Software Engineering

T.E – B

Unit 6

Faculty Incharge: Ms. Drashti Shrimal

Q.1 What is Software Testing?

Software testing can be stated as the process of verifying and validating whether a software or application is bug-free, meets the technical requirements as guided by its design and development, and meets the user requirements effectively and efficiently by handling all the exceptional and boundary cases.

The process of software testing aims not only at finding faults in the existing software but also at finding measures to improve the software in terms of efficiency, accuracy, and usability. It mainly aims at measuring the specification, functionality, and performance of a software program or application.

Software testing can be divided into two steps:
1. **Verification:** it refers to the set of tasks that ensure that the software correctly implements a specific function.
2. **Validation:** it refers to a different set of tasks that ensure that the software that has been built is traceable to customer requirements.
Verification: "Are we building the product right?"
Validation: "Are we building the right product?"

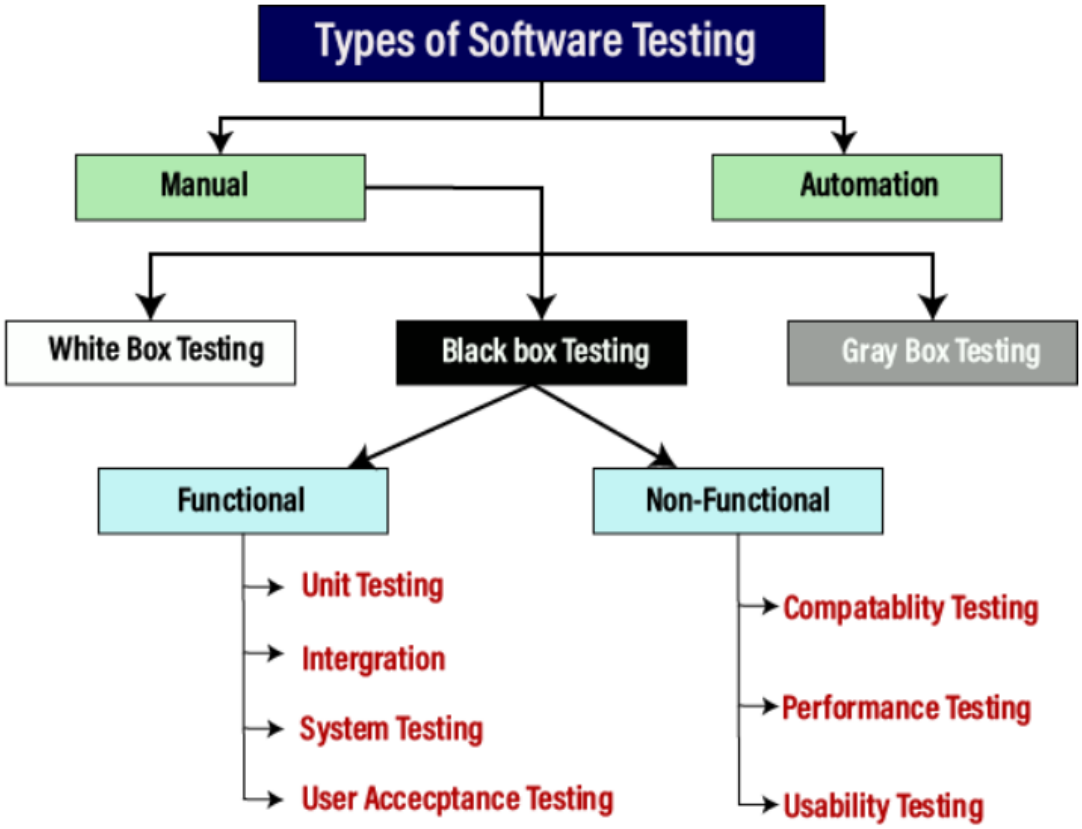
Q.2 What are different types of software testing?

Software Testing can be broadly classified into two types:

1. **Manual Testing:** Manual testing includes testing software manually, i.e., without using any automation tool or any script. In this type, the tester takes over the role of an end-user and tests the software to identify any unexpected behavior or bug. There are different stages for manual testing such as unit testing, integration testing, system testing, and user acceptance testing.
Testers use test plans, test cases, or test scenarios to test software to ensure the completeness of testing. Manual testing also includes exploratory testing, as testers explore the software to identify errors in it.

