Chap 2. Introduction to Software Testing

2.1 Software Testing Concepts and Processes

2.2 Test Management

2.1 Software Testing Concepts and Processes

1. Introduction
2. Testing Dimensions
3. Test Concepts and Tasks
4. Lifecycle Testing

1. Introduction

-Testing is the process of uncovering evidence of flaws and fixing these flaws in software systems.

÷Flaws may result from various reasons such as mistakes, misunderstandings, and omissions occurring during any phase of software development.

-Testing allows mitigating software risks.

÷Testing is conducted according to a test plan, which is more effectively driven by the specific risks faced by the system.

÷The risks determine what type of tests need to be conducted;

÷The test plan determines how much testing is needed or is acceptable from a business perspective.

-Testing remains one of the most costly and challenging aspects of the software development process.

÷From a business perspective, there is an optimum level of testing, which is acceptable.

÷Beyond the optimum level, testing becomes less cost-effective, because the cost of testing simply exceeds the gains obtained from the defects detected.

Purpose of Testing

-According to the test objectives, different approaches may be used:

÷*Defect testing*: guided by the objective of uncovering latent defects in the program, before delivering it; involves exercising the program in order to trigger some incorrect behavior, exposing some defect.

÷*Validation testing*: establish that the system behaves according to its specification; requires testing the system or correct behavior by exercising some acceptance testing criteria.

- Hence, missions of test groups can vary:

•Find defects; Maximize bug count

•Block premature product releases; Help managers make ship / no-ship decisions

•Assess/Assure quality; Conform to regulations

•Minimize technical support costs; Minimize safety-related lawsuit risk

•Assess conformance to specification/ Verify correctness of the product

•Find safe scenarios for use of the product (find ways to get it to work, in spite of the bugs)

2. Test Dimensions

-Since program errors are diverse and complex, no single test technique can cover all kinds of errors. In practice, a combination of several techniques is used to achieve adequate testing.

÷There are many levels of testing as well as many dimensions to testing, which should be considered in order to test appropriately a software system.

Test Scopes

-There are three levels of testing:

÷*Unit Testing*: targets small piece of software entity such as a method, a class, a cluster of interdependent classes, or a software component.

÷*Integration Testing*: tests the interactions between a collection of related functions or components.

÷*System Testing*: test the whole system after all the components or subsystems are combined into the final product.

Testing Dimensions

-Testing dimensions can be defined based on two criteria: task and data.

÷Testing activities can be categorized in either validation or verification tasks.

÷Based on the test data available, testing can be categorized in either functional testing or structural testing.

Validation & Verification

-***Verification*** consists of checking that the system works according to the organization's standards and processes.

÷Verification techniques include requirements review, design review, code walkthrough, and code inspection.

-***Validation*** consists of ensuring that the system functions according to the specification.

÷Involves executing the system in real-life scenarios and checking whether it behaves as expected.

÷Validation techniques include unit testing, integration testing, system testing, and user acceptance testing.

Functional vs. Structural Testing

-***Functional testing***, also called black box testing, consists of checking the system without having or using any knowledge of the internal logic in developing the test cases.

÷Mostly accomplished using validation techniques.

÷Checks that the software system is built according to the requirements.

-***Structural testing***, also called white-box testing, use knowledge of the internal logic to develop test cases.

÷Checks the structure of the software artifacts (e.g., code, design etc.).

÷Mostly accomplished using verification techniques.

