
Testing Concepts
Lesson 1: Fundamentals of Testing
Capgemini

Lesson Objectives

To understand the following topics:

- Some Facts about Software Systems
- What is Testing?
  - Typical Objectives of Testing
  - Testing and Debugging
- Why is Testing Necessary?
  - Testing's Contributions to Success
  - Quality Assurance and Testing
  - Errors, Defects, and Failures - Reasons behind Errors
  - Defects, Root Causes and Effects
  - Cost of Software Defects
  - Importance of Testing Early in SDLC phases

Lesson Objectives

- Seven Testing Principles
  - Economic of Testing
  - Scope of Software Testing
  - Factors influencing Software Testing
- Test Process
  - Test Process in Context
  - Test Activities and Tasks
  - Test Work Products
  - Traceability between the Test Basis and Test Work Products
- The Psychology of Testing
  - Human Psychology and Testing
    - Attributes of a good Tester
    - Code of Ethics for Tester
  - Tester's and Developer's Mindsets
- Limitations of Software Testing

Some Facts about Software Systems !!

- Software systems are an integral part of life, from business applications (e.g., banking) to consumer products (e.g., cars).
- Most people have had an experience with software that did not work as expected.
- Software that does not work correctly can lead to many problems, including loss of money, time, or business reputation, and even injury or death.

Examples :

1. Excel gives 77.1 x 850 = 100000 instead of 65535
2. Y2K problem in Payroll systems designed in 1974
3. Disney's Lion King – Simba

Brief Explanation on Examples :

1. Excel gives 77.1 x 850 = 100000 instead of 65535 : While multiplying two numbers in MS-Excel and if the product equals 65535, it always gives the result 100000.
2. Y2K problem in Payroll systems designed in 1974 : In 1974, when the first payroll was developed, to minimize the utilization of memory space, the year of the dates were stored in two digits instead of four digits i.e. 00 instead of 1900. But after 25 years in the yr. 2000, the question arose that how to store this. Will it be considered 1900 or 2000 ?
3. Disney's Lion King – Simba : At the fall of 1994 Christmas, the Disney company came up with its first venture – a CDROM game for children with animated story of 'The Lion King Simba'. The sale was very huge. However it was a great loss to the Disney – it failed to work on most of the systems available in the market. The very next day on December 26th, Disney's customer care phones began to ring with calls from angry parents and crying children. Disney failed to properly test the game software on different PC models available in the market and as a result, the software worked only on few systems that were just like the one used by Disney programmers.

1.1 What is Testing ?

Software testing is a way to assess the quality of the software and to reduce the risk of software failure in operation.

- Misperceptions of Testing :
  - Testing only consists of running tests i.e. executing the software and checking the results. But, software testing is a process which includes many different activities and test execution is just one of these activities. The test process includes activities such as test planning, analyzing, designing, implementing and executing the tests, reporting test progress and results, and evaluating the quality of a test object.
  - Testing focuses entirely on verification of requirements, user stories, or other specifications. But, testing also involves checking whether the system meets specified requirements, it also involves validation, which is checking whether the system will meet user and other stakeholder needs in its operational environment(s).

1.1 What is Testing ? (Cont.)

Testing involves both Dynamic testing and Static testing.

- Testing that involves the execution of the component or system being tested; such testing is called **dynamic testing.**
- Testing that involves the reviewing of work products such as requirements, user stories, and source code; such testing is called **static testing.**

Test activities are organized and carried out differently in different lifecycles, explained in lesson 2.

1.1.1 Objectives of Software Testing

- To evaluate work products such as requirements, user stories, design, and code
- To verify whether all specified requirements have been fulfilled
- To validate whether the test object is complete and works as the users and other stakeholders expect
- To build confidence in the level of quality of the test object
- To prevent defects
- To find failures and defects in the Software before User finds it
- To provide sufficient information to stakeholders to allow them to make informed decisions, especially regarding the level of quality of the test object.

1.1.1 Objectives of Software Testing (Cont.)

- To reduce the level of risk of inadequate software quality (e.g., previously undetected failures occurring in operation) and contribute to the delivery of higher quality software product
- To comply with contractual, legal, or regulatory requirements or standards, and/or to verify the test object's compliance with such requirements or standards

Objectives of testing can vary, depending upon the context of the component or system being tested, the test level, and the software development lifecycle model.

Objectives of testing can vary, depending upon the context of the component or system being tested, the test level, and the software development lifecycle model.

Examples :

- During component testing, one objective may be to find as many failures as possible so that the underlying defects are identified and fixed early. Another objective may be to increase code coverage of the component tests.
- During acceptance testing, one objective may be to confirm that the system works as expected and satisfies requirements. Another objective of this testing may be to give information to stakeholders about the risk of releasing the system at a given time.

Software Testing - Definitions

The process of executing a program (or part of a program) with the intention of finding errors - G. J. Myers

Software testing is the process of testing the functionality and correctness of software by running it

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 83a

The process of analyzing a system to detect the difference between existing and required conditions and to evaluate the feature of the system - IEEE/ANSI, 1983 [Std 829-1983]

1.1.2 Testing and Debugging

Testing and debugging are different.

- Executing tests can show failures that are caused by defects in the software.
- Debugging is the development activity that finds, analyzes, and fixes such defects.

