

Software Testing and Quality Assurance



Unit-1 Topics

1.0. Introduction

1.1. The Testing Process

1.2. What is Software Testing?

1.3. Why Should We Test? What is the Purpose?

1.4. Who Should do Testing?

1.5. How Much Should We Test?

1.6. Selection of Good Test Cases

1.7. Measurement of Testing

1.8. Incremental Testing Approach

1.9. Basic Terminology Related to Software Testing

1.10. Testing Life Cycle

1.11. When to Stop Testing?

1.12. Principles of Testing

1.13. Limitations of Testing



Introduction

- ❑ Testing is the process of **executing the program** with the **intent of finding faults**.
- ❑ **Who should do** this testing and **when should it start** are very important questions.
- ❑ As we know software testing is the **fourth phase** of the software development life cycle (SDLC)
- ❑ About **70% of development time** is spent **on testing**.



Testing Process

- ❑ Testing is different from **debugging**. **Removing errors** from your programs is known as **debugging** but **testing** aims **to locate as yet undiscovered errors**.
- ❑ We test our programs with **both valid** and **invalid inputs** and then **compare our expected outputs** as well as **the observed outputs** (after execution of software).
- ❑ Note that testing **starts from the requirements analysis phase** only and goes until the last maintenance phase.
- ❑ **During requirement analysis and designing** we do static testing wherein the **SRS is tested to check whether it is as per user requirements or not**. We use techniques of **code reviews**, **code inspections**, walkthroughs, and software technical reviews (STRs) to do **static testing**.
- ❑ **Dynamic testing** starts when the code is ready or even a unit (or module) is ready. It is dynamic testing as now the code is tested.
- ❑ We use **various techniques for dynamic testing** like **black-box**, **gray-box**, and **white-box testing**.



What is Software Testing?

- ❑ The concept of **software testing** has evolved from simple program “check-out” to a broad set of activities that cover the **entire software life-cycle**.

There are **five distinct levels** of testing that are given below:

- 1. Debug:** It is defined as the **successful correction of a failure**.
- 2. Demonstrate:** The process of showing that **major features work with typical input**.
- 3. Verify:** The process of finding **as many faults in the application under test (AUT) as possible**.
- 4. Validate:** The process of finding **as many faults in requirements, design, and AUT**.
- 5. Prevent:** To avoid errors in development of requirements, design, and implementation by self-checking techniques, including “**test before design**.”



What is Software Testing?

❑ Various definitions of testing

- “Testing is the process of exercising or evaluating a system or system components by **manual** or **automated** means to **verify that it satisfies specified requirements.**” [IEEE]
- “Software testing is the **process of executing a program** or **system with the intent of finding errors.**” [Myers]
- “It involves any activity aimed at **evaluating an attribute** or **capability of a program** or **system** and **determining that it meets its required results.**” [Hetzel]

Note: Testing is basically **a task of locating errors.**



What is **NOT** Software Testing?

- ☐ The process of demonstrating that errors are **not** present.
- ☐ The process of showing that a program performs **its intended functions correctly**.
- ☐ The process of establishing confidence that **a program does what it is supposed to do**.

Note: Testing is basically **a task of locating errors**.



What is Positive testing?

- ☐ Operate the application **as it should be operated**(normal/common Workflows).
- ☐ Does it **behave normally**? Use a **proper variety of legal test data**, including data values at the boundaries to test if it fails. Check actual test results with the expected.
- ☐ Are results correct? Does the application function correctly?
- ☐ Testing as per the **requirements** (functional testing)



What is Negative testing?

- ☐ Test for **abnormal operations**. Does the system **fail/crash**?
- ☐ Testing **NOT** as per the requirement (**quality/non functional**)
- ☐ Test with **illegal** or **abnormal data or invalid values**. Intentionally attempt to make things go wrong and to discover/detect.
- ☐ Does the program do what it **should not do**?
- ☐ **Does it fail to do** what it should?



What is Positive View of Negative Testing?

- ❑ Positive view of negative testing: The job of testing is **to discover errors** before the user does.
- ❑ A good tester is one **who is successful in making the system fail**.
- ❑ Mentality of the tester **has to be destructive**--opposite to that of the creator /author /developer, **which should be constructive**
- ❑ Tester and Developer should be collaborated to improve quality



What is Software Testing?

- ❑ One very **popular equation** of software testing is:

Software Testing = **Software Verification** + **Software Validation**

- ❑ As per IEEE definition(s):

Software verification:

“It 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.”

