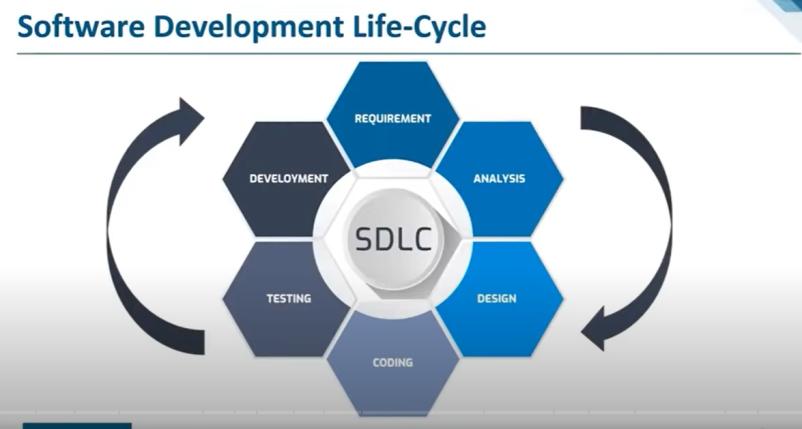




**Software Development Lifecycle**



The stages of SDLC are as follows:

Stage1: Planning and requirement analysis **(Requirement)**

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

For Example, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the **SRS (Software Requirement Specification)** , CRS document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

Stage2: Defining Requirements **(Analysis)**

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "**SRS"- Software Requirement Specification document** which contains all the product requirements to be constructed and developed during the project life cycle.

Project Manager and business analyst are involved.

Stage3: Designing the Software **(Design)**

**HLD and LLD**

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

High and low level document

Stage4: Developing the project **(Coding)**

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

Stage5: **Testing**

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.

During this stage, unit testing, integration testing, system testing, acceptance testing are done.

After QA is checked it goes to implementation

Stage6: **Deployment**

Once the software is certified, and no bugs or errors are stated, then it is deployed.

Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

Stage6: Maintenance

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

This procedure where the care is taken for the developed product is known as maintenance.

**SDLC Models**

There are various software development life cycle models defined and designed which are followed during the software development process. These models are also referred as Software Development Process Models". Each process model follows a Series of steps unique to its type to ensure success in the process of software development.

Following are the most important and popular SDLC models followed in the industry −

Waterfall Model

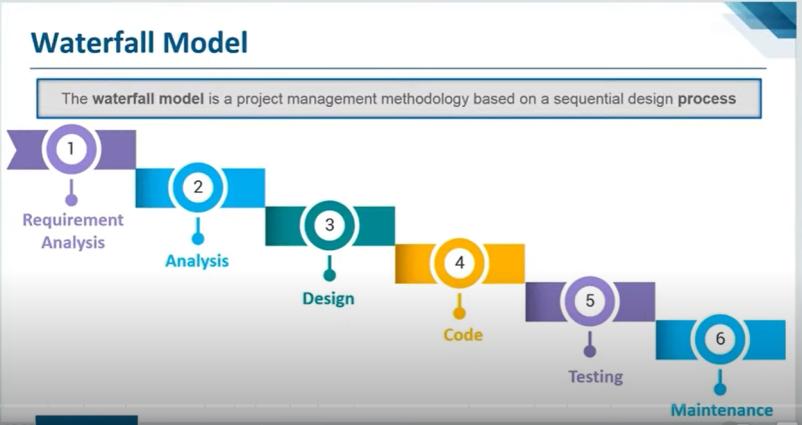
Iterative Model

Spiral Model

V-Model

Big Bang Model

Waterfall Model



The classical waterfall model divides the life cycle into a set of phases. This model considers that one phase can be started after the completion of the previous phase. That is the output of one phase will be the input to the next phase. Thus the development process can be considered as a sequential flow in the waterfall. Here the phases do not overlap with each other.

Advantages of Classical Waterfall Model

The classical waterfall model is an idealistic model for software development. It is very simple, so it can be considered the basis for other software development life cycle models. Below are some of the major advantages of this SDLC model:

This model is very simple and is easy to understand.

Phases in this model are processed one at a time.

Each stage in the model is clearly defined.

This model has very clear and well-understood milestones.

Process, actions and results are very well documented.

Reinforces good habits: define-before- design,

design-before-code.

This model works well for smaller projects and projects where requirements are well

understood.

Drawbacks of Classical Waterfall Model

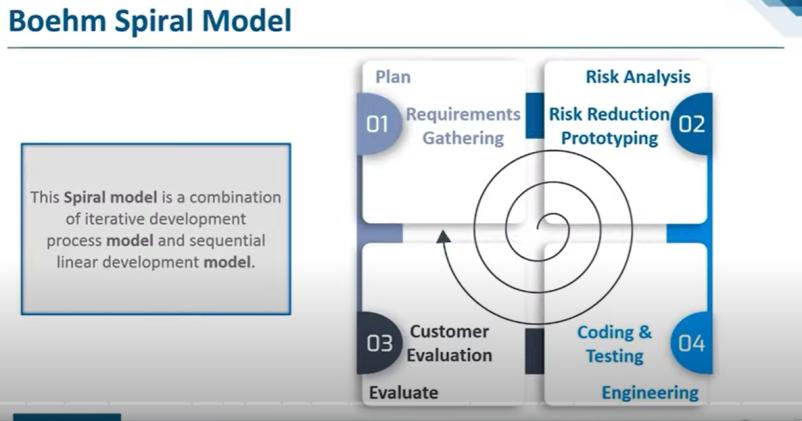
The classical waterfall model suffers from various shortcomings, basically, we can’t use it in real projects, but we use other software development lifecycle models which are based on the classical waterfall model. Below are some major drawbacks of this model:

No feedback path: In the classical waterfall model evolution of software from one phase to another phase is like a waterfall. It assumes that no error is ever committed by developers during any phase. Therefore, it does not incorporate any mechanism for error correction.

Difficult to accommodate change requests: This model assumes that all the customer requirements can be completely and correctly defined at the beginning of the project, but actually customers’ requirements keep on changing with time. It is difficult to accommodate any change requests after the requirements specification phase is complete.

No overlapping of phases: This model recommends that a new phase can start only after the completion of the previous phase. But in real projects, this can’t be maintained. To increase efficiency and reduce cost, phases may overlap.

2. Boehm Spiral Model



Spiral model is one of the most important Software Development Life Cycle models, which provides support for Risk Handling. In its diagrammatic representation, it looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a Phase of the software development process. The exact number of phases needed to develop the product can be varied by the project manager depending upon the project risks. As the project manager dynamically determines the number of phases, so the project manager has an important role to develop a product using the spiral model.

The Radius of the spiral at any point represents the expenses(cost) of the project so far, and the angular dimension represents the progress made so far in the current phase.

**Phases of Spiral Model**

Each phase of the Spiral Model is divided into four quadrants as shown in the above figure. The functions of these four quadrants are discussed below-

**Objectives determination and identify alternative solutions:** Requirements are gathered from the customers and the objectives are identified, elaborated, and analyzed at the start of every phase. Then alternative solutions possible for the phase are proposed in this quadrant.

**Identify and resolve Risks**: During the second quadrant, all the possible solutions are evaluated to select the best possible solution. Then the risks associated with that solution are identified and the risks are resolved using the best possible strategy. At the end of this quadrant, the Prototype is built for the best possible solution.

**Develop next version of the Product**: During the third quadrant, the identified features are developed and verified through testing. At the end of the third quadrant, the next version of the software is available.

**Review and plan for the next Phase**: In the fourth quadrant, the Customers evaluate the so far developed version of the software. In the end, planning for the next phase is started.

**Advantages of Spiral Model:**

Risk Handling: The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.

Good for large projects: It is recommended to use the Spiral Model in large and complex projects.

Software testing is the process of executing a program with the aim of finding the error. To make our software perform well it should be error-free. If testing is done successfully it will remove all the errors from the software.

**There are seven principles in software testing:**

* Testing shows the presence of defects
* Exhaustive testing is not possible
* Early testing
* Defect clustering
* Pesticide paradox
* Testing is context-dependent
* Absence of errors fallacy

**Testing shows the presence of defects**: The goal of software testing is to make the software fail. Software testing reduces the presence of defects. Software testing talks about the presence of defects and doesn’t talk about the absence of defects. Software testing can ensure that defects are present but it can not prove that software is defect-free. Even multiple testing can never ensure that software is 100% bug-free. Testing can reduce the number of defects but not remove all defects.

**Exhaustive testing is not possible**: It is the process of testing the functionality of the software in all possible inputs (valid or invalid) and pre-conditions is known as exhaustive testing. Exhaustive testing is impossible means the software can never test at every test case. It can test only some test cases and assume that the software is correct and it will produce the correct output in every test case. If the software will test every test case then it will take more cost, effort, etc., which is impractical.

**Early Testing**: To find the defect in the software, early test activity shall be started. The defect detected in the early phases of SDLC will be very less expensive. For better performance of software, software testing will start at the initial phase i.e. testing will perform at the requirement analysis phase.

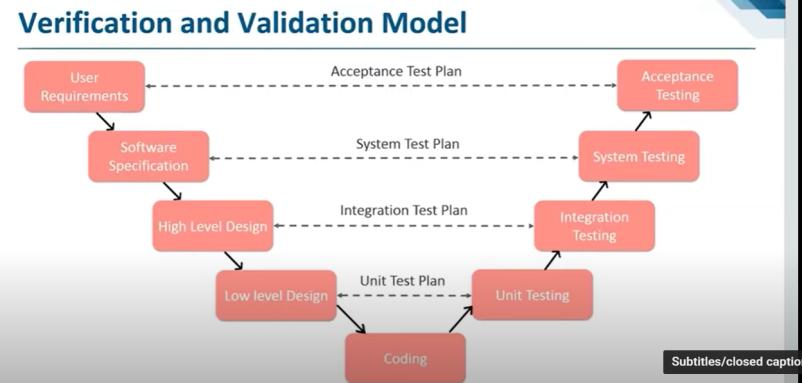
**Defect clustering**: In a project, a small number of modules can contain most of the defects. Pareto Principle to software testing state that 80% of software defect comes from 20% of modules.

**Pesticide paradox:** Repeating the same test cases, again and again, will not find new bugs. So it is necessary to review the test cases and add or update test cases to find new bugs.

**Testing is context-dependent**: The testing approach depends on the context of the software developed. Different types of software need to perform different types of testing. For example, The testing of the e-commerce site is different from the testing of the Android application.