Subsequent confirmation testing checks whether the fixes have resolved the defects.

In some cases, testers are responsible for the initial test and the final confirmation test, while developers do the debugging and associated component testing.

However, in Agile development and in some other lifecycles, testers may be involved in debugging and component testing.

1.2 Why is Software Testing necessary ?

- Rigorous testing of components and systems, and their associated documentation, can help reduce the risk of failures occurring during operation.
- When defects are detected, and subsequently fixed, this contributes to the quality of the components or systems.
- Software testing is also required to meet contractual or legal requirements or industry-specific standards.
- SDLC consists of many stages and if bugs are caught in the earlier stages it costs much less to fix them. This saves money and time.
- Software testing provides product security – the user gets a trustworthy product.
- Customer satisfaction - Software Testing helps in bringing out the best user experience possible.

Examples of Security issue:
Sometimes even the smallest security issues have brought huge problems to businesses around the world.

- Cairns Hospital – A catastrophic glitch affecting five Australian hospitals was introduced during the application of security patches designed to counter potential future cyber-attacks. It required more than two weeks for the hospitals to recover their electronic medical record systems.
- China Airlines Airbus A300 crashed due to a software bug on April 26, 1994, killing 264 innocent lives.

Example of Customer Satisfaction issue:
British Airways – For the sixth time in 2017 – a major IT software failure led to massive cancellations on local flights and significant delays on international flights. According to NPR.org – it took over three days of cancellation chaos to resolve the problems that plagued BA during this outage.

1.2.1 Testing's Contributions to Success

Use of appropriate test techniques applied with the appropriate level of test expertise, in the appropriate test levels, and at the appropriate points in the SDLC can reduce the frequency of problematic deliveries.

Examples :

- Testers involved in requirements reviews reduces the risk of incorrect functionality being developed.
- Testers working closely with system designers reduces the risk of fundamental design defects.
- Testers working closely with developers reduces the risk of defects within the code and the tests.
- Testers verifying and validating the software prior to release can detect failures that increases the likelihood that the software meets stakeholder needs and satisfies requirements.
- Achievement of defined test objectives contributes to the success of overall software development and maintenance.

Examples in detail :

- Testers involved in requirements reviews or user story refinement could detect defects in these work products. The identification and removal of requirements defects reduces the risk of incorrect or untestable functionality being developed.
- Testers working closely with system designers while system is being designed can increase each party's understanding of the design and how to test it. This increased understanding can reduce the risk of fundamental design defects and enable tests to be identified at an early stage.
- Testers working closely with developers while the code is under development can increase each party's understanding of the code and how to test it. This increased understanding can reduce the risk of defects within the code and the tests.
- Testers verifying and validating the software prior to release can detect failures that might otherwise have been missed, and support the process of removing the defects that caused the failures (i.e., debugging). This increases the likelihood that the software meets stakeholder needs and satisfies requirements.
- Achievement of defined test objectives (see section 1.1.1) contributes to the success of overall software development and maintenance.

1.2.2 Quality Assurance and Testing

- Quality Assurance (QA) and Testing are not the same, but they are related - Quality management, ties them together.
- Quality management includes both **quality assurance** and **quality control.**

1.2.2 Quality Assurance and Testing (Cont.)

Quality Assurance	Quality Control
It is an preventive approach – prevents the faults from occurring by providing rules and methods.	It is a corrective approach – corrects the faults when they occur.
It is a task conducted in the process.	It is a task conducted on the product.
Gives confidence to customer.	Gives confidence to producer.
It is a set of planned and systematic set of activities that provides adequate confidence and assures that the product conforms to specified requirements.	It is the process by which product quality is compared with applicable standards and appropriate action is taken when non-conformance is detected.
Entire team (designers, coders, testers, etc.) is responsible for QA.	Only tester is responsible for QC.
Examples : use of CASE (engineering) and CAST (testing) tools, training and	Examples: Walkthrough, inspections, etc.

Quality Assurance and Testing

- Quality assurance focuses on proper processes, in order to provide confidence that the appropriate levels of quality will be achieved. When processes are carried out properly, the work products created by those processes are generally of higher quality, which contributes to defect prevention.
- In addition, the use of root cause analysis to detect and remove the causes of defects, along with the proper application of the findings of retrospective meetings to improve processes, are important for effective quality assurance.
- Quality control involves various activities, including test activities, that support the achievement of appropriate levels of quality. Test activities are part of the overall software development or maintenance process.
- Since quality assurance is concerned with the proper execution of the entire process, quality assurance supports proper testing.
- As described in sections 1.1.1 and 1.2.1, testing contributes to the achievement of quality in a variety of ways.

1.2.3 Errors, Defects and Failures

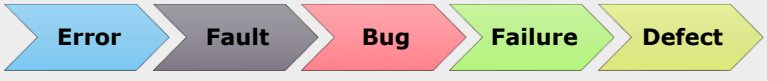
- A person can make an error (mistake), which can lead to the introduction of a defect (fault or bug) in the software code or in some other related work product.
- An error that leads to the introduction of a defect in one work product can trigger an error that leads to the introduction of a defect in a related work product.

Example :

- A requirements elicitation error can lead to a requirements defect, which then results in a programming error that leads to a defect in the code. If a defect in the code is executed, this may cause a failure, but not necessarily in all circumstances. For example, some defects require very specific inputs or preconditions to trigger a failure, which may occur rarely or never.

