MCSE 652 Software Quality Assurance

Chapter 9: Software testing techniques and strategies

Dr. Mehedi Hasan
Stamford University Bangladesh

Learning objectives

- Testing fundamentals
- Describe the processes of software testing
- Describe the general characteristics of strategic testing
- Introduce a range of testing techniques

The Software testing process has 2 distinct objectives –

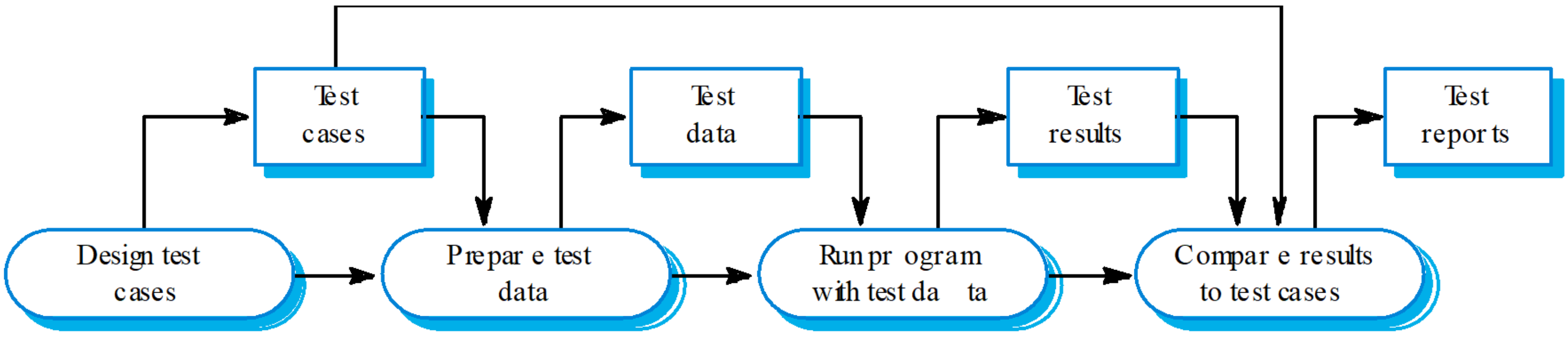
Testing is a process of executing a program with the intent of finding an error.

1. *To demonstrate to the developer and the customer that the* ***software meets its requirement*** (validation testing)
2. ***To discover faults or defects*** *in the software where the* ***behavior*** *of the software is incorrect, undesirable or does not conform to its specification* (defect testing)

Basic Principles of Software Testing

- All tests should be traceable to customer requirements.
- Tests should be planned long before testing begins
- The Pareto principle (80:20 rule) applies to software testing.
- Testing should begin ***"in the small"*** and progress toward testing ***"in the large."***
- Exhaustive testing is not possible.
- To be most effective, testing should be conducted by an independent third party.

The software testing process



Testing Approaches

- Black Box testing
  - Knowing the specified function that a product has been designed to perform, tests can be conducted that demonstrate each function is fully operational while at the same time searching for errors in each function;
- White Box testing
  - knowing the internal workings of a product, tests can be conducted to ensure that all internal logics are performed according to specifications and all internal components have been adequately exercised.

