TVVS
Software Testing, Verification and Validation

Introduction (2)

Ana Paiva

apaiva@fe.up.pt

//web.fe.up.pt/~apaiva

Let's recap some things said based on test principles

Test Principles

1. Testing Shows the Presence of Defects: The primary purpose of testing is to identify defects or issues in the software. Testing helps in revealing the presence of defects but cannot prove their absence entirely.

Test Principles

2. Exhaustive Testing is Impractical: It's impossible to test every possible input and scenario exhaustively due to time and resource constraints. Test efforts should be focused on areas with the highest risk and probability of finding defects.

Test Principles

3. Early Testing: Testing should begin as early as possible in the software development lifecycle. Early testing helps in identifying defects when they are less expensive to fix.

Test Principles

4. Defect Clustering: A small number of modules or functionalities often contain the majority of defects. In other words, defects tend to cluster in specific parts of the software.

Test Principles

5. Pesticide Paradox: If the same tests are repeated over time, they become less effective at finding new defects. Test cases need to be regularly reviewed and updated to avoid the pesticide paradox.

Test Principles

- **6. Tests are dependent on the context:** testing varies depending on the context. Examples: In the realm of security, critical industrial control software undergoes different testing procedures than a mobile e-commerce application.

Test Principles

7. Absence-of-Errors Fallacy: The absence of reported defects does not necessarily indicate that the software is error-free. It may mean that testing has not been thorough enough or that the defects have not been discovered yet.

Now,… the homework

- We must perform Verification and Validation (V&V) to find the maximum number of errors, mistakes, bugs, faults, defects, and failures ☹ in the software system under test (SUT).

- Homework

  Go to the ISTQB glossary at

  glossary.istqb.org

  and find out the meaning of

  error, mistake, bug, fault, defect, and failure

So, Error, Fault, Failure …

[According to ISTQB Glossary (glossary.istqb.org)]

- **Error/Mistake**: A human action that produces an incorrect result.
- **Defect/Fault/Bug:** An imperfection or deficiency in a work product where it does not meet its requirements or specifications
  - Usually a problem in the code, e.g., an IF that uses “less than” instead of “greater than”
- **Failure:** An event in which a component or system does not perform a required function within specified limits
  - Caused by Faults. The existence of a Fault in the source code does not necessarily lead to a failure (e.g., the code containing the fault is never executed; masked defect -- *defects that prevent or inhibit the detection of another defect*)

Software Testing

Software Testing may be static or dynamic

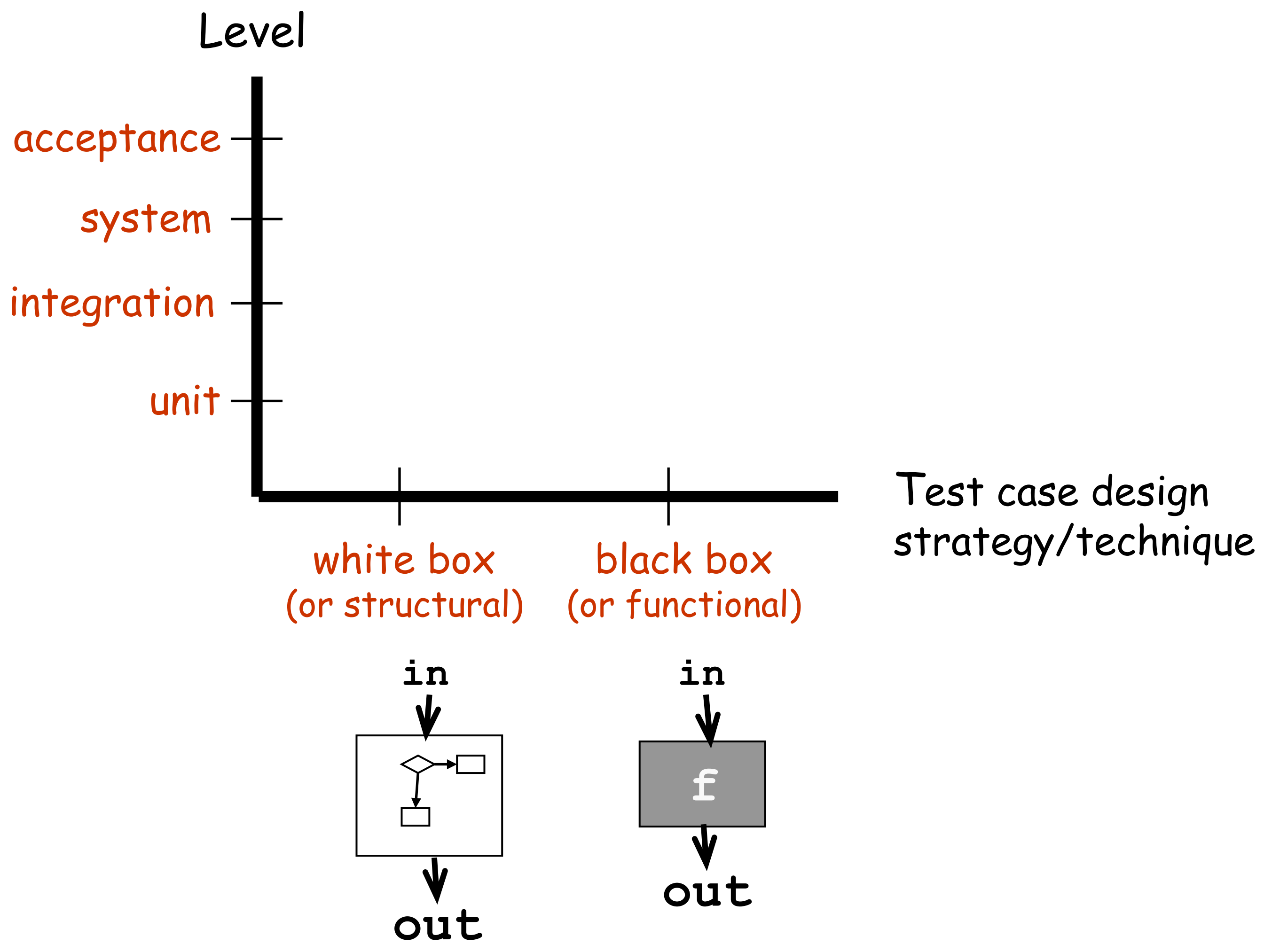
- **Static Testing**: Testing that does not involve the execution of a test item
- **Dynamic Testing**: Testing that involves the execution of the test item