1.2.3 Errors, Defects and Failures (Cont.)

- Error(Mistake): A human action that produces an incorrect result
- Fault: A stage caused by an error which leads to unintended functionality of the program
- Bug: It is an evidence of the fault. It causes the program to perform in unintended manner. It is found before application goes into beta version
- Failure: Inability of the system to perform functionality according to its requirement
- Defect: It is a mismatch of the actual and expected result identified while testing the software in the beta version



Error-Failure-Defect : Example

Consider the below program for addition of two integers;

```
#include<stdio.h>
1.  int main ()
2.  {
3.      int num1, num2, sum;
4.      num1 = 6;
5.      num2 = 4;
6.      sum = num1 – num2;
7.      printf("6 + 4 = %d", sum);
8.  }
```

Output : 6 + 4 = 2

After compiling and running this program we see that the program has failed to do what it was supposed to do. The program was supposed to add two numbers but it did not. The result of 4 + 6 should be 10, but the result is 2. For now we have detected a ***failure***. The ***fault*** in the program is the line 7 has '-' sign instead of '+' sign which led to a failure (deviation from the required functionality). In this case we can also say we have found the ***bug***. ***Error*** is the mistake programmer made by typing '-' instead of '+' sign. The tester is testing the functionality of the program and realizes the output is faulty and will raise a ***defect***.

Reasons behind Errors

- Time pressure
- Human fallibility
- Inexperienced or insufficiently skilled project participants
- Miscommunication between project participants, including miscommunication about requirements and design
- Complexity of the code, design, architecture, the underlying problem to be solved, and/or the technologies used
- Misunderstandings about intra-system and inter-system interfaces, especially when such intra-system and inter-system interactions are large in number
- New, unfamiliar technologies
- Environmental conditions - for example, radiation, electromagnetic fields, and pollution can cause defects in firmware or influence the execution of software by changing hardware conditions.

Not all unexpected test results are failures.
False positives may occur due to errors in the way tests were executed, or due to defects in the test data, the test environment, or other testware, or for other reasons.
The inverse situation can also occur, where similar errors or defects lead to false negatives. **False negatives** are tests that do not detect defects that they should have detected; false positives are reported as defects, but aren't actually defects.

1.2.4 Defects, Root Causes and Effects

- The root causes of defects are the earliest actions or conditions that contributed to creating the defects.
- Defects can be analyzed to identify their root causes.
- By focusing on most significant root causes, root cause analysis can lead to process improvements that prevent future defects from being introduced.

Example :

- Suppose incorrect interest payments, due to a single line of incorrect code, results in customer complaints. The defective code was written for a user story which was ambiguous, due to the product owner's misunderstanding of how to calculate interest. Therefore, it means that :
  - customer complaints – **effects**
  - incorrect interest payments – **failures**
  - improper calculation in the code – **defect**
  - lack of knowledge on the part of the product owner - **root cause** of this defect
  - Due to lack of knowledge, Product owner makes a **mistake** while writing user story.

Who is a Product Owner ?

Product Owner represents the Customer/User community and is responsible to for working with the user group to determine what features/functions will be in the product release. He defines the user stories and prioritizes the Requirements to streamline the execution of the software application along with the development team.

Example explained in more detail :

Suppose incorrect interest payments, due to a single line of incorrect code, results in customer complaints. The defective code was written for a user story which was ambiguous, due to the product owner's misunderstanding of how to calculate interest.

If a large percentage of defects exist in interest calculations, and these defects have their root cause in similar misunderstandings, the product owners could be trained in the topic of interest calculations to reduce such defects in the future.

In this example, the customer complaints are **effects.** The incorrect interest payments are **failures.** The improper calculation in the code is a **defect,** and the **root cause** of this defect was a lack of knowledge on the part of the product owner, which resulted in the product owner making a **mistake** while writing the user story.

Cost of Software Defects

It is Easy to find and fix defect in early stages rather than in the later phases of software.

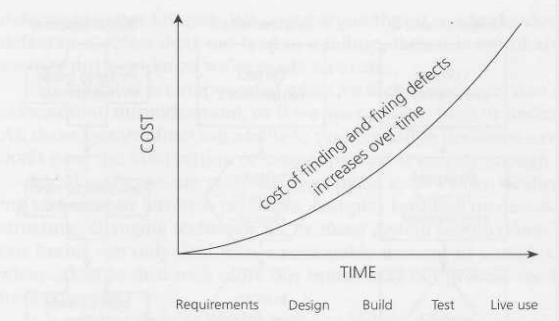


FIGURE 1.2 Cost of defects

Cost of Software Defects :
The cost of fixing a bug (defect) and making the required changes in early phases of software development is less as compared to the same detected in later phases since cost is spent at four different times in a SDLC;
First, cost is spent in writing wrong specifications, erroneous coding and incorrectly documenting the system
Second, cost is spent on detecting the errors rather on preventing errors
Third, cost is spent on removing discovered errors in specifications, coding, and documentation
Fourth, cost is spent on re-testing the system to determine whether the errors have been fixed.

Therefore, to achieve lower cost and high quality systems, testing must not be considered as single phase activity; it must be incorporated in all SDLC phases.

