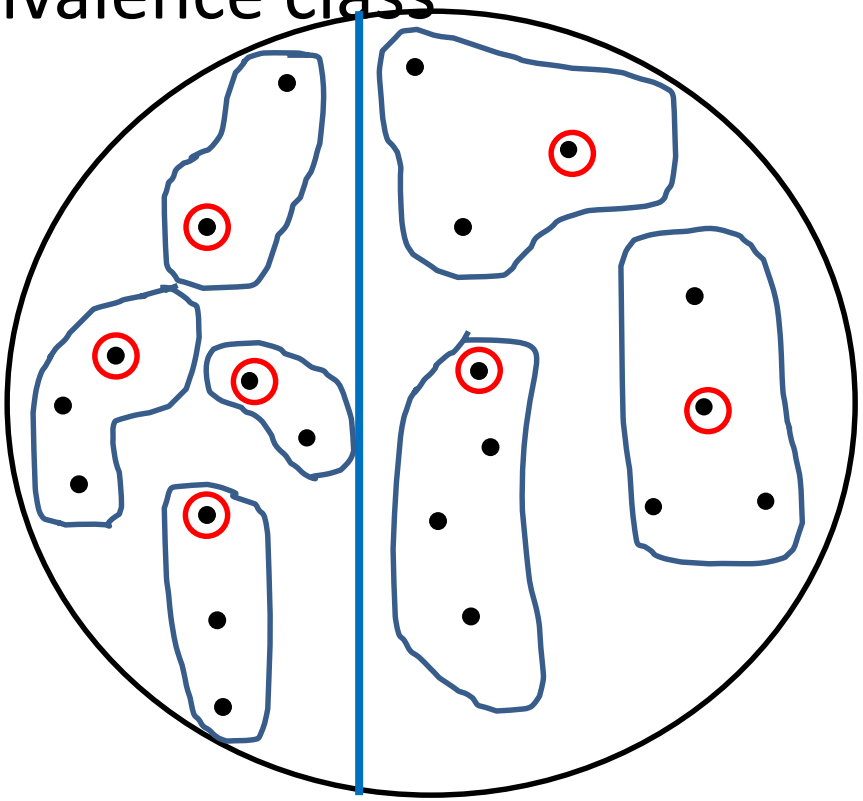
Equivalence Partitioning

- Partition valid and invalid test cases into equivalence classes



Equivalence Partitioning

- Create a test case for at least one value from each equivalence class



Equivalence Partitioning - examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	?	?
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?	?

Equivalence Partitioning - examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	?
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?	?

Equivalence Partitioning - examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	< -99 > 99 Malformed numbers {12-, 1-2-3, …} Non-numeric strings {junk, 1E2, $13} Empty value
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?	?

Equivalence Partitioning - examples

Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	< -99 > 99 Malformed numbers {12-, 1-2-3, …} Non-numeric strings {junk, 1E2, $13} Empty value
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	555-5555 (555)555-5555 555-555-5555 200 <= Area code <= 999 200 < Prefix <= 999	?

Equivalence Partitioning - examples

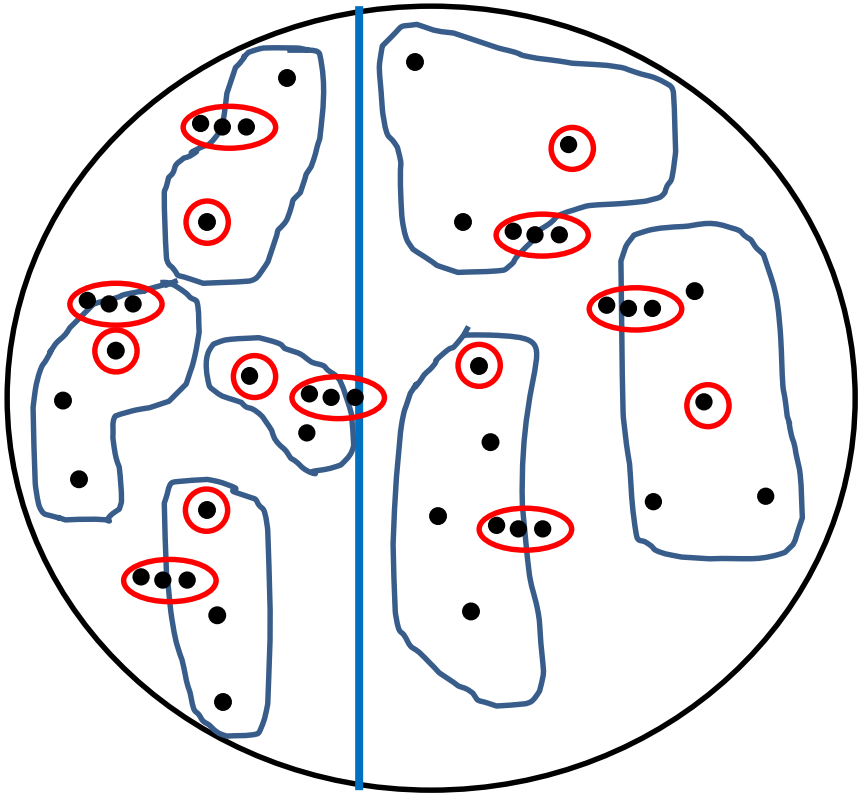
Input	Valid Equivalence Classes	Invalid Equivalence Classes
A integer N such that: -99 <= N <= 99	[-99, -10] [-9, -1] 0 [1, 9] [10, 99]	< -99 > 99 Malformed numbers {12-, 1-2-3, …} Non-numeric strings {junk, 1E2, $13} Empty value
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	555-5555 (555)555-5555 555-555-5555 200 <= Area code <= 999 200 < Prefix <= 999	Invalid format 5555555, (555)(555)5555, etc. Area code < 200 or > 999 Area code with non-numeric characters *Similar for Prefix and Suffix*