**Absence of errors fallacy**: If a built software is 99% bug-free but it does not follow the user requirement then it is unusable. It is not only necessary that software is 99% bug-free but it is also mandatory to fulfill all the customer requirements.

**Verification and Validation Model**



Validation and verification are the two steps in any simulation project to validate a model.

* **Validation** is the process of comparing two results. In this process, we need to compare the representation of a conceptual model to the real system. If the comparison is true, then it is valid, else invalid.
* **Verification** is the process of comparing two or more results to ensure its accuracy. In this process, we have to compare the model’s implementation and its associated data with the developer's conceptual description and specifications.

**Verification:**  
Verification is the process of checking that a software achieves its goal without any bugs. It is the process to ensure whether the product that is developed is right or not. It verifies whether the developed product fulfills the requirements that we have.  
Verification is **Static Testing**.

Activities involved in verification:

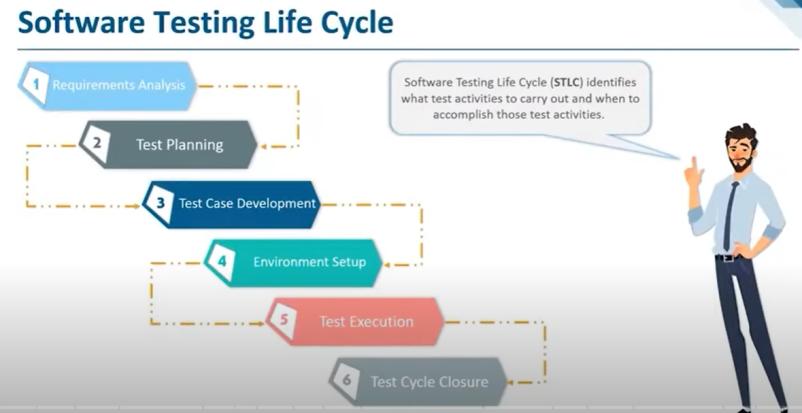
1. Inspections
2. Reviews
3. Walkthroughs
4. Desk-checking

**Validation:**  
Validation is the process of checking whether the software product is up to the mark or in other words product has high level requirements. It is the process of checking the validation of product i.e. it checks what we are developing is the right product. it is validation of actual and expected product.  
Validation is the **Dynamic Testing**.

Activities involved in validation:

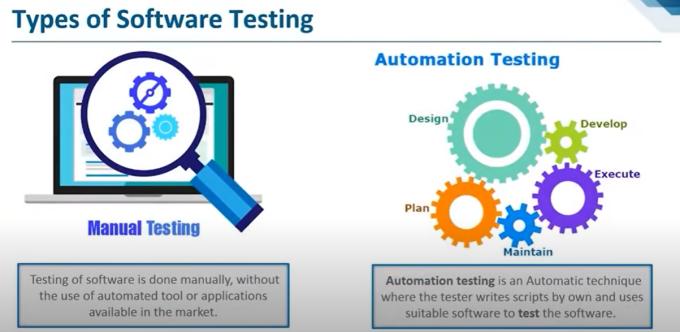
1. Black box testing
2. White box testing
3. Unit testing
4. Integration testing

**Software testing lifecycle**

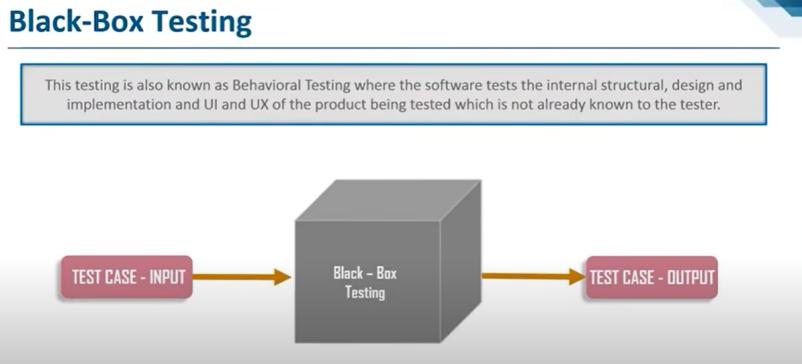


1. **Requirement Analysis:**  
   Requirement Analysis is the first step of Software Testing Life Cycle (STLC). In this phase quality assurance team understands the requirements like what is to be tested. If anything is missing or not understandable then quality assurance team meets with the stakeholders to better understand the detail knowledge of requirement.
2. **Test Planning:**  
   Test Planning is most efficient phase of software testing life cycle where all testing plans are defined. In this phase manager of the testing team calculates estimated effort and cost for the testing work. This phase gets started once the requirement gathering phase is completed.
3. **Test Case Development:**  
   The test case development phase gets started once the test planning phase is completed. In this phase testing team note down the detailed test cases. Testing team also prepare the required test data for the testing. When the test cases are prepared then they are reviewed by quality assurance team.
4. **Test Environment Setup:**  
   Test environment setup is the vital part of the STLC. Basically test environment decides the conditions on which software is tested. This is independent activity and can be started along with test case development. In this process the testing team is not involved. either the developer or the customer creates the testing environment.
5. **Test Execution:**  
   After the test case development and test environment setup test execution phase gets started. In this phase testing team start executing test cases based on prepared test cases in the earlier step.
6. **Test Closure:**  
   This is the last stage of STLC in which the process of testing is analyzed.

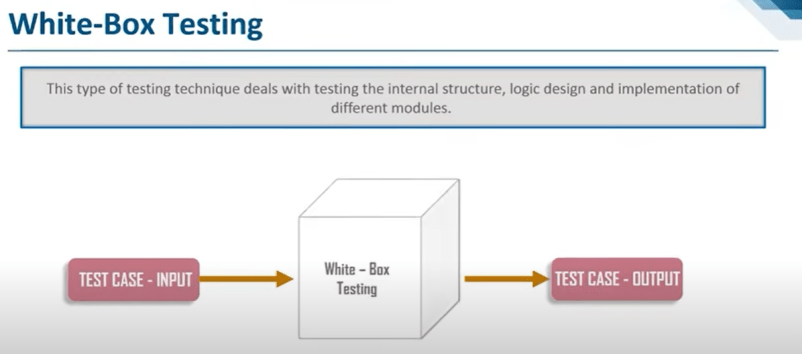
**Types of software testing**



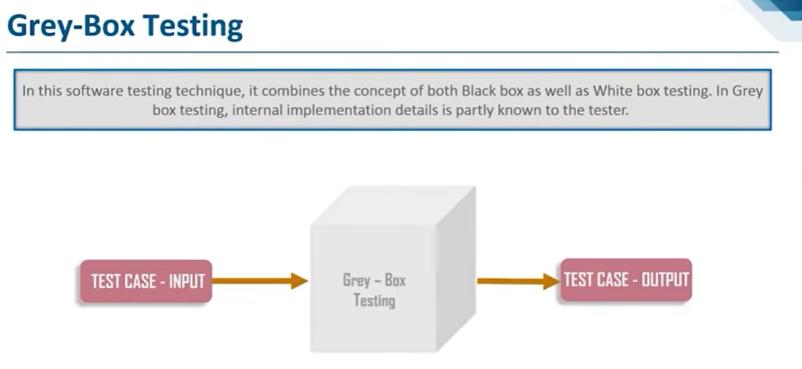
Black box testing:



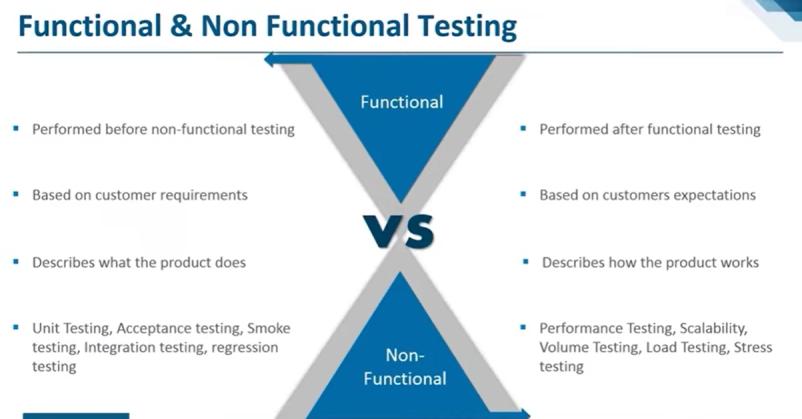
White box testing :



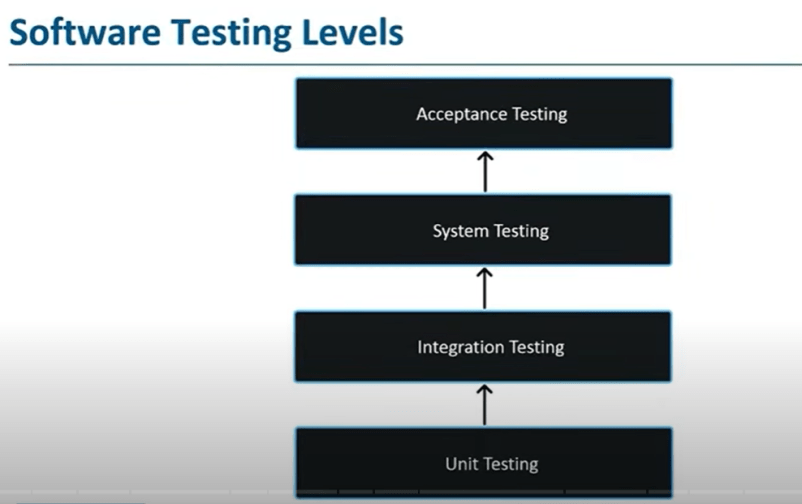
Grey box testing :