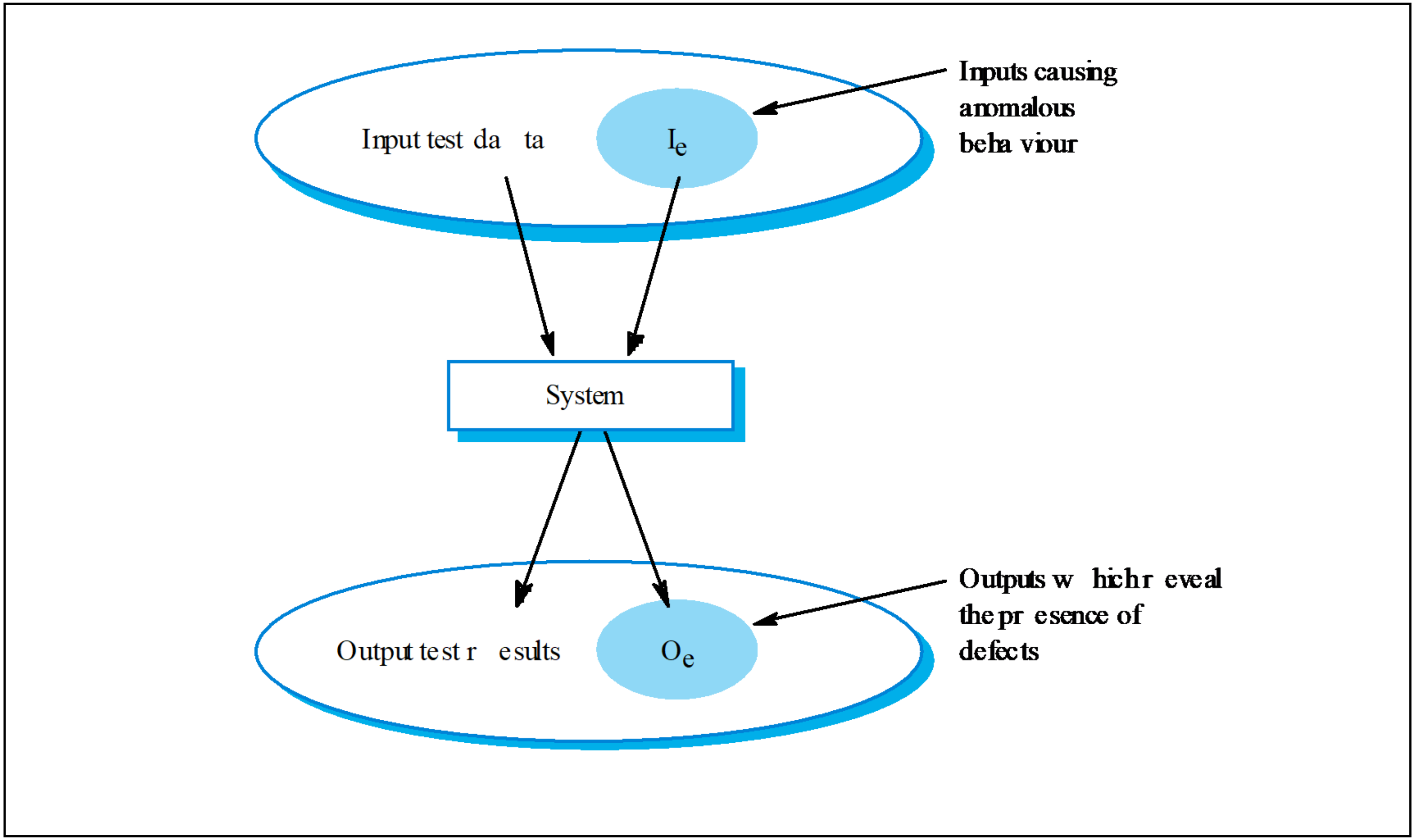
Black box testing

- Also known as *behavioral testing*
- Focuses more on the functional requirements of the software
- To determine if input is properly accepted and output is correctly produced
- It has little regard for the internal logical structure of the software and is applied at the later stages of software testing

Black box testing

- Black-box testing attempts to find errors in the following categories:
  - incorrect or missing functions,
  - interface errors,
  - errors in data structures or external database access
  - behaviour or performance errors,
  - System initialization and termination errors.

Black-box testing



Input test data

I_e

Inputs causing anomalous behaviour

System

Output test results

O_e

Outputs which reveal the presence of defects

White box testing

- 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.

- ***Although exhaustive testing is not possible***, an adequate number of important logical paths and data structures can be selected and validated

White box testing –specific examples

- Basis path testing
  - to derive a logical complexity measure of a procedural design and use this as guide for designing the basis set of execution paths (***execute every statement in the program at least one time during testing***), done with flow graph notations and cyclomatic complexity
- Control structure testing
  - exercises the logical conditions contained in a program module such as arithmetic expression error, data flow and loop testing.

Testing Strategy

Testing strategy

- A strategy for software testing integrates the design of software test cases into a well-planned series of steps that result in successful development of the software

- The strategy provides a road map that describes the steps to be taken, when, and how much effort, time, and resources will be required.

- The strategy incorporates
  - test planning,
  - test case design,
  - test execution,
  - test result collection and evaluation.