Software Testing

Now, are you able to answer the following questions?

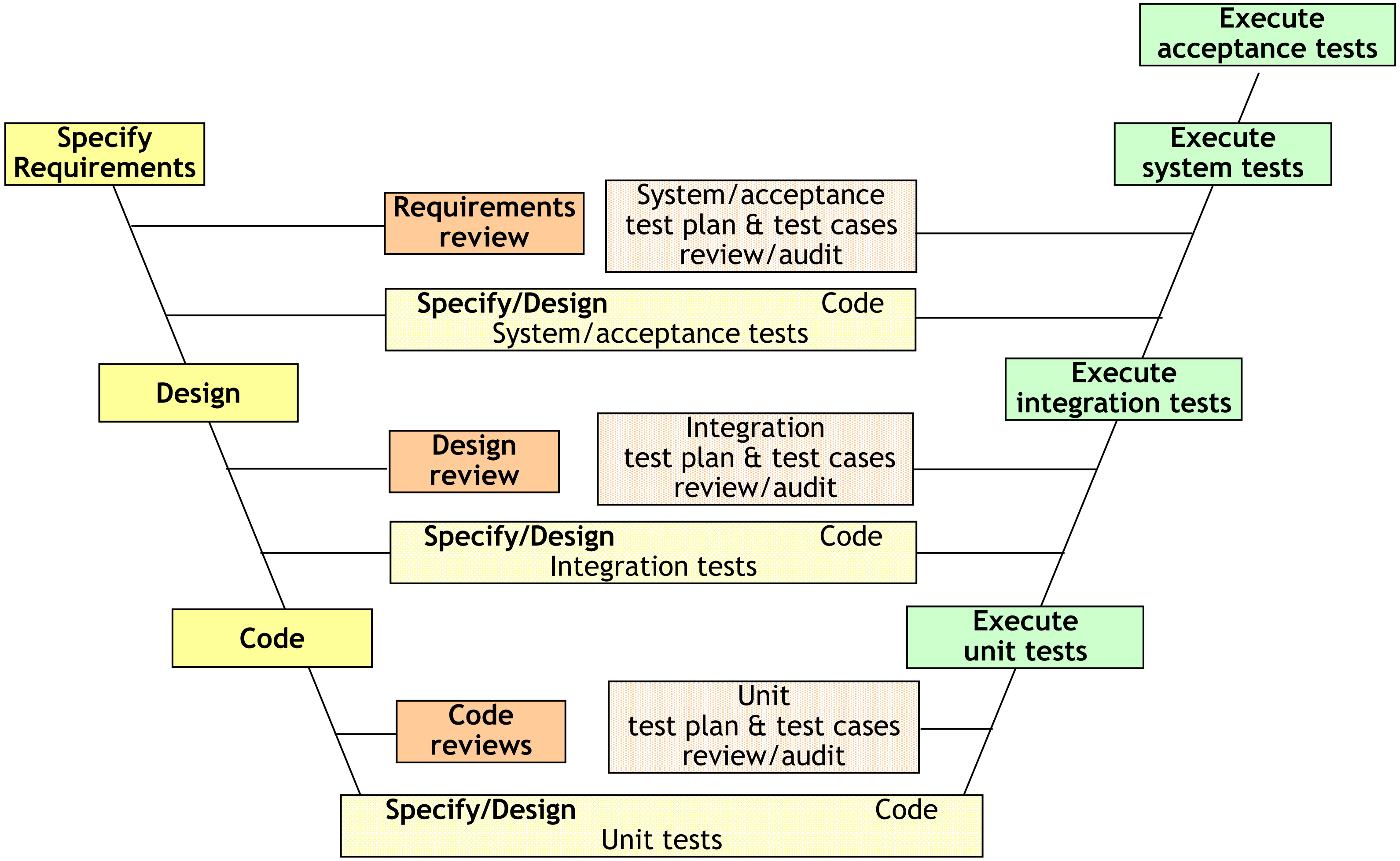
- which “fault” or “failure” may be found by static testing?
- which “fault” or “failure” may be found by dynamic testing?

