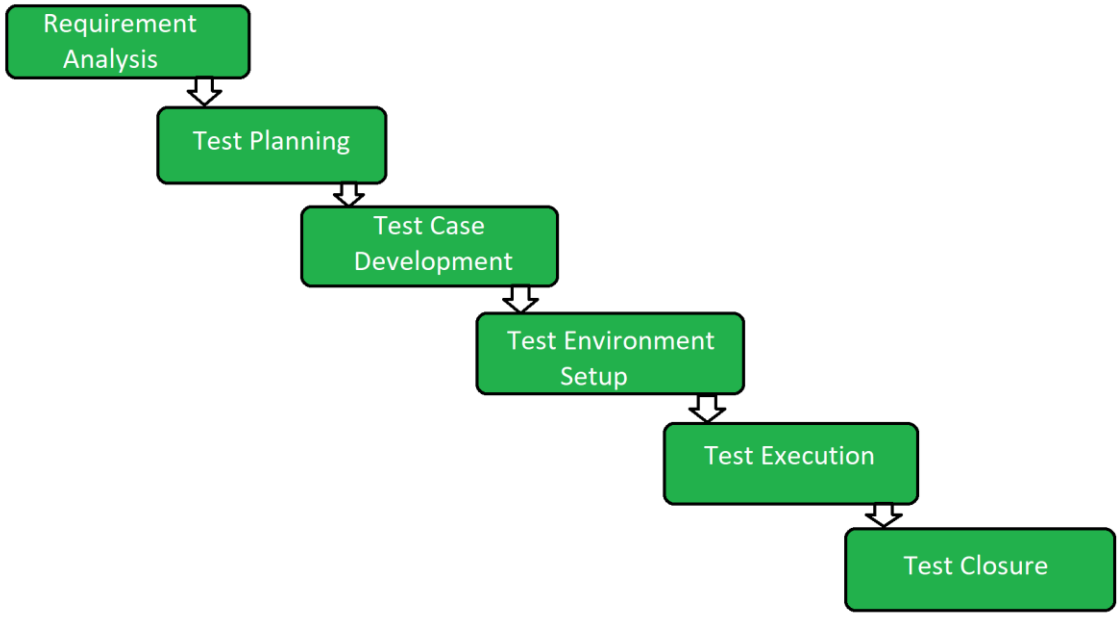
INDEX

Software Testing Lifecycle



1. Requirement Analysis

Requirement Analysis is the first step of the 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 the quality assurance team meets with the stakeholders to better understand the detailed knowledge of requirements.

The activities that take place during the Requirement Analysis stage include:

- Reviewing the software requirements document (SRD) and other related documents
- Interviewing stakeholders to gather additional information
- Identifying any ambiguities or inconsistencies in the requirements
- Identifying any missing or incomplete requirements
- Identifying any potential risks or issues that may impact the testing process

Creating a requirement traceability matrix (RTM) to map requirements to test cases
At the end of this stage, the testing team should have a clear understanding of the software requirements and should have identified any potential issues that

may impact the testing process. This will help to ensure that the testing process is focused on the most important areas of the software and that the testing team is able to deliver high-quality results.

2. Test Planning

Test Planning is the most efficient phase of the software testing life cycle where all testing plans are defined. In this phase manager of the testing, team calculates the estimated effort and cost for the testing work. This phase gets started once the requirement-gathering phase is completed.

The activities that take place during the Test Planning stage include:

- Identifying the testing objectives and scope
- Developing a test strategy: selecting the testing methods and techniques that will be used
- Identifying the testing environment and resources needed
- Identifying the test cases that will be executed and the test data that will be used
- Estimating the time and cost required for testing
- Identifying the test deliverables and milestones
- Assigning roles and responsibilities to the testing team
- Reviewing and approving the test plan

At the end of this stage, the testing team should have a detailed plan for the testing activities that will be performed, and a clear understanding of the testing objectives, scope, and deliverables. This will help to ensure that the testing process is well-organized and that the testing team is able to deliver high-quality results.

3. Test Case Development

The Test Case Development phase gets started once the test planning phase is completed. In this phase testing team notes down the detailed test cases. The testing team also prepares the required test data for the testing. When the test cases are prepared then they are reviewed by the quality assurance team.

The activities that take place during the Test Case Development stage include:

- Identifying the test cases that will be developed
- Writing test cases that are clear, concise, and easy to understand

- Creating test data and test scenarios that will be used in the test cases
- Identifying the expected results for each test case
- Reviewing and validating the test cases
- Updating the requirement traceability matrix (RTM) to map requirements to test cases

At the end of this stage, the testing team should have a set of comprehensive and accurate test cases that provide adequate coverage of the software or application. This will help to ensure that the testing process is thorough and that any potential issues are identified and addressed before the software is released.

4. Test Environment Setup

Test Environment Setup is an important part of the STLC. Basically, the 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

In Test Execution, after the test case development and test environment setup test execution phase gets started. In this phase testing team starts executing test cases based on prepared test cases in the earlier step.

The activities that take place during the test execution stage of the Software Testing Life Cycle (STLC) include:

- Test execution: The test cases and scripts created in the test design stage are run against the software application to identify any defects or issues.
- Defect logging: Any defects or issues that are found during test execution are logged in a defect tracking system, along with details such as the severity, priority, and description of the issue.
- Test data preparation: Test data is prepared and loaded into the system for test execution
- Test environment setup: The necessary hardware, software, and network configurations are set up for test execution
- Test execution: The test cases and scripts are run, and the results are collected and analyzed.

- Test result analysis: The results of the test execution are analyzed to determine the software's performance and identify any defects or issues.
- Defect retesting: Any defects that are identified during test execution are retested to ensure that they have been fixed correctly.
- Test Reporting: Test results are documented and reported to the relevant stakeholders.

It is important to note that test execution is an iterative process and may need to be repeated multiple times until all identified defects are fixed and the software is deemed fit for release.

6. Test Closure

Test Closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented. The main objective of the test closure stage is to ensure that all testing-related activities have been completed and that the software is ready for release.

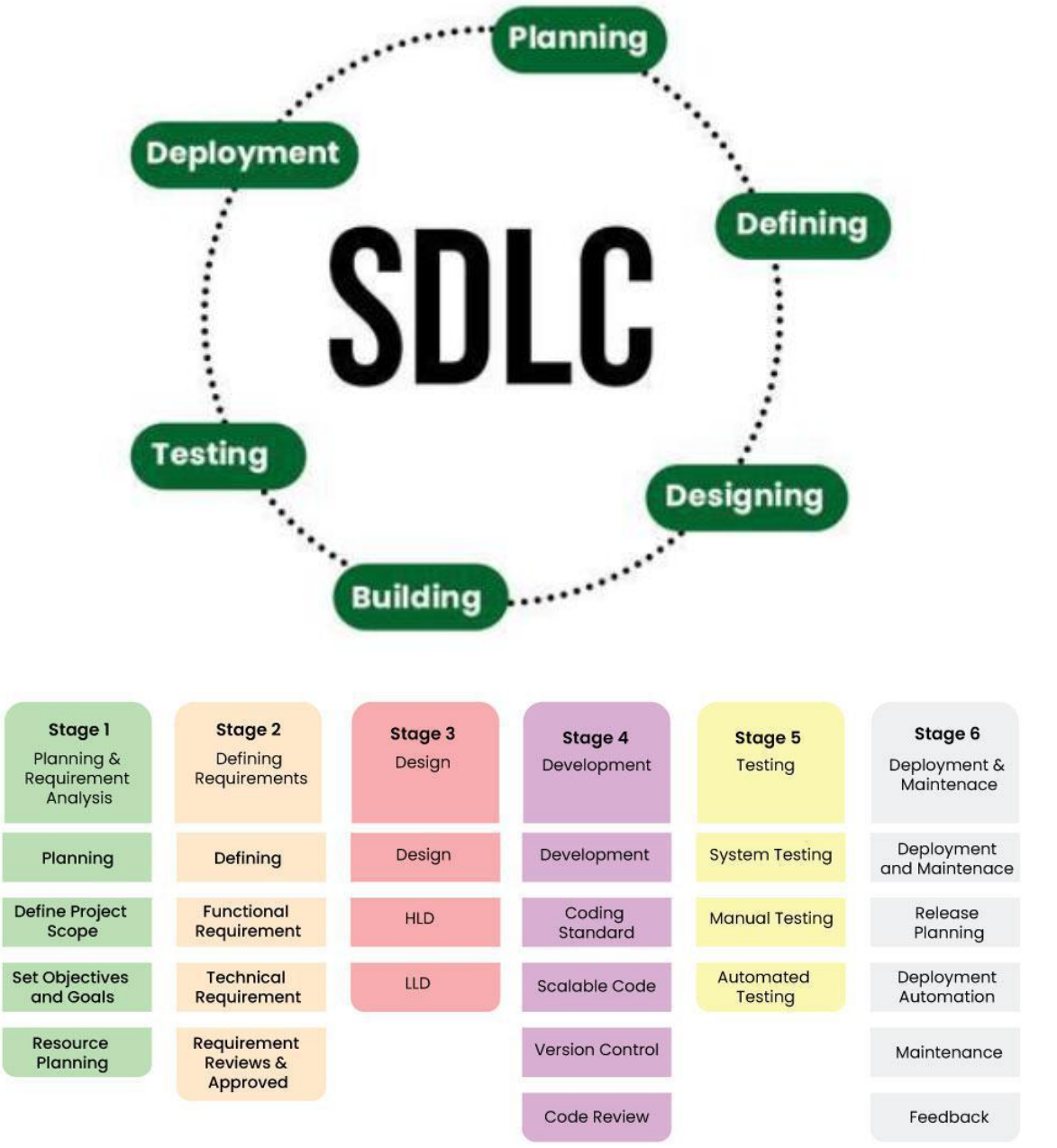
At the end of the test closure stage, the testing team should have a clear understanding of the software's quality and reliability, and any defects or issues that were identified during testing should have been resolved. The test closure stage also includes documenting the testing process and any lessons learned so that they can be used to improve future testing processes

Test closure is the final stage of the Software Testing Life Cycle (STLC) where all testing-related activities are completed and documented. The main activities that take place during the test closure stage include:

- Test summary report: A report is created that summarizes the overall testing process, including the number of test cases executed, the number of defects found, and the overall pass/fail rate.
- Defect tracking: All defects that were identified during testing are tracked and managed until they are resolved.
- Test environment clean-up: The test environment is cleaned up, and all test data and test artifacts are archived.
- Test closure report: A report is created that documents all the testing-related activities that took place, including the testing objectives, scope, schedule, and resources used.

- Knowledge transfer: Knowledge about the software and testing process is shared with the rest of the team and any stakeholders who may need to maintain or support the software in the future.
- Feedback and improvements: Feedback from the testing process is collected and used to improve future testing processes

It is important to note that test closure is not just about documenting the testing process, but also about ensuring that all relevant information is shared and any lessons learned are captured for future reference. The goal of test closure is to ensure that the software is ready for release and that the testing process has been conducted in an organized and efficient manner.



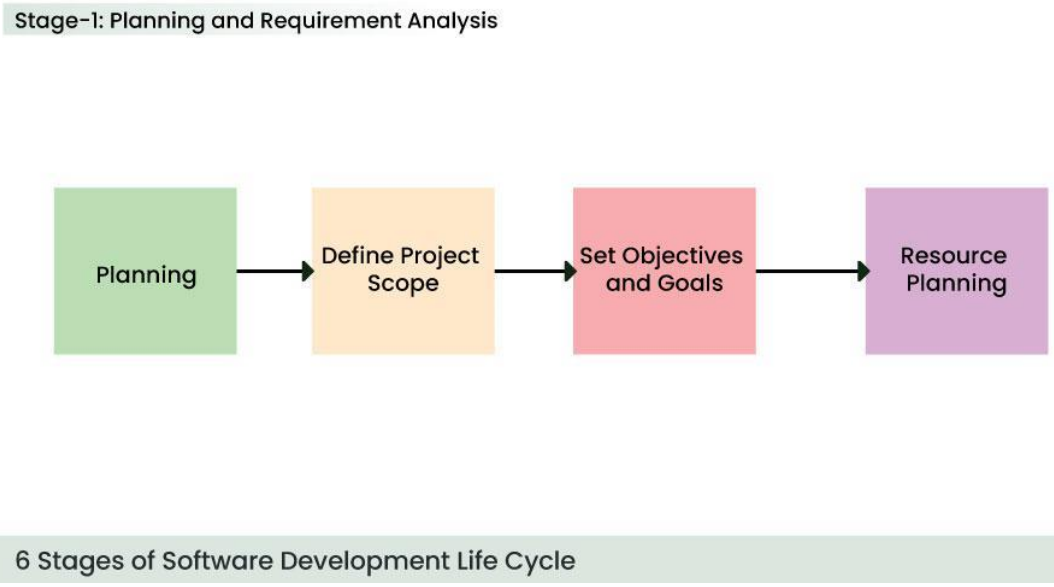
6 Stages of Software Development Life Cycle

The SDLC Model involves six phases or stages while developing any software.