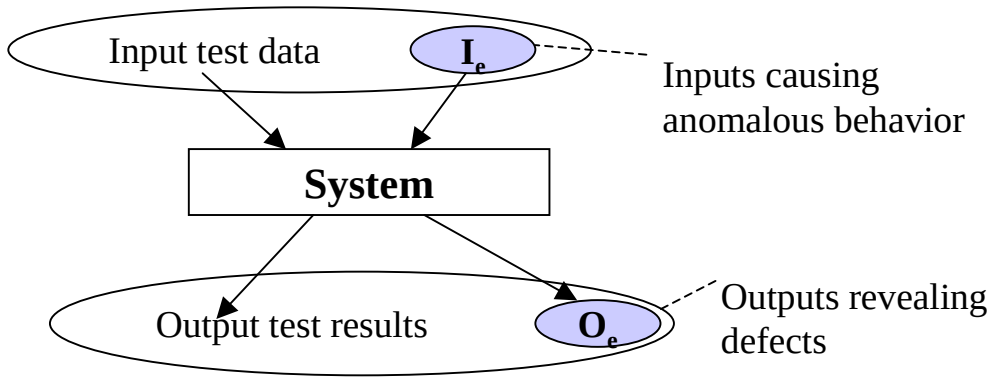
3. Test Concepts and Tasks

Test Concepts

-*Test case*: specification of inputs and outputs required to reveal defects

÷Consists of the pretest state and environment, the test inputs and expected outputs and state.

÷Expected outputs may include data, messages or exceptions generated by the application under testing.



-*Test oracle*: a technique or mechanism used to produce expected results under specific conditions.

-*Test strategy*: an algorithm and/or a collection of heuristics used to identify interesting test cases for an application under testing.

-*Test suite*: a collection of test cases, typically related by a testing goal or implementation dependency.

Example: an IUT and some corresponding test cases and test suite

```
int prog(int X, int Y) {
        x = X; y = Y;
        if (x > 100) {
                x = x – 100;
        } else {
                x = x + 100;
        }

        if (x <= y) {
                y = 2*y;
        }
        else {
                if (x > 200) {
                        x = x - 100;
                } else {
                        x = x*x;
                }
        }
        printf("x=%d. y=%d",x, y);
}
```

Sample test cases:

Test case Id	Input		Expected Output	
	X	Y	x	y
Tc1	10	100	110	200
Tc2	101	100	1	100
Tc3	201	110	1	110

Sample test suite: {Tc1, Tc2, Tc3}

Test concepts (ctd.)

-*Test run:* the execution with actual results of a test suite(s). Actual results are compared against expected one, in order to decide the outcome: ***pass*** or ***no pass***.

÷***pass:*** actual results are the same as expected one.

÷***no pass:*** actual results are different from expected ones; this reveals a bug, and is therefore considered a ***successful*** test.

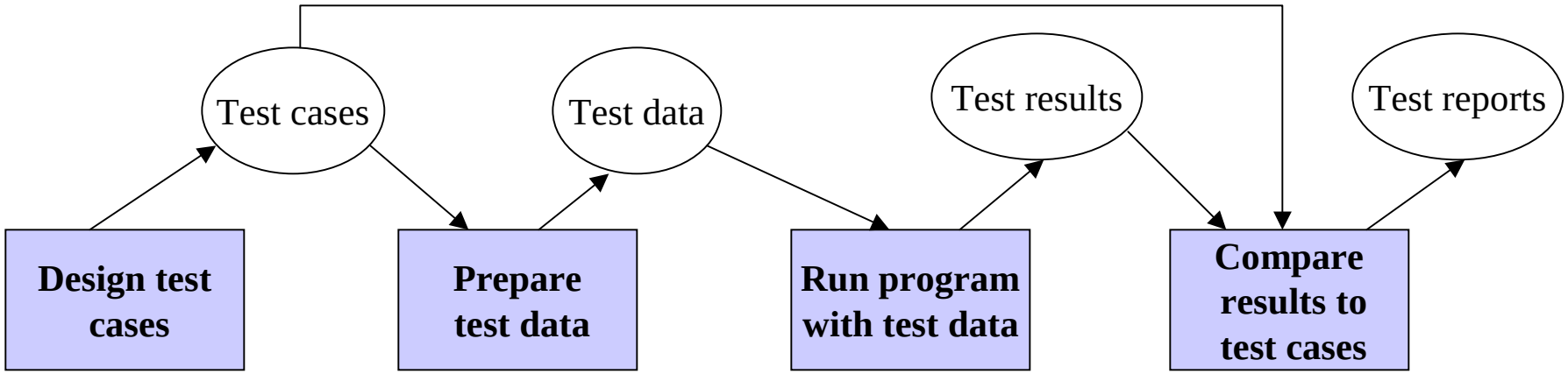
-*Test driver*: a class or utility program that applies test cases to an implementation under testing (IUT)

-*Stub:* a partial, temporary implementation of a component, that may serve as a placeholder for an incomplete component or implement testing support code.