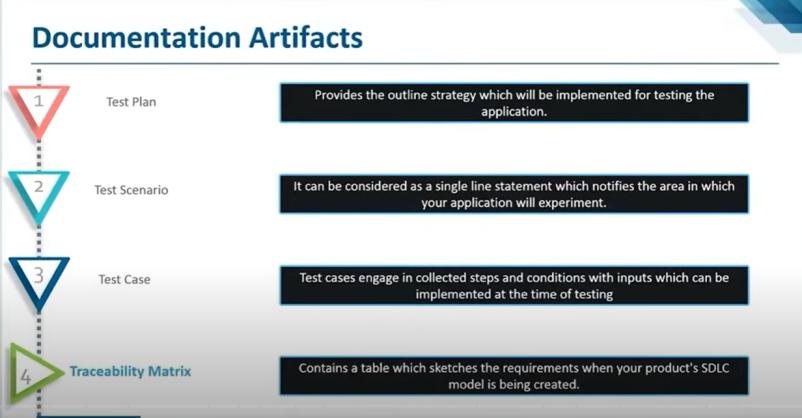
Functional VS Non–Functional Testing

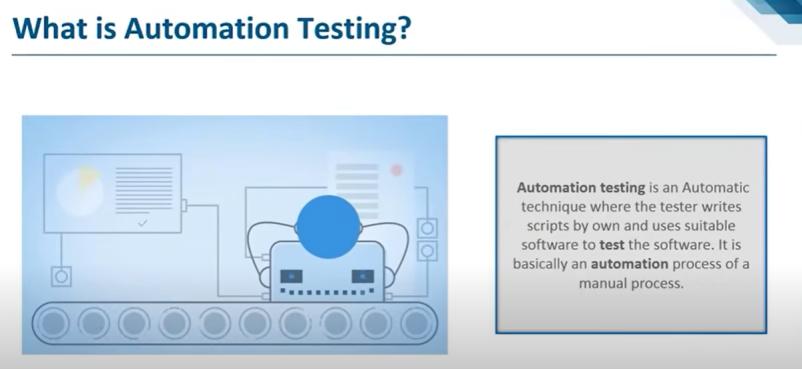


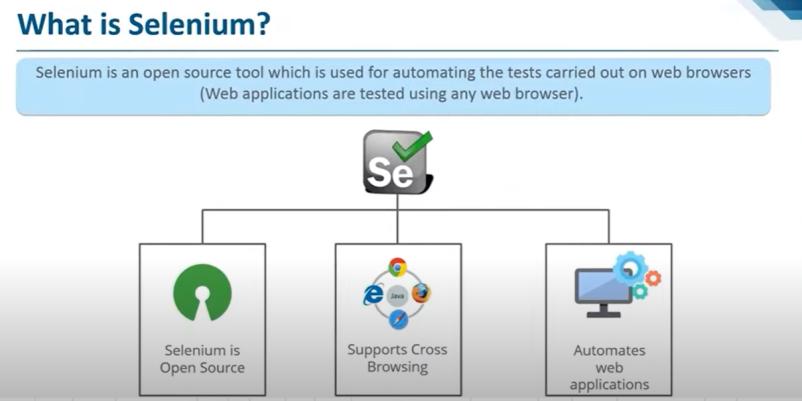
Software testing levels



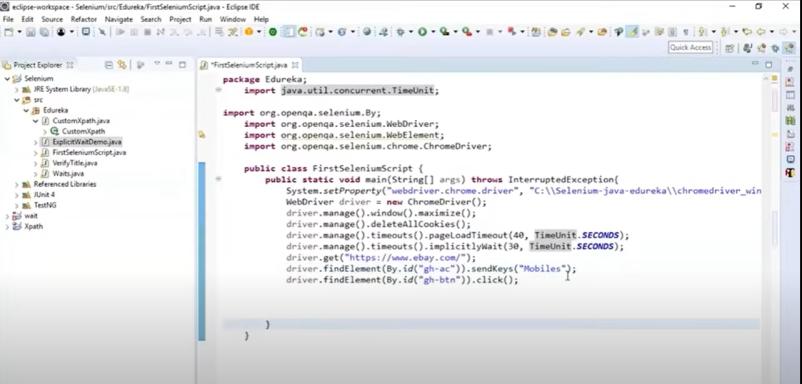
Software Testing Documentation

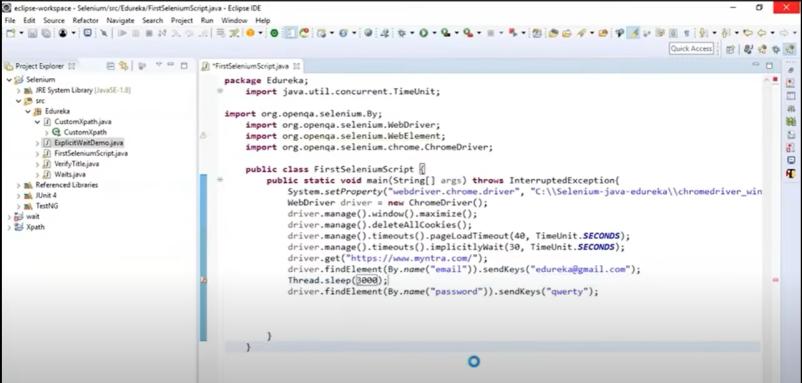


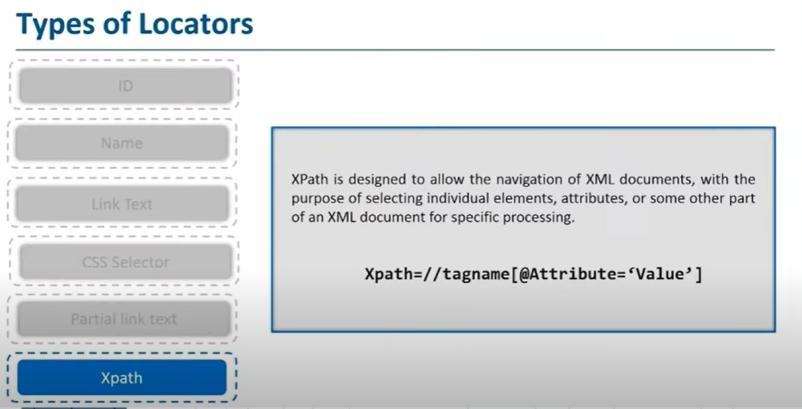




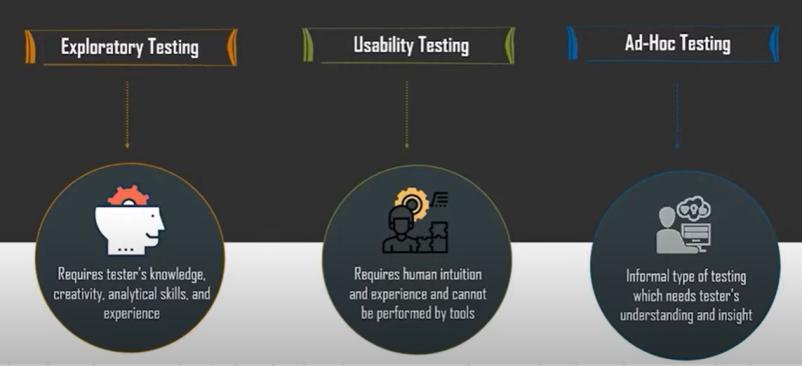




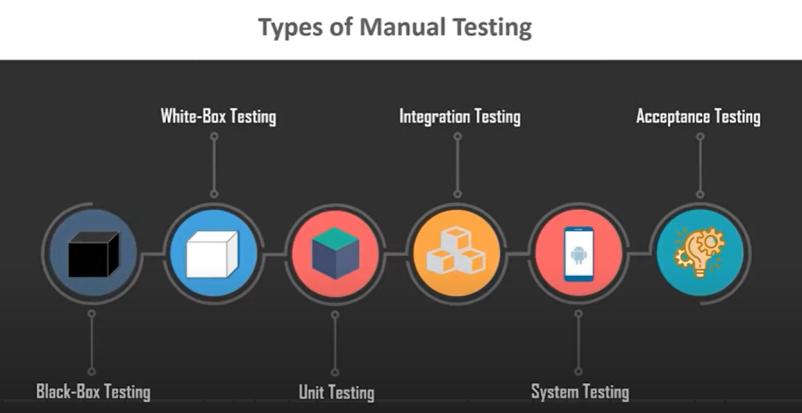


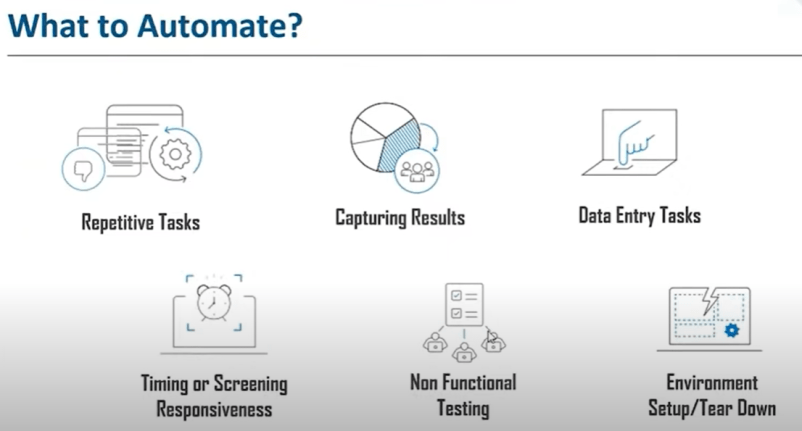


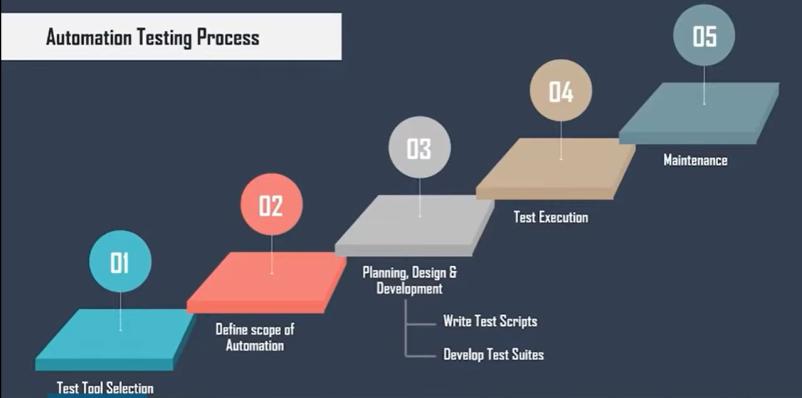