Stage-1: Planning and Requirement Analysis

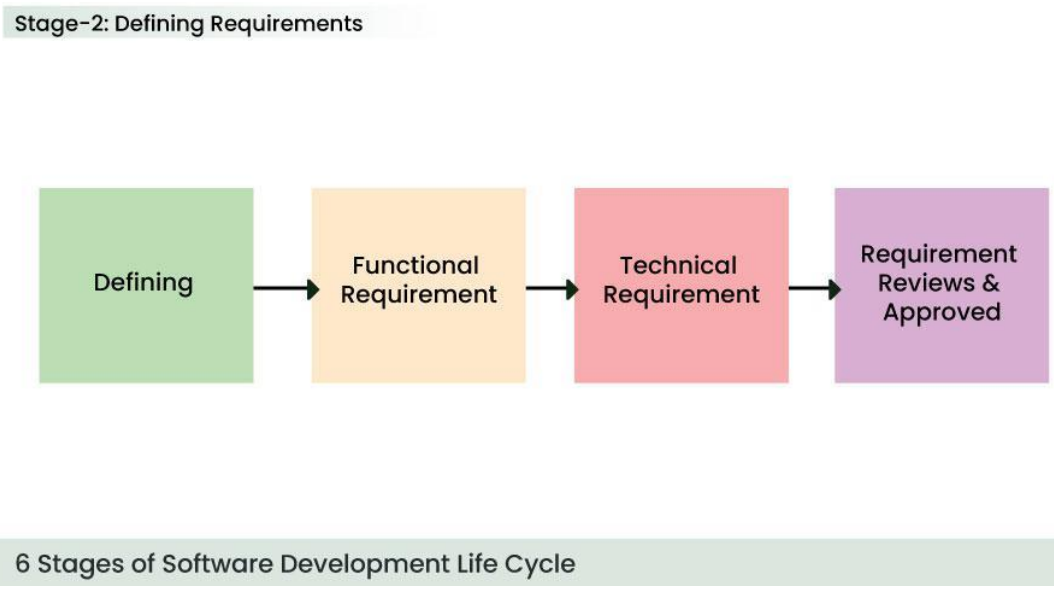
Planning is a crucial step in everything, just as in software development. In this same stage, requirement analysis is also performed by the developers of the organization. This is attained from customer inputs, and sales department/market surveys.

The information from this analysis forms the building blocks of a basic project. The quality of the project is a result of planning. Thus, in this stage, the basic project is designed with all the available information.



Stage-2: Defining Requirements

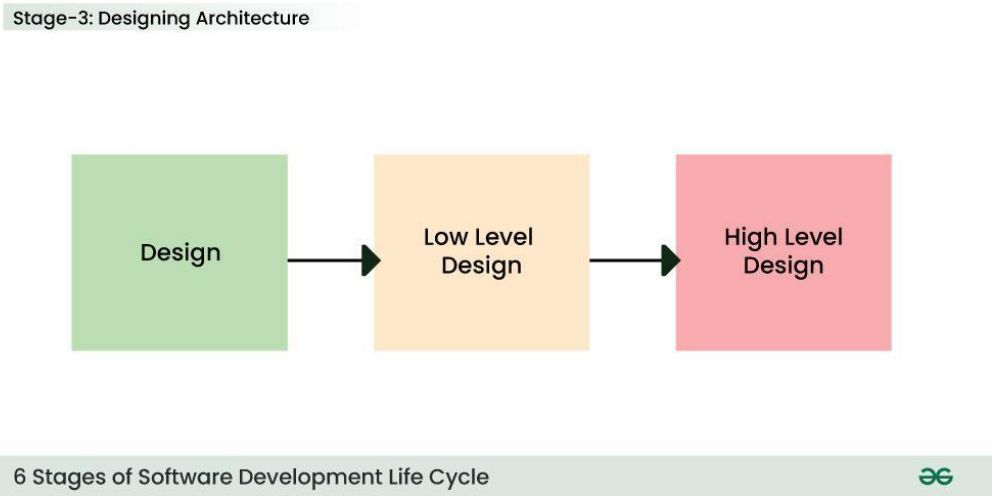
In this stage, all the requirements for the target software are specified. These requirements get approval from customers, market analysts, and stakeholders. This is fulfilled by utilizing SRS (Software Requirement Specification). This is a sort of document that specifies all those things that need to be defined and created during the entire project cycle.



Stage-3: Designing Architecture

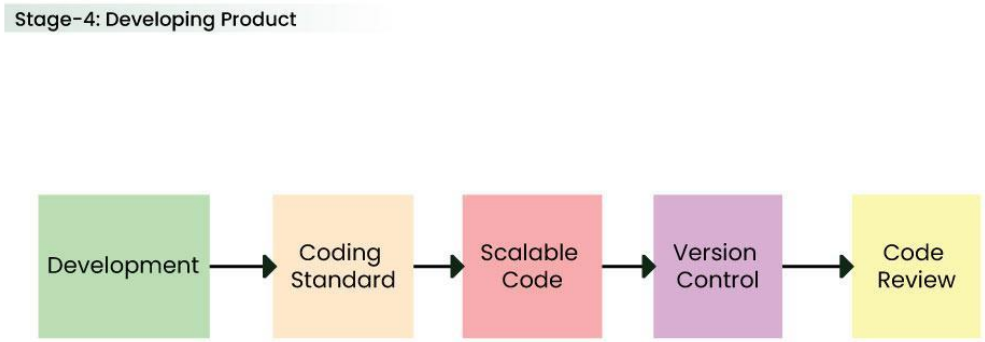
SRS is a reference for software designers to come up with the best architecture for the software. Hence, with the requirements defined in SRS, multiple designs for the product architecture are present in the Design Document Specification (DDS).

This DDS is assessed by market analysts and stakeholders. After evaluating all the possible factors, the most practical and logical design is chosen for development.



Stage-4: Developing Product

At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.



Stage-5: Product Testing and Integration

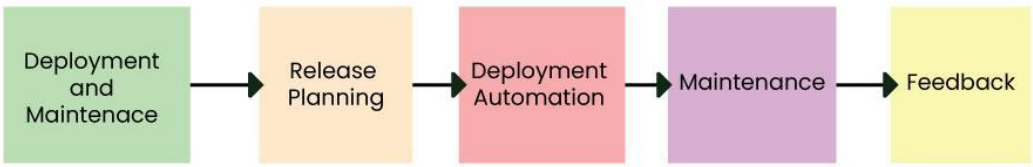
After the development of the product, testing of the software is necessary to ensure its smooth execution. Although, minimal testing is conducted at every stage of SDLC. Therefore, at this stage, all the probable flaws are tracked, fixed, and retested. This ensures that the product confronts the quality requirements of SRS. Documentation, Training, and Support: Software documentation is an essential part of the software development life cycle. A well-written document acts as a tool and means to information repository necessary to know about software processes, functions, and maintenance. Documentation also provides information about how to use the product. Training in an attempt to improve the current or future employee performance by increasing an employee's ability to work through learning, usually by changing his attitude and developing his skills and understanding.

Stage-5: Product Testing and Integration

Stage-6: Deployment and Maintenance of Products

After detailed testing, the conclusive product is released in phases as per the organization’s strategy. Then it is tested in a real industrial environment. It is important to ensure its smooth performance. If it performs well, the organization sends out the product as a whole. After retrieving beneficial feedback, the company releases it as it is or with auxiliary improvements to make it further helpful for the customers. However, this alone is not enough. Therefore, along with the deployment, the product's supervision.

Stage 6: Deployment and Maintenance of Products



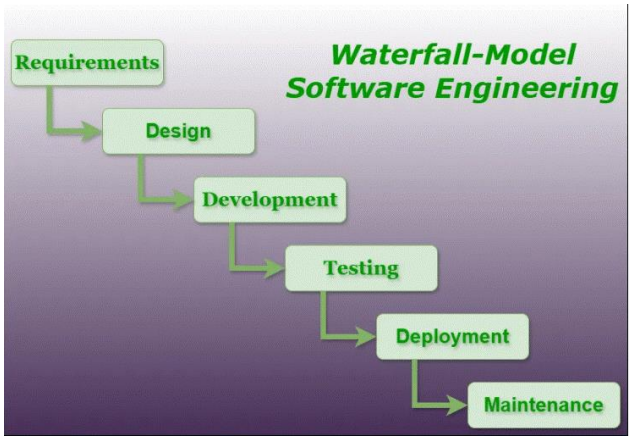
Software Development Life Cycle Models

To this day, we have more than 50 recognized SDLC models in use. But None of them is perfect, and each brings its favorable aspects and disadvantages for a specific software development project or a team.

Here, we have listed the top five most popular SDLC models:

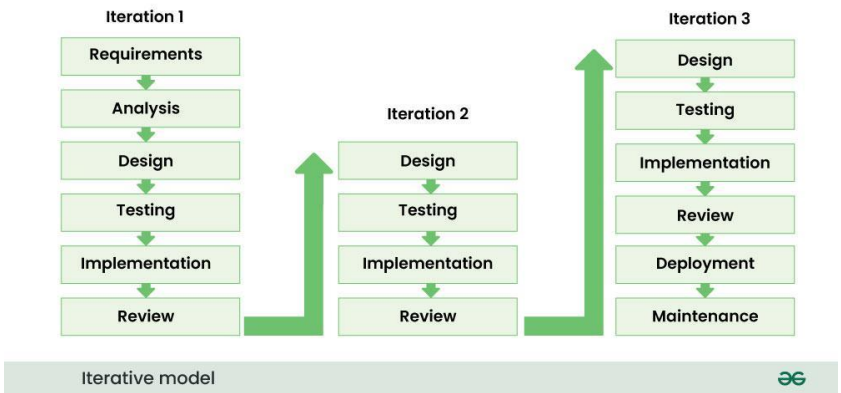
1. Waterfall Model

It is the fundamental model of the software development life cycle. This is a very simple model. The waterfall model is not in practice anymore, but it is the basis for all other SDLC models. Because of its simple structure, the waterfall model is easier to use and provides a tangible output. In the waterfall model, once a phase seems to be completed, it cannot be changed, and due to this less flexible nature, the waterfall model is not in practice anymore.



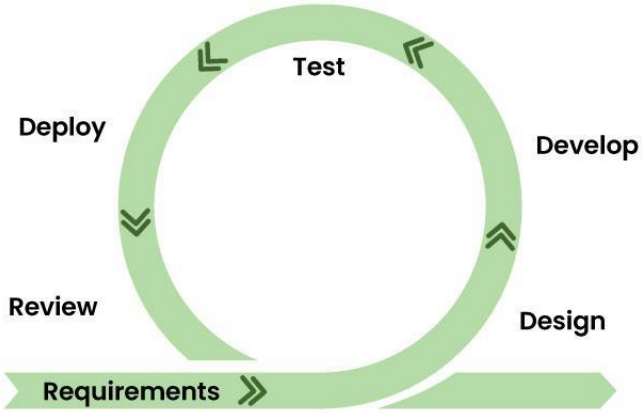
2. Iterative SDLC Models

In software development, choosing the right SDLC models is crucial for success. Among the various approaches, the Iterative SDLC model stands out as a flexible and efficient methodology that promotes continuous improvement and adaptability. In this blog post, we will explore the intricacies of the Iterative SDLC models, shedding light on its principles, benefits, and best practices



3. Agile Model

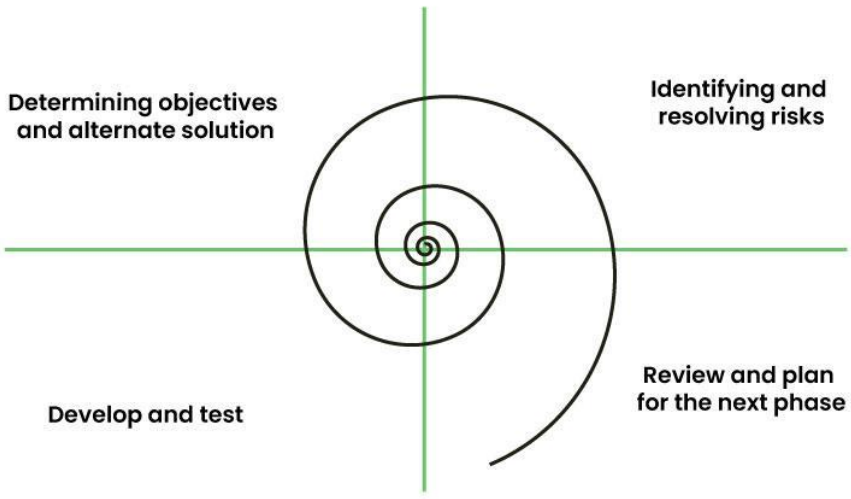
The agile model in SDLC was mainly designed to adapt to changing requests quickly. The main goal of the Agile model is to facilitate quick project completion. The agile model refers to a group of development processes. These processes have some similar characteristics but also possess certain subtle differences among themselves.



Agile Model

4. Spiral Model

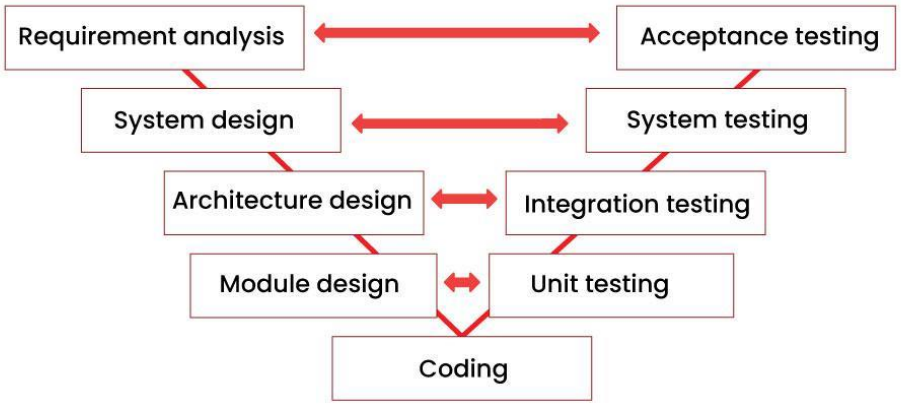
The spiral model in SDLC is one of the most crucial SDLC models that provides support for risk handling. It has various spirals in its diagrammatic representation; the number of spirals depends upon the type of project. Each loop in the spiral structure indicates the Phases of the Spiral model.