-*Test script:* a program written in a procedural script language (usually interpreted) that executes a test suite(s).

-*Test harness:* a system of test drivers and other tools supporting test execution.

Testing Tasks

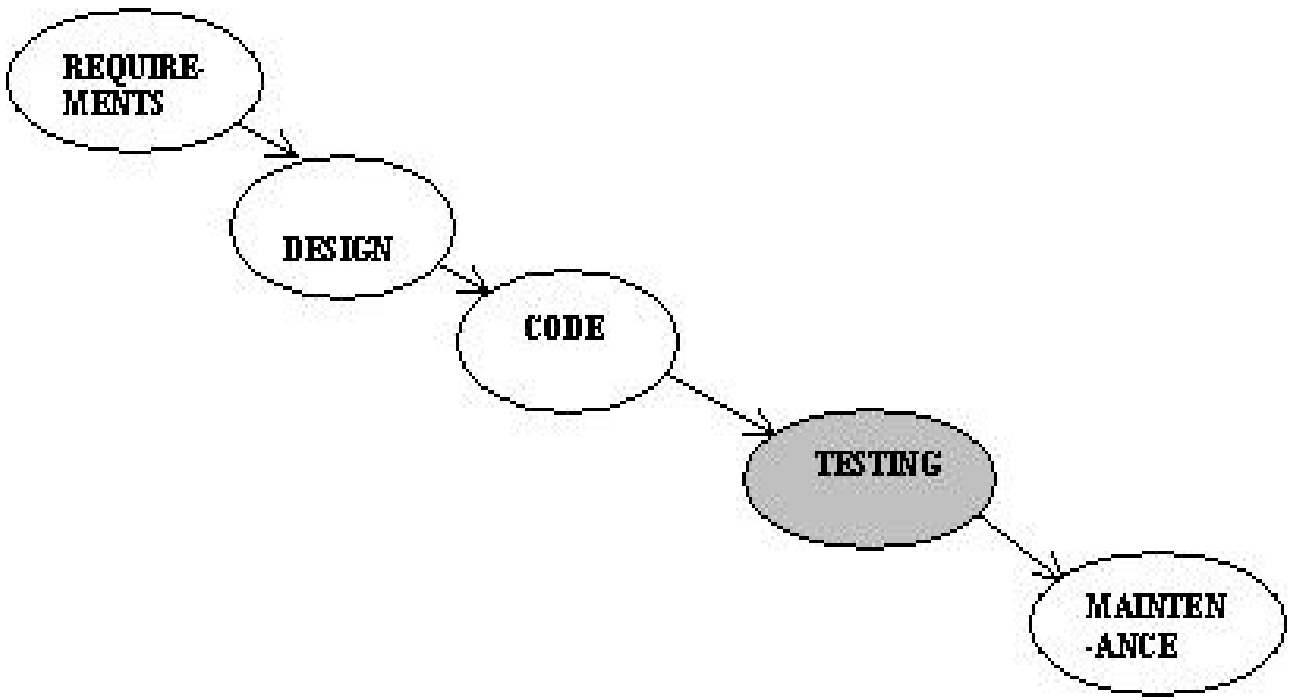


1. *Design test cases:* define and generate test cases using selected test strategies.
2. *Prepare test data:* instantiate the test cases by generating test points.
3. *Execute test cases/suites:* applies the test points to the IUT.
4. *Compare results:* compare the actual outputs with expected ones, and report corresponding test results.

4. Lifecycle Testing

Traditional Testing Approaches

-Traditionally testing occurs at the latter phases of the software development cycle



-However, restricting testing to a single phase creates the potential that errors with significant and costly consequences may occur.