Manual testing

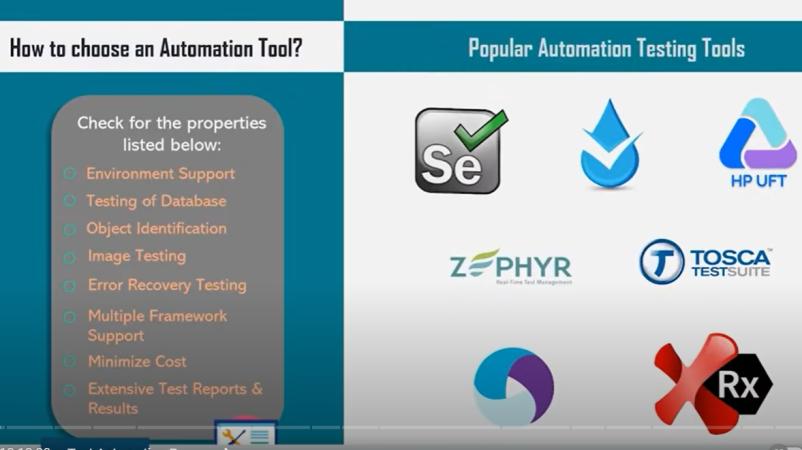


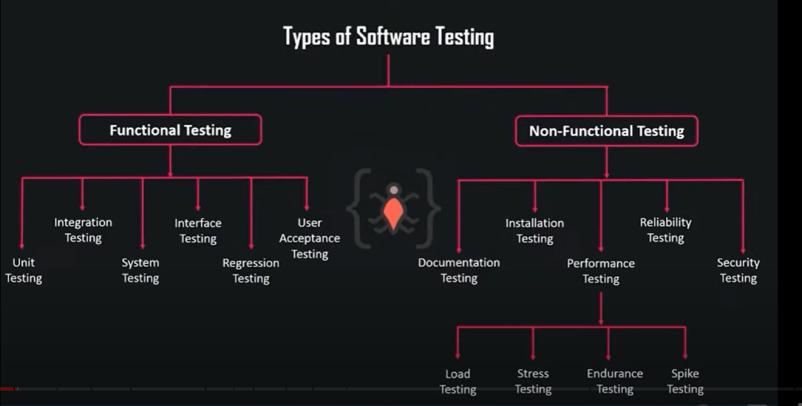


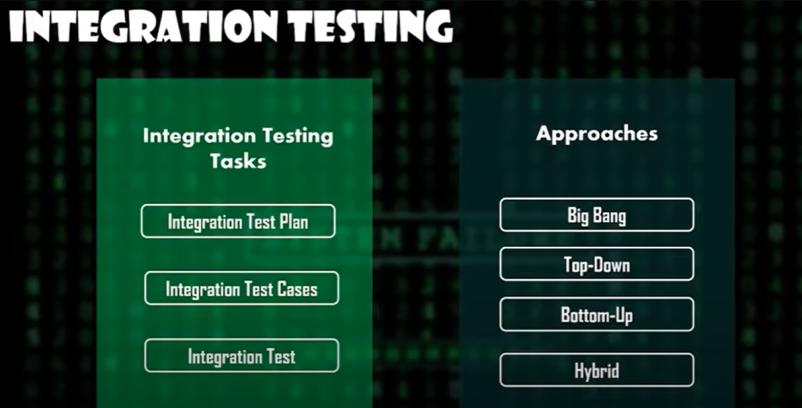








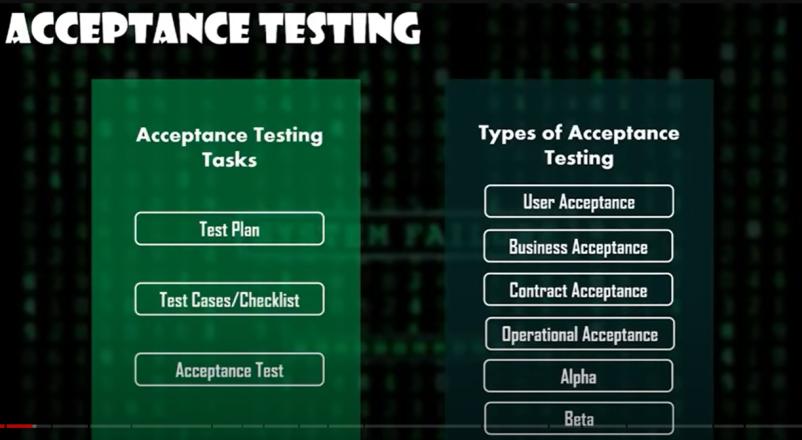




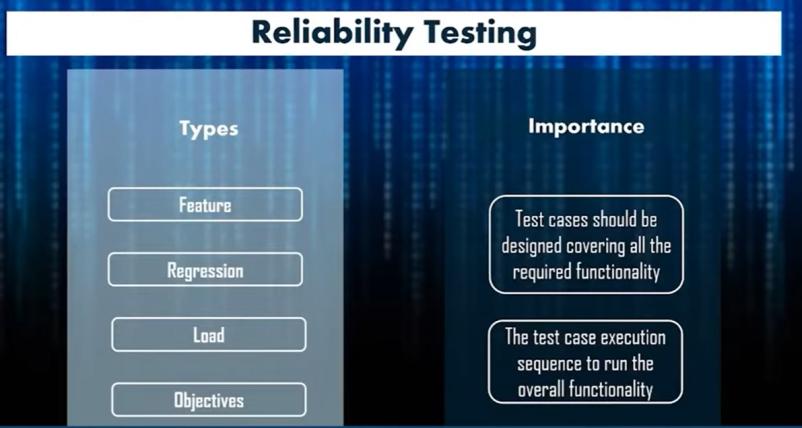


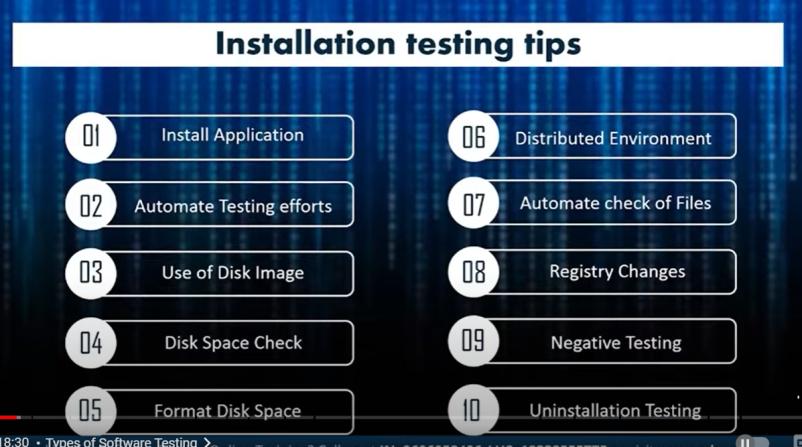


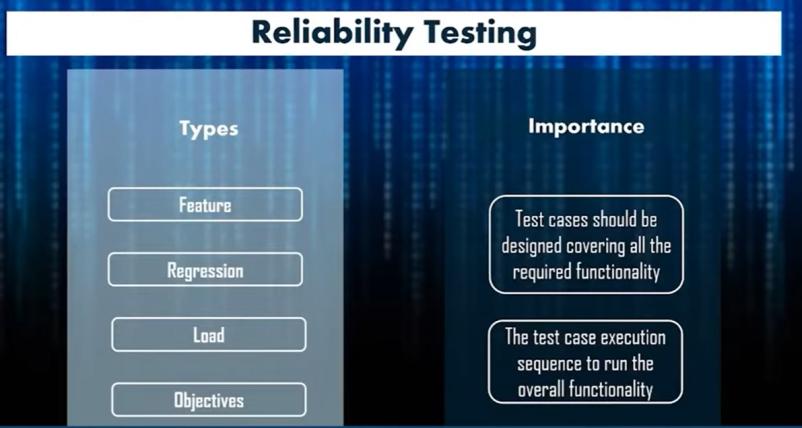


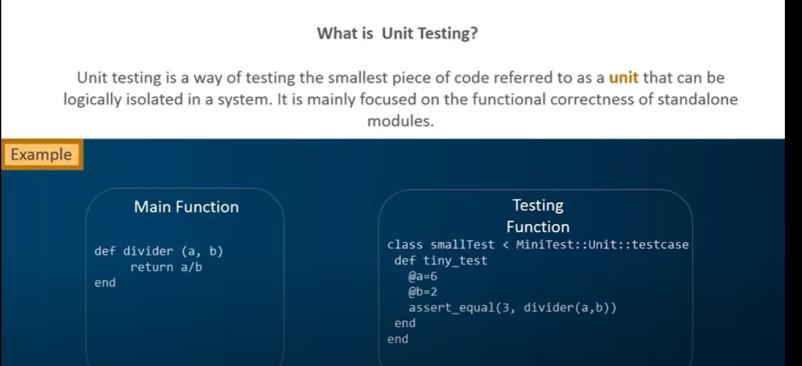






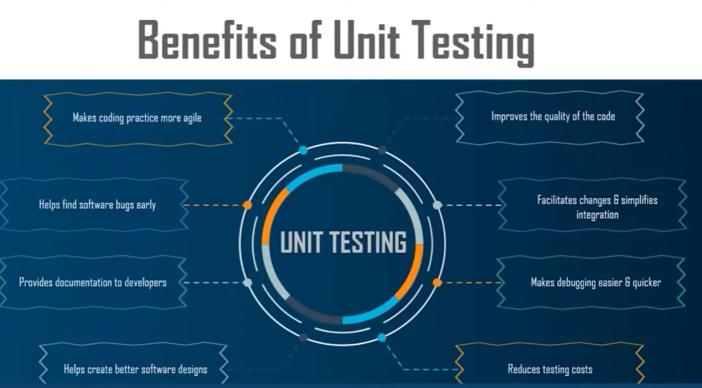


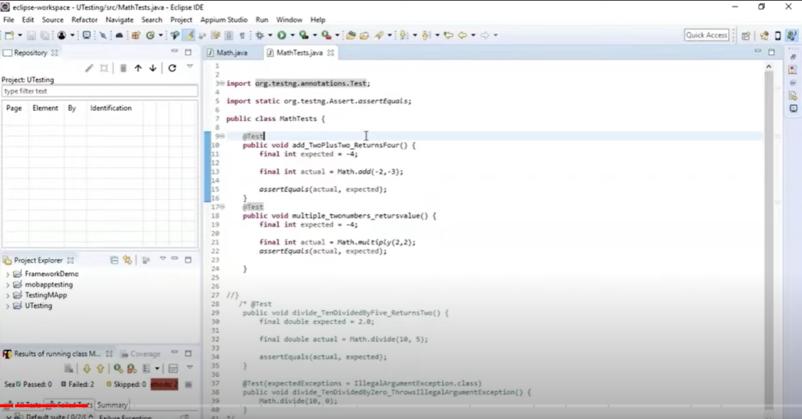


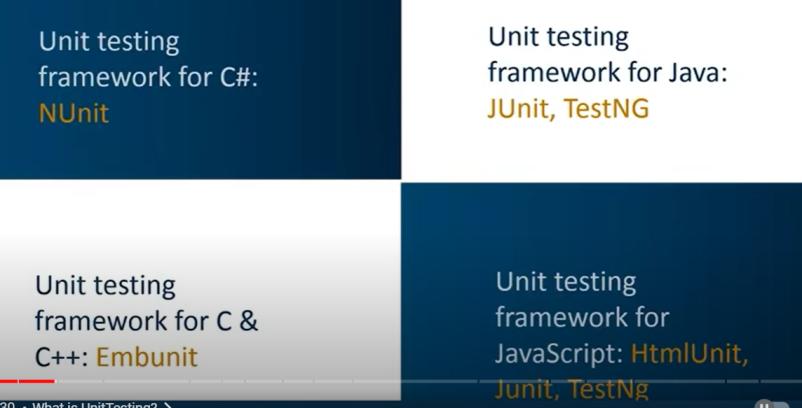




Unit Testing :