General Characteristics of Strategic Testing

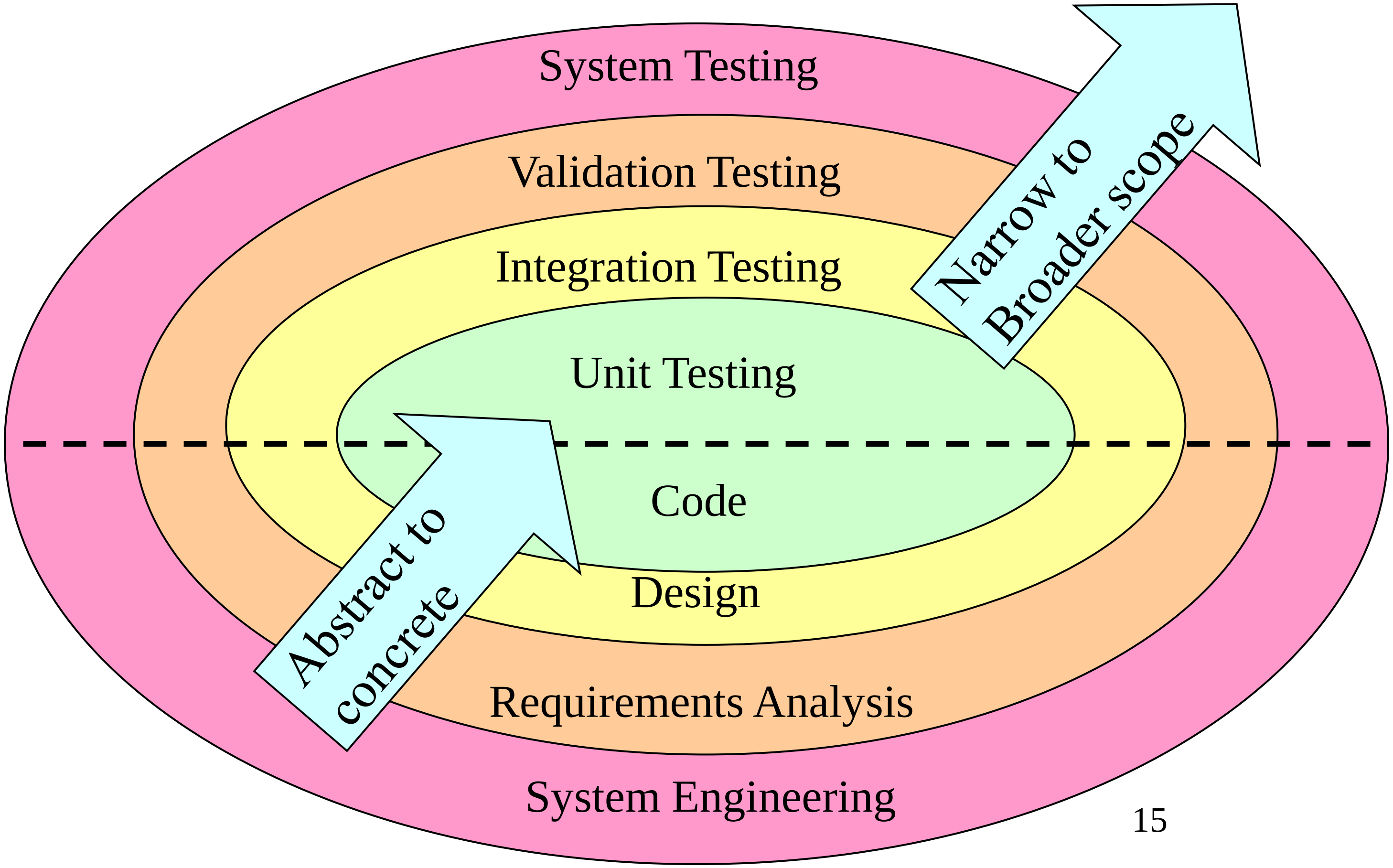
- Testing begins at the component level and work outward toward the integration of the entire computer-based system

- Different testing techniques are appropriate at different points in time

- Testing is conducted by the developer of the software and (for large projects) by an independent test group

- Testing and debugging are different activities, but debugging must be accommodated in any testing strategy

A Strategy for Testing Conventional Software



Levels of Testing for Conventional Software

1. Unit testing

   Concentrates on each component/function of the software as implemented in the source code