Boundary Value Analysis (BVA)

- Boundary value analysis is a test case design technique that complements equivalence partitioning.
- Rather than selecting any element of an equivalence class, BVA leads to the selection of test cases at the "edges" of the class.
- In other word, Rather than focusing solely on input conditions, BVA derives test cases from the output domain as well.

Boundary Value Analysis

- Create test cases to test boundaries of equivalence classes



Boundary Value Analysis - examples

Input	Boundary Cases
A number N such that: -99 <= N <= 99	?
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?

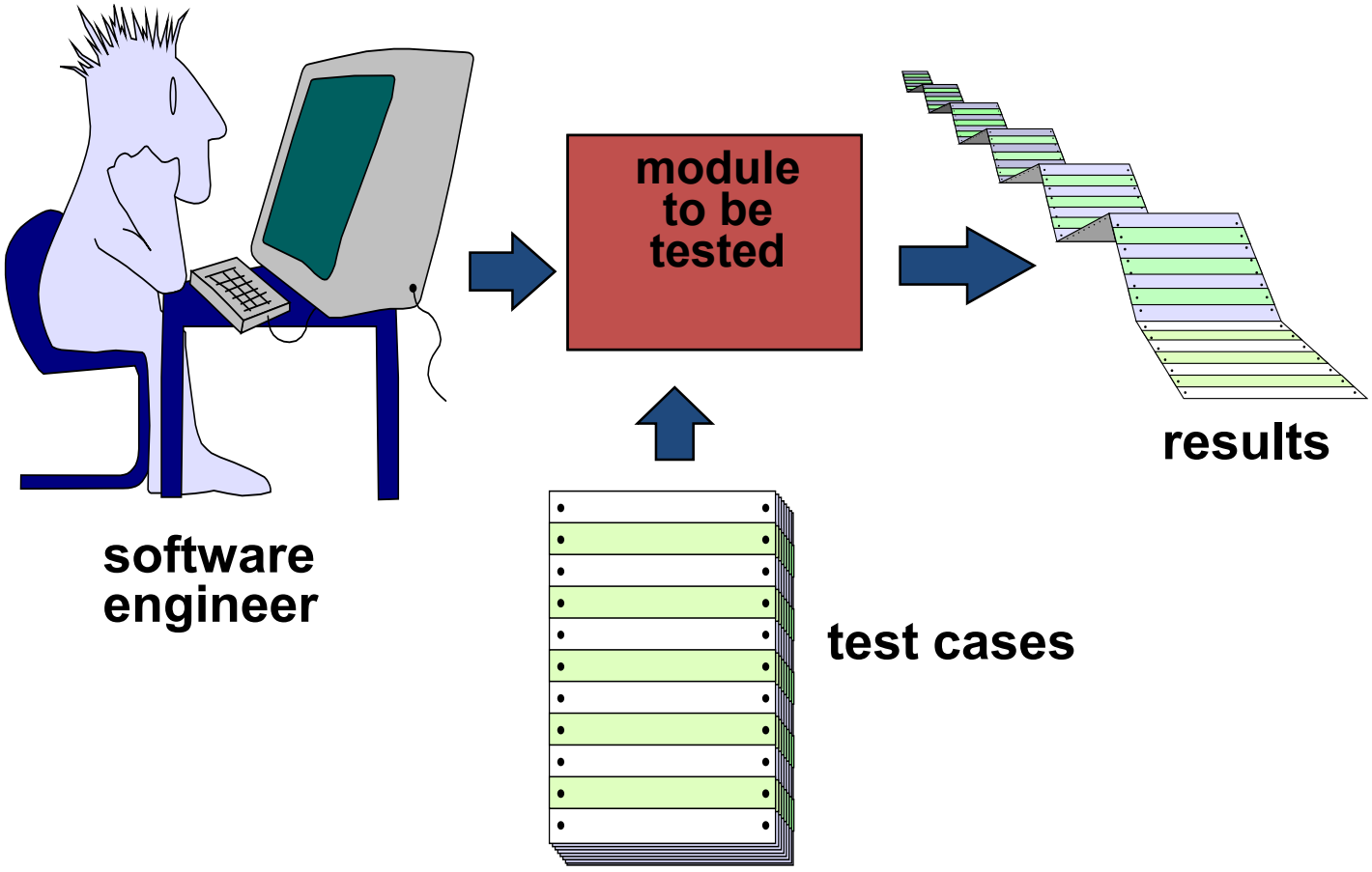
Boundary Value Analysis - examples

Input	Boundary Cases
A number N such that: -99 <= N <= 99	-100, -99, -98 -10, -9 -1, 0, 1 9, 10 98, 99, 100
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	?

Boundary Value Analysis - examples

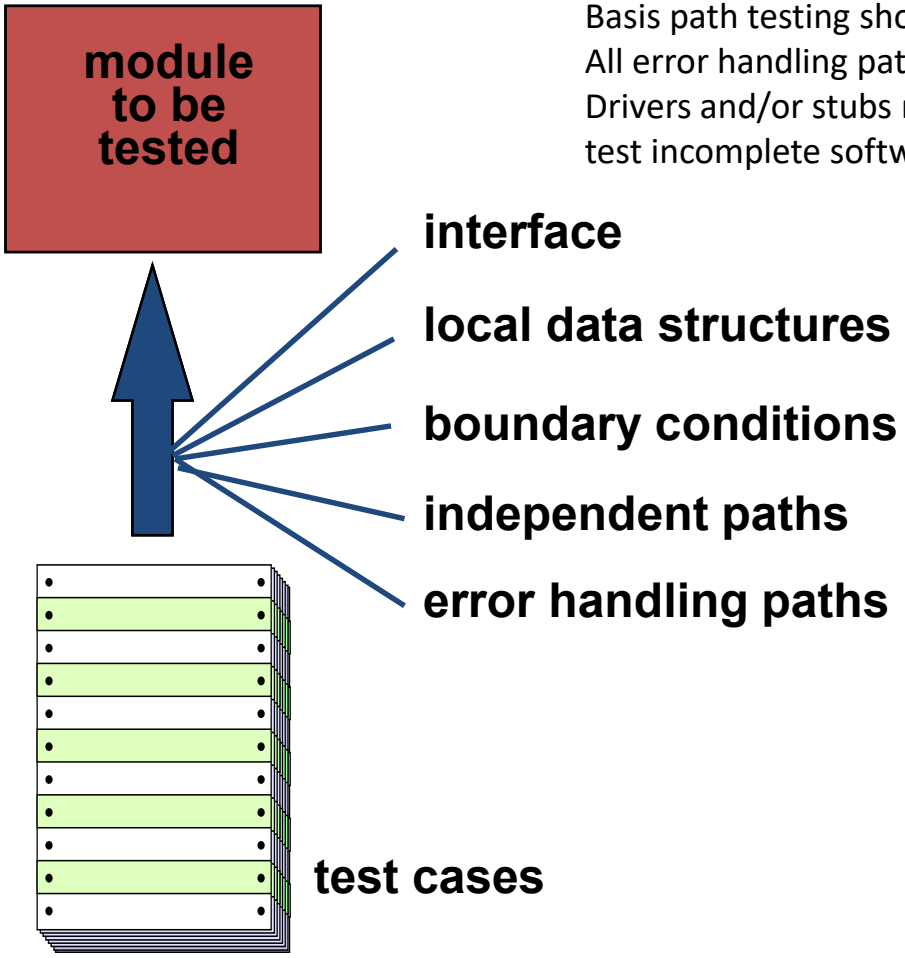
Input	Boundary Cases
A number N such that: -99 <= N <= 99	-100, -99, -98 -10, -9 -1, 0, 1 9, 10 98, 99, 100
Phone Number Area code: [200, 999] Prefix: (200, 999] Suffix: Any 4 digits	Area code: 199, 200, 201 Area code: 998, 999, 1000 Prefix: 200, 199, 198 Prefix: 998, 999, 1000 Suffix: 3 digits, 5 digits