Test types


Level
acceptance
system
integration
unit
Test case design strategy/technique
white box (or structural)
black box (or functional)
in
out
in
f
out

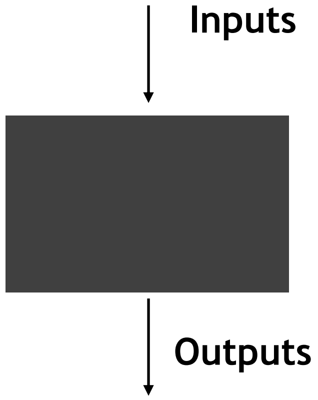
V-model

Testing of the product is planned in parallel with a corresponding phase of development in V-model.


Execute acceptance tests
Specify Requirements
Execute system tests
Requirements review
System/acceptance test plan & test cases review/audit
Specify/Design Code System/acceptance tests
Design
Execute integration tests
Design review
Integration test plan & test cases review/audit
Specify/Design Code Integration tests
Code
Execute unit tests
Code reviews
Unit test plan & test cases review/audit
Specify/Design Code Unit tests

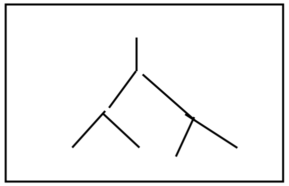
The extended V-model of software development [I.Burnstein]

Test case design strategies and techniques

Strategy	Tester's View	Knowledge sources	Techniques / Methods
Black-box testing (not code-based) (sometimes called functional testing)		Requirements document Specifications User manual Models Domain knowledge Defect analysis data Intuition Experience	Equivalence class partitioning Boundary value analysis Cause effect graphing Pairwise testing Error guessing Random testing State-transition testing Scenario-based testing

(adapted from: I. Burnstein, pg.65)

Test case design strategies and techniques

Strategy	Tester's View	Knowledge sources	Techniques / Methods
White-box testing (also called code-based or structural testing)		Program code Control flow graphs Data flow graphs Cyclomatic complexity High-level design Detailed design	Control flow testing/coverage: •Statement coverage •Branch (or decision) coverage •Condition coverage •Branch and condition coverage •Modified condition/decision coverage •Multiple condition coverage •Independent path coverage •Path coverage •LCSAJ coverage Data flow testing/coverage Class testing/coverage Mutation testing

(adapted from: I. Burnstein, pg.65)

SO, BUT WHAT IS A TEST CASE?

What is a test case?

[ISTQB Glossary - 2023]

Test case

- A set of preconditions, inputs, actions (where applicable), expected results and post conditions, developed based on test conditions

Test condition

- A testable aspect of a component or system identified as a basis for testing

Test design

- The activity that derives and specifies test cases from test conditions

What is a test case?

[ISTQB Glossary - 2023]

Test specification

- The complete documentation of the test design, test cases, and test scripts for a specific test item

Test item

- A part of a test object used in the process

Example

Test Condition: User Authentication

- Description: This test condition covers the overall authentication process for a website, including valid and invalid login attempts, password reset functionality, and account lockout behaviour.