Spiral model

5. V-Shaped Model

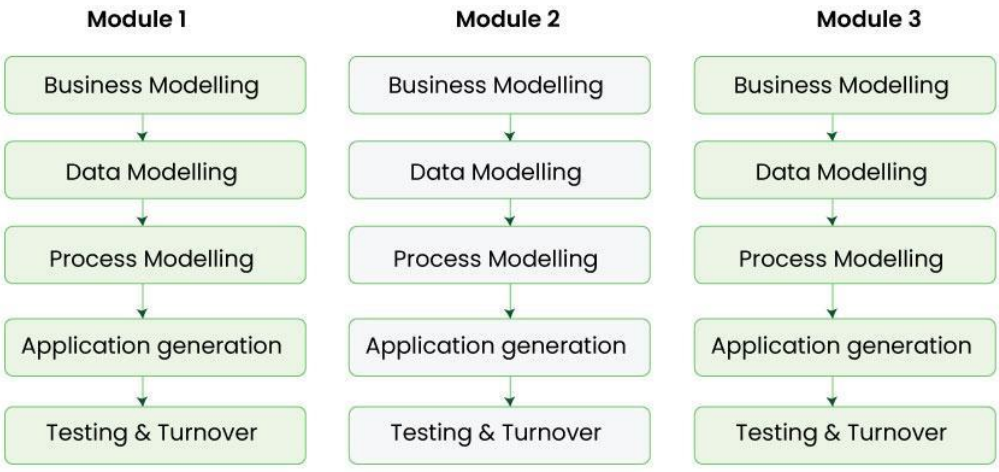
The V-shaped model in SDLC is executed in a sequential manner in V-shape. Each stage or phase of this model is integrated with a testing phase. After every development phase, a testing phase is associated with it, and the next phase will start once the previous phase is completed, i.e., development & testing. It is also known as the verification or validation model.



V-Model

6. Rapid Application Development (RAD) SDLC Models

Rapid Application Development is an iterative and incremental model that prioritizes quick development and iteration cycles. It places a strong emphasis on user feedback and involvement throughout the development process. RAD aims to deliver functional prototypes rapidly, allowing stakeholders to provide feedback and guide ongoing development.



RAD Model

What is the need for SDLC?

SDLC is a method, approach, or process that is followed by a software development organization while developing any software. SDLC models were introduced to follow a disciplined and systematic method while designing software. With the software development life cycle, the process of software design is divided into small parts, which makes the problem more understandable and easier to solve. SDLC comprises a detailed description or step-by-step plan for designing, developing, testing, and maintaining the software.

How does SDLC Address Security?

A frequent issue in software development is the delay of security-related tasks until the testing phase, which occurs late in the software development life cycle (SDLC) and occurs after the majority of crucial design and implementation has been finished. During the testing phase, security checks may be minimal and restricted to scanning and penetration testing, which may fail to identify more complicated security flaws.

Security issue can be address in SDLC by following DevOps. Security is integrated throughout the whole SDLC, from build to production, through the use of DevSecOps. Everyone involved in the DevOps value chain have responsibility for security under DevSecOps.

Real Life Example of SDLC

Developing a banking application using SDLC:

- Planning and Analysis: During this stage, business stakeholders' requirements about the functionality and features of banking application will be gathered by program managers and business analysts. Detailed SRS (Software Requirement Specification) documentation will be produced by them. Together with business stakeholders, business analysts will analyse and approve the SRS document.
- Design: Developers will receive SRS documentation. Developers will read over the documentation and comprehend the specifications. Web pages will be designed by designers. High level system architecture will be prepared by developers.
- Development: During this stage, development will code. They will create the web pages and APIs needed to put the feature into practice.

- Testing: Comprehensive functional testing will be carried out. They will guarantee that the banking platform is glitch-free and operating properly.
- Deployment and Maintenance: The code will be made available to customers and deployed. Following this deployment, the customer can access the online banking. The same methodology will be used to create any additional features.

How to Choose an SDLC Model?

Choosing the right SDLC (Software Development Life Cycle) model is essential for project success. Here are the key factors to consider:

1. **Project Requirements:**
    - Clear Requirements: Use Waterfall or V-Model if requirements are well-defined and unlikely to change.
    - Changing Requirements: Use Agile or Iterative models if requirements are unclear or likely to evolve.
2. **Project Size and Complexity:**
    - Small Projects: Use Waterfall or RAD for small, simple projects.
    - Large Projects: Use Agile, Spiral, or DevOps for large, complex projects that need flexibility.
3. **Team Expertise:**
    - Experienced Teams: Use Agile or Scrum if the team is familiar with iterative development.
    - Less Experienced Teams: Use Waterfall or V-Model for teams needing structured guidance.
4. **Client Involvement:**
    - Frequent Client Feedback: Use Agile, Scrum, or RAD if regular client interaction is needed.
    - Minimal Client Involvement: Use Waterfall or V-Model if client involvement is low after initial planning.
5. **Time and Budget Constraints:**
    - Fixed Time and Budget: Use Waterfall or V-Model if you have strict time and budget limits.

- Flexible Time and Budget: Use Agile or Spiral if you can adjust time and budget as needed.

6. **Risk Management:**
    - High-Risk Projects: Use Spiral for projects with significant risks and uncertainties.
    - Low-Risk Projects: Use Waterfall for projects with minimal risks.
7. **Product Release Timeline:**
    - Quick Release Needed: Use Agile or RAD to deliver products quickly.
    - Longer Development Time: Use Waterfall or V-Model for projects with no urgent deadlines.
8. **Maintenance and Support:**
    - Long-Term Maintenance: Use Agile or DevOps for projects needing continuous updates and support.
    - Minimal Maintenance: Use Waterfall or V-Model if little future maintenance is expected.
9. **Stakeholder Expectations:**
    - High Stakeholder Engagement: Use Agile or Scrum if stakeholders want ongoing involvement.
    - Low Stakeholder Engagement: Use Waterfall or V-Model if stakeholders prefer involvement only at major milestones.

Note:

- *Waterfall: Best for clear, stable projects with minimal changes.*
- *V-Model: Good for projects with clear requirements and a strong focus on testing.*
- *Agile/Scrum: Ideal for projects with changing requirements and frequent client interaction.*
- *Spiral: Suitable for high-risk projects with evolving requirements.*
- *RAD: Useful for projects needing rapid development.*

What is a Bug/Defect?

A defect is an error or bug in an application that is created during the building or designing of software due to which software starts to show abnormal behaviors during its use. So it is one of the important responsibilities of the tester to find as many as defects possible to ensure the quality of the product is not affected and the end product is fulfilling all requirements perfectly for which it has been designed and provides the required services to the end-user. Because as much as defects will be identified and resolved then the software will behave perfectly as per expectation.

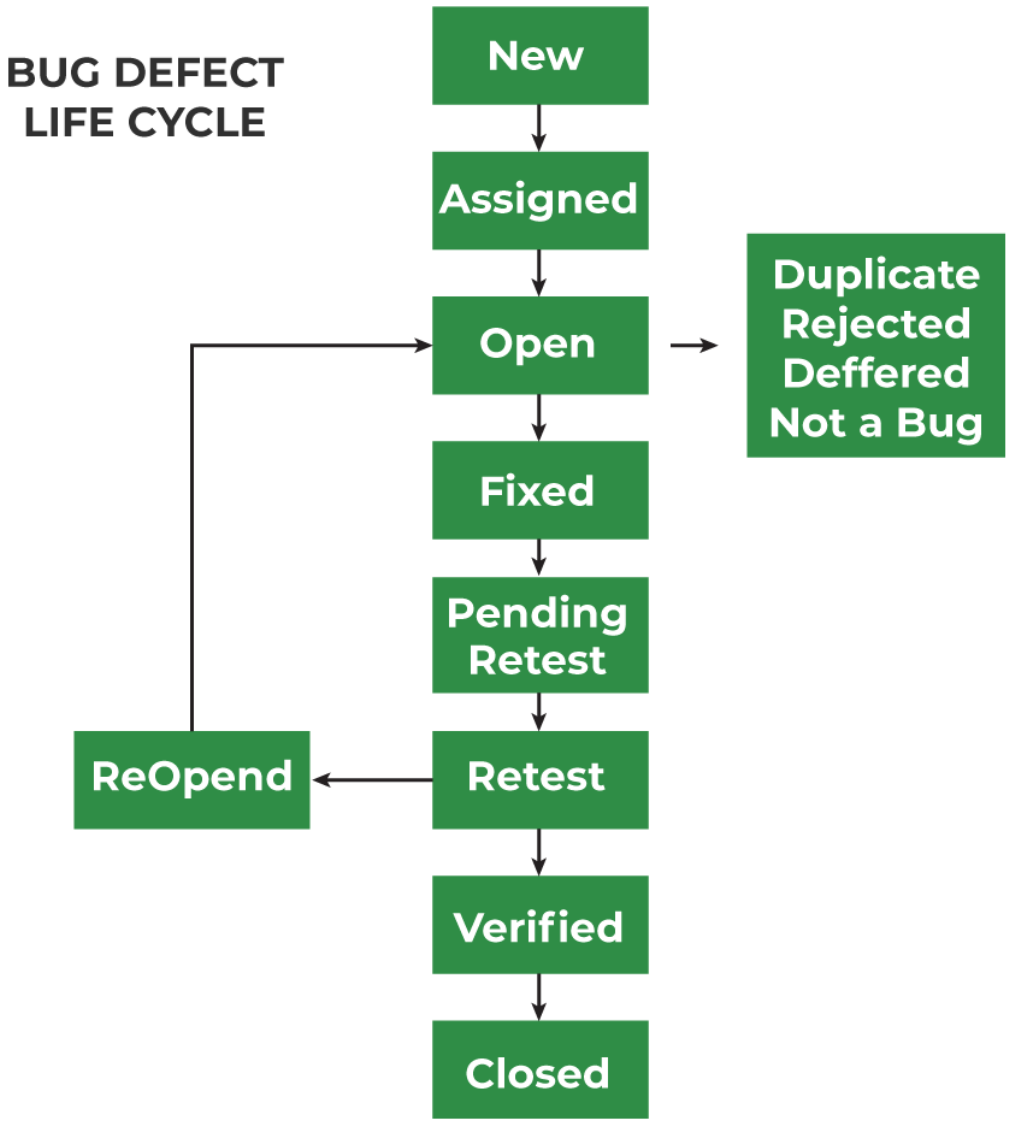
What is the Defect Life Cycle?

In the Software Development Process, the Defect Life Cycle is the life cycle of a defect or bug that it goes through covering a specific set of states in its entire life. Mainly bug life cycle refers to its entire state starting from a new defect detected to the closing off of that defect by the tester. Alternatively, it is also called a Bug Life Cycle.

- The journey of the Defect Cycle varies from organization to organization and also from project to project because development procedures and platforms as well as testing methods and testing tools differ depending upon organizations and projects.
- The number of states that a defect goes through also varies depending upon the different tools used and processes followed during the software testing.
- The objective of the defect lifecycle is to easily coordinate and communicate the current status of the defect and thus help to make the defect-fixing process efficient.

Defect Status

Defect status or Bug status is the current state from which the defect is currently going through. The number of states the defect goes through varies from project to project. The below diagram illustrates all the possible states of the defect:



1. New: When any new defect is identified by the tester, it falls in the 'New' state. It is the first state of the Bug Life Cycle. The tester provides a proper Defect document to the Development team so that the development team can refer to Defect Document and can fix the bug accordingly.

2. Assigned: Defects that are in the status of 'New' will be approved and that newly identified defect is assigned to the development team for working on the defect and to resolve that. When the defect is assigned to the developer team the status of the bug changes to the 'Assigned' state.

3. Open: In this 'Open' state the defect is being addressed by the developer team and the developer team works on the defect for fixing the bug. Based on some specific reason if the developer team feels that the defect is not appropriate then it is transferred to either the 'Rejected' or 'Deferred' state.

4. Fixed: After necessary changes of codes or after fixing identified bug developer team marks the state as 'Fixed'.

5. Pending Retest: During the fixing of the defect is completed, the developer team passes the new code to the testing team for retesting. And the code/application is pending for retesting on the Tester side so the status is assigned as 'Pending Retest'.

6. Retest: At this stage, the tester starts work of retesting the defect to check whether the defect is fixed by the developer or not, and the status is marked as 'Retesting'.

7. Reopen: After 'Retesting' if the tester team found that the bug continues like previously even after the developer team has fixed the bug, then the status of the bug is again changed to 'Reopened'. Once again bug goes to the 'Open' state and goes through the life cycle again. This means it goes for Re-fixing by the developer team.

8. Verified: The tester re-tests the bug after it got fixed by the developer team and if the tester does not find any kind of defect/bug then the bug is fixed and the status assigned is 'Verified'.

9. Closed: It is the final state of the Defect Cycle, after fixing the defect by the developer team when testing found that the bug has been resolved and it does not persist then they mark the defect as a 'Closed' state.

Few More States that also come under this Defect Life Cycle:

1. Rejected: If the developer team rejects a defect if they feel that defect is not considered a genuine defect, and then they mark the status as ‘Rejected’. The cause of rejection may be any of these three i.e Duplicate Defect, NOT a Defect, Non-Reproducible.