2. Integration testing

   Focuses on the design and construction of the software architecture

3. Validation testing

   Requirements are validated against the constructed software

4. System testing

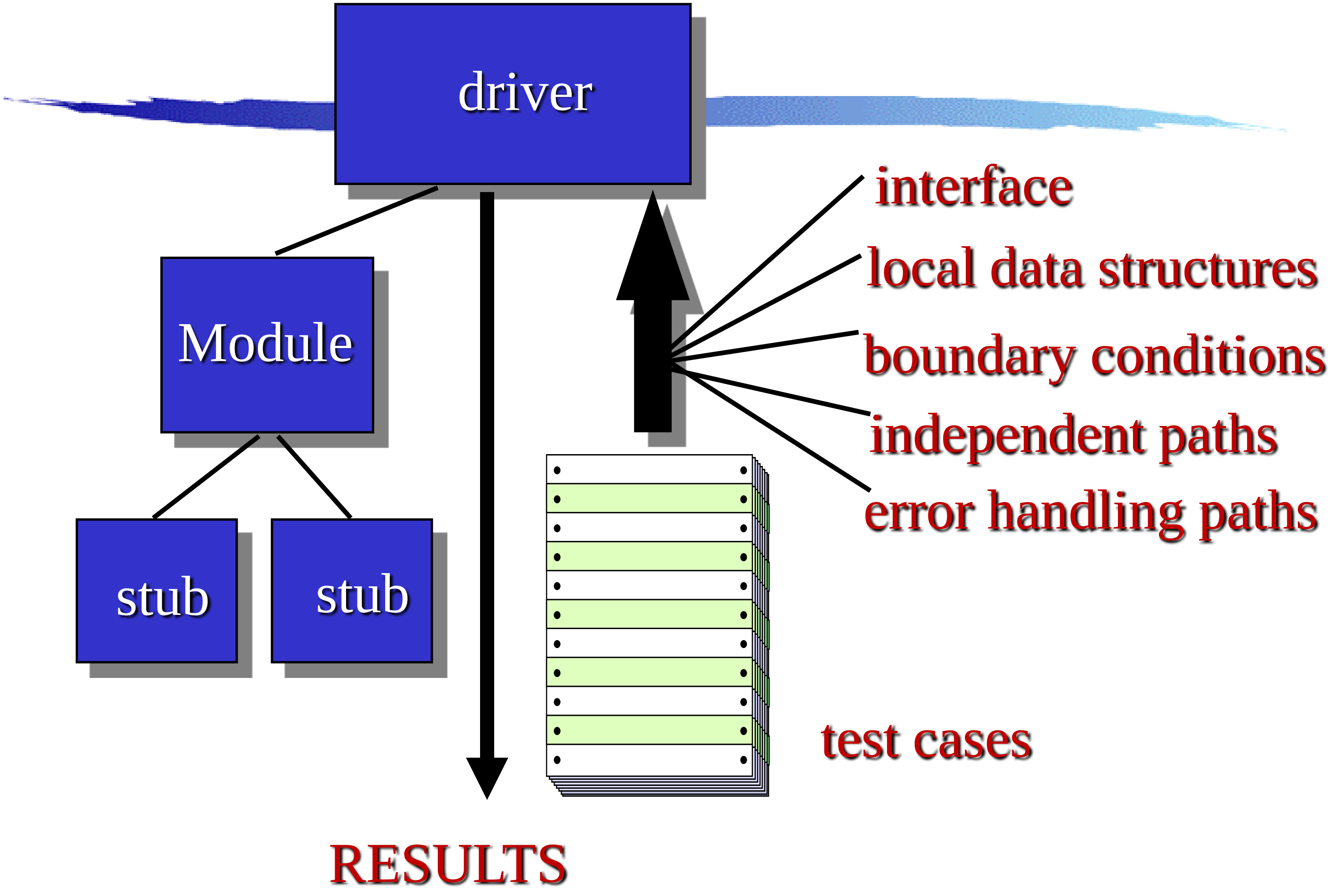
   The software and other system elements are tested as a whole

1. Unit Testing

- Focuses testing on the function or software module
- Concentrates on the internal processing logic and data structures
- Is simplified when a module is designed with high cohesion
  - Reduces the number of test cases
  - Allows errors to be more easily predicted and uncovered

- Concentrates on critical modules and those with **high complexity** when testing resources are limited

Unit Test Environment



driver

Module

stub

stub

interface

local data structures

boundary conditions

independent paths

error handling paths

test cases

RESULTS

Drivers and Stubs for Unit Testing

- **Driver**
  - A **simple main program** that accepts test case data, passes such data to the component being tested, and prints the returned results
- **Stubs**
  - Serve to replace modules that are subordinate to (called by) the component to be tested

  - It uses the module's exact interface, may do minimal data manipulation, provides verification of entry, and returns control to the module undergoing testing

Targets for Unit Test Cases

- Module interface
  - Ensure that information flows properly into and out of the module
- Local data structures
  - Ensure that data stored temporarily maintains its integrity during all steps in an algorithm execution
- Boundary conditions
  - Ensure that the module operates properly at boundary values established to limit or restrict processing
- Independent paths
  - Paths are exercised to ensure that all statements in a module have been executed at least once
- Error handling paths
  - Ensure that the algorithms respond correctly to specific error conditions

Common Computational Errors in Execution Paths

- Misunderstood or incorrect arithmetic precedence
- Mixed mode operations (e.g., int, float, char)
- Incorrect initialization of values
- Precision inaccuracy and round-off errors
- Incorrect symbolic representation of an expression (int vs. float)