2. **Automation Testing:** Automation testing, which is also known as Test Automation, is when the tester writes scripts and uses software to test the product. This process involves the automation of a manual process. Automation Testing is used to re-run the test scenarios quickly and repeatedly, that were performed manually in manual testing.
Apart from regression testing, automation testing is also used to test the application from a load, performance, and stress point of view. It increases the test coverage, improves accuracy, and saves time and money when compared to manual testing.

Q.3 What are Software Testing techniques?



1. Black box testing:

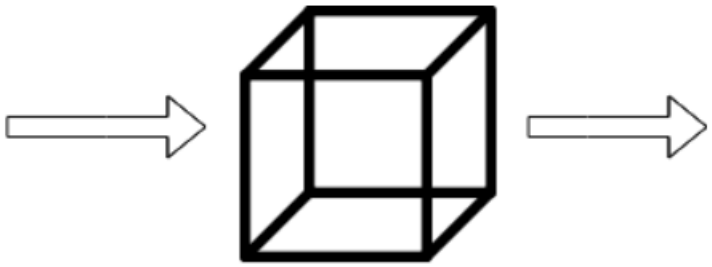
Black box testing is a technique of software testing which examines the functionality of software without peering into its internal structure or coding. The primary source of black box testing is a specification of requirements that is stated by the customer.

In this method, tester selects a function and gives input value to examine its functionality, and checks whether the function is giving expected output or not. If the function produces correct output, then it is passed in testing, otherwise failed. The test team reports the result to the development team and then tests the next function. After completing testing of all functions if there are severe problems, then it is given back to the development team for correction.

2. White box testing:

White box testing techniques analyze the internal structures the used data structures, internal design, code structure, and the working of the software rather than just the functionality as in black box testing. It is also called glass box testing or clear box testing or structural testing. White Box Testing is also known as transparent testing, open box testing.
White box testing is a software testing technique that involves testing the internal structure and workings of a software application. The tester has access to the source code and uses this knowledge to design test cases that can verify the correctness of the software at the code level.

White box testing is also known as structural testing or code-based testing, and it is used to test the software's internal logic, flow, and structure. The tester creates test cases to examine the code paths and logic flows to ensure that they meet the specified requirements.



Whitebox Testing

Other testing techniques:

1. Unit testing-

Unit testing is a type of software testing that focuses on individual units or components of a software system. The purpose of unit testing is to validate that each unit of the software works as intended and meets the requirements. Unit testing is typically performed by developers, and it is performed early in the development process before the code is integrated and tested as a whole system.

The objective of Unit Testing is:

- To isolate a section of code.
- To verify the correctness of the code.
- To test every function and procedure.
- To fix bugs early in the development cycle and to save costs.

- To help the developers to understand the code base and enable them to make changes quickly.
- To help with code reuse.

Here are some commonly used Unit Testing tools:

- Jtest
- Junit
- NUnit
- EMMA
- PHPUnit

2. Integration Testing:

Integration testing is the process of testing the interface between two software units or modules. It focuses on determining the correctness of the interface. The purpose of integration testing is to expose faults in the interaction between integrated units. Once all the modules have been unit tested, integration testing is performed.

Integration testing is a software testing technique that focuses on verifying the interactions and data exchange between different components or modules of a software application. The goal of integration testing is to identify any problems or bugs that arise when different components are combined and interact with each other. Integration testing is typically performed after unit testing and before system testing. It helps to identify and resolve integration issues early in the development cycle, reducing the risk of more severe and costly problems later on.

3. System Testing:

System Testing is a level of testing that validates the complete and fully integrated software product. The purpose of a system test is to evaluate the end-to-end system specifications. Usually, the software is only one element of a larger computer-based system. Ultimately, the software is interfaced with other software/hardware systems. System Testing is defined as a series of different tests whose sole purpose is to exercise the full computer-based system.