Unit Testing

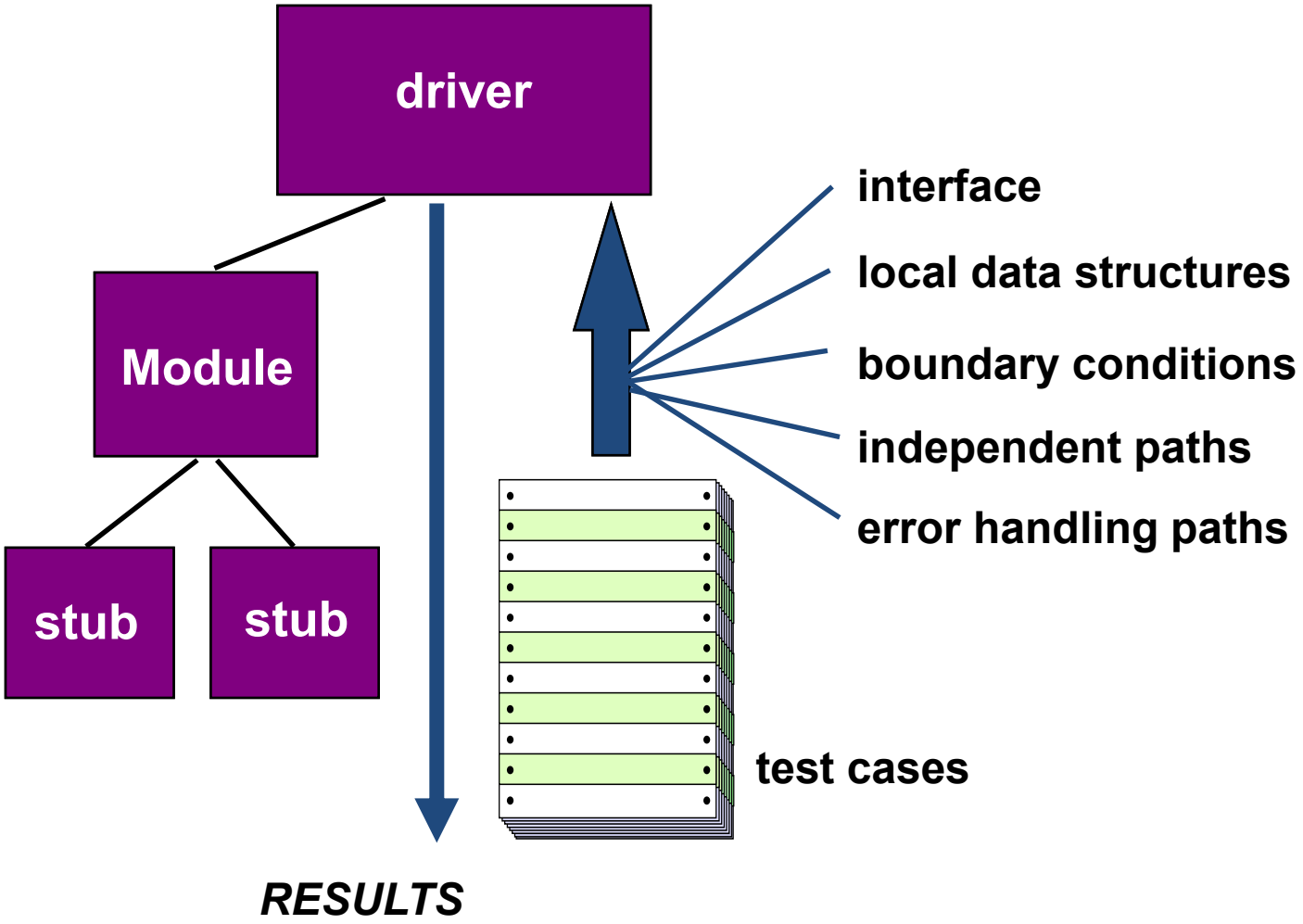


Unit Testing

Interfaces tested for proper information flow.
Local data are examined to ensure that integrity is maintained.
Boundary conditions are tested.
Basis path testing should be used.
All error handling paths should be tested.
Drivers and/or stubs need to be developed to test incomplete software.