Other Errors to Uncover

- Comparison of different data types
- Incorrect logical operators or precedence
- Expectation of equality when precision error makes equality unlikely (using == with float types)
- Incorrect comparison of variables
- Improper or nonexistent loop termination
- Failure to exit when divergent iteration is encountered
- Boundary value violations

2. Integration Testing

- Defined as a systematic technique for constructing the software architecture
  - At the same time integration is occurring, conduct tests to uncover errors associated with interfacing

Objective:

To take unit tested modules /components and build a program structure based on the prescribed design

Two Approaches

- Non-incremental Integration Testing
- Incremental Integration Testing

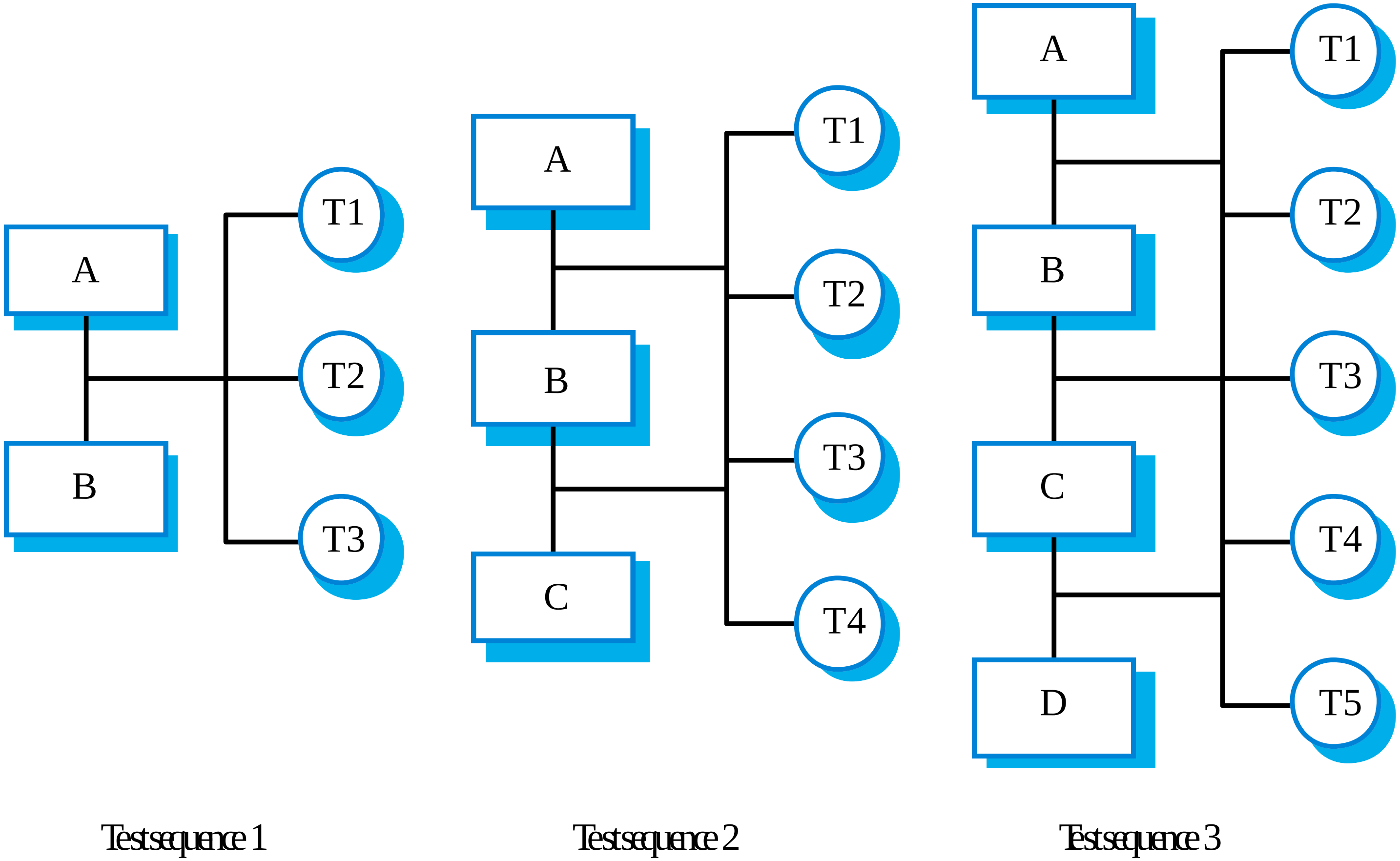
Non-incremental Integration Testing

- Commonly called the **“Big Bang”** approach
- All components are combined in advance
- The entire program is tested as a whole
- Chaos results
- Many seemingly-unrelated errors are encountered
- Correction is difficult because isolation of causes is complicated
- Once a set of errors are corrected, more errors occur, and testing appears to enter an endless loop