2. Deferred: All defects have a bad impact on developed software and also they have a level based on their impact on software. If the developer team feels that the defect that is identified is not a prime priority and it can get fixed in further updates or releases then the developer team can mark the status as 'Deferred'. This means from the current defect life cycle it will be terminated.

3. Duplicate: Sometimes it may happen that the defect is repeated twice or the defect is the same as any other defect then it is marked as a 'Duplicate' state and then the defect is 'Rejected'.

4. Not a Defect: If the defect has no impact or effect on other functions of the software then it is marked as 'NOT A DEFECT' state and 'Rejected'.

5. Non-Reproducible: If the defect is not reproduced due to platform mismatch, data mismatch, build mismatch, or any other reason then the developer marks the defect as in a 'Non-Reproducible' state.

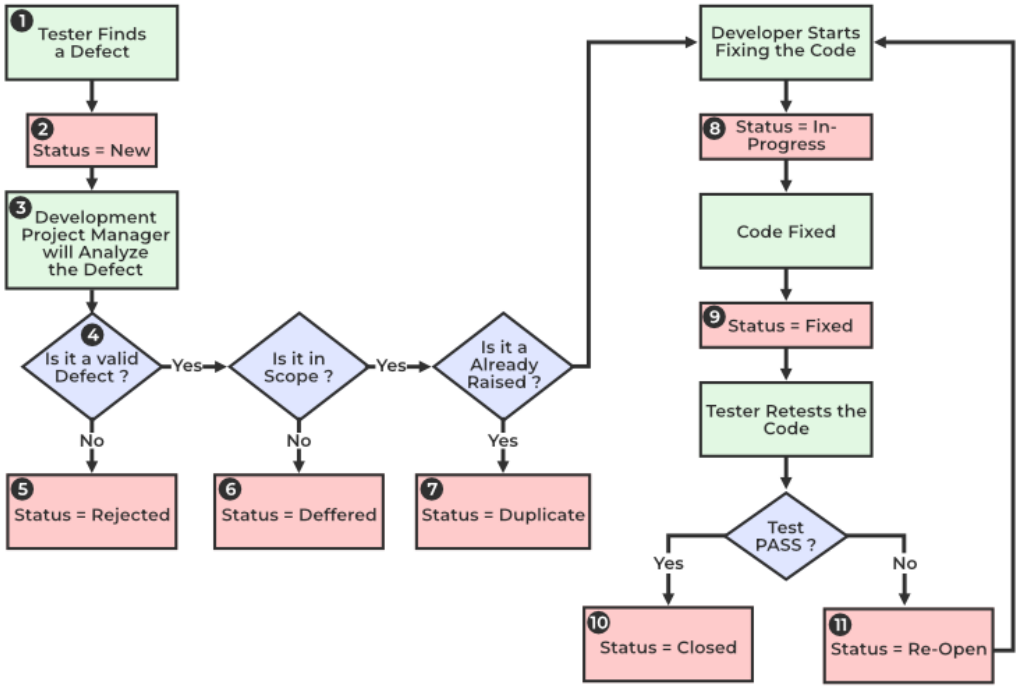
6. Can't be Fixed: If the developer team fails to fix the defect due to Technology support, the Cost of fixing a bug is more, lack of required skill, or due to any other reasons then the developer team marks the defect as in 'Can't be fixed' state.

7. Need more information: This state is very close to the 'Non-reproducible' state. But it is different from that. When the developer team fails to reproduce the defect due to the steps/document provided by the tester being insufficient or the Defect Document is not so clear to reproduce the defect then the developer team can change the status to “Need more information’. When the Tester team provides a good defect document the developer team proceeds to fix the bug.

Defect Lifecycle

Consider the flow chart below to understand the defect lifecycle.

1. The tester finds a defect.
2. The defect status is changed to New.
3. The development manager will then analyze the defect.
4. The manager determines if the defect is valid or not.
5. If the defect is not valid then the development manager assigns the status Rejected to defect.
6. If the defect is valid then it is checked whether the defect is in scope or not. If no then the defect status is changed to Deferred.
7. If the defect is in scope then the manager checks whether a similar defect was raised earlier. If yes then the defect is assigned a status duplicate.
8. If the defect is not already raised then the manager starts fixing the code and the defect is assigned the status In-Progress.
9. Once the defect is fixed the status is changed to fixed.
10. The tester retests the code if the test is passed then the defect status is changed to closed.
11. If the test fails again then the defect is assigned status reopened and assigned to the developer.



Benefits of Defect Lifecycle

- Deliver High-Quality Product: The defect lifecycle helps in identifying and fixing bugs throughout the development process, that helps in delivering high-quality product.
- Improve Return on Investment (ROI): Finding and fixing defects early in the lifecycle is cheaper and less time-consuming than addressing them later, which saves money and resources.
- Better Communication: The defect lifecycle provides a structured process for logging, tracking, and resolving defects, which provides better communication and collaboration among team members.
- Early Issue Detection: The lifecycle process allows for early detection of defects, enabling the team to understand patterns and trends, and take preventive measures for future development.
- Customer Satisfaction: By systematically managing and resolving defects, the final product is more reliable and user-friendly, leading to higher customer satisfaction and loyalty.

Limitations in Defect Lifecycle

- Time-Consuming: Tracking and managing defects can be lengthy and complex.
- Resource Intensive: Requires dedicated resources for effective defect management.
- Possible Overhead: Bug Lifecycle can introduce additional administrative tasks, slowing down development.
- Delayed Releases: Extensive defect management might delay product release timelines.

SYSTEM TESTING

A specific level of software testing where the complete, integrated software system is tested.

Objectives of System Testing

- Validate the end-to-end flow of the application.
- Ensure that all modules work together correctly.
- Verify the software meets business and technical requirements.
- Detect defects in the system behavior, not individual components.

Prerequisites for System Testing

1. Integration Testing must be completed.
2. The software should be fully developed and integrated.
3. Test environment setup (hardware, software, network) should be ready.
4. Test plans and test cases should be created.
5. Entry criteria are met:
    - All units integrated
    - High-priority bugs fixed
    - Stable build is ready

Types of System Testing

- Functional Testing
    - Focuses on what the system does.
    - Tests features like login, search, payment, etc.
- Non-Functional Testing
    - Focuses on how well the system performs.
    - Includes:
        - Performance Testing
        - Load Testing

- Stress Testing
- Security Testing
- Usability Testing
- Compatibility Testing
- Recovery Testing

System Testing Techniques

1. Black-box testing
2. Requirement-based testing
3. Use-case testing
4. Risk-based testing
5. End-to-end scenario testing

System Testing Process

Step	Description
1. 📋 Test Plan Preparation	Define objectives, scope, resources, and schedule
2. 🧪 Test Case Design	Create detailed test cases and expected results
3. 🏗️ Test Environment Setup	Ensure system, network, and data are ready
4. 🚀 Test Execution	Run test cases on the system
5. 🐞 Defect Logging	Log bugs and communicate with developers
6. 🔁 Regression Testing	Re-test after bug fixes
7. ✅ Test Closure	Prepare test summary report, metrics, and review

🛠️ Popular Tools for System Testing

Tool Name	Use
Selenium	Functional UI testing
JMeter	Performance and load testing
TestComplete	Automated functional testing
LoadRunner	Load testing
Postman	API testing
QTP/UFT	Automation of regression and functional tests
Bugzilla / Jira	Bug tracking

🧠 Example Scenario of System Testing

Scenario: Online Shopping App

Test Area	Example
Functional	Can the user search for products? Can they add to cart?
Usability	Is the checkout process user-friendly?
Performance	How fast is the page loading under 1000 users?
Security	Can unauthorized users access user data?
Compatibility	Does it work on Chrome, Firefox, Safari?
Recovery	What happens if the payment gateway crashes?

✅ Entry and Exit Criteria

Entry Criteria:

- Software is fully integrated
- All lower-level testing (unit/integration) is complete
- Test cases are ready

Exit Criteria:

- All planned test cases executed

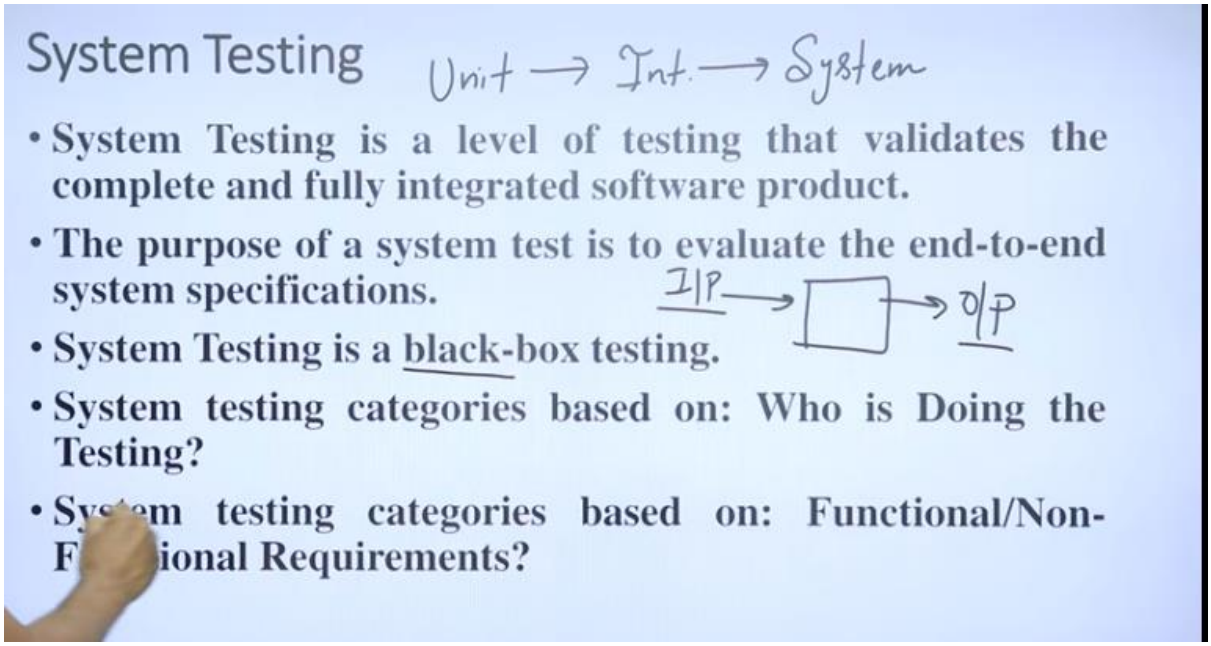
- Critical defects fixed or accepted
- Test reports are prepared

📌 Key Features of System Testing

- Independent Testing Team: Often done by testers who did not write the code.
- Focuses on End-to-End Scenarios: Ensures realistic user experiences are validated.
- Does Not Rely on Code Knowledge: It's black-box oriented.
- Helps Reduce Risk: Ensures no critical issue is left before delivery.

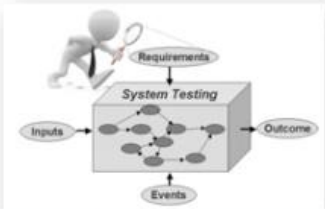
🚫 Common Mistakes in System Testing

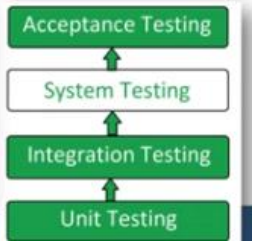
- Incomplete test coverage
- Missing edge cases
- Testing without real-world data
- Poor documentation of test cases
- Not updating test cases after requirement changes



System Testing

- System Testing is validates the complete and fully integrated software product.
- The purpose of a system test is to evaluate the end-to-end system specifications.
- It's a type of black box testing perform by Quality Assurance Team in Testing phase.
- It focus on the Functionality, Accuracy, Quality, Expected Output & Overall behavior of an application rather than the inner working of system.
- **System Testing Tools:** Selenium, LoadRunner, Jmeter, Microsoft Test Manager, SoapUI
- The choice of tool depends on various factors like Technology used, Size of the project, Budget & Testing requirements etc.