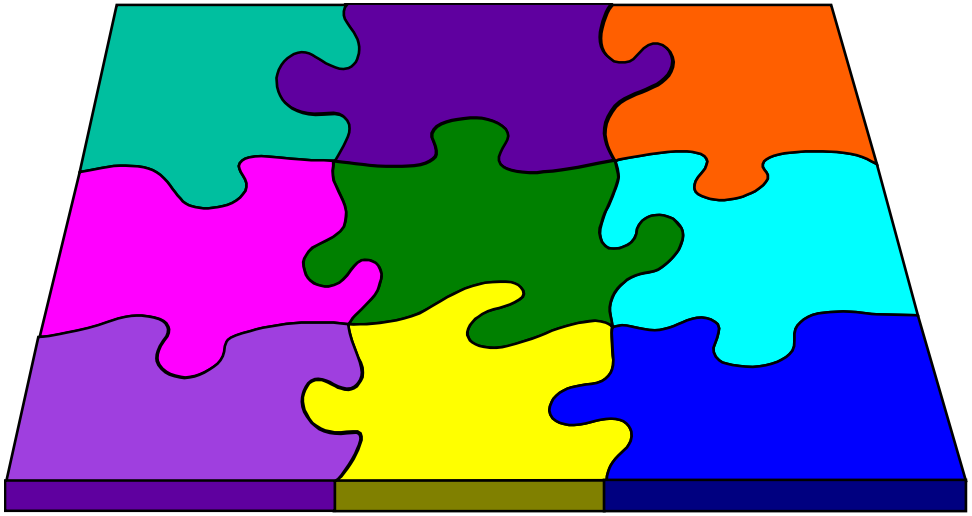
Unit Test Environment



Integration Testing Strategies

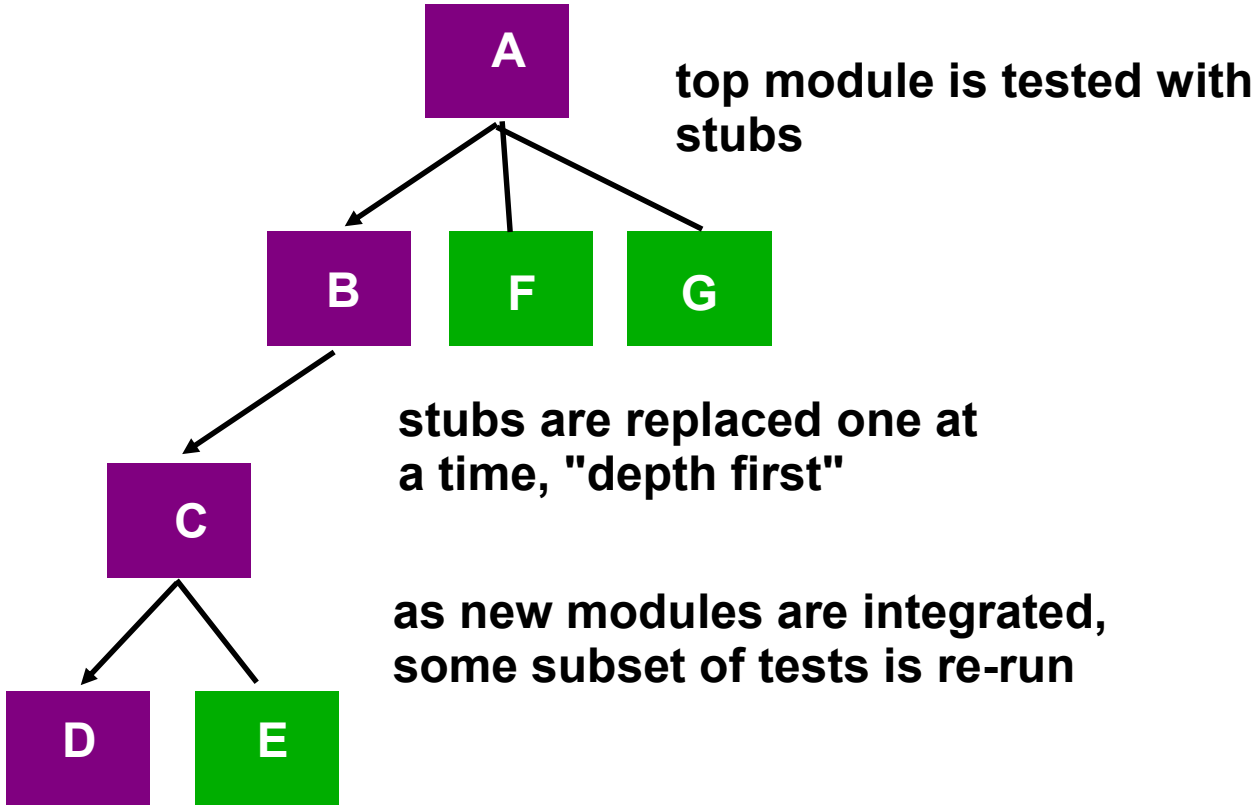
Options:

- **the “big bang” approach**
- **an incremental construction strategy**



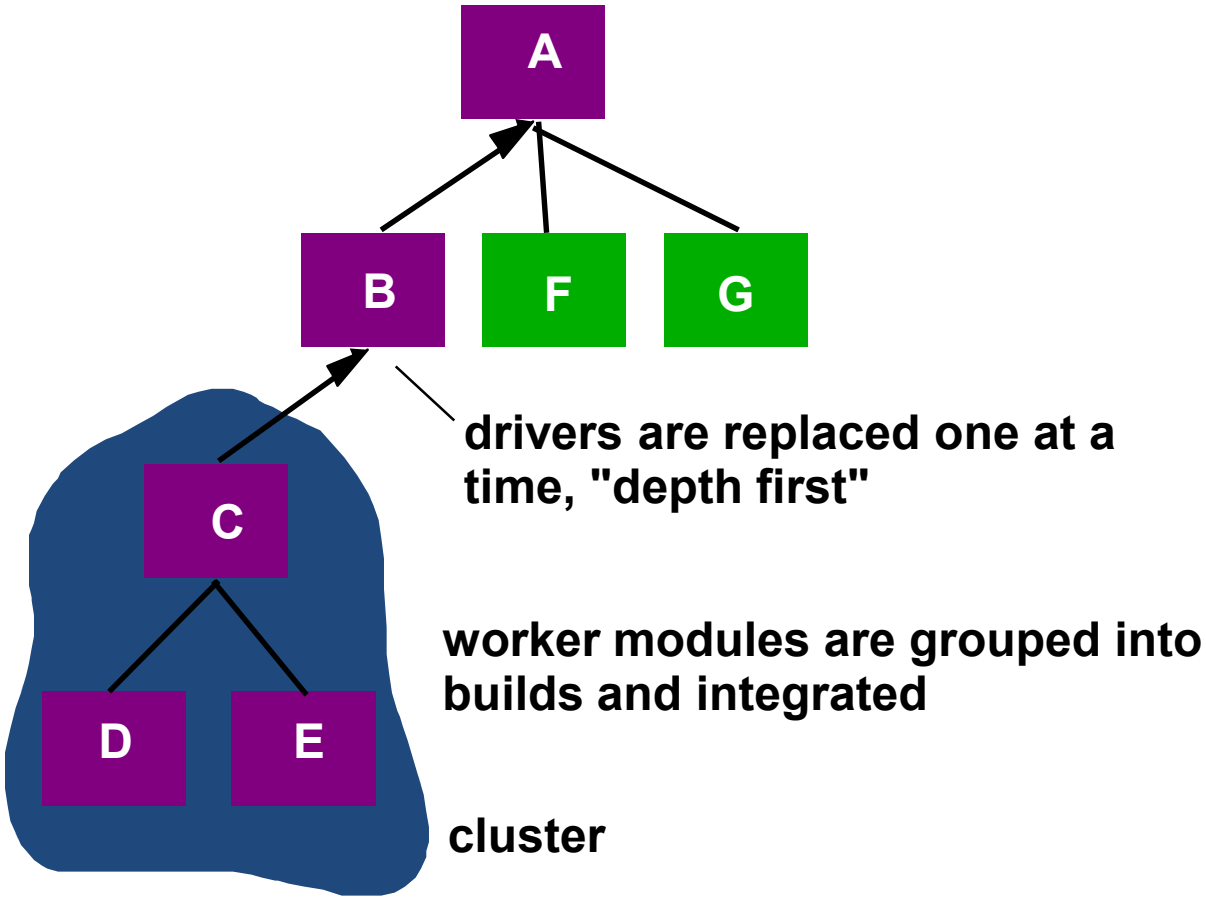
Top Down Integration

- Main program used as a test driver and stubs are substitutes for components directly subordinate to it.
- Subordinate stubs are replaced one at a time with real components (following the depth-first or breadth-first approach).
- Tests are conducted as each component is integrated.
- On completion of each set of tests and other stub is replaced with a real component.
- Regression testing may be used to ensure that new errors not introduced.



Bottom-Up Integration

- Low level components are combined in clusters that perform a specific software function.
- A driver (control program) is written to coordinate test case input and output.
- The cluster is tested.
- Drivers are removed and clusters are combined moving upward in the program structure.



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 behavior or additional errors.
- Regression testing may be conducted manually, by re-executing a subset of all test cases or using automated capture/playback tools.