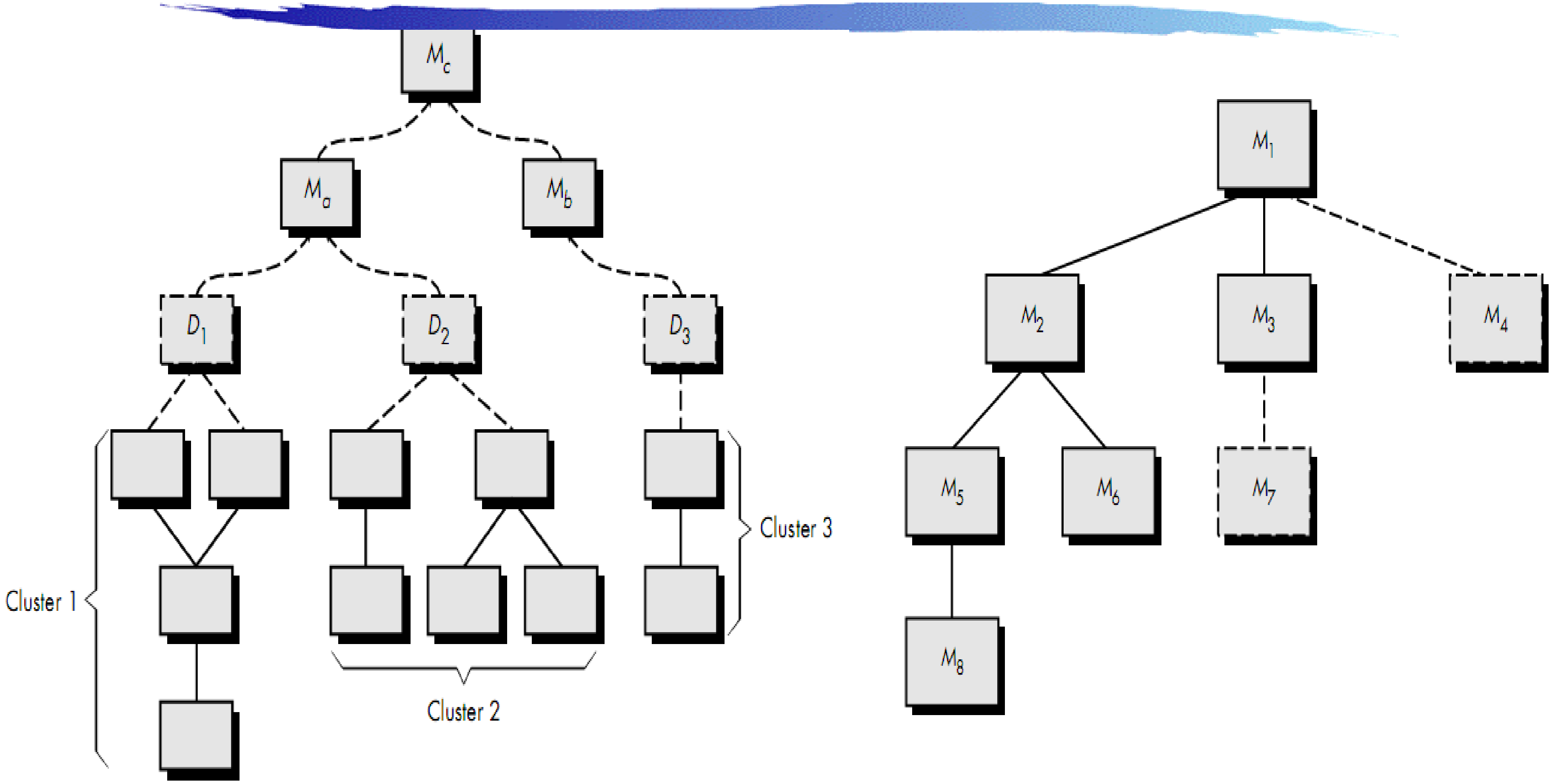
Incremental Integration Testing

- Three kinds
  - Top-down integration
    - Develop the skeleton of the system and populate it with components.
  - Bottom-up integration
    - Integrate infrastructure components then add functional components.
  - Sandwich integration
- The program is constructed and tested in small increments
- Errors are easier to isolate and correct
- Interfaces are more likely to be tested completely
- A systematic test approach is applied to simplify error localisation.

