Testing types, levels and techniques

Software Testing

261449 / 269496

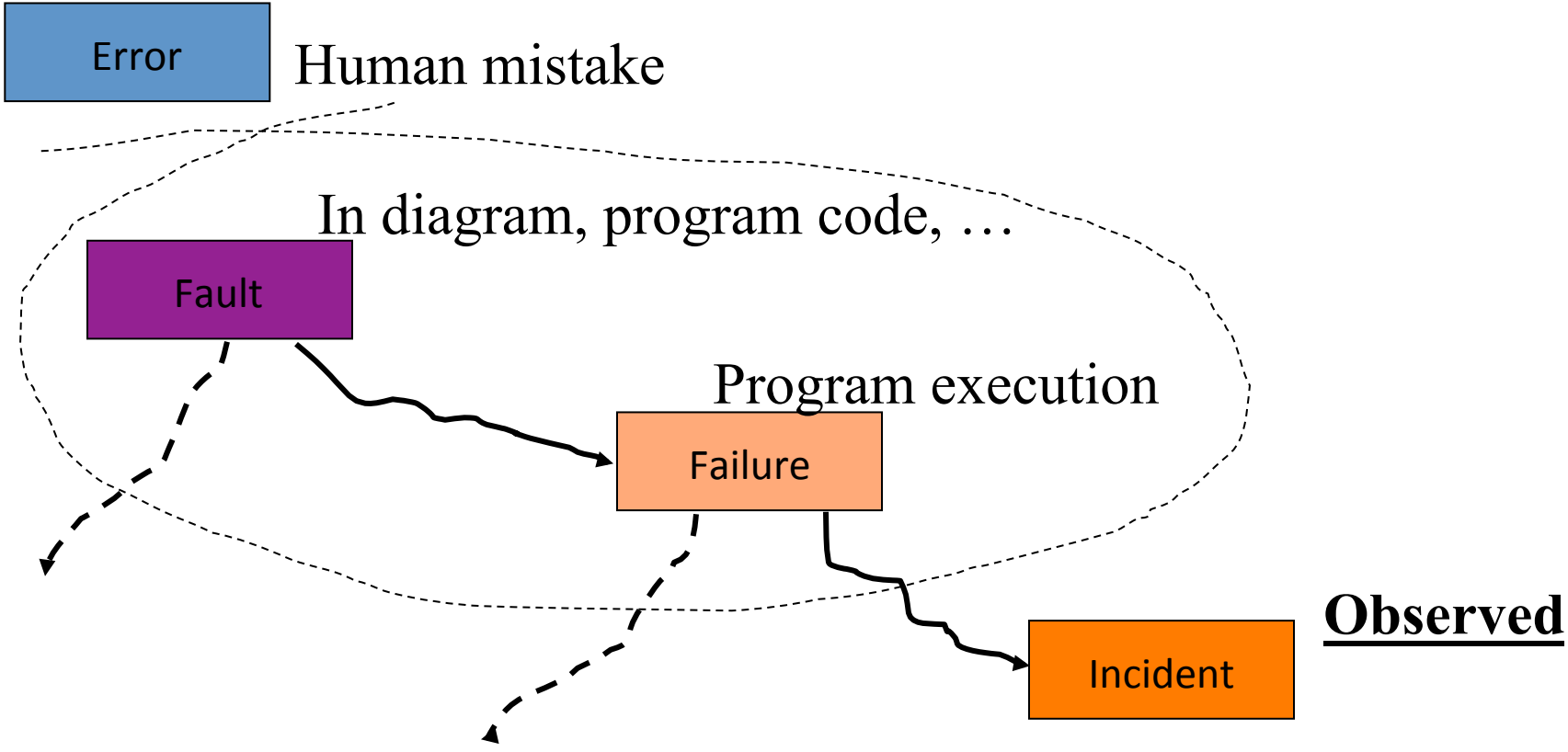
Software Testing

- Course outline
  - Basic principles of software testing
  - **Test levels and types**
  - **Test case design**
  - Supporting tools
  - Testing process.
  - Test planning.
  - Writing test report