Example

- Test Cases Associated with the "User Authentication" Test Condition:
- Abstract Test Case 1 (Verify Successful Login) This test case checks if a user can successfully log in with valid credentials.
  - Precondition: be on URL
  - Enter a valid username and password
  - Click login
  - Verify successful login
- Concrete Test Case 1
  - Precondition: be on //www.url.pt
  - Enter “ANA” and “PASS”
  - Click login
  - Verify if on URL //www.url.pt/~apaiva

Example

Test Cases Associated with the "User Authentication" Test Condition:

- Abstract Test Case 2 (Verify Invalid Username and Password Handling) This test case checks how the system handles an invalid username and password combination.
  - Precondition: be on URL
  - Enter an invalid username and password
  - click Login
  - verify error message
- Concrete Test Case 2
  - Precondition: be on //www.url.pt
  - Enter “WRONGLOGIN” and “PASSWRONG”
  - click Login
  - verify if message “invalid login” on screen

Homework

Build concrete test cases for

- Abstract Test Case 3 (Verify Password Reset Functionality) This test case verifies that the password reset feature allows users to reset their passwords successfully.
  - Test Steps: Request a password reset, follow the reset link or code, and set a new password.
- Abstract Test Case 4 (Verify Account Lockout) This test case checks the system's behavior when a user exceeds the maximum number of login attempts.
  - Test Steps: Attempt to log in with incorrect credentials multiple times, exceeding the lockout threshold, and verify the account is locked.

HOW TO MEASURE THE QUALITY OF THE TEST CASES?

What is test adequacy?

- Consider a program *P* written to meet a set *R* of functional requirements.
  - Let *R* contain *n* requirements labeled R_1, R_2,…, R_n .
  - A set *T* containing *k* tests has been constructed to test *P*.
- *P* has been executed against each test in *T* and has produced correct behaviour.
- **We now ask:**
  - Is *T* good enough?
  - Has *P* been tested thoroughly?
  - Is *T* adequate?

Test adequacy/coverage criteria

Adequacy criteria

- Criteria to decide if a given test suite is adequate, i.e., to give us "enough" confidence that "most" of the defects are revealed
  - Used in the evaluation and in the design/selection of test cases
  - In practice, reduced to **coverage criteria**

Test adequacy

[ISTQB Glossary 2023]

Coverage criteria

- The criteria to define the coverage items required to reach a test objective

Coverage

- The degree to which specified coverage items are exercised by a test suite, expressed as a percentage

Coverage criteria

Coverage criteria

- criterion C is a **white-box test adequacy criterion** if the corresponding coverage domain depends solely on program P under test
- Criterion C is a **black-box test adequacy criterion** when it depends solely on requirements for the program P under test.

Black box

- Requirements/specification coverage (black-box)
  - At least one test case for each requirement/specification statement
- Model coverage
  - State-transition coverage
  - Use case and scenario coverage

White box

- Code coverage (white-box)
  - Control flow coverage (statement, decision, MC/DC coverage …)
  - Data flow coverage
- Fault coverage

Measurement of adequacy example

- C: A test *T* for the program (*P*, *R*) is considered adequate if **<u>for each requirement R_i in *R*, there is at least one test case</u>** in *T* that tests the correctness of *P* with respect to R_i.
- Program *sumProduct* must meet the following requirements:
  - R1 - Input two integers, ***x*** and ***y***, from the standard input device.
  - R2.1 - if x<y, print the sum of x and y.
  - R2.2 - if x≥ y, print the product of x and y.
- Is T={t: <x=2, y=3>} adequate according to criterion C? (Does it "cover" all Requirements?)
- Test case t in T tests R1 and R2.1, but not R2.2. So, T is not adequate (*does not achieve 100% coverage*) with respect to C.
- The coverage of T with respect to C, P, and R is 2/3 = 66%

Exercise

- Suppose that condition **C=C1 AND C2 AND C3** has been coded as **C'=C1 AND C3**. Four tests that form an MC/DC* adequate set for C' are in the following table. Verify that the following set of four tests is MC/DC adequate but does not reveal the error.

	Test	C	C'	Error detected?
	C1, C2, C3	C1 and C2 and C3	C1 and C3	
T1	True, True, True			
T2	False, False, False			
T3	True, True, False			
T4	False, False, True			

*[MC/DC – Modified Condition Decision Coverage]