Incremental integration testing



Bottom up and Top down integration



Regression Testing

- Each new addition or change to base lined software may cause problems with functions that previously worked flawlessly

- Regression testing re-executes a small subset of tests that have already been conducted
  - Ensures that changes have not propagated unintended side effects
  - Helps to ensure that changes do not introduce unintended behavior or additional errors
  - May be done manually or through the use of automated capture/playback tools

3. Validation Testing

- Focuses on **user-visible actions and output** from the system
- Achieved through a **series of black box testing** that demonstrates conformity with requirements
- Designed to ensure that
  - All functional requirements are satisfied
  - All behavioral characteristics are achieved
  - All performance requirements are attained
  - Documentation is correct
  - Usability and other requirements are met (e.g., transportability, compatibility, error recovery, maintainability)

3. Validation Testing

- **After each validation test, one of the two possible conditions exist:**

1. The function or performance characteristic conforms to specification and is accepted

2. A deviation from specification is uncovered and a deficiency list is created

Validation through Acceptance testing

It is virtually impossible for software developers to foresee how the customer will really use the system

- Instructions may be misinterpreted
- Strange input data may be used
- Output that is clear to the tester may seem vague and confusing to users in the field

Acceptance testing

- When **custom software** is built for one customer, a series of acceptance tests are conducted to enable the customer to validate all requirements.

- Most software product builders use a process called **alpha and beta testing** to ***uncover errors that only the end-user seems able to find***.

Alpha Testing

- Conducted at the developer’s site by a customer
- Software is used in a natural setting with the developers watching intently and recording errors and usage problems
- Testing is conducted in a controlled environment

Beta Testing

- Conducted at one or more customers’ site
- Developer is generally not present
- It serves as a live application of the software in an environment that cannot be controlled by the developer
- The customer records all problems that are encountered and reports these to the developers at regular intervals

After beta testing is complete, software engineers make software modifications and prepare for release of the software product to the entire customer base

4. **System testing**

- Software is most of the time, one element of a larger computer-based system ***(like system software, video and image processing software in iPhone, inbuilt video screening software in aircraft, GPS software in cars)***
- Software is incorporated with other system elements (hardware, people and information)
- Many stakeholders for system testing
  - Classic problem was "finger pointing"

4. **System testing**

- It is a black box testing to validate the overall system accuracy and completeness in performing the function as designed or specified.
- Must ensure that all unit and integration test results are reviewed and that all problems are resolved.
- Important to understand unresolved problems that originate at unit or integration test levels

Types of system testing

i. Recovery testing

ii. Security testing

iii. Stress testing

iv. Performance testing

i. Recovery testing

- Tests for recovery from system faults
- Forces the software to fail in a variety of ways and verifies that recovery is properly performed
- If recovery is automatic, reinitialization, check pointing mechanisms, data recovery, and restart are evaluated for correctness

ii. Security testing

Verifies that protection mechanisms built into a system to protect it from improper access by hackers and frauds

- During security testing, tester plays the role(s) of an individual who desires to penetrate the system

- ***Tester attempt to get the password, use custom-designed software to breakdown the access protection, cause system error, penetrate during recovery time and use insecure data.***

iii. Stress testing

- Stressing the system often causes defects to come to light.

- It is to test the system with abnormal situations by executing in a manner that *demands resources in abnormal quantity, frequency, and volume.*
- Exercises the system beyond its maximum design load:
  - special tests may be designed to 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

iii. Stress testing

- Stressing the system test failure behaviour.
- Systems should not fail catastrophically. Stress testing checks for unacceptable loss of service or data.
- Stress testing is particularly relevant to distributed systems that can exhibit severe degradation as a network becomes overloaded.

iv. Performance testing

- 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. However, it is not until all system elements are fully integrated that the true performance of a system can be ascertained.