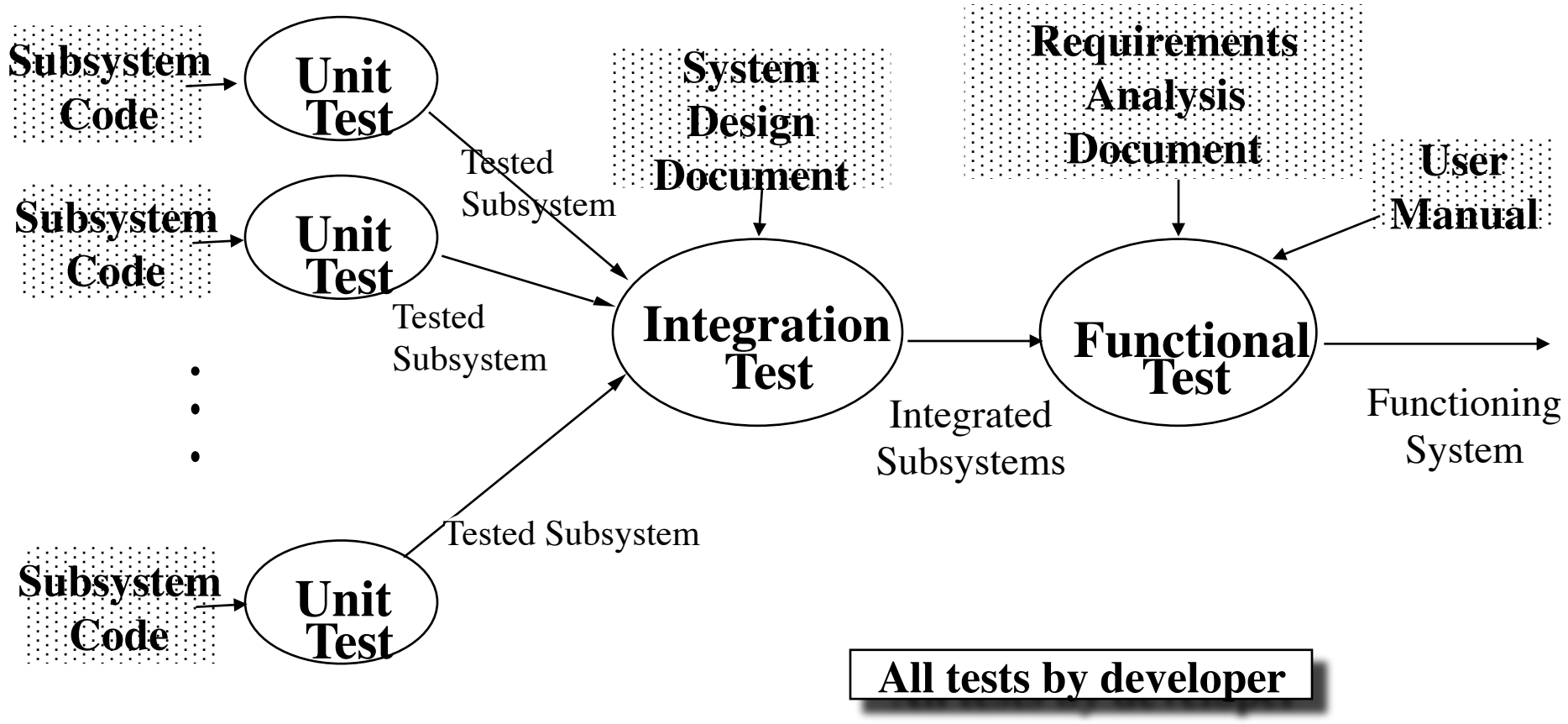
Review

- Software defects are costly and, therefore, needs to be prevented
- Software testing is the process to ensure that software has no defect and, thus, is good quality
- Different software development models pose different perspective to software testing

