TESTING BASICS AND TEST CASE DESIGN

DEPARTMENT OF COMPUTER SCIENCES
UNIVERSITY OF KASHMIR
North Campus

Basic Terminology

Error can defined as incorrect or missing human action that result in software containing a fault (i.e. incorrect software). Examples include omission or misinterpretation of user requirements.

Fault can be defined as abnormal condition that may cause a reduction in, or loss of, the capability of a functional unit to perform a required function. It can also be defined as a requirements, design, or implementation flaw or deviation from a desired or intended state

Failure can be defined as the inability of a system to perform its required functions within specified requirements or to perform in an unexpected way.

$$\begin{tikzcd}
\boxed{\text{Error}} \arrow[r, "\text{results in}", "\text{during development}"'] & \boxed{\text{Fault}} \arrow[r, "\text{causes}", "\text{during execution}"'] & \boxed{\text{Failure}}
\end{tikzcd}$$

An Example

LOC	Source Code
1	Program to double a number;
2	var x,y: integer;
3	Begin
4	read(x);
5	y := x * x;
6	write(y)
7	End

Failure: x = 3 means y =9 →Failure!

- This is a failure of the system since the correct output should have been *6*

Fault: The fault that causes the failure is in line 5. The * operator is used instead of +.

Error: The error that conduces to this fault may be:

- a typing error (the developer has written * instead of +)
- a conceptual error (e.g., the developer doesn't know how means *to double a* ***number***)

What is Testing

1. “Testing is the process of exercising or evaluating a system or system component by manual or automated means to verify that it satisfies specified requirements [**IEEE**].

2. “The process consisting of all life cycle activities, both static and dynamic, concerned with planning, preparation and evaluation of software products and related work products to determine that they satisfy specified requirements, to demonstrate that they are fit for purpose and to detect defects.” [**ISQTB**]

3. Testing can be stated as the process of validating and verifying that a software program/application/product:
   a. meets the business and technical requirements that guided its development;
   b. Works as expected and can be executed successfully in the Intended environment.
   c. Is fit for the purpose and satisfies the needs of stakeholders.

4. Testing is the process of executing a program with the intent of finding defects. [**Myers Definition**] *

** Myers Definition does not consider Static Testing as Testing.*

1. Testing is not the process of demonstrating that faults are not present.

2. The purpose of testing is not only to show that a program performs its intended functions correctly and/or establishing confidence that a program does what it is supposed to do.

A problem with such definitions as “testing is the process of demonstrating that faults are not present” is that such a goal is impossible to achieve for virtually all programs, even trivial programs. For Example **CROSSWORD PUZZLE Problem.**

Another problem with such definitions such as “testing is the process of demonstrating that a program does what it is supposed to do” is that programs that do what they are supposed to do still can contain faults. That is, an fault is clearly present if a program does not do what it is supposed to do, but faults are also present **if a program does what it is not supposed to do. For Example Triangle Problem.**

Testing is a destructive process rather than Constructive one.

Testing can demonstrate the presence of errors, not their absence. (Dijkstra)

SOFTWARE VERIFICATION AND VALIDATION

- Verification
  - Are you building the product right?
  - Software must conform to its specification
- Validation
  - Are you building the right product?
  - Software should do what the user really requires

- Verification is the process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase.

- Validation is the process of evaluating a system or component during or at the end of the development process to determine whether it satisfies requirements specified by the user.

Why do Software Testing

“A clever person solves a problem. A wise person avoids it.”

Albert Einstein

- All definitions agree on that Testing is a process. So what does that process entail?
  - **Process:** IEEE defines process as a sequence of steps performed for a given purpose. However according to SEI-CMM, software process is a set of activities, methods, practices, and transformations that people use to develop and maintain software and associated products.

- IEEE 610 and BS 7925-1, respectively, talk about "operating" and "exercising"; that is, the idea that testing requires the software to run on a computer. This is what is also called "dynamic testing." IEEE 829 broadens the idea to "analyzing," thus including "static testing." And ISTQB takes the full step and includes both "dynamic and static." Testing is both dynamic and static.

- Then what do we do dynamic and static testing on? The object of the testing in the definitions ranges from "system or component," "software item," and "software" to "software products and related work products." In line with testing being both dynamic and static, we have to conclude: Testing can be done on any work product or product

- Checking properties of the implementation of the software against its specification
  - by reading it (without executing) it ------------ Static Testing
  - by executing it ------------ Dynamic Testing

Testing Related Terminology

- ✓ **Test case:** a set of inputs, execution conditions, and a pass/fail criterion.
- ✓ **Test case specification**: a requirement to be satisfied by one or more test Cases.
- ✓ **Test obligation:** a partial test case specification, requiring some property deemed important to thorough testing.
- ✓ **Test suite:** a set of test cases.
- ✓ **Test or test execution:** the activity of executing test cases and evaluating their results.
- ✓ **Adequacy criterion:** a predicate that is true (satisfied) or false (not satisfied) of a ⟨program, test suite⟩ pair.