Types of System Testing:

- **Performance Testing:** Performance Testing is a type of software testing that is carried out to test the speed, scalability, stability and reliability of the software product or application.
- **Load Testing:** Load Testing is a type of software Testing which is carried out to determine the behavior of a system or software product under extreme load.
- **Stress Testing:** Stress Testing is a type of software testing performed to check the robustness of the system under the varying loads.
- **Scalability Testing:** Scalability Testing is a type of software testing which is carried out to check the performance of a software application or system in terms of its capability to scale up or scale down the number of user request load.

4. Basis Path Testing:

Basis Path Testing is a white-box testing technique based on the control structure of a program or a module. Using this structure, a control flow graph is prepared and the various possible paths present in the graph are executed as a part of testing. Therefore, by definition, Basis path testing is a technique of selecting the paths in the control flow graph, that provide a basis set of execution paths through the program or module. Since this testing is based on the control structure of the program, it requires complete knowledge of the program's structure.

To conduct *basis path testing* of a program, follow the steps below:

1. Draw the control flow graph of the program.
2. Calculate the **cyclomatic complexity** of the control flow graph. This will be the maximum number of independent paths in the graph.
3. Identify independent paths in the control flow graph.
4. Design test cases based on the independent paths identified so that the test cases execute all independent paths.

Q.5 Describe the test plan.

- A test plan is a detailed document which describes software testing areas and activities. It outlines the test strategy, objectives, test schedule, required resources (human resources, software, and hardware), test estimation and test deliverables.
- The test plan is a base of every software's testing. It is the most crucial activity which ensures availability of all the lists of planned activities in an appropriate sequence.
- The test plan is a template for conducting software testing activities as a defined process that is fully monitored and controlled by the testing manager. The test plan is prepared by the Test Lead (60%), Test Manager (20%), and by the test engineer (20%).

Types of Test Plan:

1. Master Test Plan

Master Test Plan is a type of test plan that has multiple levels of testing. It includes a complete test strategy.

2. Phase Test Plan

A phase test plan is a type of test plan that addresses any one phase of the testing strategy. For example, a list of tools, a list of test cases, etc.

3. Specific Test Plans

Specific test plan designed for major types of testing like security testing, load testing, performance testing, etc. In other words, a specific test plan designed for non-functional testing.

Sample Test Plan: *attached separately excel file.

Test Attributes:

There is no hard and fast rule of preparing a test plan but it has some **standard 15 attributes** that companies follow:

A. **Objective:** It describes the aim of the test plan, whatever the good process and procedure they are going to follow in order to give quality software to customers. The overall objective of the test is to find as many defects as possible and to make software bug free.
B. **Test Strategy:** It is a crucial document that is to be performed and usually designed by the Test Manager. It helps to determine Test Effort and Test cost.
C. **Testing Methodology:** The methods that are going to be used for testing depend on application to application. The testing methodology is decided based on the feature and application requirements.
D. **Approach:** The approach of testing different software is different.
E. **Assumptions:** In this phase, certain assumptions will be made.
F. **Risk:** All the risks that can happen if the assumption is breaking. For Example, in the case of wrong budget estimation, the cost may overrun.
G. **Backup/Mitigation Plan-** If any risk is involved then the company must have a backup plan, the purpose is to avoid errors.
H. **Roles and Responsibilities:** All the responsibilities and role of every member in a particular testing team has to be recorded.
I. **Scheduling:** Under this, it will record the start and the end date of each and every testing-related activity.
J. **Defect Tracking:** It is an important process in software engineering as lots of issue arises when you develop a critical system for business. If there is any defect found while testing and that defect must be given to the developer team.
K. **Test Environment-** It is the environment which the testing team will use i.e. the list of hardware and software, while testing the application, the things which are said to be tested will be written under this section.
L. **Entry and Exit Criteria:** The set of conditions that should be met in order to start any new type of testing or to end any kind of testing.
M. **Test Automation:** It consists of the features that are to be automated and which features are not to be automated.

N. **Deliverables-** It is the outcome from the testing team and that is to be given to the customers at the end of the project.