-Studies conducted by IBM over several major projects have shown that:

÷ an average of 60 defects could be detected during application development:

÷50% of the defects could be detected when testing is conducted before the implementation phase,

÷80% could be detected when testing is conducted after implementation.

÷It is at least 10 times more expensive to fix a defect after implementation than before, and 100 times more expensive during maintenance.

-Lessons learned:

÷Importance of starting testing as early as possible.

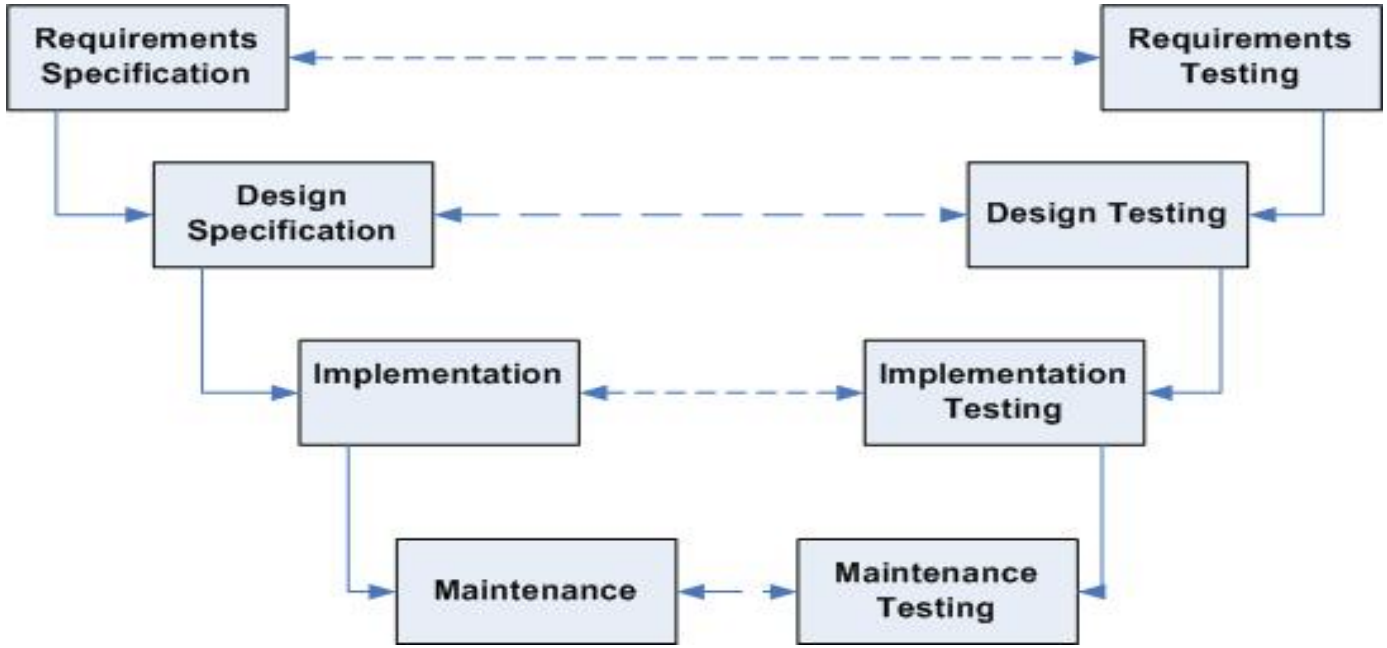
÷Necessity to integrate testing in the entire development life cycle, and not only after coding or during production.

-*Lifecycle testing* is recommended as alternative to traditional testing:

÷Incorporates testing in all the phases of the development process, and as such it spans the entire software lifecycle.

÷Starts at the same time as the product development process; both processes run concurrently and should be conducted using well structured but different methodologies.

Lifecycle Testing



÷*Requirements Testing:* ensure that the requirements are recorded or documented properly, and address user and business concerns.

÷*Design Testing:* ensure that the design is complete, accurate and matches with the requirements.

÷*Implementation Testing:* ensure that the design specification has been correctly implemented.

÷*Maintenance Testing:* involve deployment testing at the installation, and regression testing during operation.