VALIDATION TESTING

- Determine if the software meets all of the requirements defined in the SRS
- Having written requirements is essential
- Regression testing is performed to determine if the software still meets all of its requirements in light of changes and modifications to the software
- Regression testing involves selectively repeating existing validation tests, not developing new tests

Alpha and Beta Testing:

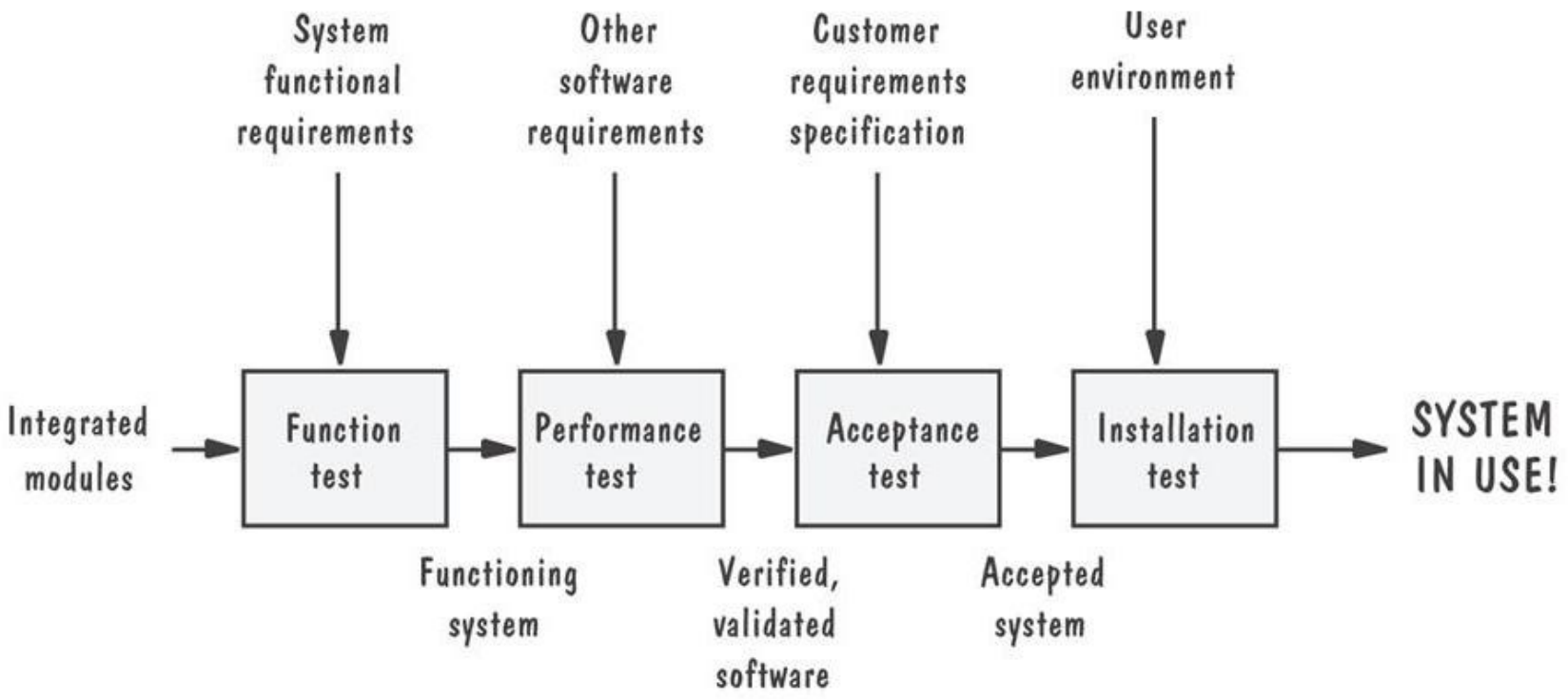
- It’s best to provide customers with an outline of the things that you would like them to focus on and specific test scenarios for them to execute.
- Provide with customers who are actively involved with a commitment to fix defects that they discover.

System Testing

- The objective of system testing is to ensure that the software does what the customer wants it to do.
- System Testing Process involves the following steps:
  - Function testing: does the integrated system perform as promised by the requirements specification?
  - Performance testing: are the non-functional requirements met?
  - Acceptance testing: is the system what the customer expects?
  - Installation testing: does the system run at the customer site(s)?

Cont..