(OR)

“It is the process of **evaluating**, **reviewing**, **inspecting** and doing desk checks of work products such as requirement specifications, design specifications and code.”

(OR)

“It is a human testing activity as it involves looking at the **documents on paper**.”



What is Software Testing?

- ❑ As per IEEE definition(s):
- ❑ Software validation:
- ❑ “It is defined as the process of **evaluating a system or component during or at the end of development process to determine whether it satisfies the specified requirements.**
- ❑ **Software validation** involves **executing the actual software.**
- ❑ Both verification and validation (V&V) are **complementary to each other.**
- ❑ As mentioned earlier, **good testing** expects **more than just running a program.**



Why Should We Test? What is the Purpose?

Testing is necessary- **Reasons from various prospective**

1. The Technical Case:

- a. **Competent developers** are not always effective.
- b. The implications of requirements **are not always predictable**.
- c. The behavior of a system is not necessarily predictable **from its components**.
- d. Languages, databases, user interfaces, and operating systems have bugs **that can cause application failures**.
- e. Reusable classes and objects **must be trustworthy**.

2. The Business Case:

- a. **If you don't find bugs** your customers or users will.
- b. Post-release debugging is the **most expensive form of development**.
- c. Buggy software **hurts operations, sales, and reputation**.
- d. Buggy software can be **hazardous to life and property**.

3. The Professional Case:

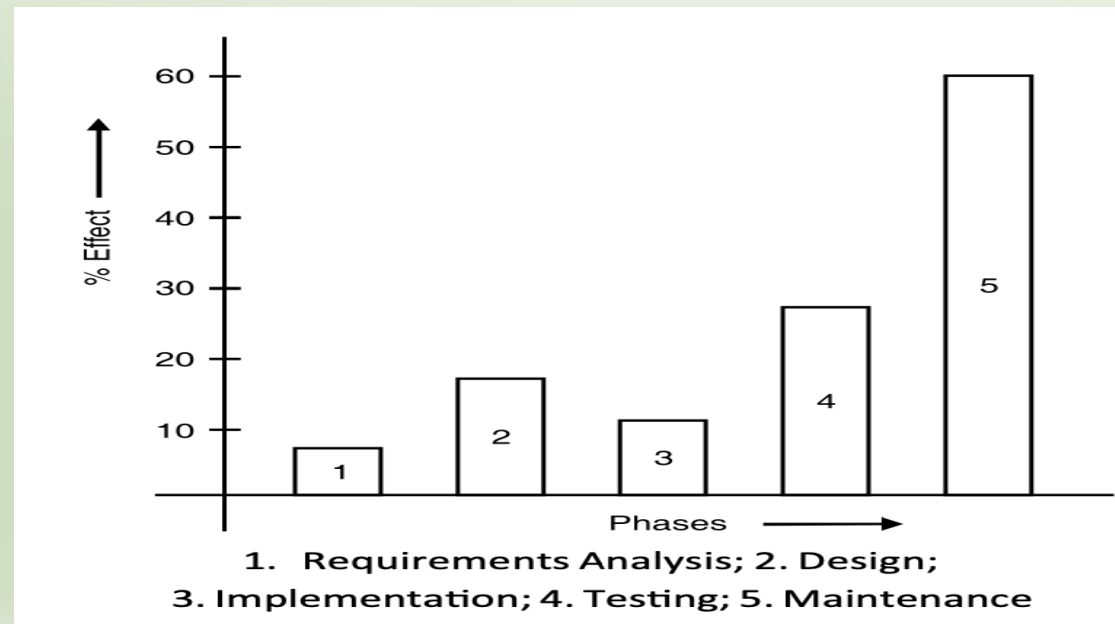
- a. **Test case design** is a challenging and rewarding task.
- b. Good testing allows **confidence in your work**.
- c. Systematic testing allows you to be **most effective**.
- d. Your **credibility is increased** and you have pride in your efforts.



Why Should We Test? What is the Purpose?

4.The Economics Case: Practically speaking, defects get introduced in every phase of SDLC. Pressman has described a defect amplification model wherein he says that **errors get amplified** by a certain factor **if that error is not removed in that phase only**.

- ❑ This may increase the **cost of defect removal**. This principle of detecting errors as close to their point of introduction as possible is known as **phase containment of errors**.



Efforts During SDLC



Why Should We Test? What is the Purpose?