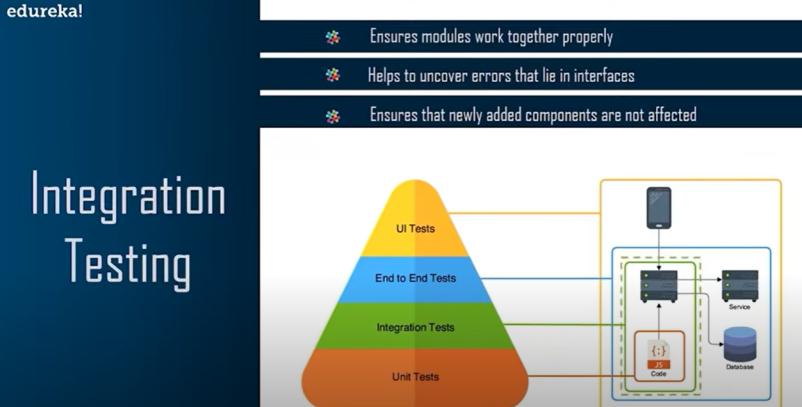


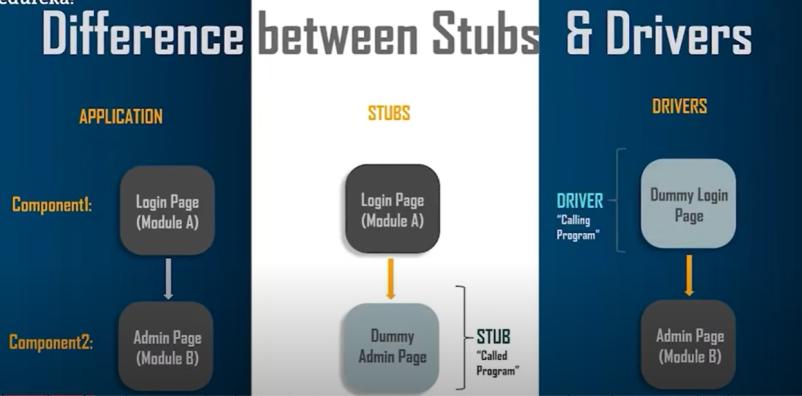
**INTEGRATION TESTING** is a level of software testing where individual units / components are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and test stubs are used to assist in Integration Testing.

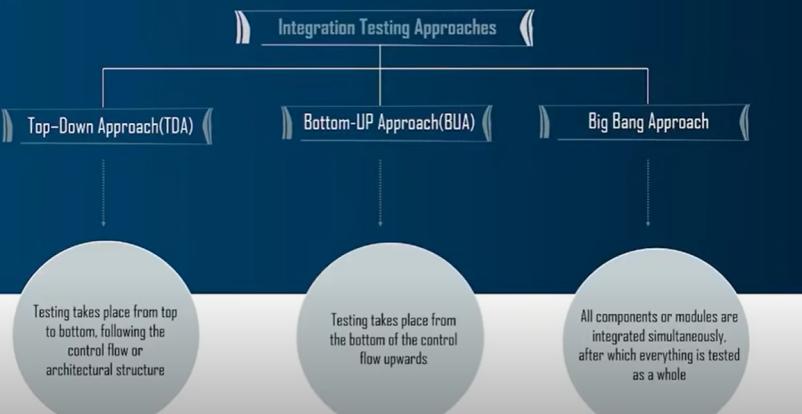
**Integration testing**: Testing performed to expose defects in the interfaces and in the interactions between integrated components or systems. See also component integration testing, system integration testing.

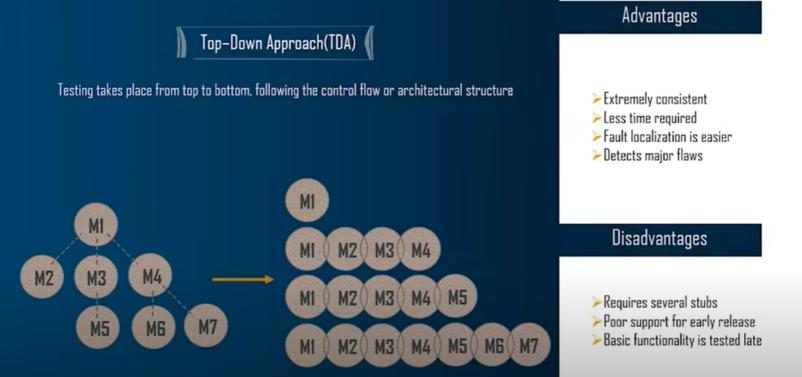
**Component integration testing:** Testing performed to expose defects in the interfaces and interaction between integrated components.

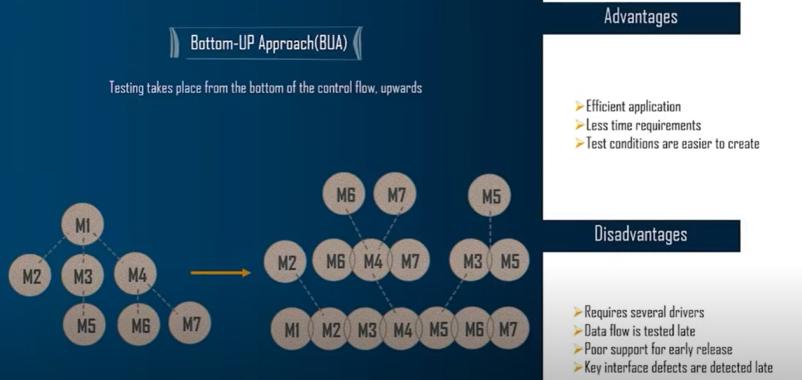
**System integration testing**: Testing the integration of systems and packages; testing interfaces to external organizations