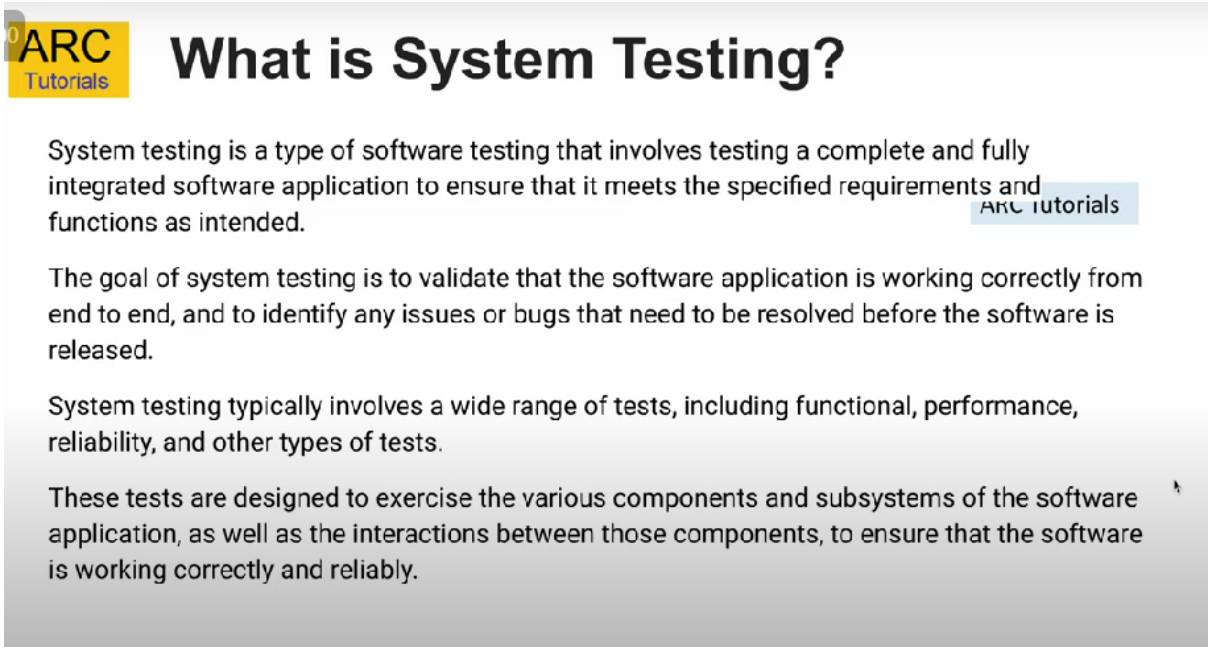


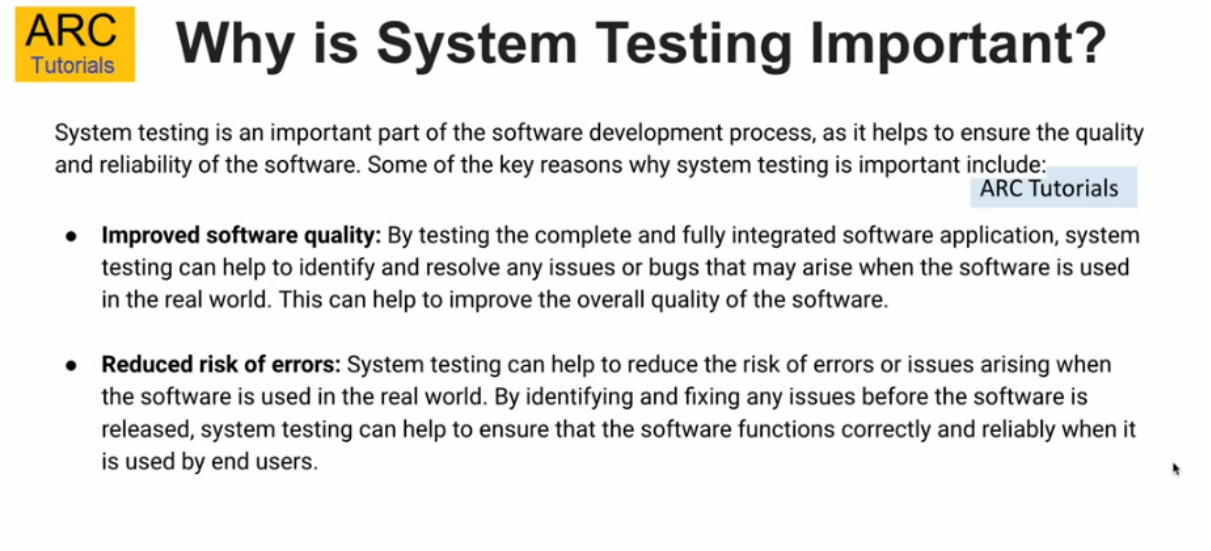


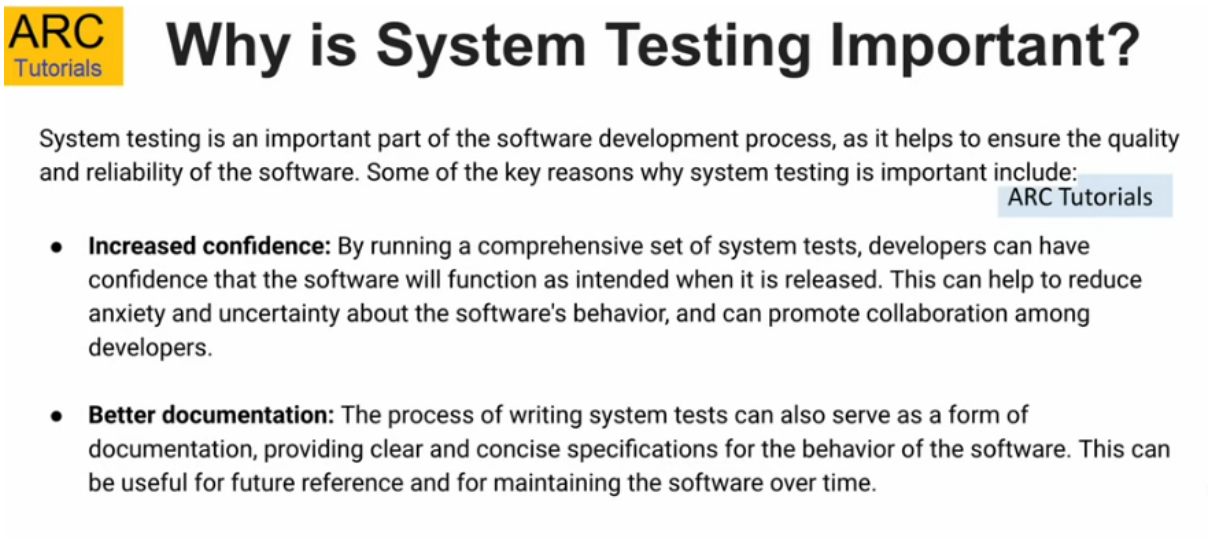
CS & IT Tutorials by Vrushali

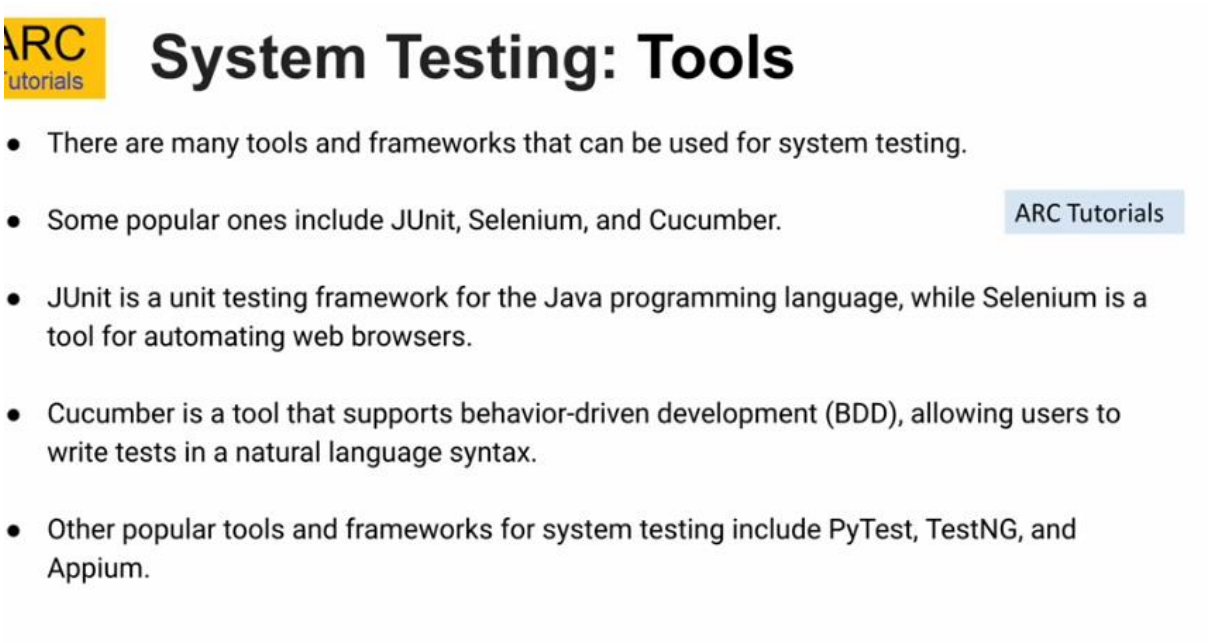
Importance of System Testing

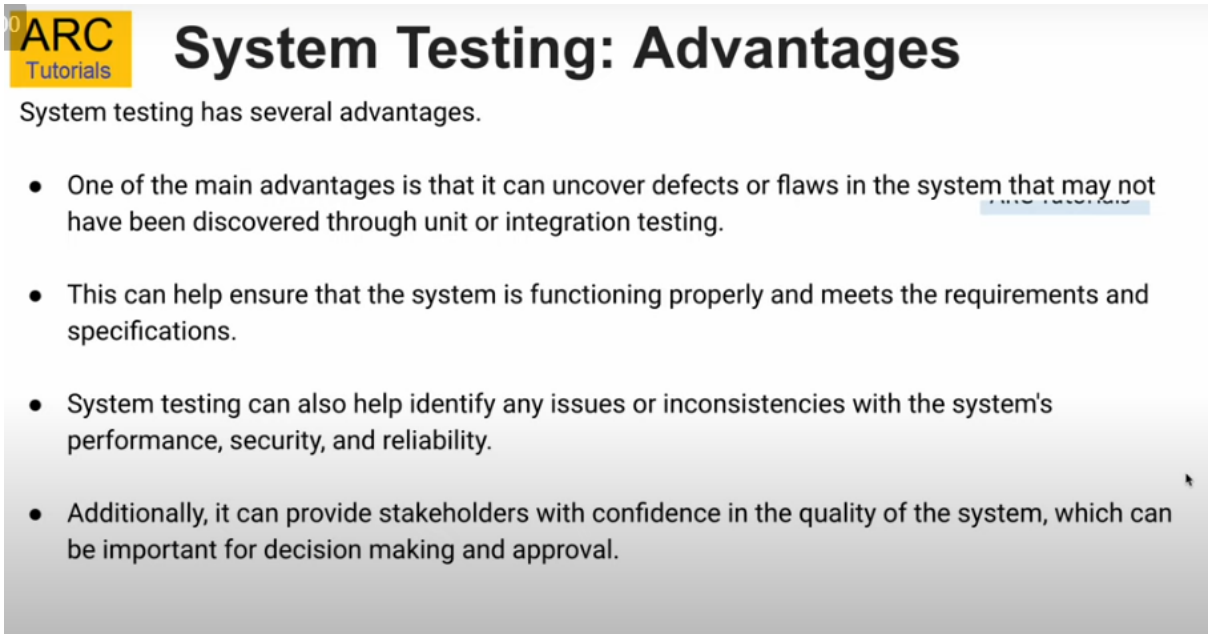
1. **Improved Product Quality:** It test product can successfully work across different platforms and environments.
2. **Error Reduction:** It verifies a system's code & functionality as per requirements, so errors that aren't detected during integration and unit testing can be exposed during system testing.
3. **Cost Savings:** Conducting timely & continuous system testing reduces unexpected costs & project delays.
4. **Security:** They ensure that the tested system doesn't contain potential vulnerabilities or bugs that can put end users system data at risk.
5. **Customer Satisfaction:** It builds customer confidence & improves overall user experience.
6. **Software Performance:** It track complete system performance. Also help to understand changes in a system's performance & behavior, such as memory consumption, central processing unit utilization etc. Inform developers to take proactive action.

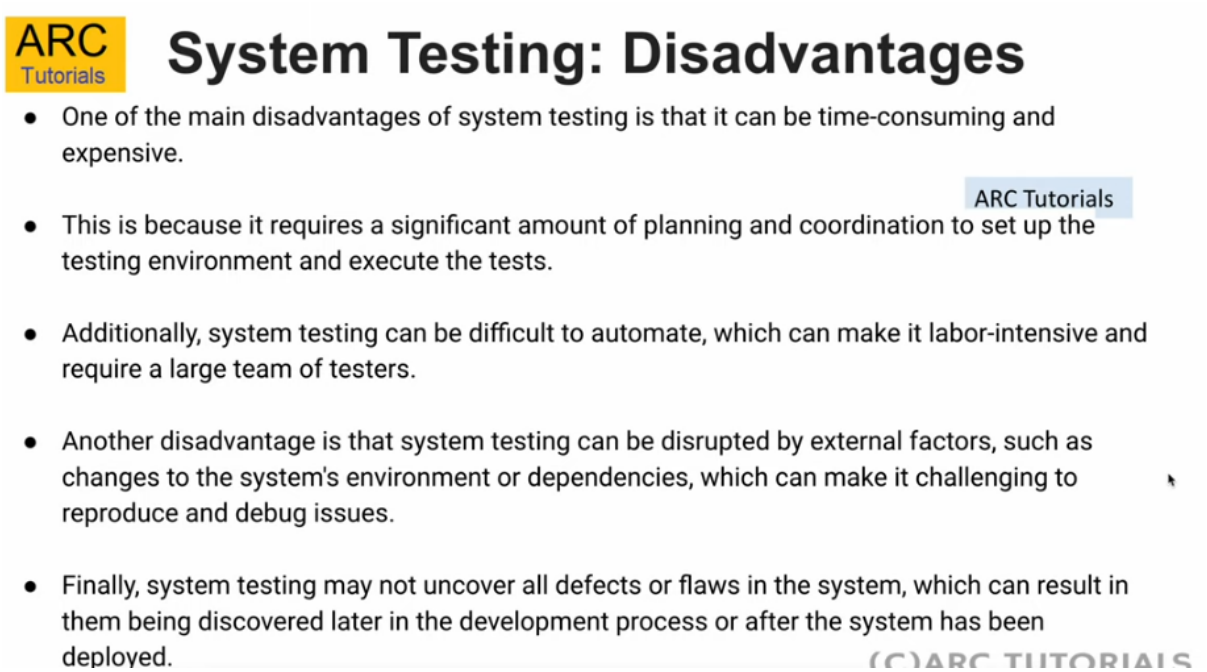

ARC
Tutorials
What is System Testing?
System testing is a type of software testing that involves testing a complete and fully integrated software application to ensure that it meets the specified requirements and functions as intended.
ARC Tutorials
The goal of system testing is to validate that the software application is working correctly from end to end, and to identify any issues or bugs that need to be resolved before the software is released.
System testing typically involves a wide range of tests, including functional, performance, reliability, and other types of tests.
These tests are designed to exercise the various components and subsystems of the software application, as well as the interactions between those components, to ensure that the software is working correctly and reliably.