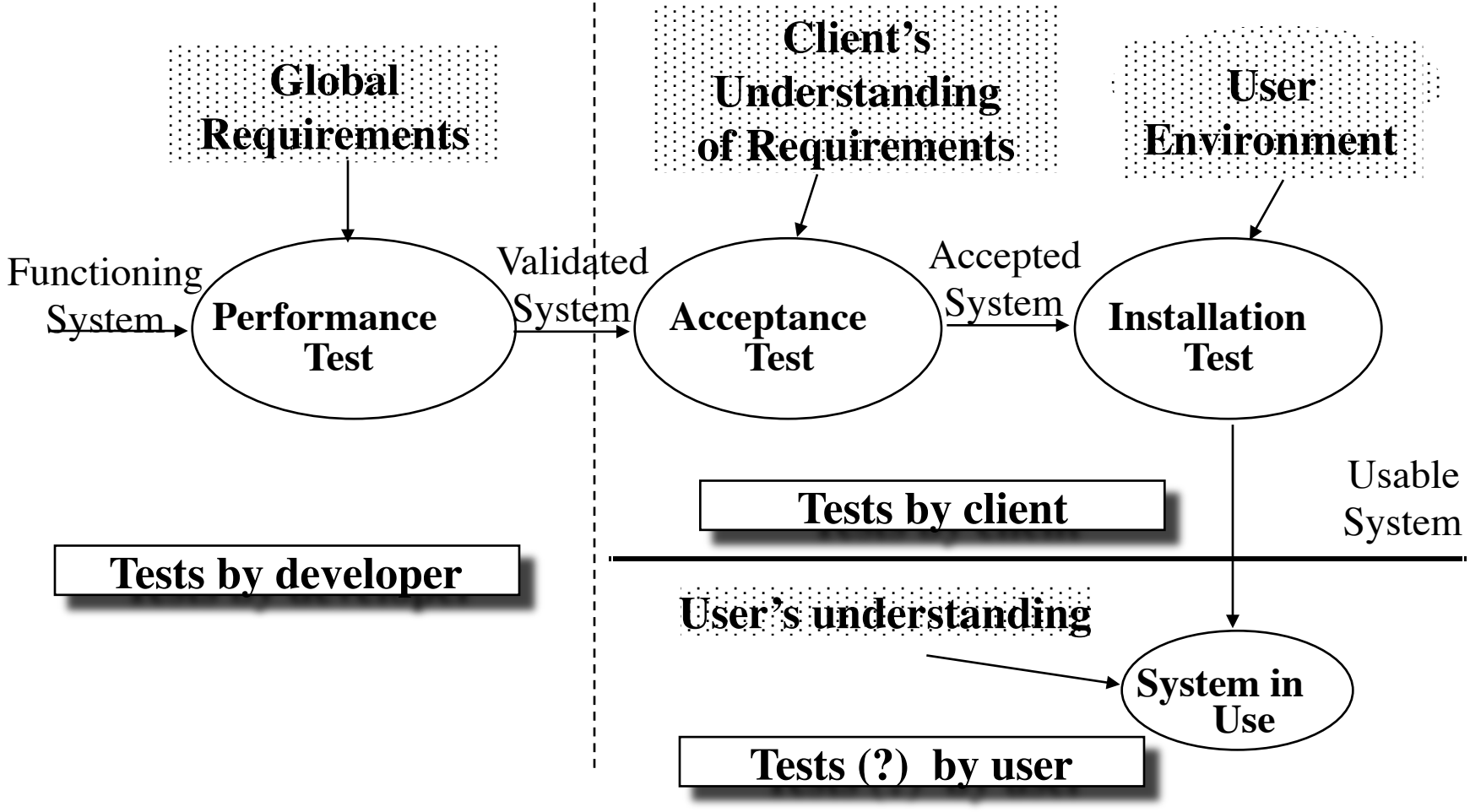
Review


Error
Human mistake
In diagram, program code, …
Fault
Program execution
Failure
Incident
Observed