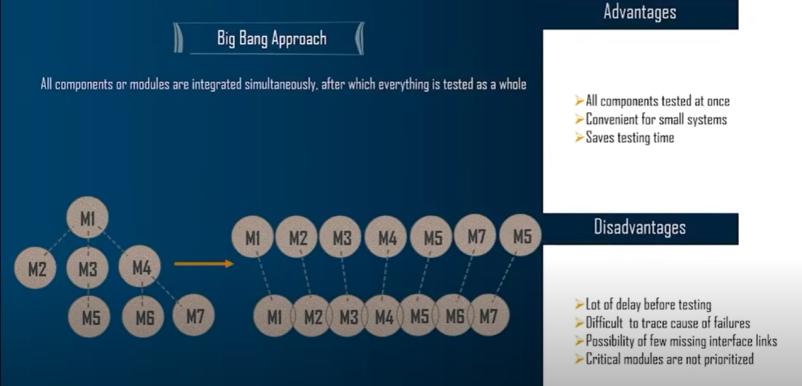


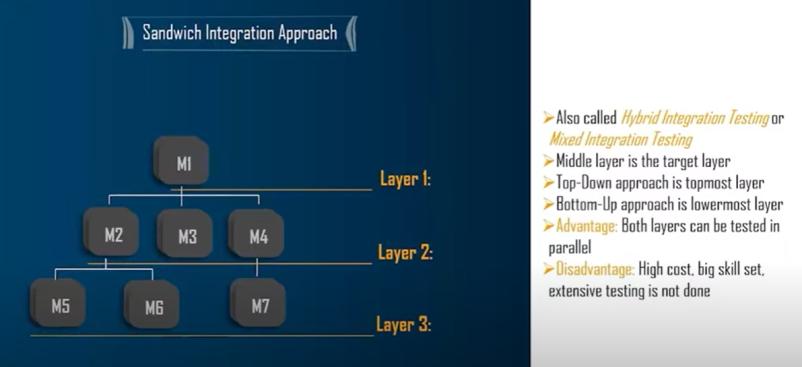


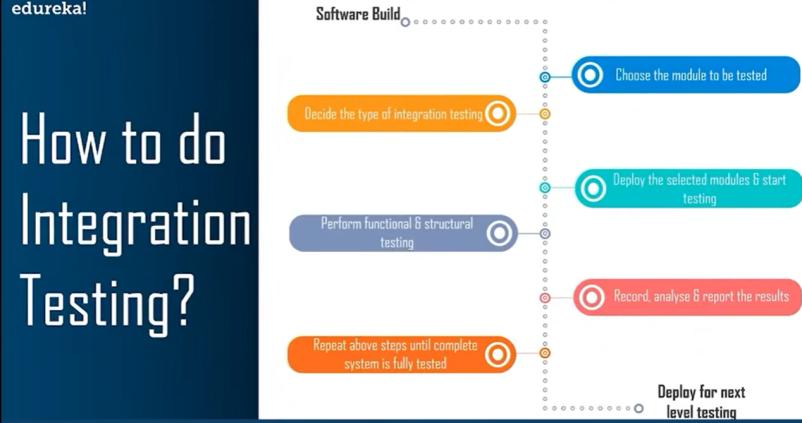




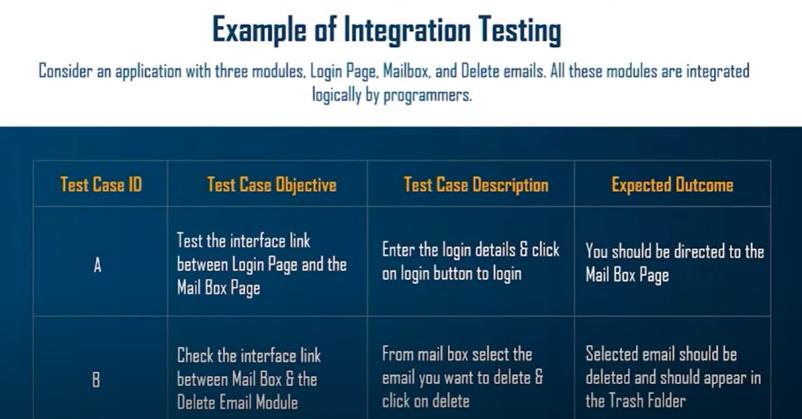


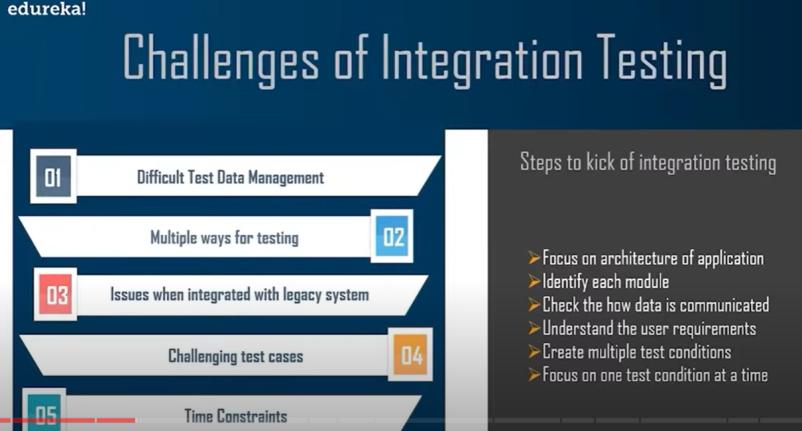






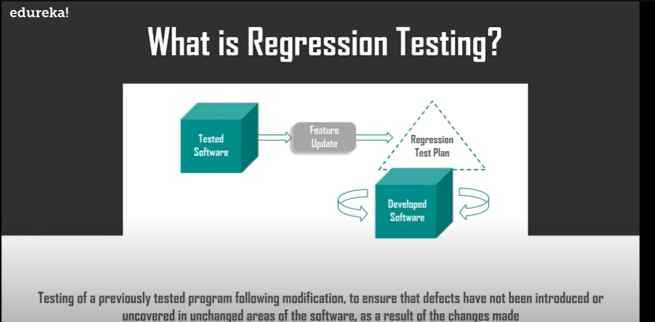




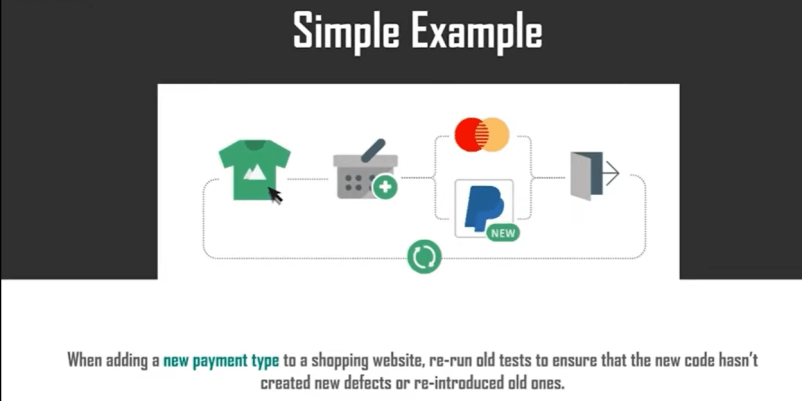


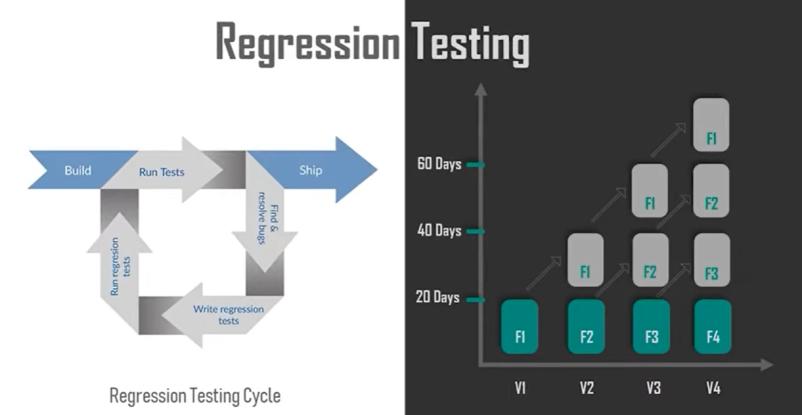
Regression Testing:



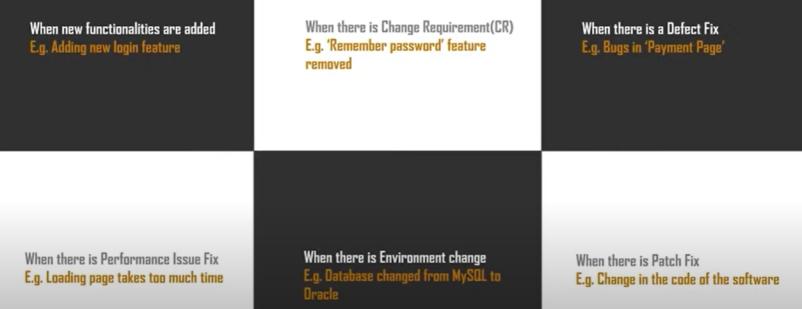


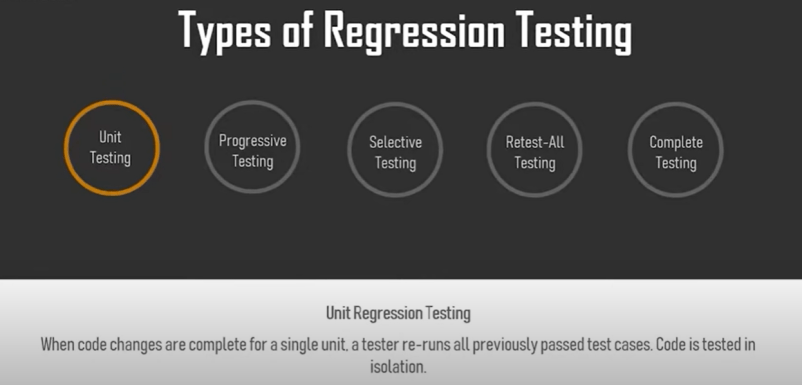












1. **Corrective** Regression Testing

2. **Retest-all** Regression Testing

3. **Selective** Regression Testing

4. **Progressive** Regression Testing

5. **Complete** Regression Testing

6. **Partial** Regression Testing

7. **Unit** Regression Testing

1. Re-test All:

Re-Test is one of the approaches to do regression testing. In this approach, all the test case suits should be re-executed. Here we can define re-test as when a test fails, and we determine the cause of the failure is a software fault. The fault is reported, we can expect a new version of the software in which defect fixed. In this case, we will need to execute the test again to confirm that the fault fixed. This is known as re-testing. Some will refer to this as confirmation testing.

The re-test is very expensive, as it requires enormous time and resources.

2. Regression test Selection:

In this technique, a selected test-case suit will execute rather than an entire test-case suit.

The selected test case suits divided in two cases

Reusable Test cases.

Obsolete Test cases.

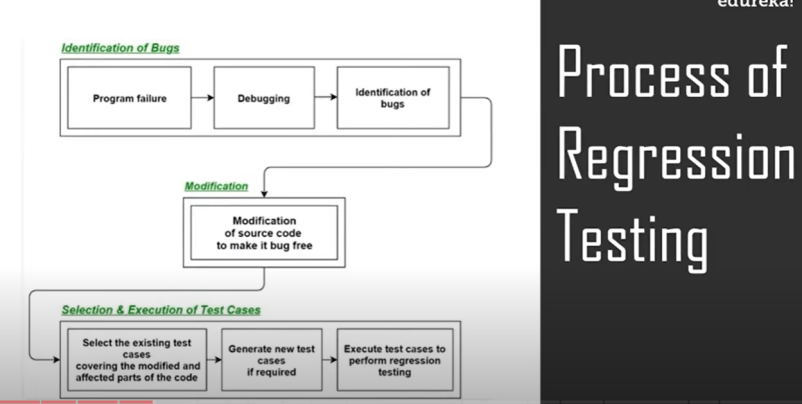
Reusable test cases can use in succeeding regression cycle.

Obsolete test cases can't use in succeeding regression cycle.

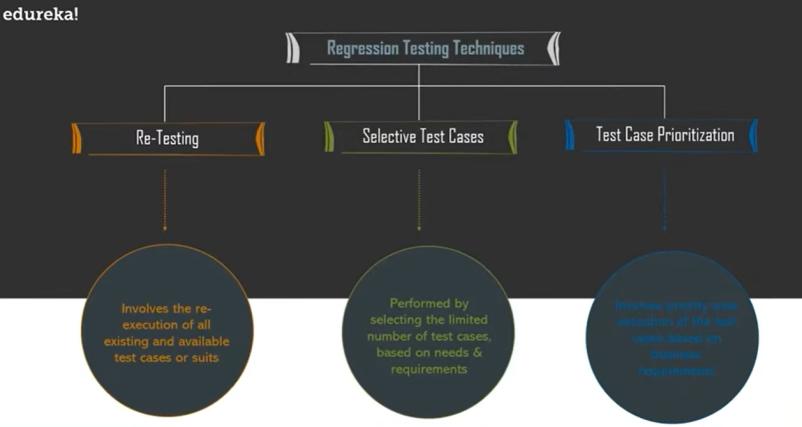
3. Prioritization of test cases:

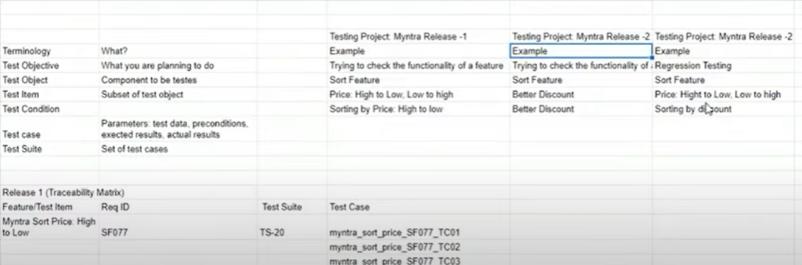
Prioritize the test case depending on business impact, critical and frequently functionality used. Selection of test cases will reduce the regression test suite.