Importance of Testing Early in SDLC Phases

- Prevents future Problems thus lowering the cost
- Testing will not be a bottleneck anymore
- Testers become more familiar with the software, as they are more involved with the evolution of the product.
- Reduces the chances of failure
- The test environment can be prepared in advance
- The risk of having a short time for testing is greatly reduced
- Maintains "quality culture" in the organization

Importance of Testing Early in SDLC Phases

- Many problems raise during planning or design. Requirements testing can prevent future problems thus lowering the cost.
- Since the testing process is involved with all phases of the SDLC, Management will not feel as if testing is a bottleneck to release the product.
- Test cases written during requirements and shared with the Dev. team before the construction phase can help developers to reduce the chances of failure
- The test environment can be prepared in advance
- The risk of having a short time for testing is greatly reduced
- Involving quality assurance in all phases of the SDLC helps creating a 'quality culture' inside the organization.

1.3 Seven Testing Principles

Principle 1 - Testing shows presence of defects, not their absence

Principle 2 - Exhaustive testing is impossible

Principle 3 - Early testing saves time and money

Principle 4 - Defects Cluster together

Principle 5 – Beware of the Pesticide Paradox

Principle 6 - Testing is context dependent

Principle 7 - Absence of Errors is fallacy

Seven Testing Principles

1. Testing shows the presence of defects not their absence. Testing can show that defects are present, but cannot prove that there are no defects. Testing reduces the probability of undiscovered defects remaining in the software but, even if no defects are found, testing is not a proof of correctness.
2. Exhaustive testing is impossible. Testing all combinations of inputs and preconditions is not feasible except for trivial cases. Rather than attempting to test exhaustively, risk analysis, test techniques, and priorities should be used to focus test efforts.
3. Early testing saves time and money. To find defects early, both static and dynamic test activities should be started as early as possible in the SDLC. Early testing is sometimes referred to as shift left.
4. Defects cluster together. A small number of modules usually contain most of the defects discovered during pre-release testing, or is responsible for most of the operational failures. Predicted defect clusters, and the actual observed defect clusters in test or operation, are an important input into a risk analysis used to focus the test effort.
5. Beware of the pesticide paradox If the same tests are repeated over and over again, eventually these tests no longer find any new defects. To detect new defects, existing tests and test data may need changing, and new tests may need to be written. (Tests are no longer effective at finding defects, just as pesticides are no longer effective at killing insects after a while.) In some cases, such as automated regression testing, the pesticide paradox has a beneficial outcome, which is the relatively low number of regression defects.
6. Testing is context dependent. Testing is done differently in different contexts. For example, safety-critical industrial control software is tested differently from an e-commerce mobile app; testing in an Agile project is done differently than testing in a sequential lifecycle project (explained in section 2.1).
7. Absence-of-errors is a fallacy. Some organizations expect that testers can run all possible tests and find all possible defects, but principles 2 and 1, respectively, tell us that this is impossible. Further, it is a fallacy (i.e., a mistaken belief) to expect that just finding and fixing a large number of defects will ensure the success of a system. For example, thoroughly testing all specified requirements and fixing all defects found could still produce a system that is difficult to use, that does not fulfill the users' needs and expectations, or that is inferior compared to other competing systems.

Economics of Testing

Economics of Testing

- It is both the driving force and the limiting factor

Driving - Earlier the errors are discovered and removed in the lifecycle, lowers the cost of their removal.

Limiting - Testing must end when the economic returns cease to make it worth while i.e. the costs of testing process significantly outweigh the returns

Although testing is itself an expensive activity, the cost of not testing is potentially much higher. The most damaging errors are those which are not discovered during the testing process and therefore remain when the system goes live.

Limiting - It is infeasible to test exhaustively all but the most simple or the most vital of s/w.

Here we discuss how exhaustive testing is impossible to achieve.

Consider the following example:

Exhaustive input testing (Black box)

Lets assume that we have written a function say ax2 + bx +c=0
Assume 16 bit numbers

So each input is 2power16
And so total test cases is 2 power16*2power16*2power16=2power48 test cases which is impractical.

Scope of Software Testing

Bad news : You can't test everything
Good news : There is such a thing as "good enough"
Bad news : Good enough may cost too much

What do we do : Increase focus via systematic process of elimination

What you might test?

- Those areas which are within the scope of your project

What you should test?

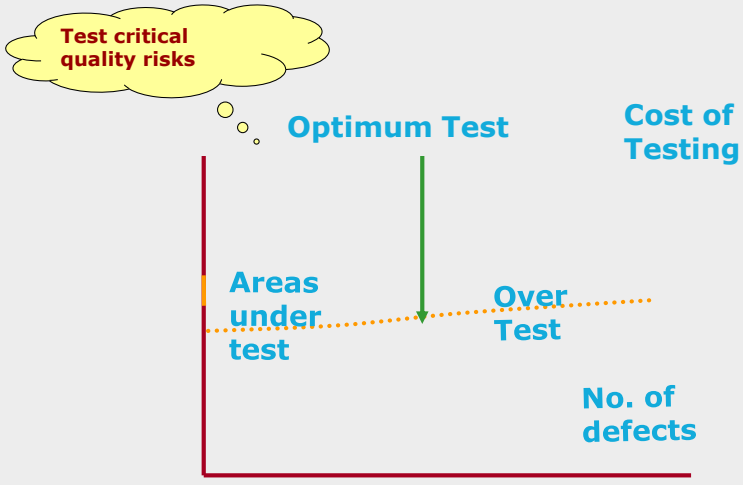
- The critical system functionality which effects the customers & users experience of quality

What you can test?