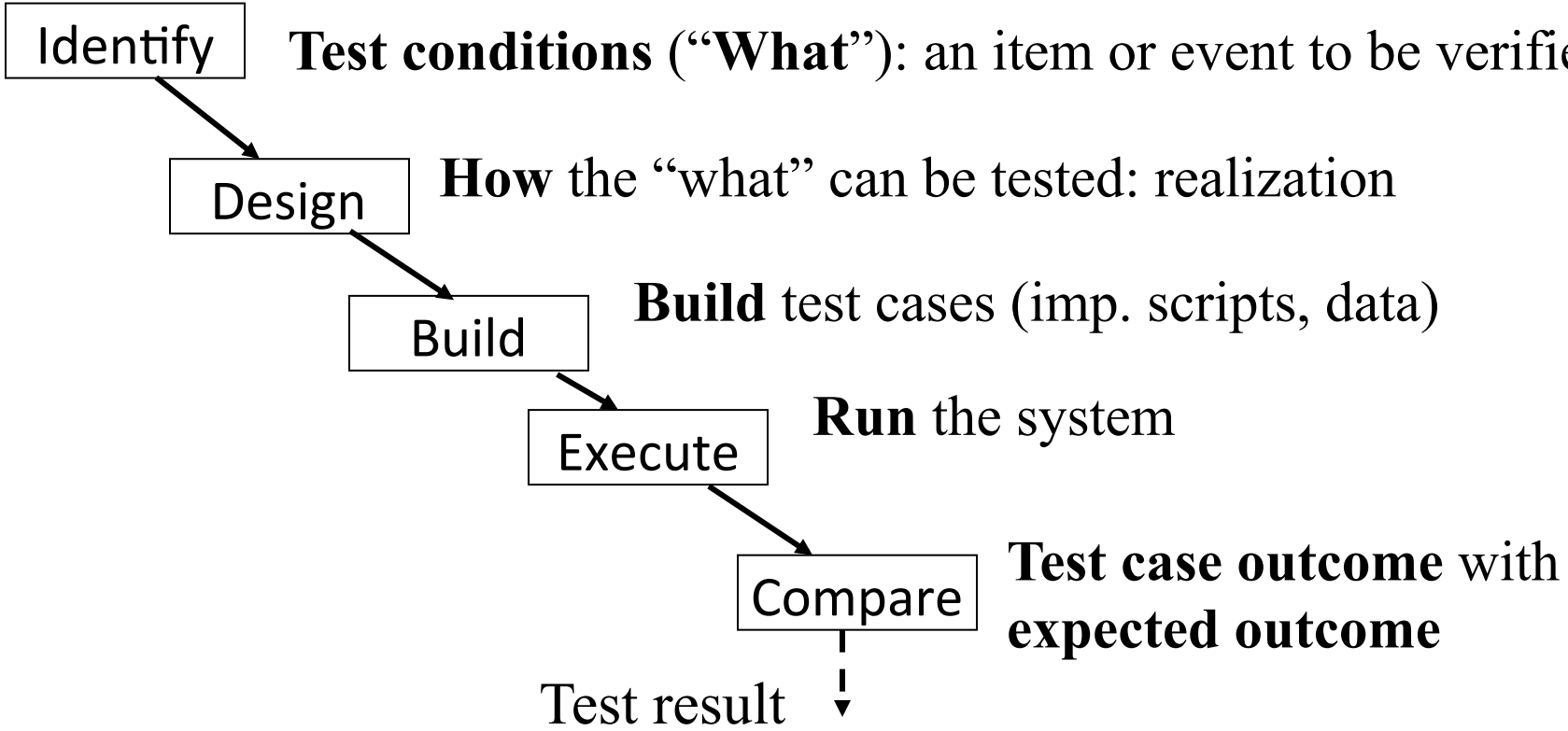
Testing Activities



Testing Activities continued



Testing Activities



Testing Activities

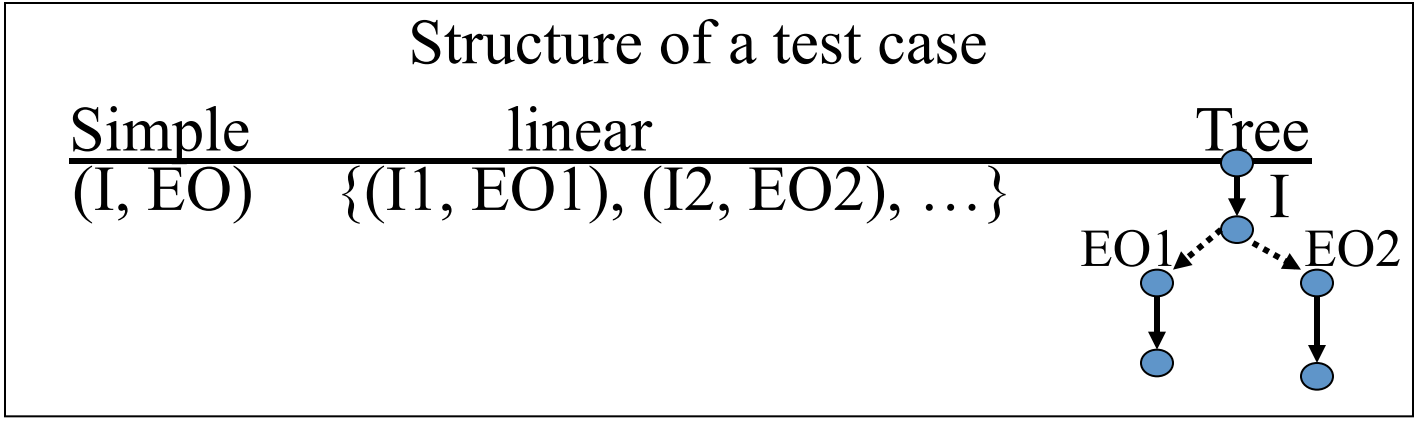
- Test condition
  - **What**: Descriptions of **circumstances** that could be examined (**event** or **item**).
  - **Categories**: functionality, performance, stress, robustness…
  - **Derive**
    - Using **testing techniques (to be discussed)**
    - (Refer to the V-Model)

Testing Activities

- Design **test cases**: the details
  - Input values
  - Expected outcomes
    - Things created (output)
    - Things changed/updated ➔ database?
    - Things deleted
    - Timing
    - …
  - Environment prerequisites: file, net connection …

Testing Activities

- Build test cases (implement)
  - Implement the **preconditions** (set up the environment)
  - Prepare test **scripts** (may use test automation tools)



Testing Activities

- **Scripts** contain data and instructions for testing
  - Comparison information
  - What screen data to capture
  - When/where to read input
  - Control information
    - Repeat a set of inputs
    - Make a decision based on output
  - Testing concurrent activities

Testing Activities

- **Compare** (test outcomes, expected outcomes)
  - Simple/complex
  - Different types of outcomes
    - Variable values (in memory)
    - Disk-based (textual, non-textual, database, binary)
    - Screen-based (char., GUI, images)
    - Others (multimedia, communicating apps.)

Testing Activities

- Compare:
  actual output == expected output??
  - Yes
    - Pass (Assumption: Test case was “instrumented.”)
  - No
    - Fail (assuming that there is no error in test case, preconditions)

Overview

- Basics of Testing
- Testing & Debugging Activities
- ➔ Testing Strategies
  - Black-Box Testing
  - White-Box Testing
- Testing in the Development Process
  - Unit Test
  - Integration Test
  - System Test
  - Acceptance Test
  - Regression Test
- Practical Considerations

Goodness of test cases

- Exec. of a **test case** against a **program** P
  - **Covers** certain **requirements** of P;
  - **Covers** certain parts of P's **functionality**;
  - **Covers** certain parts of P's **internal logic**.

➔ **Idea of *coverage* guides test case selection**.