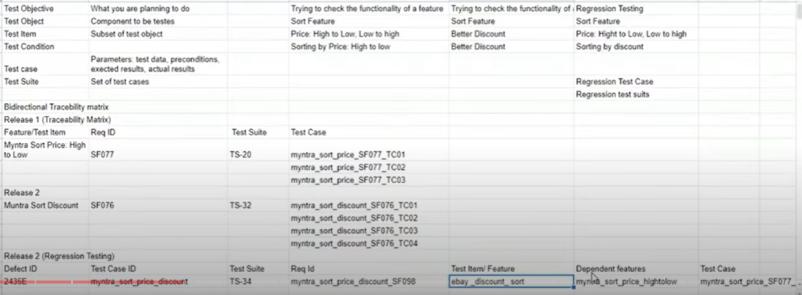




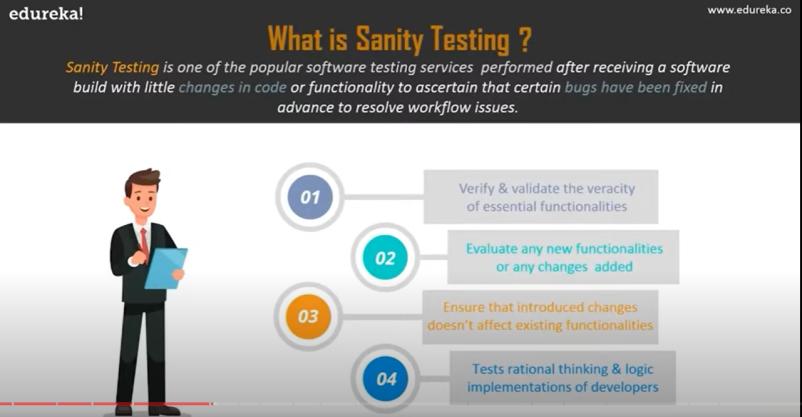


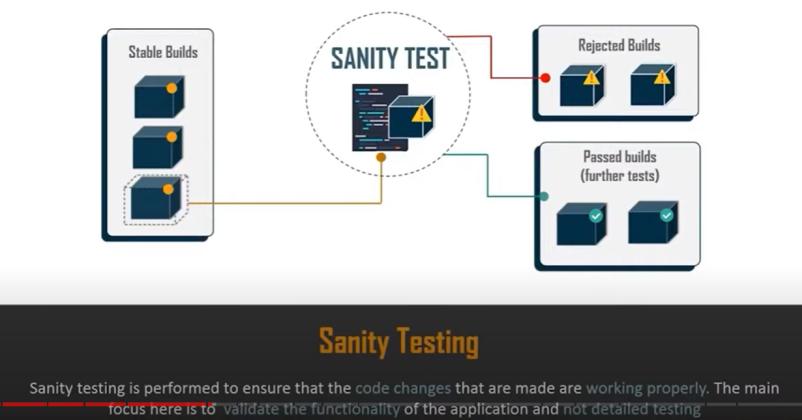


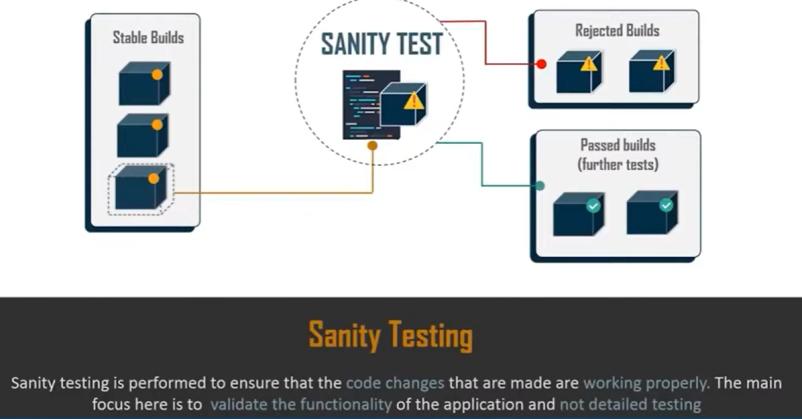


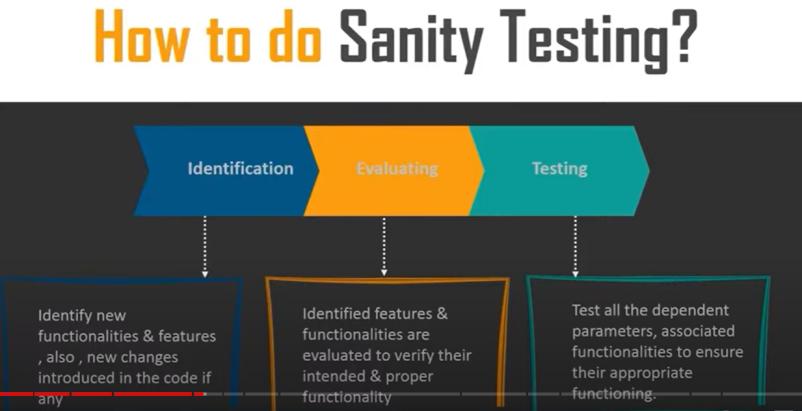






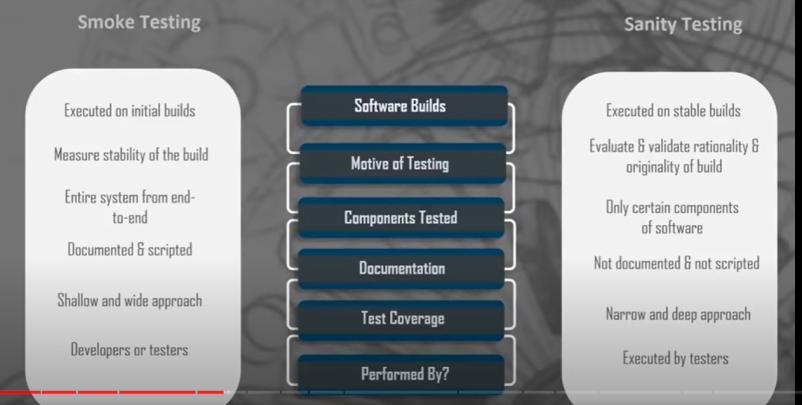




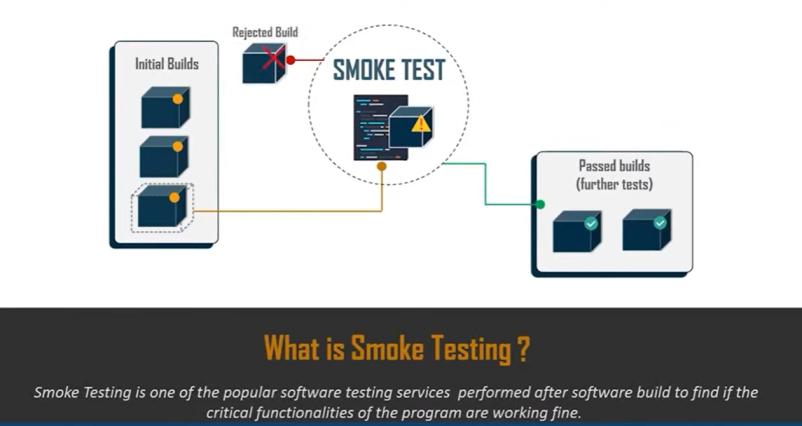










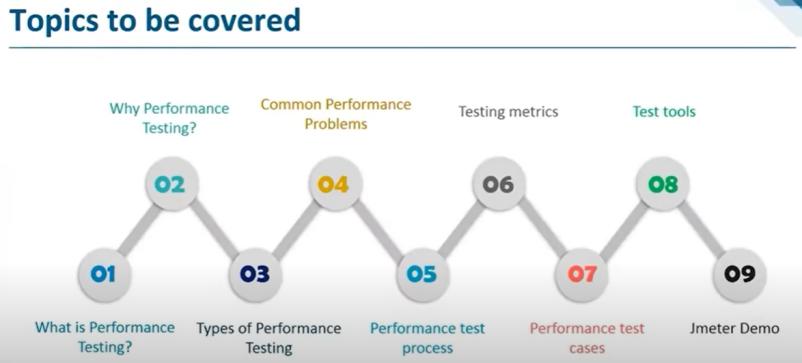




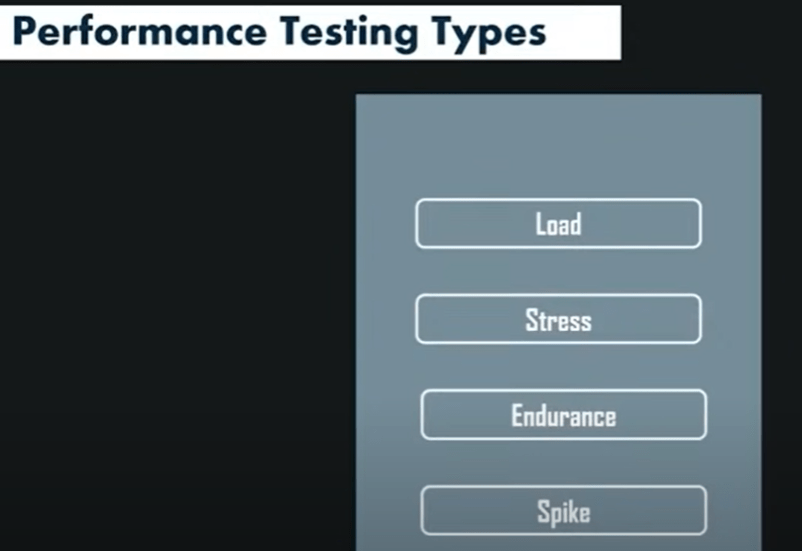


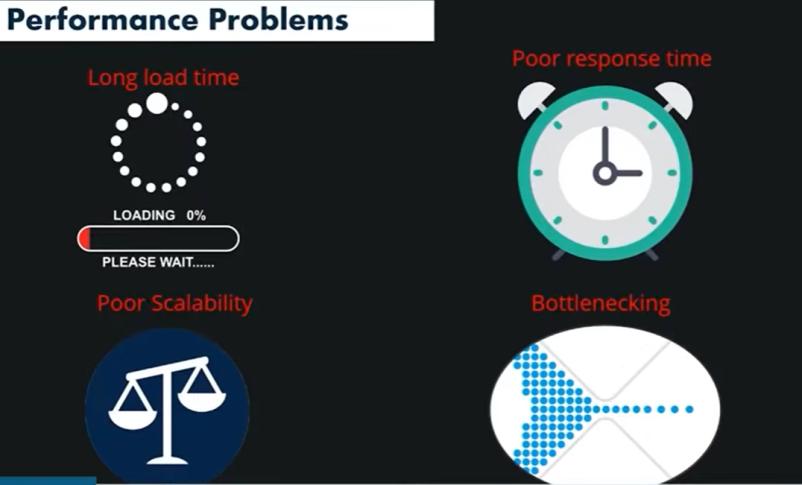


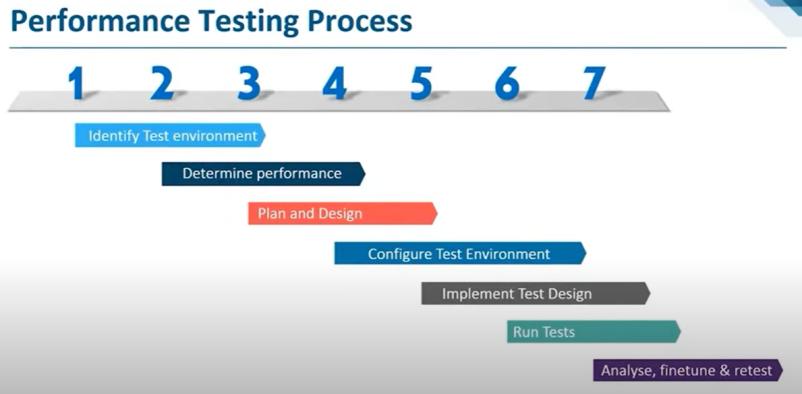
Non- Functional Testing:

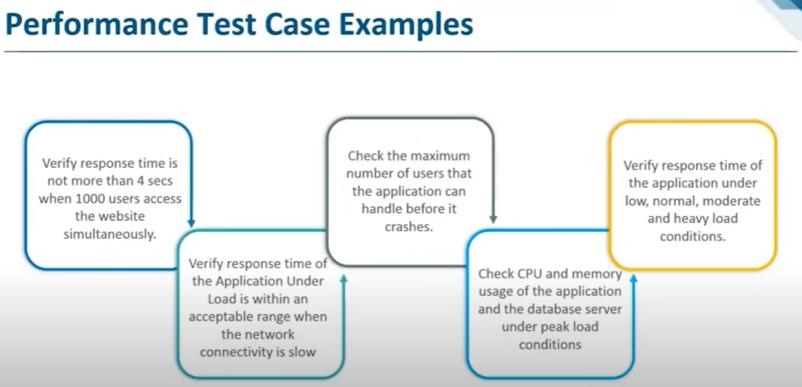






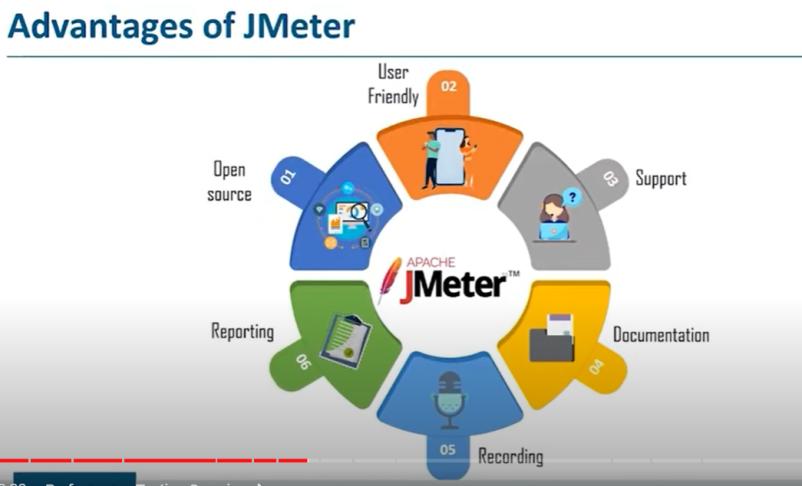


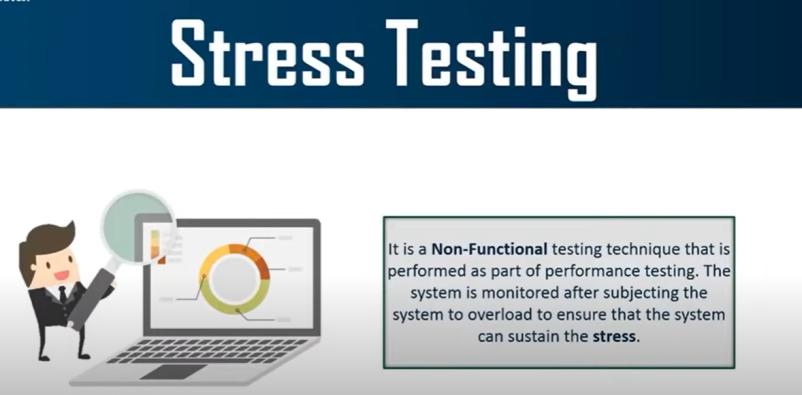










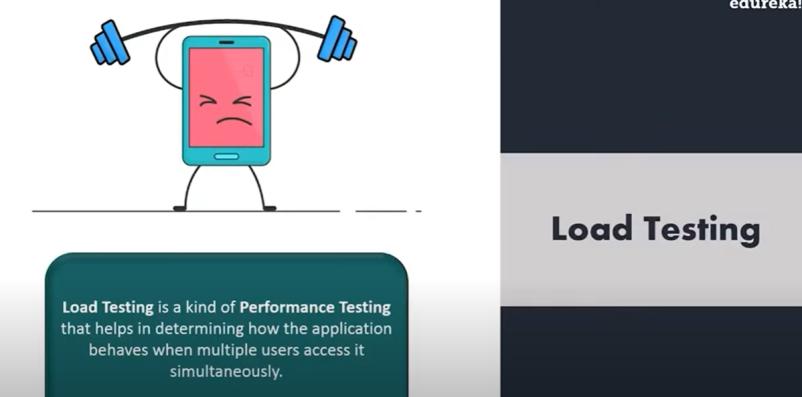




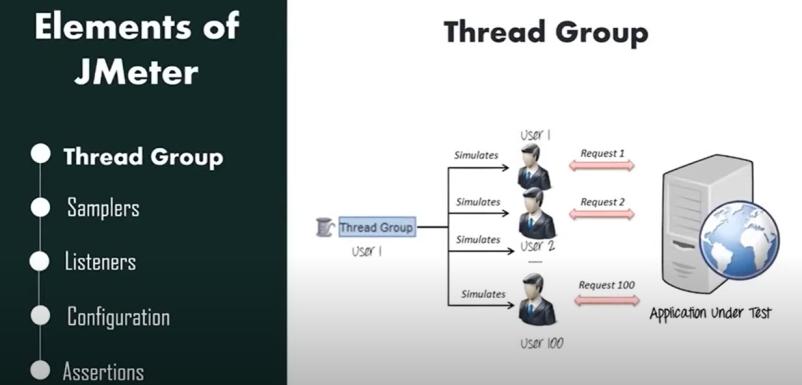


Apache jmeter used for stress, performance testing,load testing,neo load(show appropriate with heavy load).

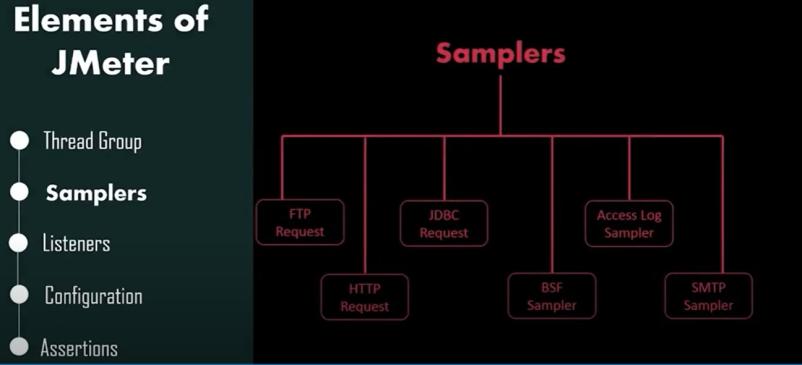


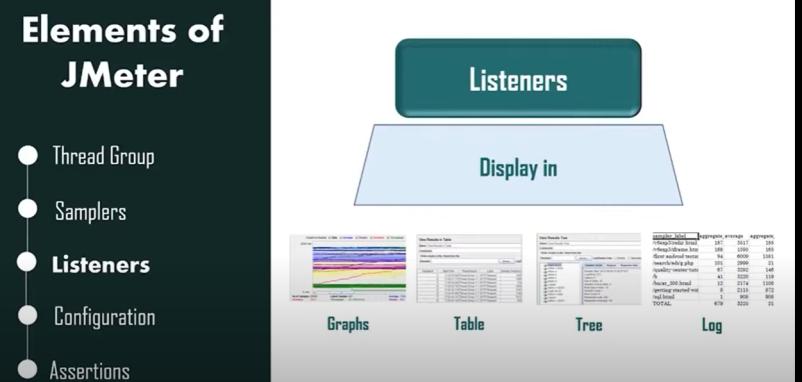


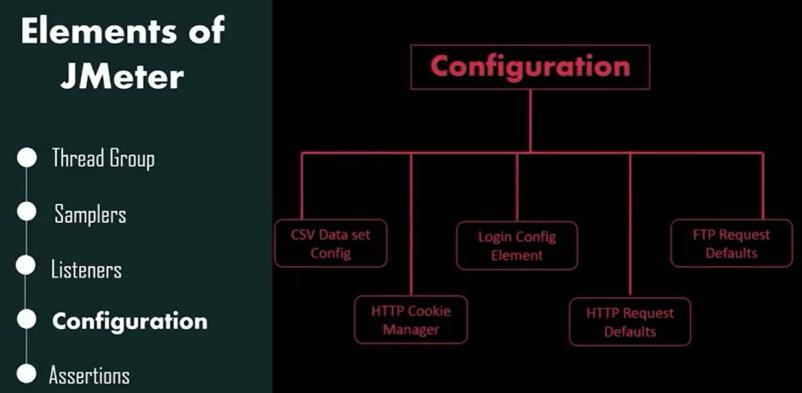


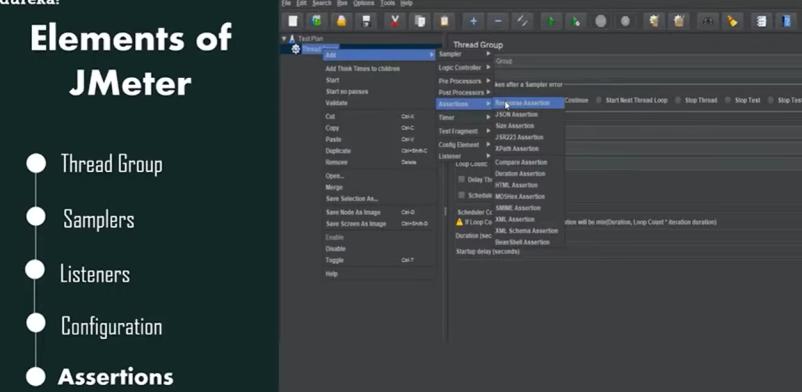


Response arresertion:









Verify actual and expected results