- Estimate time & resource for risk driven effort

Scope of Software Testing

"Understanding risk is the key to Optimum testing"



Factors influencing the Scope of Testing

Contractual requirements

Legal requirements

- Privacy related laws
- Non-disclosure of identity

Industry-specific requirements

- Aircraft safety equipment

Scope of Testing is about identifying the correct test cases for automation

The steps involved are:

- Identify various factors that form the basis of identifying the candidate test cases
- Apply 'Divide & rule' strategy : Break the application into smaller modules
- Analyze each module to identify the candidate test cases
- Calculate ROI

Factors influencing the scope of testing :

- In small projects
  - Test case writing
  - Test case execution
  - Regression testing
- In Large projects
  - Setting up test bed
  - Generating test data, test scripts, etc.

1.4 Test Process

Testing is a process rather than a single activity.

The quality and effectiveness of software testing is primarily determined by the quality of the test processes used.

The activities of testing can be divided into the following basic steps:

- Planning and Control
- Analysis and Design
- Implementation and Execution - Evaluating and Reporting
- Test Completion/Closure activities

Test process :
It provides a phenomenon of executing the testing activities in a systematic and planned way.
Test process is a set of various testing activities carried out to improve the quality of the product.

1.4.1 Test Process in Context

The proper, specific software test process in any given situation depends on many factors.

Few of the Contextual factors that influence the test process are :

- SDLC model and project methodologies being used
- Test levels and test types being considered
- Product and project risks
- Business domain
- Operational constraints that include :
  - Budgets and resources
  - Timescales
  - Complexity
  - Contractual and regulatory requirements
- Organizational policies and practices
- Required internal and external standards

Operational constraints that include Budgets and resources : Inaccurate or unreasonable assumptions can quickly make a budget unrealistic. Budgets can lead to inflexibility in decision-making.
Organizational Policies and practices typically stem from the company vision and objectives. They are designed to influence and determine all major decisions and actions, and all activities take place within the boundaries set by them.

1.4.1 Test Process in Context (Cont.)

The following sections describe general aspects of organizational test processes in terms of the following:

- Test activities and tasks
- Test work products
- Traceability between the test basis and test work products

It is very useful if the test basis (for any level or type of testing that is being considered) has measurable coverage criteria defined.

The coverage criteria can act effectively as key performance indicators (KPIs) to drive the activities that demonstrate achievement of software test objectives (defined in section 1.1.1).

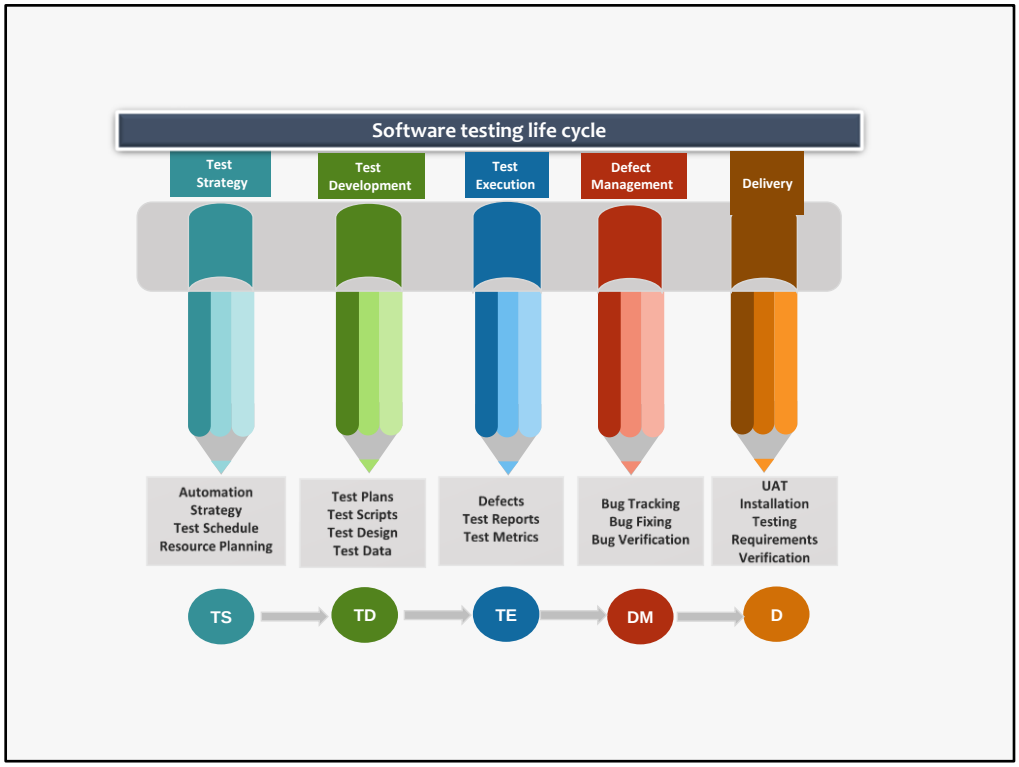
Example:

For a mobile application, the test basis may include a list of requirements and a list of supported mobile devices. Each requirement is an element of the test basis. Each supported device is also an element of the test basis. The coverage criteria may require at least one test case for each element of the test basis. Once executed, the results of these tests tell stakeholders whether specified requirements are fulfilled and whether failures were observed on supported devices.