5. To Improve Quality: As computers and software are used in critical applications, the **outcome of a bug can be severe.**

- ☐ Bugs can **cause huge losses.**
- ☐ Bugs in critical systems have caused airplane crashes, allowed space shuttle systems to go awry, and halted trading on the stock market. **Bugs can kill.**
- ☐ Bugs can **cause disasters.**
- ☐ In a computerized embedded world, the **quality** and **reliability** of software is a matter of life and death. This can be achieved only **if thorough testing is done.**

6. For Verification and Validation (V&V): Testing can serve as metrics.

- ☐ It is heavily used as a tool in the **V&V process.**
- ☐ We can compare the quality among different products under the same specification based on results from the same test.
- ☐ Good testing can **provide measures for all relevant quality factors.**



Why Should We Test? What is the Purpose?

7. For Reliability Estimation: Software reliability has important relationships with many aspects of software, including the structure **and the amount of testing done to the software.**

- ☐ Based on an operational profile (an estimate of the relative frequency of use) of various inputs to the program, testing can serve as a statistical sampling **method to gain failure data for reliability estimation.**



Who Should do Testing?

- ☐ As mentioned earlier, testing **starts right from the very beginning**. This implies that testing is **everyone's responsibility**.
- ☐ By “everyone,” we mean **all project team members**. So, we cannot rely on one person only. Naturally, it is **a team effort**.
- ☐ We **cannot** only designate the tester responsible. Even the **developers are responsible**.
- ☐ Developers build the code but **do not indicate any errors** as **they have written their own code**.



How Much Should We Test?

- ❑ Consider that there is a while loop that has three paths. If this loop is executed twice, we have (3×3) paths and so on. So, the total number of paths through such code will be:
$$= 1 + 3 + (3 \times 3) + (3 \times 3 \times 3) + \dots$$
$$= 1 + \sum 3^n \text{ (where } n > 0 \text{)}$$

Note: This means an infinite number of test cases. Thus, testing is **not** 100% exhaustive.

- ❑ **If multiple testing of the functionality** of the software is not showcasing any defects, then it is the right time to stop testing the software.
- ❑ **If you found defects with a small portion of overall functionality**, you should continue the testing.



Selection of Good Test Cases

Designing a good test case is a complex art. It is complex because:

- a. **Different types of test cases** are needed for different classes of information.
- b. **All test cases within a test suite will not be good.** Test cases may be good in variety of ways.
- c. People create test cases according to certain testing styles like domain testing or risk-based testing. And good domain tests are different from good risk-based tests.

Brian Marick coined a new term to a lightly documented test case—the test idea. According to Brian, “A test idea is a brief statement of something that should be tested.” For example, if we are testing a square-root function, one test idea would be—“test a number less than zero.” The idea here is again to check if the code handles an error case.

Cem Kaner said—“The best test cases are the **ones that find bugs.**” Our efforts should be on the test cases that finds issues. Do broad or deep coverage testing **on the trouble spots.**

A test case is a question that you ask of the program. The point of running the test is to gain information like whether the program will **pass or fail the test.**



Measurement of Testing

- ❑ There is **no single scale** that is available to measure the testing progress.
- ❑ A good project manager (PM) wants **worse conditions to occur in the very beginning of the project** instead of in the later phases.
- ❑ If errors are large in numbers, we can say either **testing was not done thoroughly** or it was done so thoroughly that all errors were covered. So there is no standard way to measure our testing process. But metrics can be computed at the **organizational**, **process**, **project**, and **product** levels. Each set of these measurements has its value in monitoring, planning, and control.

Note: Metrics is assisted by four core components—schedule, quality, resources, and size.



Incremental Testing Approach

To be effective, a **software tester** should be knowledgeable in two key areas:

1. Software testing techniques

2. The application under test (AUT)

For each new testing assignment, a tester must invest time **in learning about the application**. A tester with no experience must also learn testing techniques, including general testing concepts and how to define test cases. Our goal is to define a suitable list of tests to perform within a tight deadline.

There are 8 stages for this approach:

Stage 1: Exploration

Purpose: To gain familiarity with the application

Stage 2: Baseline test

Purpose: To devise and execute a simple test case

Stage 3: Trends analysis

Purpose: To evaluate whether the application performs as expected when actual output cannot be predetermined

Stage 4: Inventory

Purpose: To identify the different categories of data and create a test for each category item



Incremental Testing Approach

Stage 5: Inventory combinations

Purpose: To combine different input data

Stage 6: Push the boundaries

Purpose: To evaluate application behavior at data boundaries

Stage 7: Devious data

Purpose: To evaluate system response when specifying bad data

Stage 8: Stress the environment

Purpose: To attempt to break the system

The schedule is tight, so we may not be able to perform all of the stages. The time permitted by the delivery schedule determines how many stages one person can perform. After executing the baseline test, later stages could be performed in parallel if more testers are available.