Black-box Testing

- Focus: I/O behavior. If for any given input, we can predict the output, then the module passes the test.
  - Almost always impossible to generate all possible inputs ("test cases")

- Goal: Reduce number of test cases by equivalence partitioning:
  - Divide input conditions into equivalence classes
  - Choose test cases for each equivalence class. (Example: If an object is supposed to accept a negative number, testing one negative number is enough)

White-box Testing

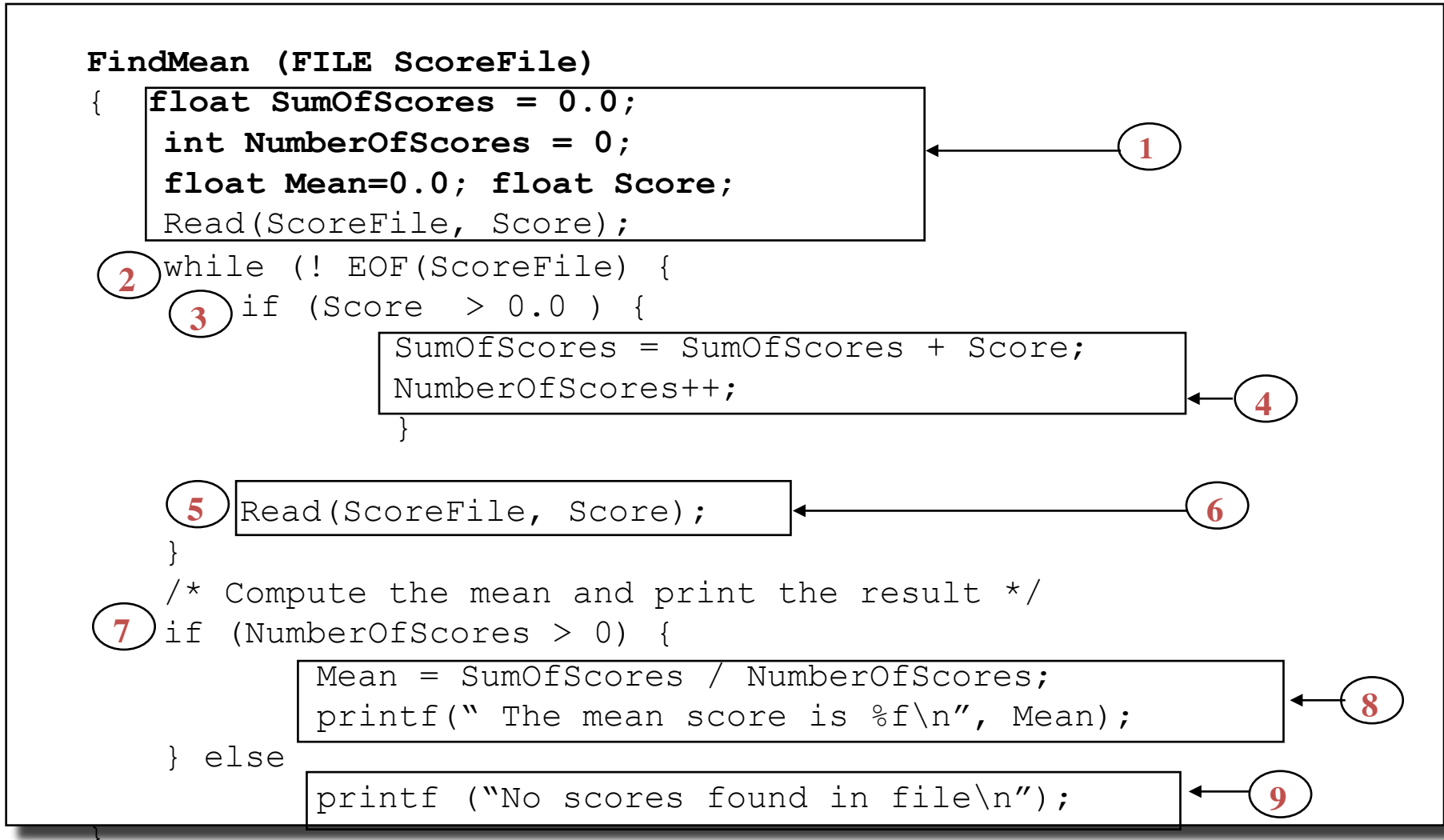
- Statement Testing: Test single statements
- Loop Testing:
  - Cause execution of the loop to be skipped completely. (Exception: Repeat loops)
  - Loop to be executed exactly once
  - Loop to be executed more than once
- Path testing:
  - Make sure all paths in the program are executed
- Branch Testing (Conditional Testing): Make sure that each possible outcome from a condition is tested at least once

```
if ( i =TRUE) printf("YES\n"); else printf("NO\n");
Test cases: 1) i = TRUE; 2) i = FALSE
```