1.4.2 Test Activities and Tasks

- Test planning
- Test monitoring and control
- Test analysis
- Test design
- Test implementation
- Test execution
- Test completion

Although many of these activity groups may appear logically sequential, they are often implemented iteratively. For example, Agile development involves small iterations of software design, build, and test that happen on a continuous basis, supported by on-going planning. So test activities are also happening on an iterative, continuous basis within this development approach. Even in sequential development, the stepped logical sequence of activities will involve overlap, combination, concurrency, or omission, so tailoring these main activities within the context of the system and the project is usually required.


Software testing life cycle
Test Strategy
Test Development
Test Execution
Defect Management
Delivery
Automation Strategy Test Schedule Resource Planning
Test Plans Test Scripts Test Design Test Data
Defects Test Reports Test Metrics
Bug Tracking Bug Fixing Bug Verification
UAT Installation Testing Requirements Verification
TS
TD
TE
DM
D

Test Planning

- Test planning involves activities that define the testing objectives (refer section 1.1.1) and the approach for meeting test objectives within constraints imposed by the context.

  For example, specifying suitable test techniques and tasks, and formulating a test schedule for meeting a deadline.
- Test plans may be revisited based on feedback obtained from monitoring and control activities.

Test planning is the first step of the testing process. In this phase we identify the activities and resources which would help to meet the testing objectives.
During planning we also try to identify the metrics, the method of gathering and tracking those metrics.
Identify the need and feasibility of tools to be used based on project profitability, enhanced delivery capability and ease & effectiveness of testing.
Project Manager develops a Test strategy document that defines the testing approaches to achieve testing objectives.

Test Monitoring and Control

- Test monitoring involves an on-going comparison of actual progress against the test plan using any test monitoring metrics defined in the test plan.
- Test control involves taking actions necessary to meet the objectives of the test plan.
- Test monitoring and control are supported by the evaluation of exit criteria.
- For example, the evaluation of exit criteria for test execution as part of a given test level may include:
  - Checking test results and logs against specified coverage criteria
  - Assessing the level of component or system quality based on test results and logs
  - Determining if more tests are needed (e.g., if tests originally intended to achieve a certain level of product risk coverage failed to do so, requiring additional tests to be written and executed)

Test progress against the plan is communicated to stakeholders in test progress reports, including deviations from the plan and information to support any decision to stop testing.

Test monitoring and control are further explained in section 5.3.

Test Analysis

Test analysis determines "what to test" in terms of measurable coverage criteria.

Test analysis includes the following major activities:

1. Analyzing the test basis appropriate to the test level being considered.
   - Requirement specifications, such as business requirements, functional requirements, system requirements, user stories, epics, use cases, or similar work products that specify desired functional and non-functional component or system behavior
   - Design and implementation information, such as system or software architecture diagrams or documents, design specifications, call flows, modelling diagrams (e.g., UML or ER diagrams), interface specifications, or similar work products that specify component or system structure
   - The implementation of the component or system itself, including code, database metadata and queries, and interfaces
   - Risk analysis reports, which may consider functional, non-functional, and structural aspects of the component or system

Analyse: Defines "WHAT" to be tested.

Test Analysis (Cont.)

2. Evaluating the test basis and test items to identify defects of various types, such as:
   - Ambiguities
   - Omissions
   - Inconsistencies
   - Inaccuracies
   - Contradictions
   - Superfluous statements

Test Analysis (Cont.)

3. Identifying features and sets of features to be tested
4. Defining and prioritizing test conditions for each feature based on analysis of the test basis, and considering functional, non-functional, and structural characteristics, other business and technical factors, and levels of risks
5. Capturing bi-directional traceability between each element of the test basis and the associated test conditions (see sections 1.4.3 and 1.4.4)

The application of black-box, white-box, and experience-based test techniques can be useful in the process of test analysis (see chapter 4) to reduce the likelihood of omitting important test conditions and to define more precise and accurate test conditions.
In some cases, test analysis produces test conditions which are to be used as test objectives in test charters. Test charters are typical work products in some types of experience-based testing (see section 4.4.2). When these test objectives are traceable to the test basis, coverage achieved during such experience-based testing can be measured.
The identification of defects during test analysis is an important potential benefit, especially where no other review process is being used and/or the test process is closely connected with the review process. Such test analysis activities not only verify whether the requirements are consistent, properly expressed, and complete, but also validate whether the requirements properly capture customer, user, and other stakeholder needs.
For example, techniques such as behavior driven development (BDD) and acceptance test driven development (ATDD), which involve generating test conditions and test cases from user stories and acceptance criteria prior to coding, also verify, validate, and detect defects in the user stories and acceptance criteria.

Test Design

- During test design, the test conditions are elaborated into high-level test cases, sets of high-level test cases, and other testware. So, test analysis answers the question "what to test?" while test design answers the question "how to test?"
- Test design includes the following major activities:
- Designing and prioritizing test cases and sets of test cases
- Identifying necessary test data to support test conditions and test cases
- Designing the test environment and identifying any required infrastructure and tools
- Capturing bi-directional traceability between the test basis, test conditions, test cases, and test procedures (see section 1.4.4)

The elaboration of test conditions into test cases and sets of test cases during test design often involves using test techniques (see chapter 4).