iv. Performance testing

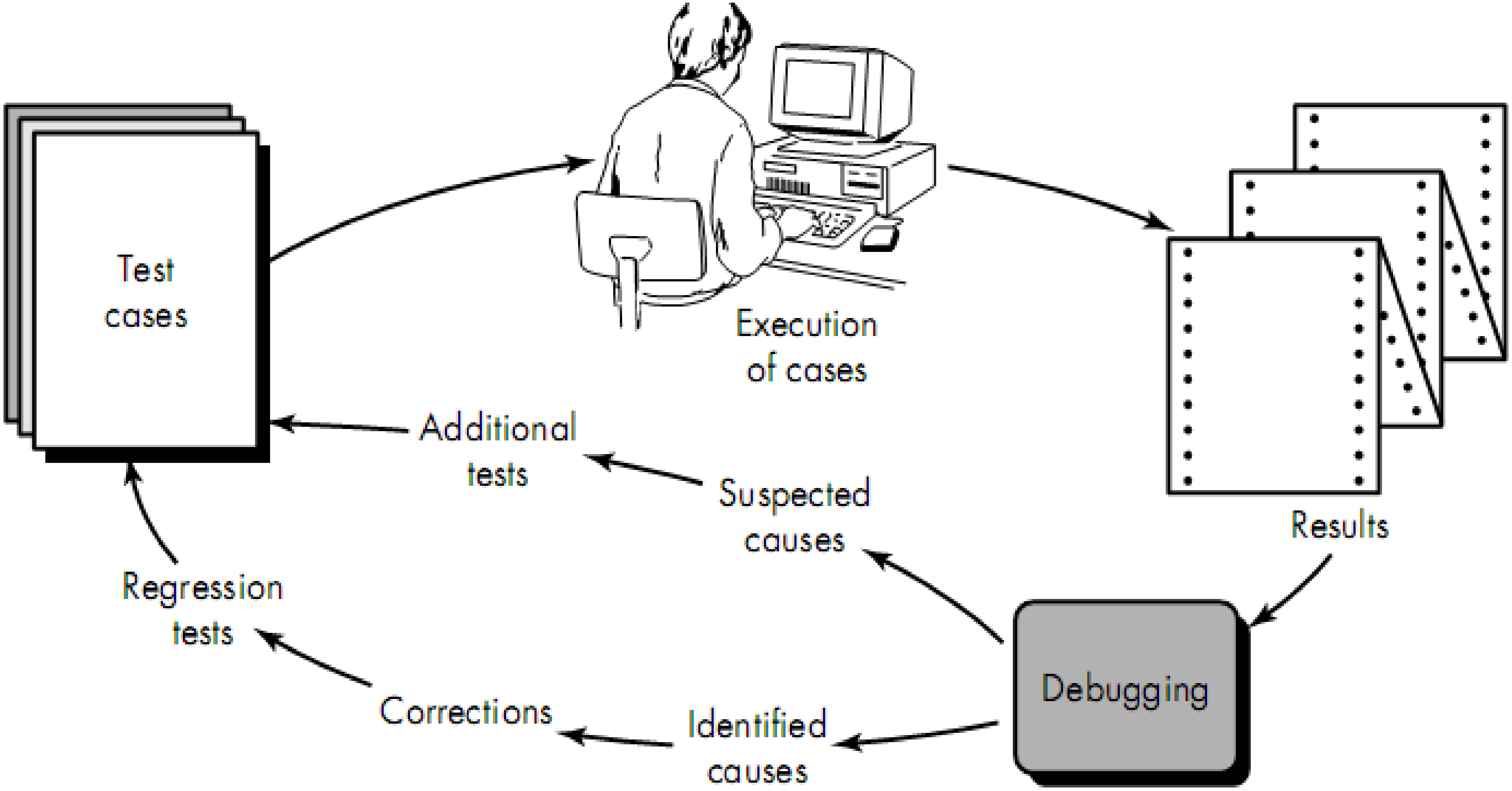
- Performance testing is often coupled with stress testing and it requires both software and hardware instrumentation

- ***Performance tests can uncover situations that lead to degradation and possible system failure by monitoring resource utilization, execution intervals (interrupts) and event logs on a regular basis***

Debugging

- Occurs as a consequence of successful testing. ***i.e. when a test case uncovers an error,***
  debugging is the process that results in the removal of the error.

Debugging process



Debugging Process

- Debugging occurs as a consequence of successful testing
- It is still very much an art rather than a science

- The debugging process begins with the execution of a test case
- Results are assessed and the difference between expected and actual performance is encountered
- This difference is a symptom of an underlying cause that lies hidden
- The debugging process attempts to match symptom with cause, thereby leading to error correction

Why is Debugging so Difficult?

- The symptom and the cause may be geographically remote. *i.e., the symptom may appear in one part of a program, while the cause may actually be located at a site that is far removed.* ***Highly coupled program*** *structures exacerbate this situation.*
- It may be **difficult to accurately reproduce** input conditions, such as asynchronous real-time information
- The symptom may be **irregular** such as in embedded systems involving both hardware and software
- The symptom may be due to causes that are **distributed** across a number of tasks running on different processes

Debugging Approaches

Objective of debugging is to find and correct the cause of a software error

Although debugging can and should be an orderly process, it is still very much an art.

- Debugging methods and tools are not a substitute for careful evaluation based on a complete design model and clear source code

- There are three main debugging approaches
  - Brute force
  - Backtracking
  - Cause elimination