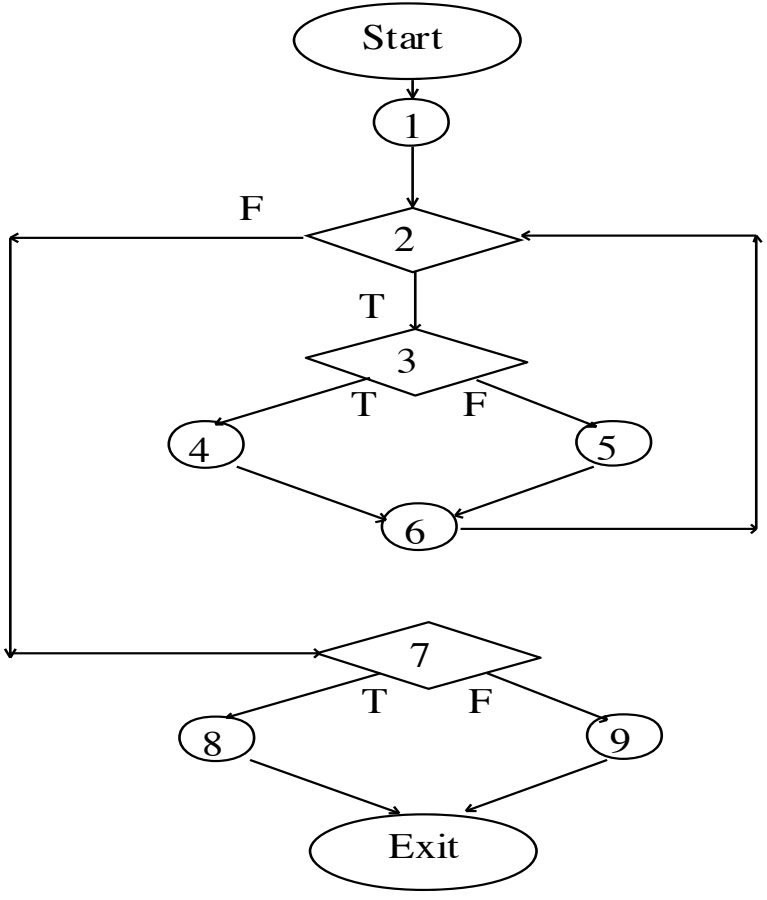
Code Coverage

- **Statement coverage**
  - Elementary statements: assignment, I/O, call
  - Select a test set T such that by executing P in all cases in T, **each statement of P is executed at least once.**
  - **read(x); read(y);**
    if x > 0 then write("1");
    else write("2");
    if y > 0 then write("3");
    else write("4");
  - **T: {<x = -13, y = 51>, <x = 2, y = -3>}**

White-box Testing: Determining the Paths



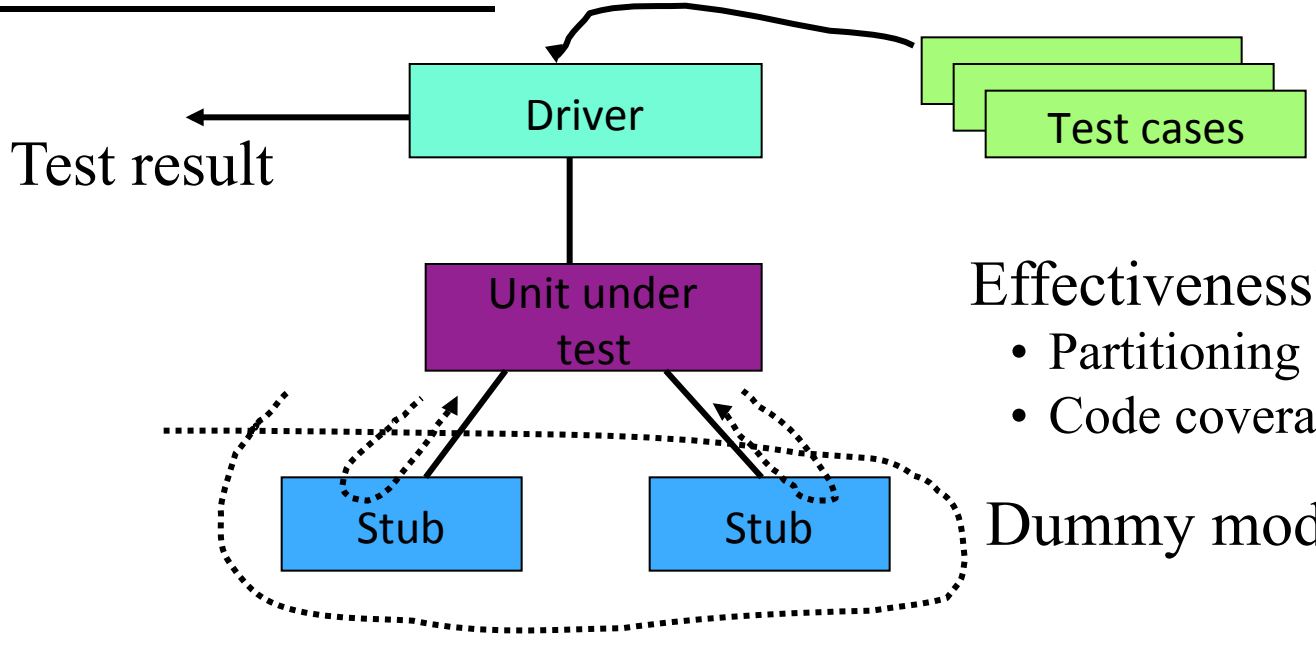
Constructing the Logic Flow Diagram



Unit Testing

Objective: Find differences between specified **units** and their imps.
Unit: component (module, function, class, objects, …)

Unit test environment:



Effectiveness?