TEST CASE

Test case: a set of inputs, execution conditions, and a pass/fail criterion.

- ✓ Test case contains:
  - Test data to be used
  - A sequence of Steps describing actions to be performed,
  - An expected response for each action performed.
- ✓ Each Test case is associated with a specific program behavior.
- ✓ Test Cases are written based on Business and Functional/Technical requirements, use cases and Technical design documents.
- ✓ There can be 1:1 or 1:N or N:1 or N:N relationship between requirements and test cases.

Software testing is the act of evaluating software with a suite of test cases so that it can either find defects in the program or demonstrate the program is correct.

It is difficult to design a suite of test cases that can prove a program is correct.

Sample Test cases for a web page

- Testing without entering any username and password.
- Test it only with Username or password.
- User name with wrong password
- Right username and right password
- Try copy/paste in the password text box.
- .
- After successful sign-out, try "Back" option from your browser. Check whether it gets you to the "signed-in" page.

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Results	Actual Results	Pass/ Fail
TU01	Check Customer Login with valid Data	Go to Site Enter UserId Enter Password Click Submit	Userid = umar Password = 1234abc	User should Login	As Expected	Pass

TEST ADEQUACY CRITERIA

A software test adequacy criterion is a predicate that defines what properties of a program must be exercised to constitute a thorough test.

- ✓ Specifies requirements for testing.
- ✓ Can be used as *stopping rule*: stop testing if 100% of the statements have been tested
- ✓ Can be used as *measurement*: a test set that covers 80% of the test cases is better than one which covers 70%.
- ✓ Can be used as *test case generator*: look for a test which exercises some statements not covered by the tests so far
- ✓ A given test adequacy criterion and the associated test technique are opposite sides of the same coin

Test Adequacy Criteria

- Consider a program *P* written to meet a set *R* of functional requirements. We notate such a P and R as (P,R). Let *R* contain n requirements labeled *R1*,*R2*,...,*R*n.

- Suppose now that a set T containing k tests has been constructed to test P to determine whether or not it meets all the requirements in R. Also, P has been executed against each test in T and has produced correct behavior.

- We now ask: *Is* T *good enough ?* This question can be stated differently as: *Has* P *been tested thoroughly ?*, or as: *Is* T *adequate ?* Regardless of how the question is stated, it assumes importance when one wants to test P thoroughly in the hope that all errors have been discovered and removed when testing is declared complete and the program P declared usable by the intended users.

Test Adequacy Criteria

- In the context of software testing, the terms "*thorough*," "*good enough*," and "*adequate*," used in the questions above, have the same meaning.

- We prefer the term "adequate" and the question *Is* T *adequate ?*. Adequacy is measured for a given test set designed to test P to determine whether or not P meets its requirements. This measurement is done against a given criterion C .

- A test set is considered adequate with respect to criterion C when it *satisfies* C . The determination of whether or not a test set T for program P satisfies criterion C depends on the criterion itself.

Example of Test Adequacy Criteria

Consider the problem of writing a program named SumProduct that meets the following requirements:

- ✓ R1 : Input two integers, say x and y, from the standard input device.
- ✓ R2 : Print to the standard output device the sum of x and y if $x < y$.
- ✓ R3 : Print to the standard output device the product of x and y if $x \geq y$.

Suppose now that the ***test adequacy criterion C*** is specified as follows:

- C : *A test* T *for program (*P , R*) is considered adequate if for each requirement* r *in* R *there is at least one test case in* T *that tests the correctness of* P *with respect to* r.
- It is obvious that $T = \{t :< x = 2, y = 3 >\}$ is ***inadequate*** with respect to C for program SumProduct.
- The lone test case t in T tests R1 and R2, but not R3.

Uses of Test Adequacy Criteria

A software test adequacy criterion

- Specify a software testing requirement
    - Determine test cases to satisfy requirement
- Determine observations that should be made during testing
- Control the cost of testing
    - Avoid redundant and unnecessary tests
- Help assess software dependability
- Build confidence in the integrity estimate

Why to Choose Test Case Design Strategy

Development of Test Cases

Complete testing is impossible

$\Downarrow$

Testing cannot guarantee the absence of faults

$\Downarrow$

How to select subset of test cases from all possible test cases with a high chance of detecting most faults ?