- Pictorial representation of steps in testing process



Performance Tests

Types of Performance Tests

- Stress tests
- Volume tests
- Configuration tests
- Compatibility tests
- Regression tests
- Security tests
- Timing tests
- Environmental tests
- Quality tests
- Recovery tests
- Maintenance tests
- Documentation tests
- Human factors (usability) tests

Acceptance Tests

Types of Acceptance Tests

- Pilot test: install on experimental basis
- Alpha test: in-house test
- Beta test: customer pilot
- Parallel testing: new system operates in parallel with old system

Installation Testing

- Before the testing
  - Configure the system
  - Attach proper number and kind of devices
  - Establish communication with other system
- The testing
  - Regression tests: to verify that the system has been installed properly and works

Test Life Cycle

- Establish test objectives.
- Design criteria (review criteria).
    - Correct.
    - Feasible.
    - Coverage.
    - Demonstrate functionality.
- Writing test cases.
- Testing test cases.
- Execute test cases.
- Evaluate test results

Error Code Design

ONLINE LIBRARY MANAGEMENT SYSTEM

14. ERROR CODE DESIGN

These are the anticipated errors and the accompanying messages.

Error #	Error Description	Name of Form
1	Invalid loginid Or password	Login
2	ERROR: Same Book id	Book Transaction
3	ERROR: Book id already present	Book Acquisition
4	ERROR: User id already present	Registration
5	ERROR: Missing Field	Purchase Request
6	ERROR: Missing Field	Registration
7	ERROR: Missing Field	Book Acquisition
8	ERROR: Missing Field	Book Issue
9	ERROR: Missing Field	Book Return
10	ERROR: Missing Field	Issue Details
11	ERROR: Missing Field	Fine Details
12	ERROR: Missing Field	Change Password
13	ERROR: Missing Field	Search

Designing the test case

ONLINE LIBRARY MANAGEMENT SYSTEM

15. DESIGN OF TEST CASES

Test Case #	Test	Expected Response
1. Login: Login Form		
1.1	Enter a zero length user id	Enter Your Userid
1.2	Enter a zero length password	Enter Your Password
1.3	Enter correct login name and password	User logged into system. Login form disposed.
2.Book Transaction : Book Transaction Form		
2.1	Enter same book id for more than one transaction.	ERROR:Same book id
2.2	Enter Book Transaction without filling member id	ERROR:Missing Field
3.Book Acquisition: Book Acquisition Form		
3.1	Enter already present book id for new book.	ERROR: Same book id
3.2	Enter without filling any field and submit	ERROR: Missing Field
4.Registration: Registration Form		
4.1	Enter already present member id for new member.	ERROR: Same member id
4.2	Enter without filling any field and submit	ERROR: Missing Field
5. Search: Book Search Form		
5.1	To search without any search criteria	ERROR:Missing Field
6. Purchase Request: Purchase Request Form		
6.1	To send a request without filling any field	ERROR: Missing Field.
6.2	To leave some field empty and submit	ERROR: Missing Field.
7. Issue Details: Issue Details Form		
7.1	To leave the user id field empty & submit	ERROR: Missing Field.
8. Fine Details: Fine Details Form		
8.1	To leave the user id field empty & submit	ERROR: Missing Field.

Strategic Approach to Testing

- Testing begins at the component level and works outward toward the integration of the entire computer-based system.
- Different testing techniques are appropriate at different points in time.
- The developer of the software conducts testing and may be assisted by independent test groups for large projects.
- The role of the independent tester is to remove the conflict of interest inherent when the builder is testing his or her own product.

Strategic Approach to Testing

- Testing and debugging are different activities.
- Debugging must be accommodated in any testing strategy.
- Need to consider verification issues
  - are we building the product right?
- Need to Consider validation issues
  - are we building the right product?

Strategic Testing Issues - 1

- Specify product requirements in a quantifiable manner before testing starts.
- Specify testing objectives explicitly.
- Identify the user classes of the software and develop a profile for each.
- Develop a test plan that emphasizes rapid cycle testing.

Strategic Testing Issues - 2

- Build robust software that is designed to test itself (e.g. use anti-bugging).
- Use effective formal reviews as a filter prior to testing.
- Conduct formal technical reviews to assess the test strategy and test cases.

Debugging

- Debugging (removal of a defect) occurs as a consequence of successful testing.
- Some people better at debugging than others.
- Is the cause of the bug reproduced in another part of the program?
- What “next bug” might be introduced by the fix that is being proposed?
- What could have been done to prevent this bug in the first place?

Debugging Approaches

- Brute force
  - memory dumps and run-time traces are examined for clues to error causes
- Backtracking
  - source code is examined by looking backwards from symptom to potential causes of errors
- Cause elimination
  - uses binary partitioning to reduce the number of locations potential where errors can exist