- Partitioning
- Code coverage

Dummy modules

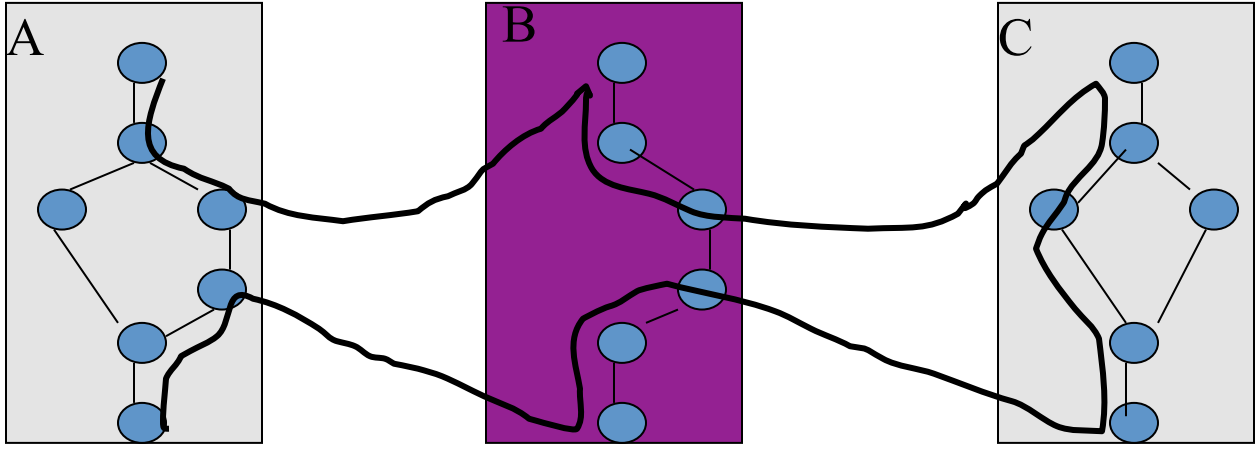
Integration Testing

- **Objectives**:
  - To **expose** problems arising from the combination
  - To quickly obtain a **working solution** from components.
- **Problem areas**
  - **Internal**: between components
    - **Invocation**: call/message passing/…
    - **Parameters**: type, number, order, value
    - **Invocation return**: identity (who?), type, sequence
  - **External**:
    - Interrupts (wrong handler?)
    - I/O timing
  - **Interaction**

Integration Testing

- **Types of integration**
  - **Structural**
    - **"Big bang"** ← no error localization
    - **Bottom-up**: terminal, driver/module, (driver ← module)
    - **Top-down**: top, stubs, (stub ← module), early demo
  - **Behavioral**
    - **(next slide)**

Integration Testing (Behavioral: **Path-Based**)



MM-path: Interleaved sequence of **module exec path** and **messages**
Module exec path: **entry-exit** path in the same module

Atomic System Function: port input, … {MM-paths}, … port output

Test cases: exercise ASFs

System Testing

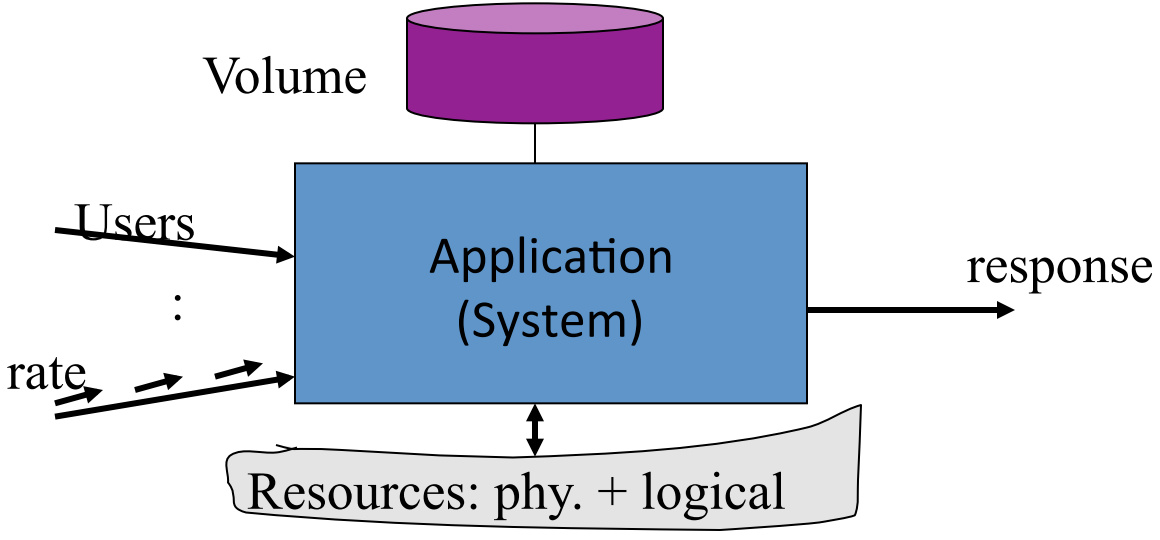
- Concerns with the app's **externals**
- Much more than **functional**
  - Load/stress testing
  - Usability testing
  - Performance testing
  - Resource testing