Design : Defines "HOW" to test.
The testers will develop the test cases based against the requirements of the customer. There are usually three levels of requirements, to be understood by the testers before they can proceed to write the test cases for the product
•HLI (High level Information)
•LLI / Use Cases (Low level Information)
•Snapshots (Prototype or images of a similar product or framework.)

Test Implementation

- During test implementation, the testware necessary for test execution is created and/or completed, including sequencing the test cases into test procedures.
- Test design answers the question “how to test?” while test implementation answers the question “do we now have everything in place to run the tests?”

Test Implementation :
One of the best techniques is to refer the use cases, pick up the standard test case templates and prepare test cases based on different categories. While do remember to maintain Traceability matrix alongside test execution to avoid rework and confusions at a later stage.

Test Implementation (Cont.)

Test implementation includes the following major activities:

- Developing and prioritizing test procedures, and, potentially, creating automated test scripts
- Creating test suites from the test procedures and automated test scripts
- Arranging the test suites within a test execution schedule in a way that results in efficient test execution (see section 5.2.4)
- Building the test environment (including, potentially, test harnesses, service virtualization, simulators, and other infrastructure items) and verifying that everything needed has been set up correctly
- Preparing test data and ensuring it is properly loaded in the test environment
- Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test suites (see section 1.4.4)

Test design and test implementation tasks are often combined.
In exploratory testing and other types of experience-based testing, test design and implementation may occur, and may be documented, as part of test execution. Exploratory testing may be based on test charters (produced as part of test analysis), and exploratory tests are executed immediately as they are designed and implemented (see section 4.4.2).

Test Execution

During test execution, test suites are run in accordance with the test execution schedule.

Test execution includes the following major activities:

- Recording the IDs and versions of the test item(s) or test object, test tool(s), and testware
- Executing tests either manually or by using test execution tools
- Comparing actual results with expected results
- Analyzing anomalies to establish their likely causes (e.g., failures may occur due to defects in the code, but false positives also may occur (see section 1.2.3))
- Reporting defects based on the failures observed (see section 5.6)
- Logging the outcome of test execution (e.g., pass, fail, blocked)
- Repeating test activities either as a result of action taken for an anomaly, or as part of the planned testing (e.g., confirmation testing, or regression testing)
- Verifying and updating bi-directional traceability between the test basis, test conditions, test cases, test procedures, and test results.

Bi-directional traceability : is the ability to trace both forward and backward i.e., from requirements to end products and from end product back to requirements.
Test basis : It is defined as the source of information or the document that is needed to write test cases and perform test analysis. For example, SRS.
Test Condition : These are the constraints that you should follow to test an application. For example, When User Name and Password are valid then application will move forward.
Test Cases : It is a set of test conditions under which a tester will determine whether a system under test satisfies requirements and works correctly.
Test Procedures : It is a formal script specification of test cases to be applied to a target module.
Test Results : It is a kind of confirmation that gradually leads us to the right track towards obtaining a robust software application.

Test Completion

Test completion includes the following major activities:

- Checking whether all defect reports are closed, entering change requests or product backlog items for any defects that remain unresolved at the end of test execution
- Creating a test summary report to be communicated to stakeholders
- Finalizing and archiving the test environment, the test data, the test infrastructure, and other testware for later reuse
- Handing over the testware to the maintenance teams, other project teams, and/or other stakeholders who could benefit from its use
- Analyzing lessons learned from the completed test activities to determine changes needed for future iterations, releases, and projects
- Using the information gathered to improve test process maturity.

Test completion activities collect data from completed test activities to consolidate experience, testware, and any other relevant information.

Test completion activities occur at project milestones such as when a software system is released, a test project is completed (or cancelled), an Agile project iteration is finished (e.g., as part of a retrospective meeting), a test level is completed, or a maintenance release has been completed.

1.4.3 Test Work Products

Test work products are created as part of the test process. Many of the test work products can be captured and managed using test management tools and defect management tools (see chapter 6).

- Test planning work products
- Test monitoring and control work products
- Test analysis work products
- Test design work products
- Test implementation work products
- Test execution work products
- Test completion work products

Test planning work products
This includes one or more test plans. The test plan includes information about the test basis, to which the other test work products will be related via traceability information (see section 1.4.4), as well as exit criteria which will be used during test monitoring and control.
Test plans are described in section 5.2.

Test monitoring and control work products

- Test monitoring and control work products typically include various types of test reports, including test progress reports (produced on an ongoing and/or a regular basis) and test summary reports (produced at various completion milestones).
- All test reports should provide audience-relevant details about the test progress as of the date of the report, including summarizing the test execution results once those become available.
- Test monitoring and control work products should also address project management concerns, such as task completion, resource allocation and usage, and effort.
- Test monitoring and control, and the work products created during these activities, are further explained in section 5.3 of this syllabus.

1.4.4 Traceability between Test Basis and Test Work Products

It is important to establish and maintain traceability throughout the test process between each element of the test basis and the various test work products associated with it.