ARC
Tutorials
Why is System Testing Important?
System testing is an important part of the software development process, as it helps to ensure the quality and reliability of the software. Some of the key reasons why system testing is important include:
ARC Tutorials
Improved software quality: By testing the complete and fully integrated software application, system testing can help to identify and resolve any issues or bugs that may arise when the software is used in the real world. This can help to improve the overall quality of the software.
Reduced risk of errors: System testing can help to reduce the risk of errors or issues arising when the software is used in the real world. By identifying and fixing any issues before the software is released, system testing can help to ensure that the software functions correctly and reliably when it is used by end users.


ARC
Tutorials
Why is System Testing Important?
System testing is an important part of the software development process, as it helps to ensure the quality and reliability of the software. Some of the key reasons why system testing is important include:
ARC Tutorials
Increased confidence: By running a comprehensive set of system tests, developers can have confidence that the software will function as intended when it is released. This can help to reduce anxiety and uncertainty about the software's behavior, and can promote collaboration among developers.
Better documentation: The process of writing system tests can also serve as a form of documentation, providing clear and concise specifications for the behavior of the software. This can be useful for future reference and for maintaining the software over time.


ARC
Tutorials
System Testing: Tools
There are many tools and frameworks that can be used for system testing.
Some popular ones include JUnit, Selenium, and Cucumber.
ARC Tutorials
JUnit is a unit testing framework for the Java programming language, while Selenium is a tool for automating web browsers.
Cucumber is a tool that supports behavior-driven development (BDD), allowing users to write tests in a natural language syntax.
Other popular tools and frameworks for system testing include PyTest, TestNG, and Appium.


ARC
Tutorials
System Testing: Advantages
System testing has several advantages.
One of the main advantages is that it can uncover defects or flaws in the system that may not have been discovered through unit or integration testing.
This can help ensure that the system is functioning properly and meets the requirements and specifications.
System testing can also help identify any issues or inconsistencies with the system's performance, security, and reliability.
Additionally, it can provide stakeholders with confidence in the quality of the system, which can be important for decision making and approval.



Advantages of System Testing

- In System Testing The testers do not require more knowledge of programming to carry out this testing.
- It will test the entire product or software so that we will easily detect the errors or defects which cannot be identified during the unit testing and integration testing.
- The testing environment is similar to that of the real time production or business environment.
- It checks the entire functionality of the system with different test scripts and also it covers the technical and business requirements of clients.
- After this testing, the product will almost cover all the possible bugs or errors and hence the development team will confidently go ahead with acceptance testing
- Verifies the overall functionality of the system.
- Detects and identifies system-level problems early in the development cycle.
- Helps to validate the requirements and ensure the system meets the user needs.
- Improves system reliability and quality.

- Facilitates collaboration and communication between development and testing teams.
- Enhances the overall performance of the system.
- Increases user confidence and reduces risks.
- Facilitates early detection and resolution of bugs and defects.
- Supports the identification of system-level dependencies and inter-module interactions.
- Improves the system's maintainability and scalability.

Disadvantages of System Testing

- System Testing is time consuming process than another testing techniques since it checks the entire product or software.
- The cost for the testing will be high since it covers the testing of entire software.
- It needs good debugging tool otherwise the hidden errors will not be found.
- Can be time-consuming and expensive.
- Requires adequate resources and infrastructure.
- Can be complex and challenging, especially for large and complex systems.
- Dependent on the quality of requirements and design documents.
- Limited visibility into the internal workings of the system.
- Can be impacted by external factors like hardware and network configurations.
- Requires proper planning, coordination, and execution.
- Can be impacted by changes made during development.
- Requires specialized skills and expertise.
- May require multiple test cycles to achieve desired results.

Best Practices for System Testing

To make system testing effective, follow these best practices:

1. Clear Test Plan and Requirements: Make sure you have a well-defined test plan and clear requirements. This ensures all parts of the system are covered and tested correctly.
2. Test Early and Often: Start testing as soon as possible in the development process and continue testing regularly. Finding and fixing issues early helps save time and money.
3. Use Automation for Repetitive Tests: Automate tests that you run repeatedly, like regression or performance tests. This saves time and allows you to focus on more complex testing.
4. Create Realistic Test Scenarios: Design test cases based on how users will actually use the system. This includes testing for edge cases, unusual inputs, and potential system failures.
5. Collaborate with Developers: Work closely with developers throughout the testing process. They can help identify problems faster and provide insights into how the system functions.

Basic requirements of System testing

A system test's objective is to assess the complete system requirements. System testing consists of several tests to exercise the entire computer-based system. To achieve your end goal, you must abide by the factors listed below while executing system test operations.

- Industry type: To comprehend the testing process and ensure you have the resources to do the assignment, be aware of the industry vertical to which your organization belongs. An organization can employ more automatic analyses, like something of a functionality evaluation, if it chooses a more practical methodology. Rather than using a conventional analysis like exploratory testing, an organization may choose to adopt a user's perspective to discover flaws.
- Testing time needed: System testing may execute several tasks and take more time to complete in some contexts, particularly when regular monitoring becomes necessary. You must be aware of the time commitment you can make to testing. It will offer you a realistic notion of the progress and assist you in arranging your work.

An organization may highlight system tests that require minimal steps if its product release deadlines are shorter, or it may adjust screening procedures to accommodate its specifications.

- Resources at your disposal for testing: As previously said, when making plans for System testing, you must consider your test team's size, expertise, and experience. You may need to train your current personnel or hire extra testers based on the size and complexity of your application.
- Background of tester: Planning for System testing should consider the testers' experience. The test cycle could take longer to complete if the testers are inexperienced. However, if they had previous experience, it would take less time.
- Entire testing expense: When developing your test strategy, it's crucial to keep the entire cost of testing in mind. System testing may be a costly procedure. The size and complexity of the system, the number of test cases needed, and the time and resources required to carry out the tests are just a few of the numerous variables that might affect the cost of System testing.

An example of a System test case

This section of the System testing tutorial will help you to understand how System testing works with an example.

- Firstly, a tester must check if the website loads smoothly, includes all the necessary functions, pages, and logos, and if a user can access the site and log in. The user must be able to add items to his cart, complete the purchase, and receive a confirmation via email, SMS, or phone call if he can see the available products.
- Next, the tester must check that the site can simultaneously accommodate the number of users (as specified in the requirement specification). Further, the primary features, such as searching, filtering, sorting, adding, altering, wishlist, etc., must function as intended.
- A tester should check if the website functions well across all platforms, including Windows, Linux, mobile, etc., loads in all popular browsers, and its recent versions with the text on the pages correctly spaced, organized, and free of typos. Further, the tester also verifies if the session timeout is working as expected.
- Lastly, the tester must verify that the user manual or guide, return policy, privacy policy, and terms of service are all provided as separate documents. The ability to download the same by new or first-time users would help ensure that the transactions made on the website through a particular user are safe. The end-user must be pleased with the website after using it and should face no hiccups.

Types of System Testing

1. **Performance Testing**
2. **Load Testing**
3. **Usability Testing**
4. **Regression Testing**
5. **Migration Testing**
6. **Functional Testing**
7. **Recovery Testing**
8. **Stress Testing**
9. **Software & Hardware Testing**

Types of System Testing

1. Performance Testing:

- It measures Speed, Load time, Stability, Reliability & Response times of the system under various conditions.

2. Load Testing:

- It determine how system or software performs under a real-life extreme load.
- Such as Throughput, Number of users etc.

3. Usability Testing:

- It evaluate system is easy to use and functional for the end user.
- Such as User error rates, Task success rates, Time takes a user to complete a task & User satisfaction etc.

Types of System Testing

4. Regression Testing:

- It ensure that any changes done during the development process have not introduced a new defects or bugs.
- Also ensure that old defects or bugs will not exist on the addition of new software over the time.

5. Migration Testing:

- It ensure that if system needs to be modified in new infrastructure so it should be modified without any issue.

6. Functional Testing:

- Tester find out any missing function in the system & make list of it.
- It can be added during functional testing and should improve quality of the system.

Types of System Testing

7. Recovery Testing:

- It ensure that system is capable of recovering from certain system errors, crashes and failures.

8. Stress Testing:

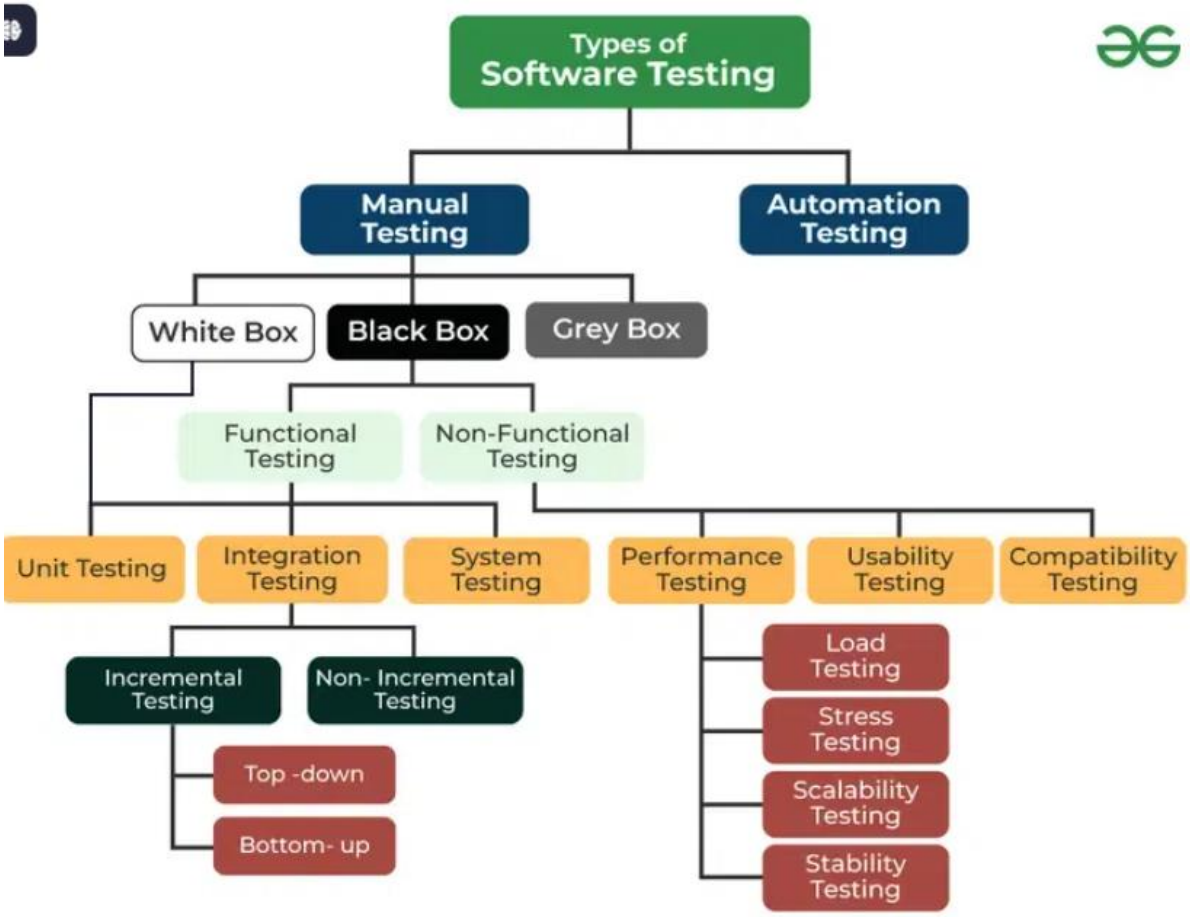
- It ensure that robustness of the system under the various loads & capacities.

9. Software & Hardware Testing:

- This testing of the system check hardware and software compatibility & interaction.
- The hardware configuration must be compatible with the software to run it without any issue.

Types of System Testing

Here are the Types of System Testing are follows:



Types of testing

- Functional Testing: This checks if the system's features work as expected and meet the defined requirements.
- Performance Testing: This tests how the system performs under different conditions, like high traffic or heavy use, to ensure it can handle the expected load.
- Security Testing: This ensures the system's security measures protect sensitive data from unauthorized access or attacks.
- Compatibility Testing: This makes sure the system works well across different hardware, software, and network environments.
- Usability Testing: This evaluates how easy and user-friendly the system is, making sure it provides a good experience for users.
- Regression Testing: This ensures that any new code or features don't break or negatively affect the system's existing functionality.

- Acceptance Testing: This tests the system at a high level to make sure it meets customer expectations and requirements before release.