O. **Template:** It is followed by every kind of report that is going to be prepared by the testing team.

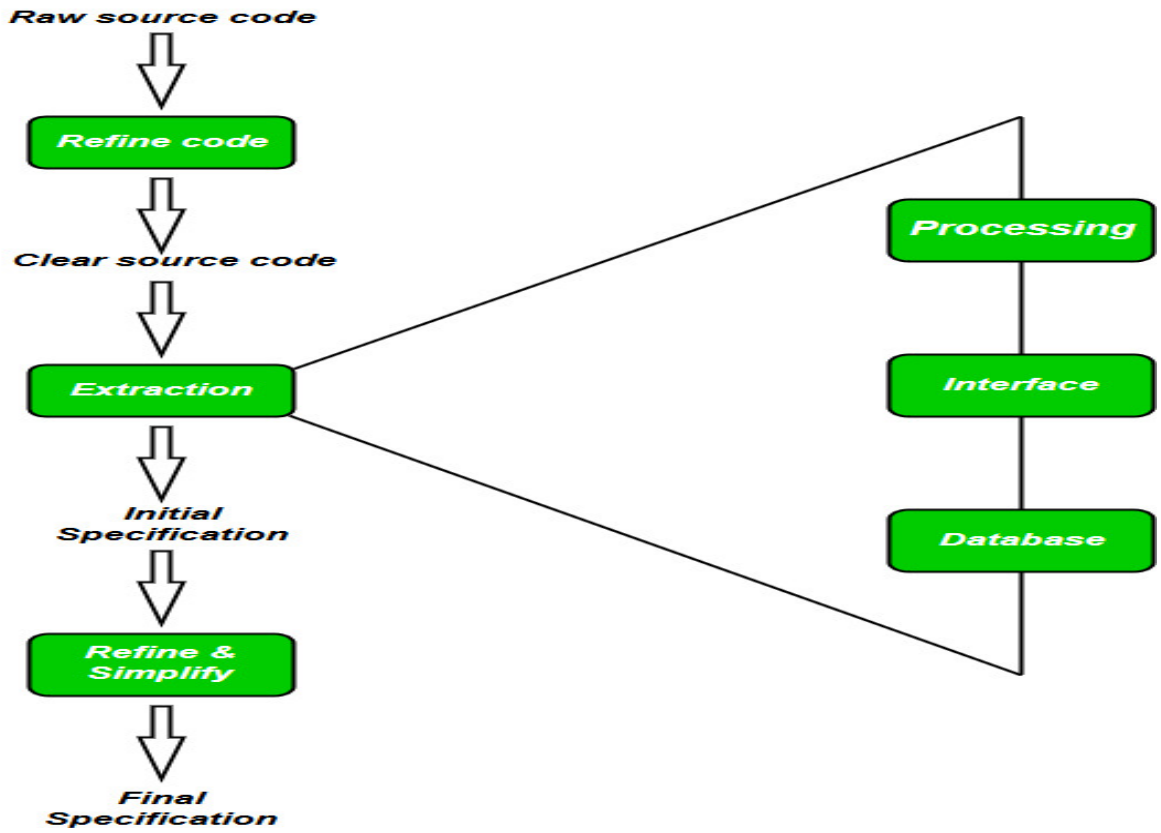
Software Reverse Engineering:

Software Reverse Engineering is a process of recovering the design, requirement specifications, and functions of a product from an analysis of its code. It builds a program database and generates information from this.

The purpose of reverse engineering is to facilitate the maintenance work by improving the understandability of a system and producing the necessary documents for a legacy system.

Reverse Engineering Goals:

- Cope with Complexity.
- Recover lost information.
- Detect side effects.
- Synthesise higher abstraction.
- Facilitate Reuse.



Steps of Software Reverse Engineering:

1. **Collection Information:**
   This step focuses on collecting all possible information (i.e., source design documents, etc.) about the software.

2. **Examining the information:**
   The information collected in step-1 is studied so as to get familiar with the system.

3. **Extracting the structure:**
   This step concerns identifying program structure in the form of a structure chart where each node corresponds to some routine.

4. **Recording the functionality:**
   During this step processing details of each module of the structure, charts are recorded using structured language like decision table, etc.