$\Downarrow$

Test Case Design Strategies

Static vs. Dynamic Testing

- Static testing covers the area of performing all kinds of tests on the *software related documents* or *source code* of software without executing it.

- Dynamic testing covers the area of performing all kinds of tests on the object code and executable that is determined from the source code by executing it.

- The difference between these types of testing is usually determined by the state of the software program (source code vs. executable).

- The other difference is that static testing techniques can be used from *requirement phase to implementation phase*, dynamic testing techniques can be applied from *implementation phase* onwards only.

- Therefore, any product obtained during development (including code) can be evaluated by means of static testing. However, dynamic testing exclusively evaluates executable code.

- Another difference is that while static testing looks for *faults*, dynamic testing looks for *failures*.

- ***Static testing/non execution based techniques*** focuses on the range of ways that are used to verify the program without reference to actual execution of the program.

- Static techniques are concerned with the analysis and checking of system representations such as the requirements documents, design diagrams and the program source code, either manually or automatically, without actually executing the code.

- Techniques in this area include code inspection, program analysis, symbolic analysis, and model checking etc. Documentation in the form of text, models or code are analyzed, often by hand. In a number of cases, e.g. in the compilation of program code, tools are used.

- ***Dynamic Testing / Execution based Techniques*** focuses on the range of ways that are used to ascertain software quality and validate the software through actual executions of the software under test.

- Dynamic testing is used to evaluate dynamic criteria, which means that it examines the result of operating the system as opposed to the product directly.

- In dynamic testing we create input test cases to physically test the software under test by executing it and observing its behavior. To test the software means to operate it with real or simulated inputs, both normal and abnormal, under controlled and expected conditions to check how a software system reacts to various input test data.

- The dynamic testing of a software product implies execution, as only by studying the result of this execution is it possible to decide whether or not (or to what extent) the quality levels set for the dynamic aspects evaluated are met. Therefore, dynamic testing looks for failures. They involve the execution of a piece of software with test data and a comparison of the results with the expected output which must satisfy the users' requirements.

Black Box and White Box Testing

- Dynamic testing techniques are generally divided into the two broad categories depending on a criterion whether we require the knowledge of source code or not for test case design, if it does not require knowledge of the source code, it is known as ***black box testing*** otherwise it is known as ***white box testing***, which correspond with two different starting points for dynamic software testing: the requirements specification and internal structure of the software.

- Black box testing is focused on results whereas white box testing is focused on details. Black box testing is typically used to check the product without examining the code. But we do not know how much of it is being tested. This is where the white–box techniques come in. They allow you to examine the code in detail and be confident that at least you have achieved a certain level of test coverage.

- White box testing is in itself insufficient since the software under examination may not perform one of its desired tasks–the function to do this may even be missing–and the examination of the code is unlikely to reveal this.

- The main purpose of black box testing is to spot such missing functionality. These tests can be functional or non-functional, though usually functional.

- Considering the purpose of these testing strategies, it appears that both of them should be used in order to test a product fully.

- White box strategy is used only at the unit testing and program integration testing stages, whereas black box testing strategy can be used at all testing stages (unit testing to acceptance testing).

Advantages of Black Box Testing

The main advantage of black box testing is that, testers no need to have knowledge on specific programming language, not only programming language but also knowledge on implementation. In black box testing both programmers and testers are independent of each other. Another advantage is that testing is done from user's point of view. The significant advantage of black box testing is that it helps to expose any ambiguities or inconsistencies in the requirements specifications.

Advantages of White Box Testing

White box testing is mainly used for detecting logical errors in the program code. It is used for debugging a code, finding random typographical errors, and uncovering incorrect programming assumptions. White box testing is done at low level design and implementable code.

Summing up Black and White Box Testing

- **<u>Black-box testing</u> : Deriving tests from external descriptions of the software, including specifications, requirements, and design (without regard for its internal logic).**

A criterion C is a black-box test adequacy criterion if the corresponding coverage domain Ce depends solely on requirements R for the program P under test.

- **<u>White-box testing</u> : Deriving tests from the source code internals of the software, specifically including branches, individual conditions, and statements.**

A criterion C is a white-box test adequacy criterion if the corresponding coverage domain Ce depends solely on program P under test.

All other test adequacy criteria are of a mixed nature

Gray box testing

- Gray Box Testing is a software testing method which is a combination of *Black Box Testing* method and *White Box Testing* method. In Black Box Testing, the internal structure of the item being tested is unknown to the tester and in White Box Testing the internal structure in known.

- In Gray Box Testing, the internal structure is partially known. This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level.

- Gray Box Testing is named so because the software program, in the eyes of the tester is like a gray/semi-transparent box; inside which one can partially see.