In addition to the evaluation of test coverage, good traceability supports:

- Analyzing the impact of changes
- Making testing auditable
- Meeting IT governance criteria
- Improving the understandability of test progress reports and test summary reports to include the status of elements of the test basis (e.g., requirements that passed their tests, requirements that failed their tests, and requirements that have pending tests)
- Relating the technical aspects of testing to stakeholders in terms that they can understand
- Providing information to assess product quality, process capability, and project progress against business goals.

Some test management tools provide test work product models that match these test work products. Some organizations build their own management systems to organize the work products and provide the information traceability they require.

Attributes of a good Tester

- A good test engineer has a 'test to break' attitude
- Need to take a different view, a different mindset (“What if it isn’t?”, “What could go wrong?”)
- An ability to understand the point of view of the customer
- A passion for quality and attention to detail
- Notice little things that others miss/ignore (See symptom not bug)
- Ability to communicate fault information to both technical (developers) and non-technical (managers and customers)
- Tact and diplomacy for maintaining a cooperative relationship with developers
- Work under worst time pressure (at the end)
- “Patience”

1.5 Psychology of Testing

- Identifying defects during a static test or identifying failures during dynamic test execution, was perceived as criticism of the product and of its author.
- Since developers expect their code to be correct, they have a confirmation bias that makes it difficult to accept that the code is incorrect.
- As a result of these psychological factors, some people may perceive testing as a destructive activity, even though it contributes greatly to project progress and product quality
- To try to reduce these perceptions and tensions between the testers and the analysts, product owners, designers, and developers, the information about defects and failures should be communicated in a constructive way.

Code of Ethics for Tester

Involvement in software testing enables individuals to learn confidential and privileged information. A code of ethics is therefore necessary, among other reasons to ensure that the information is not put to inappropriate use.

PUBLIC - Tester shall act consistently with public interest

CLIENT AND EMPLOYER - Tester shall act in best interests of their client and employer

PRODUCT - Tester shall ensure that the deliverables they provide meet highest professional standards possible

JUDGMENT - Tester shall maintain integrity and independence in their professional judgment

MANAGEMENT - Tester managers and leaders shall subscribe to and promote an ethical approach to manage software testing

PROFESSION - Tester shall advance the integrity and reputation of the profession consistent with the public interest

COLLEAGUES - Tester shall be fair to and supportive of their colleagues, and promote cooperation with software developers

SELF - Tester shall participate in lifelong learning and shall promote an ethical approach to the practice of the profession

Independent testers bring a perspective which is different than that of the authors since they have different cognitive biases from the authors.

Limitations of Software Testing

- We use testing to disclose many hidden errors but this methodology never guarantees the absence of errors. It is only used to identify the known errors. It never gives any information about those defects which remain uncovered.
- Testing do not provide you any help when you have to make a decision either “you should release the product consisting errors for meeting the deadline” or “you should release late by compromising the deadline”.
- Software testing does not predicts or estimate the proper functioning of the product under different conditions, but it may prove to be helpful in delivering the information w.r.t. Incorrect or improper functioning of the product.
- While injecting the defects, software testing unable to find the root causes which may help in placing defects at the first place. Identifying the root causes of defects/errors helps in injection of defects for future purposes.
- Testing cannot be done against system requirements. We also cannot detect any errors in requirements or ambiguous requirement leads the complete testing process to inadequate testing.
- When it comes to major constraints like Time & Budget, it requires attentive planning of test effort. Mostly, we compromise in between thoroughness and budget at the time of testing.

Limitations of Software Testing
Manual Testing Limitations

- It requires more time and resources to accomplish quality goal. Many times, it requires both.
- When it comes to **performance testing**, it seems to be very impractical in terms of manual testing.
- With manual testing, it possess least accuracy.
- Executing recursive tests again and again take too much amount of time which lead to project delay and incomplete testing.
- GUI object size difference and color combination are the two essential measures which are very difficult to find by using manual testing.
- Manual testing process is not suitable for large scale projects and limited time projects.
- Comparing huge amount of data will be unrealistic in terms of manual testing.
- At the time of software maintenance, processing of change requests takes huge amount of time.

Limitations of Software Testing
Testing Tools Limitations

- Unrealistic expectation from tools.
- Many times, testers make mistakes by underestimating factors like time, cost, and effort for initial introduction of tool.
- Testers repeated the miscalculation of time and effort required to achieve prominent amount of benefit from tool.
- Testers underestimate the effort required to maintain the test assets which are generated by tool.
- Testing teams are too much dependent on tools

Summary

In this lesson, you have learnt:

- Testing is an extremely creative & intellectually challenging task
- No software exists without bug
- Testing is conducted with the help of users requirements, design documents, functionality, internal structures & design, by executing code
- Scope of testing
- The cost of not testing is potentially much higher
- Testing is in a way a destructive process
- A successful test case is one that brings out an error in program
- Various principles of testing

Review Question

Question 1: What is visible to end-users is a deviation from the specific or expected behavior is called as

- Defect
- Bug
- Failure
- Fault

Question 2: _____ is a planned sequence of actions.

Question 3: Pick the best definition of Quality :

- Quality is job done
- Zero defects
- Conformance to requirements
- Work as designed

Question 4: One cannot test a program completely to guarantee that it is error free (T/F)

Question 5: One can find missing paths due to coding errors (T/F)