🔹 1. Based on Testing Method

a. Manual Testing

- Testing is done by human testers without automation tools.
- Used for exploratory testing, usability, and ad-hoc scenarios.
- Slower but better at finding UI/UX issues.

b. Automated Testing

- Uses tools like Selenium, JUnit, TestNG, etc.
- Scripts run automatically to validate features.
- Ideal for regression testing and large-scale projects.

🔹 2. Based on Testing Level

a. Unit Testing

- Tests individual units/components of code (e.g., functions, classes).
- Done by developers using tools like JUnit (Java), pytest (Python).
- Helps catch bugs early.

b. Integration Testing

- Tests how modules interact with each other.
- Example: checking how the login module connects to the database.
- Approaches: Top-down, Bottom-up, Big Bang.

c. System Testing

- Tests the entire integrated software system as a whole.
- Checks end-to-end functionality.

d. Acceptance Testing

- Verifies if the system meets business requirements.

- Done by the client/user.
- Types:
  - **Alpha Testing**: Done by internal team.
  - **Beta Testing**: Done by real users in a live environment.

🔹 3. Based on Execution Time

a. Static Testing

- Done without executing the code.
- Includes code reviews, walkthroughs, and static analysis.

b. Dynamic Testing

- Code is executed to check functional behavior.
- Includes unit, integration, system, and acceptance testing.

🔹 4. Based on Test Purpose

a. Functional Testing

- Verifies if the application works as per the functional requirements.
- Includes:
  - Smoke Testing
  - Sanity Testing
  - Regression Testing
  - Usability Testing

b. Non-Functional Testing

- Tests non-functional aspects like performance, usability, security.
- Includes:
  - Performance Testing
  - Load Testing
  - Stress Testing

- Scalability Testing
- Security Testing
- Compatibility Testing

🔹 5. Other Important Types

a. Smoke Testing

- A quick check to see if the build is stable.
- Called "Build Verification Testing".

b. Sanity Testing

- Focuses on one or few areas after a bug fix or change.

c. Regression Testing

- Re-tests the application after changes to ensure existing functionality isn't broken.

d. Exploratory Testing

- Tester explores the system freely without predefined test cases.
- Often uncovers hidden bugs.

e. Ad-hoc Testing

- Informal testing without planning or documentation.
- Based on tester's intuition.

f. Compatibility Testing

- Checks how software works in different environments, devices, browsers.

g. Security Testing

- Ensures data and resources are protected from breaches.
- Includes penetration testing, vulnerability scanning.

h. Performance Testing

- Measures how the system behaves under load.
  - **Load Testing**: Under expected load.
  - **Stress Testing**: Under extreme load.

- **Spike Testing**: Sudden large increase in load.
- **Soak Testing**: Over extended periods.

i. Usability Testing

- Measures user-friendliness and ease of use.

j. Accessibility Testing

- Ensures the application is usable by people with disabilities.

k. Localization & Internationalization Testing

- **Localization**: Verifies content in specific languages and regions.
- **Internationalization**: Ensures support for multiple languages and regions.

🔹 6. Based on Test Approach

a. Black Box Testing

- Tester doesn't know internal code structure.
- Focuses on input-output behavior.

b. White Box Testing

- Tester knows the internal logic of the code.
- Tests paths, conditions, loops.

c. Gray Box Testing

- Partial knowledge of internal workings.
- Combines both black and white box approaches.

🔹 7. Mobile and Web Specific Testing

- **Responsive Testing**: Layout adjusts to screen size.
- **Cross-Browser Testing**: Works across different web browsers.
- **Interrupt Testing**: For mobile (e.g., incoming calls, SMS interruptions).
- **App Store Testing**: Behavior post-publishing on Google Play/App Store.

System Testing Example

Test Case Type	Description	Test Step	Expected Result	Status
Functionality	Area should accommodate up to 20 characters	Input up to 20 characters	All 20 characters in the request should be appropriate	Pass or Fail
Security	Verify password rules are working	Create a new password in accordance with rules	The user's password will be accepted if it adheres to the rules	Pass or Fail
Usability	Ensure all links are working properly	Have users click on various links on the page	Links will take users to another web page according to the on-page URL	Pass or Fail

Important Interview Topics

Question	What to Say
What is system testing?	"Testing the complete integrated system to ensure it meets requirements."
Black-box vs white-box?	"Black-box = no code visibility, only inputs & outputs. System testing is black-box."
What comes before system testing?	"Integration testing"
What comes after?	"Acceptance testing"
What tools do you know?	"Selenium for automation, JMeter for load testing, and Postman for API testing." *(Even if you haven't used them, knowing names helps.)*
How do you write test cases?	"By studying requirements and writing step-by-step scenarios with expected results."
What if a bug is found?	"Log it in a defect tracking tool like JIRA and assign it to the developer."

System Testing Process

1. Requirement Analysis

- Review software requirements, business specifications, and system architecture.
- Identify functional and non-functional requirements.
- Understand expected system behavior from an end-user perspective.

2. Test Planning

- Define scope, objectives, and types of system tests to be conducted.
- Identify resources (hardware, software, personnel).
- Prepare a System Test Plan including:
    - Test strategy
    - Test objectives
    - Deliverables
    - Risk assessment
    - Schedule

3. Test Case Design

- Write detailed test cases and test scenarios covering:
    - Functional testing
    - Performance testing
    - Security testing
    - Usability testing
    - Compatibility testing
- Prepare test data (valid, invalid, boundary cases, etc.)
- Trace each test case to requirements (RTM – Requirements Traceability Matrix).

4. Test Environment Setup

- Set up the environment that closely resembles the production environment.
- Install the integrated system software.
- Configure servers, databases, and network as needed.
- Ensure all dependent systems and tools are in place.

5. Test Execution

- Execute the test cases as per the test plan.
- Log test results (pass/fail) for each case.
- Report any defects/bugs using a defect tracking tool (e.g., JIRA, Bugzilla).
- Retest after fixes and perform regression testing if required.

6. Defect Reporting & Tracking

- Report defects with details (steps to reproduce, environment, screenshots, logs).
- Work with the development team to verify fixes.
- Maintain a defect log for tracking and reporting.

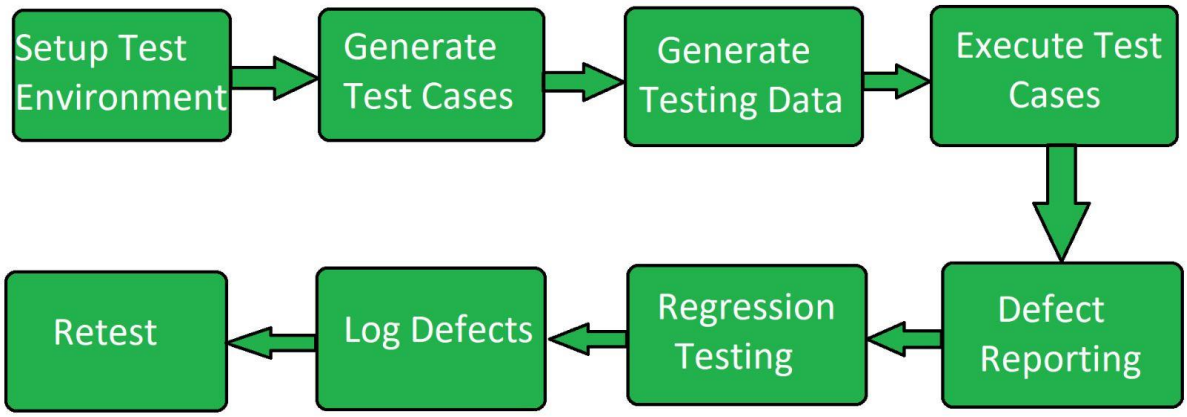
7. Test Closure

- Ensure all test cases have been executed and all defects are resolved or documented.
- Prepare System Test Report including:
    - Summary of testing
    - Coverage
    - Defects and resolutions
    - Lessons learned
- Conduct a Test Closure Meeting with stakeholders.

System Testing Process

System Testing is performed in the following steps:

- Test Environment Setup: Create testing environment for the better quality testing.
- Create Test Case: Generate test case for the testing process.
- Create Test Data: Generate the data that is to be tested.
- Execute Test Case: After the generation of the test case and the test data, test cases are executed.
- Defect Reporting: Defects in the system are detected.
- Regression Testing: It is carried out to test the side effects of the testing process.
- Log Defects: Defects are fixed in this step.
- Retest: If the test is not successful then again test is performed.



Tools used for System Testing Here are the Tools used for System Testing are follows:

---

1. JMeter

- Purpose: Performance and load testing.
- Use Case: Simulates heavy load on a server, group of servers, or network to test its strength and analyze performance under different load types.

---

2. Galen Framework

- Purpose: Layout and visual testing.
- Use Case: Verifies UI layout across different screen sizes and resolutions. Especially useful for responsive design testing.

---

3. HP Quality Center / ALM (Application Lifecycle Management)

- Purpose: Test management.
- Use Case: Manages test planning, execution, and defect tracking. Supports full testing lifecycle and integrates with various automation tools.

---

4. IBM Rational Quality Manager

- Purpose: Collaborative test planning and execution.
- Use Case: Integrates with development and quality assurance processes to track test artifacts, executions, and defects in real time.

---

5. Microsoft Test Manager

- Purpose: Manual and automated test management.
- Use Case: Integrated with Visual Studio, it helps teams manage, execute, and track manual and automated tests.

---

6. Selenium

- Purpose: Web automation testing.
- Use Case: Automates browsers across different platforms and programming languages. Widely used for functional and regression testing of web applications.

---

7. Appium

- Purpose: Mobile application testing.
- Use Case: Automates testing for native, hybrid, and mobile web applications on Android and iOS platforms.

---

8. LoadRunner

- Purpose: Load and performance testing.
- Use Case: Simulates virtual users to analyze system behavior and performance under load.

---

9. Gatling

- Purpose: Performance testing.
- Use Case: Especially useful for testing web applications and APIs. Known for high performance and detailed reports.

---

10. Apache JServ

- Purpose: Java Servlet execution (deprecated tool).
- Use Case: Formerly used to run Java Servlets in web applications. Now largely replaced by modern Java application servers like Tomcat.

---

11. SoapUI

- Purpose: API testing (SOAP and REST).
- Use Case: Functional testing, security testing, and load testing of web services. Supports both SOAP and REST APIs.

Comprehensive List of System Testing Tools

Functional Testing Tools

1. Selenium – Web app testing (open-source)
2. TestComplete – Automated UI testing (by SmartBear)
3. Ranorex – GUI testing
4. Katalon Studio – Functional and API testing
5. UFT (Unified Functional Testing) – Formerly QTP, by Micro Focus
6. Watir – Ruby-based web testing
7. Leapwork – No-code automation testing
8. Tricentis Tosca – Model-based test automation
9. Cypress – Modern web app testing for JavaScript apps
10. Robot Framework – Generic automation framework

Performance & Load Testing Tools

1. Apache JMeter – Open-source load testing
2. LoadRunner – By Micro Focus, enterprise-grade
3. Gatling – High-performance load testing
4. BlazeMeter – Cloud-based JMeter testing
5. NeoLoad – Performance testing for web & mobile
6. k6 (by Grafana Labs) – Developer-centric load testing
7. Locust – Python-based load testing
8. WebLOAD – Enterprise load testing
9. Appvance IQ – AI-driven performance testing

- **Test Management Tools**

1. HP ALM / Quality Center
2. TestRail
3. PractiTest
4. Zephyr (by SmartBear)
5. qTest (by Tricentis)
6. Xray – For Jira test management
7. TestLink – Open-source test case management
8. Microsoft Test Manager (MTM)

- **API Testing Tools**

1. Postman – RESTful API testing
2. SoapUI – SOAP and REST API testing
3. ReadyAPI – Enterprise version of SoapUI
4. REST Assured – Java-based library for API testing
5. Karate DSL – API and UI testing
6. Apigee – Google's API testing and monitoring
7. JMeter – Can also be used for API testing

- **Mobile Testing Tools**

1. Appium – Cross-platform mobile app testing
2. Espresso – Android UI testing (by Google)
3. XCUITest – iOS UI testing (by Apple)
4. Detox – React Native mobile testing
5. TestProject – Cloud-based mobile testing
6. Perfecto Mobile – Cloud-based mobile testing
7. Kobiton – Mobile experience platform

- **Security Testing Tools**

1. OWASP ZAP (Zed Attack Proxy) – Open-source security testing
2. Burp Suite – Web app security testing
3. Nessus – Vulnerability scanning
4. Acunetix – Web vulnerability scanner
5. Wapiti – Open-source vulnerability scanner

- **Cross-Browser Testing Tools**

1. BrowserStack – Real device and browser cloud
2. Sauce Labs – Cross-browser testing at scale
3. LambdaTest – Online testing platform
4. CrossBrowserTesting.com – UI testing in real browsers

- **CI/CD & DevOps Integration Tools**

1. Jenkins – Open-source CI/CD
2. GitLab CI/CD
3. Azure DevOps
4. CircleCI
5. TeamCity
6. Travis CI
7. Bamboo (by Atlassian)

- **Visual and Layout Testing Tools**

1. Galen Framework – Layout testing for responsive UIs
2. Applitools Eyes – Visual AI testing
3. Percy – Visual testing by BrowserStack

- **Other Useful Tools**

1. Allure – Test report generation
2. ExtentReports – Reporting for test frameworks
3. TestNG / JUnit / NUnit – Test execution frameworks
4. PyTest – Python testing framework
5. Cucumber – BDD testing tool (Gherkin language)