System Testing

- **Functional testing**
  - **Objective**: Assess whether the app does what it is supposed to do
  - **Basis**: Behavioral/functional specification
  - **Test case**: A **sequence** of **ASFs** (thread)

System Testing

- **Stress testing:** push it to its limit + beyond



System Testing

- **Performance testing**
  - Performance seen by
    - **users:** delay, throughput
    - **System owner:** memory, CPU, comm
  - Performance
    - Explicitly specified or expected to do well
    - Unspecified ➔ find the limit
- **Usability testing**
  - Human element in system operation
    - GUI, messages, reports, …

Test Stopping Criteria

- Meet **deadline,** exhaust **budget,** … ← management
- Achieved desired coverage
- Achieved desired level failure intensity

Acceptance Testing

- **Purpose**: ensure that end **users are satisfied**
- **Basis:** user expectations (documented or not)
- **Environment**: real
- **Performed**: **for** and **by end users** (commissioned projects)
- **Test cases**:
  - May reuse from system test
  - Designed by end users

Regression Testing

- Whenever a system is modified (fixing a bug, adding functionality, etc.), the entire test suite needs to be rerun
  - Make sure that features that already worked are not affected by the change
- Automatic re-testing before checking in changes into a code repository
- Incremental testing strategies for big systems

Comparison of White & Black-box Testing

- White-box Testing:
  - Potentially infinite number of paths have to be tested
  - White-box testing often tests what is done, instead of what should be done
  - Cannot detect missing use cases
- Black-box Testing:
  - Potential combinatorical explosion of test cases (valid & invalid data)
  - Often not clear whether the selected test cases uncover a particular error
  - Does not discover extraneous use cases ("features")

- Both types of testing are needed
- White-box testing and black box testing are the extreme ends of a testing continuum.
- Any choice of test case lies in between and depends on the following:
  - Number of possible logical paths
  - Nature of input data
  - Amount of computation
  - Complexity of algorithms and data structures

The 4 Testing Steps

1. Select what has to be measured
   - Analysis: Completeness of requirements
   - Design: tested for cohesion
   - Implementation: Code tests
2. Decide how the testing is done
   - Code inspection
   - Proofs (Design by Contract)
   - Black-box, white box,
   - Select integration testing strategy (big bang, bottom up, top down, sandwich)
3. Develop test cases
   - A test case is a set of test data or situations that will be used to exercise the unit (code, module, system) being tested or about the attribute being measured
4. Create the test oracle
   - An oracle contains of the predicted results for a set of test cases
   - The test oracle has to be written down before the actual testing takes place

Guidance for Test Case Selection

- Use *analysis knowledge* about functional requirements (black-box testing):
  - Use cases
  - Expected input data
  - Invalid input data
- Use *design knowledge* about system structure, algorithms, data structures (white-box testing):
  - Control structures
    - Test branches, loops, ...
  - Data structures
    - Test records fields, arrays, ...

- Use *implementation knowledge* about algorithms:
  - Examples:
  - Force division by zero
  - Use sequence of test cases for interrupt handler

Unit-testing Heuristics

1. Create unit tests as soon as object design is completed:
   - Black-box test: Test the use cases & functional model
   - White-box test: Test the dynamic model
   - Data-structure test: Test the object model
2. Develop the test cases
   - Goal: Find the minimal number of test cases to cover as many paths as possible
3. Cross-check the test cases to eliminate duplicates
   - Don't waste your time!
4. Desk check your source code
   - Reduces testing time
5. Create a test harness
   - Test drivers and test stubs are needed for integration testing
6. Describe the test oracle
   - Often the result of the first successfully executed test
7. Execute the test cases
   - Don't forget regression testing
   - Re-execute test cases every time a change is made.
8. Compare the results of the test with the test oracle
   - Automate as much as possible

Testing Techniques

- Refer to Chapter 5's slides from Jorgensen's book.

Summary

- Test Activities
- Test levels
- Test types
- Test techniques (intro)

References

- Paul C. Jorgensen, Software Testing : a Craftsmanship's approach.
- Paul Ammann & Jeff Offutt, Introduction to Software Testing.
- Paulo Alencar, Computer Science, University of Waterloo, Lecture slides from cs447