Basic Terminology Related to Software Testing

We must define the following terminologies one by one:

1. Error (or mistake or bugs): People make errors. When people make mistakes while coding, we call these mistakes bugs. Errors tend to propagate.

A requirements error may be magnified during design and still amplified during coding. So, an error is a mistake during SDLC.

2. Fault (or defect): A missing or incorrect statement in a program resulting from an error is a fault. So, a fault is the representation of an error.

Representation here means the mode of expression, such as a narrative text, data flow diagrams, hierarchy charts, etc. Defect is a good synonym for fault. Faults can be elusive. They requires fixes.

3. Failure: A failure occurs when a fault executes. The manifested inability of a system or component to perform a required function within specified limits is known as a failure. A failure is evidenced by incorrect output, abnormal termination, or unmet time and space constraints. It is a dynamic process.

Eg: Error (or mistake or bug) → Fault (or defect) → Failure.



Basic Terminology Related to Software Testing

4. Incident: When a failure occurs, it may or may not be readily apparent to the user. An incident is the symptom associated with a failure that alerts the user to the occurrence of a failure. It is an unexpected occurrence that requires further investigation. It may not need to be fixed.

5. Test: Testing is concerned with errors, faults, failures, and incidents. A test is the act of exercising software with test cases. A test has two distinct goals—to find failures or to demonstrate correct execution.

6. Test case: A test case has an identity and is associated with program behavior. A test case also has a set of inputs and a list of expected outputs. The essence of software testing is to determine a set of test cases for the item to be tested.

The test case template is shown below.

Test Case ID, Purpose , Preconditions, Inputs, Expected Outputs, Postconditions, Execution History, Date, Result, Version, Run By



Basic Terminology Related to Software Testing

There are 2 types of inputs:

- a. Preconditions: Circumstances that hold prior to test case execution.
- b. Actual inputs: That were identified by some testing method.

Expected outputs are also of two types:

- a. Post conditions
- b. Actual outputs

7. Test suite: A collection of test scripts or test cases that is used for validating bug fixes (or finding new bugs) within a logical or physical area of a product. For example, an acceptance test suite contains all of the test cases that were used to verify that the software has met certain predefined acceptance criteria.

8. Test script: The step-by-step instructions that describe how a test case is to be executed. It may contain one or more test cases.

9. Test ware: It includes all of testing documentation created during the testing process. For example, test specification, test scripts, test cases, test data, the environment specification.

10. Test oracle: Any means used to predict the outcome of a test.

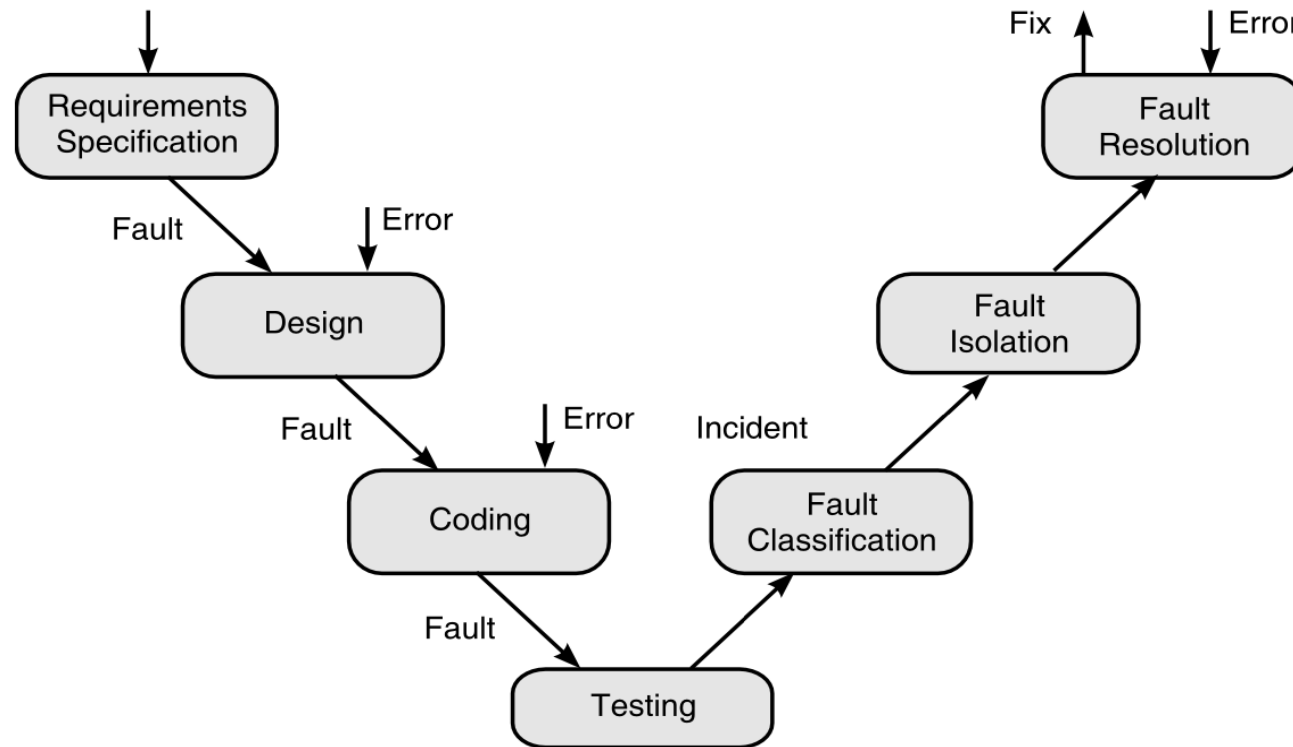
11. Test log: A chronological record of all relevant details about the execution of a test.