5. **Recording data flow:**
   From the information extracted in step-3 and step-4, a set of data flow diagrams is derived to show the flow of data among the processes.

6. **Recording control flow:**
   The high-level control structure of the software is recorded.

7. **Review extracted design:**
   The design document extracted is reviewed several times to ensure consistency and correctness. It also ensures that the design represents the program.

8. **Generate documentation:**
   Finally, in this step, the complete documentation including SRS, design document, history, overview, etc. is recorded for future use.

Reverse Engineering Tools:
Reverse engineering if done manually would consume a lot of time and human labor and hence must be supported by automated tools. Some of the tools are given below:

- **CIAO and CIA:** A graphical navigator for software and web repositories and a collection of Reverse Engineering tools.
- **Rigi:** A visual software understanding tool.
- **Bunch:** A software clustering/modularization tool.
- **GEN++:** An application generator to support the development of analysis tools for the C++ language.
- **PBS:** Software Bookshelf tools for extracting and visualizing the architecture of programs.

Software Re engineering:

Software Re-engineering is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form. This process encompasses a combination of sub-processes like reverse engineering, forward engineering, reconstructing etc.

Software re-engineering, also known as software restructuring or software renovation, refers to the process of improving or upgrading existing software systems to improve their quality, maintainability, or functionality. It involves reusing the existing software artifacts, such as code, design, and documentation, and transforming them to meet new or updated requirements.

The primary goal of software re-engineering is to improve the quality and maintainability of the software system, while minimizing the risks and costs associated with the redevelopment of the system from scratch.

The process of software re-engineering involves the following steps:

1. Planning: The first step is to plan the re-engineering process, which involves identifying the reasons for re-engineering, defining the scope, and establishing the goals and objectives of the process.
2. Analysis: The next step is to analyze the existing system, including the code, documentation, and other artifacts. This involves identifying the system's strengths and weaknesses, as well as any issues that need to be addressed.
3. Design: Based on the analysis, the next step is to design the new or updated software system. This involves identifying the changes that need to be made and developing a plan to implement them.
4. Implementation: The next step is to implement the changes by modifying the existing code, adding new features, and updating the documentation and other artifacts.
5. Testing: Once the changes have been implemented, the software system needs to be tested to ensure that it meets the new requirements and specifications.
6. Deployment: The final step is to deploy the re-engineered software system and make it available to end-users.

Software re-engineering can be initiated for various reasons, such as:

1. Improving software quality: Re-engineering can help improve the quality of software by eliminating defects, improving performance, and enhancing reliability and maintainability.
2. Updating technology: Re-engineering can help modernize the software system by updating the technology used to develop, test, and deploy the system.
3. Enhancing functionality: Re-engineering can help enhance the functionality of the software system by adding new features or improving existing ones.
4. Resolving issues: Re-engineering can help resolve issues related to scalability, security, or compatibility with other systems.

Advantages of software re-engineering include:

1. Improved system performance and reliability: Re-engineering can improve the performance and reliability of the software system, making it more efficient and less prone to errors or failures.

2. Reduced maintenance costs: Re-engineering can reduce the cost of maintaining and supporting the software system over time, by improving its maintainability and reducing the frequency of required maintenance activities.
3. Increased user satisfaction: Re-engineering can improve the usability and functionality of the software system, enhancing user satisfaction and increasing adoption rates.

Disadvantages of software re-engineering include:

1. Increased development time and cost: Re-engineering requires significant time and resources, which can increase development costs and delay time-to-market.
2. Risk of introducing new errors or defects: Re-engineering can introduce new errors or defects into the software system if not done carefully and thoroughly.
3. Overall, software re-engineering can be an effective way to improve the quality and functionality of an existing software system. However, it is important to carefully assess the risks and benefits of re-engineering and to plan and execute the re-engineering effort carefully to ensure a successful outcome.

Lecture takeaway:

1. **Define Testing**
2. **Explain all types of testing with examples.**
3. **Differentiate between Black and white boxes testing (diagram necessary).**
4. **Explain Software Reverse engineering.**
5. **Explain Re engineering.**
6. **Write a manual test case for an example scenario.**