12. Test report: A document describing the conduct and results of testing carried out for a system.



Testing Life Cycle

In the development phase, three opportunities arise for errors to be made resulting in faults that propagate through the remainder of the development process. The first three phases are putting bugs IN, the testing phase is finding bugs, and the last three phases are getting bugs OUT. The fault resolution step is another opportunity for errors and new faults. When a fix causes formerly correct software to misbehave, the fix is deficient.



When to Stop Testing?

- ❑ Testing is potentially **endless**. We cannot test until all defects are unearthed and removed. It is simply impossible. At some point, we have to stop testing and ship the software. The question is when?
- ❑ Realistically, testing is a trade-off between **budget**, **time**, and **quality**. It is driven **by profit models**.
- ❑ The pessimistic approach is to stop testing whenever some or any of the allocated resources—time, budget, or test cases—**are exhausted**.
- ❑ The optimistic stopping rule is to stop testing when either reliability **meets the requirement**, or the benefit from continuing testing **cannot justify the testing cost**. [Yang]



Principles of Testing

To make software testing effective and efficient we follow certain principles.

- 1. Testing should be based on user requirements:** This is in order to uncover any defects that might cause the program or system to fail to meet the client's requirements.
- 2. Testing time and resources are limited:** Avoid redundant tests.
- 3. Exhaustive testing is impossible:** As stated by Myer, it is impossible to test everything due to huge data space and the large number of paths that a program flow might take.
- 4. Use effective resources to test:** This represents the most suitable tools, procedures, and individuals to conduct the tests. The test team should use tools like:
 - a. Deja Gnu: It is a testing frame work for interactive or batch-oriented applications. It is designed for regression and embedded system testing. It runs on UNIX platform. It is a cross-platform operating system.



Principles of Testing

5. **Test planning should be done early:** This is because test planning can begin independently of coding and as soon as the client requirements are set.
6. **Testing should begin “in small” and progress toward testing “in large”:** The smallest programming units (or modules) should be tested first and then expanded to other parts of the system.
7. Testing should be conducted by **an independent third party**.
8. All tests should be traceable **to customer requirements**.
9. Assign **best people for testing**. Avoid programmers.
10. Test should be planned to show **software defects** and not their absence.
11. Prepare test reports including test cases and test results **to summarize the results of testing**.
12. Advance test planning is a must and **should be updated in a timely manner**.



Limitations of Testing

1. Testing can show the **presence of errors**-not their absence.
2. No matter how hard you try, **you will never find the last bug in any application.**
3. The domain of possible inputs is **too large to test.**
4. There are too many possible paths **through the program to test.**
5. In short, **maximum coverage through minimum test-cases.** That is the challenge of testing.
6. Various testing techniques are **complementary in nature** and it is only through their combined use that one **can hope to detect most errors.**



QUESTIONS PLEASE 😊