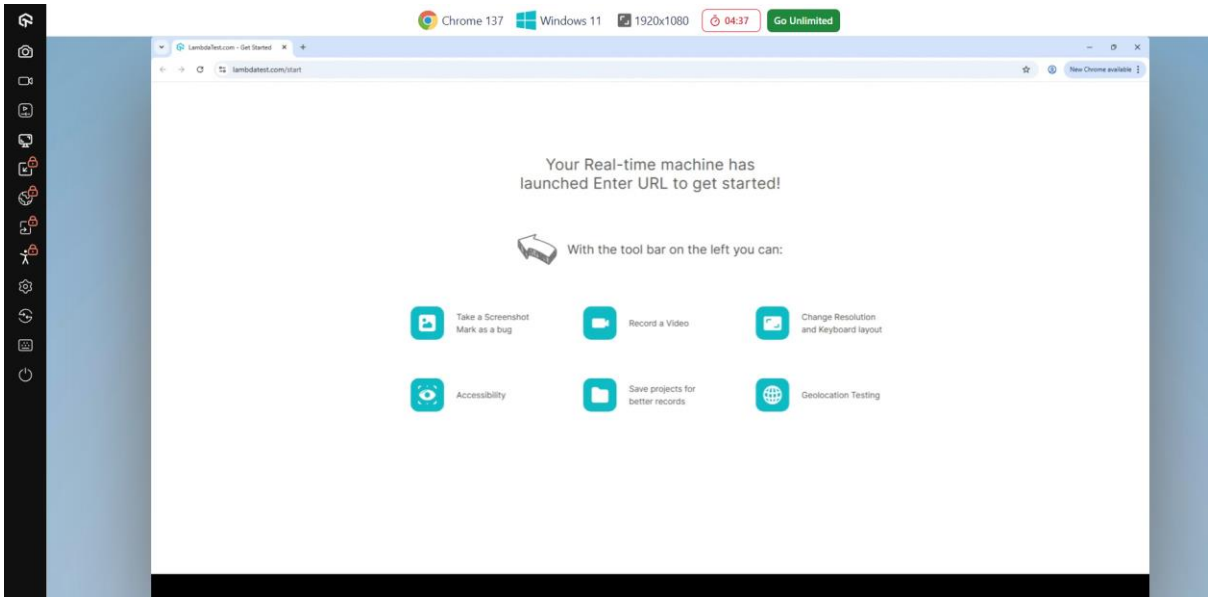
LAMBDA TESTING

What is LambdaTest?

LambdaTest is a **cloud-based platform** that lets you perform:

- **Cross-browser testing**
- **Responsive UI testing**
- **Automation testing using Selenium, Cypress**
- **Visual regression testing**
- **Mobile app testing (real device and emulators)**

It saves time by allowing you to test across multiple OS/browser combinations **without setting up local environments.**

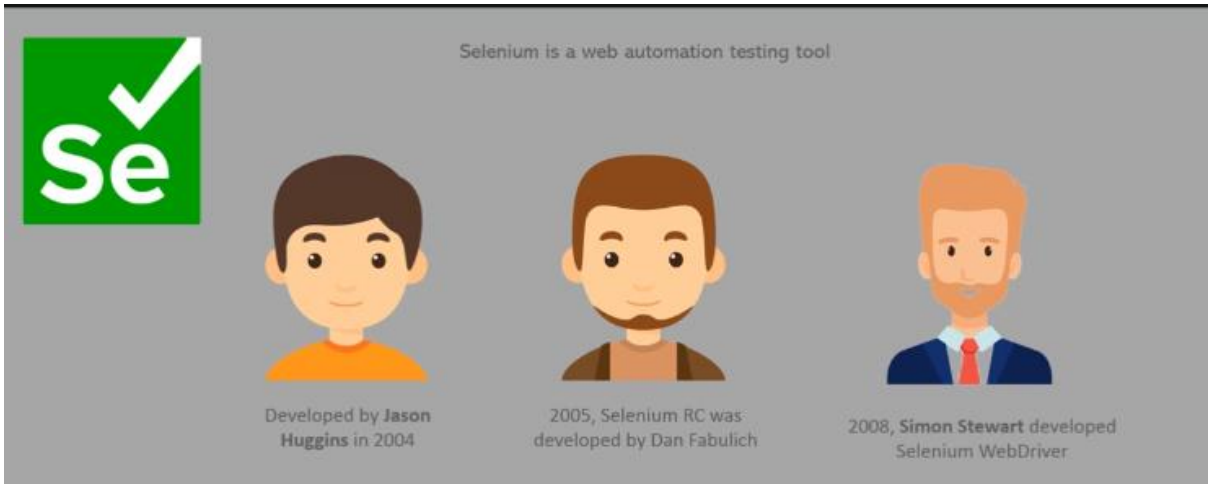


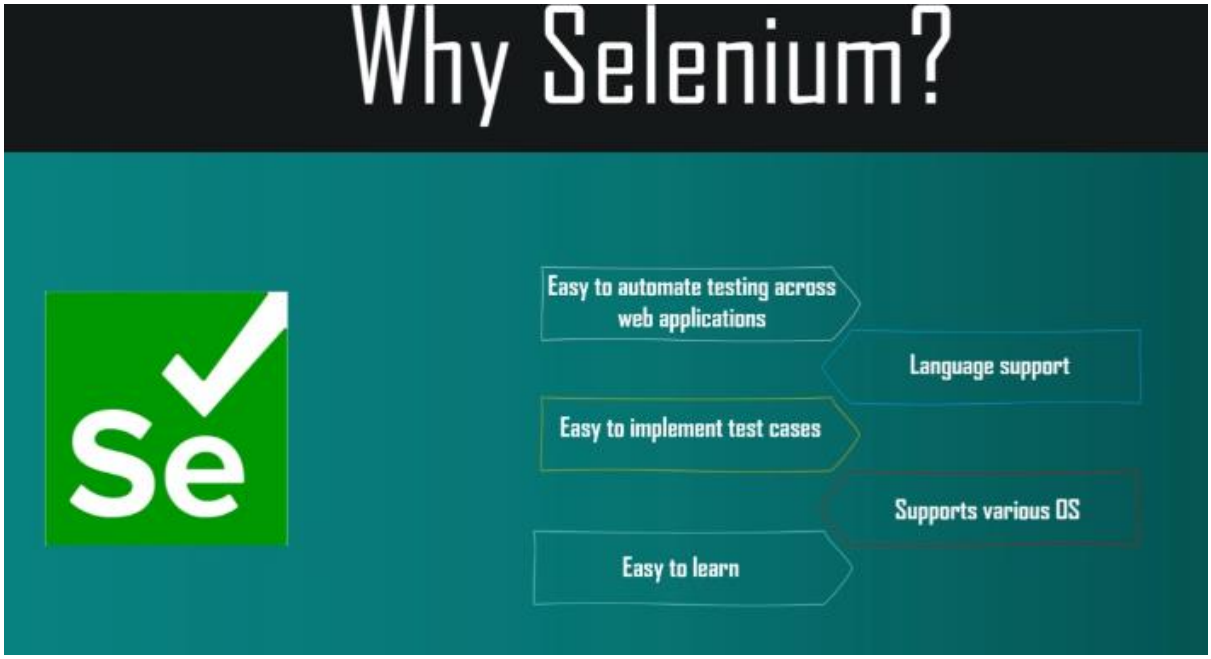
this is lambda testing of its platfrom , we can test app or real time browser here , it's a cloud testing basis

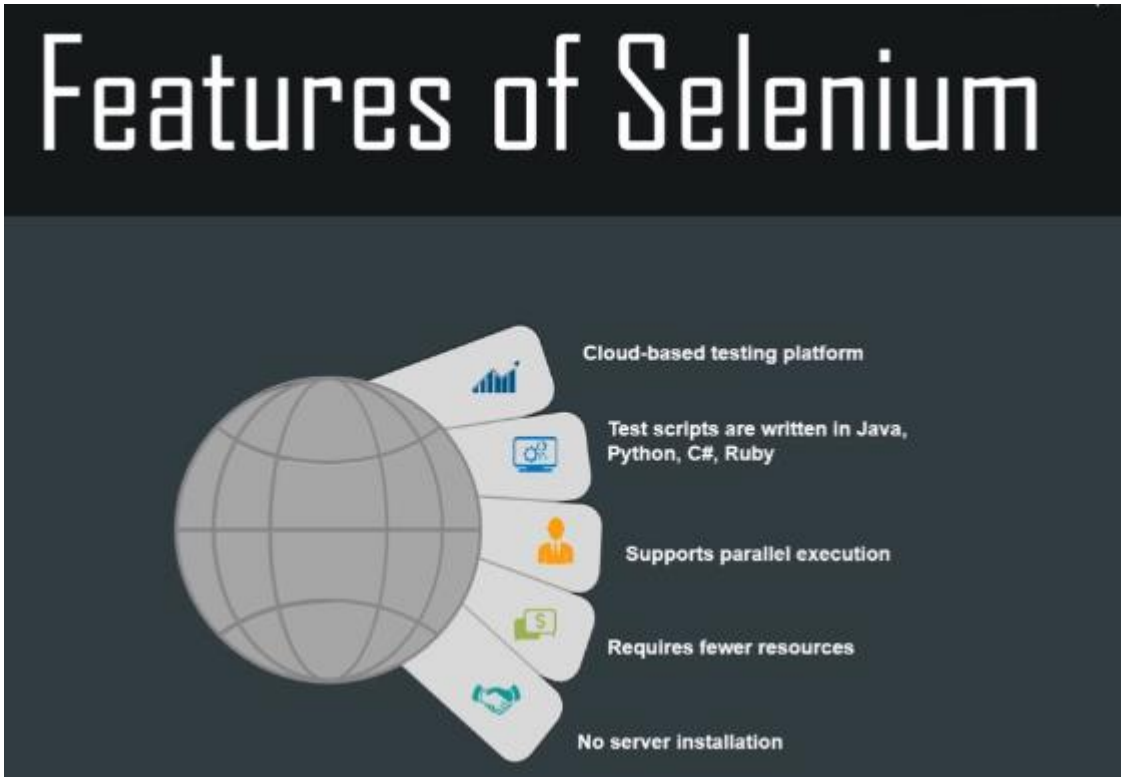
"Why LambdaTest or what's its benefit?", you can say:

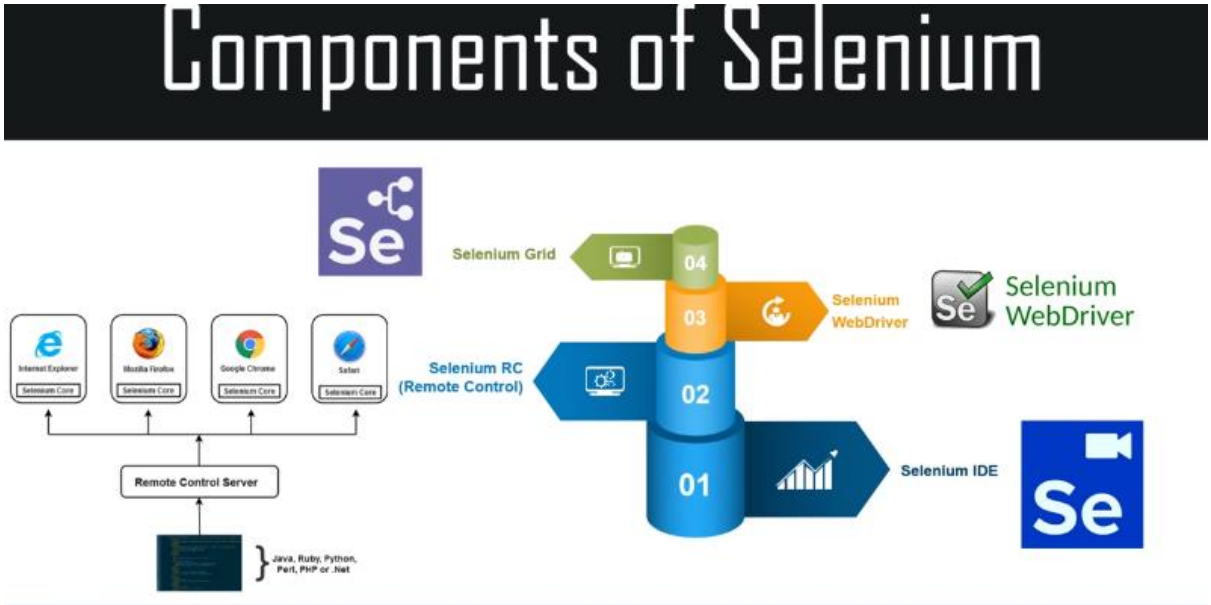
"LambdaTest reduces the time and cost of maintaining local test environments by offering real devices and browsers on the cloud. It enables fast, scalable cross-browser and automation testing, improving release quality and speed. It's especially useful in Agile teams with frequent deployments."

SELENIUM


Selenium is a web automation testing tool
Developed by Jason Huggins in 2004
2005, Selenium RC was developed by Dan Fabulich
2008, Simon Stewart developed Selenium WebDriver


Why Selenium?
Easy to automate testing across web applications
Language support
Easy to implement test cases
Supports various OS
Easy to learn


Features of Selenium
Cloud-based testing platform
Test scripts are written in Java, Python, C#, Ruby
Supports parallel execution
Requires fewer resources
No server installation


Components of Selenium
Selenium Grid
04
Selenium WebDriver
03
Selenium WebDriver
Selenium RC (Remote Control)
02
01
Selenium IDE
Remote Control Server
Java, Ruby, Python, Perl, PHP or .Net

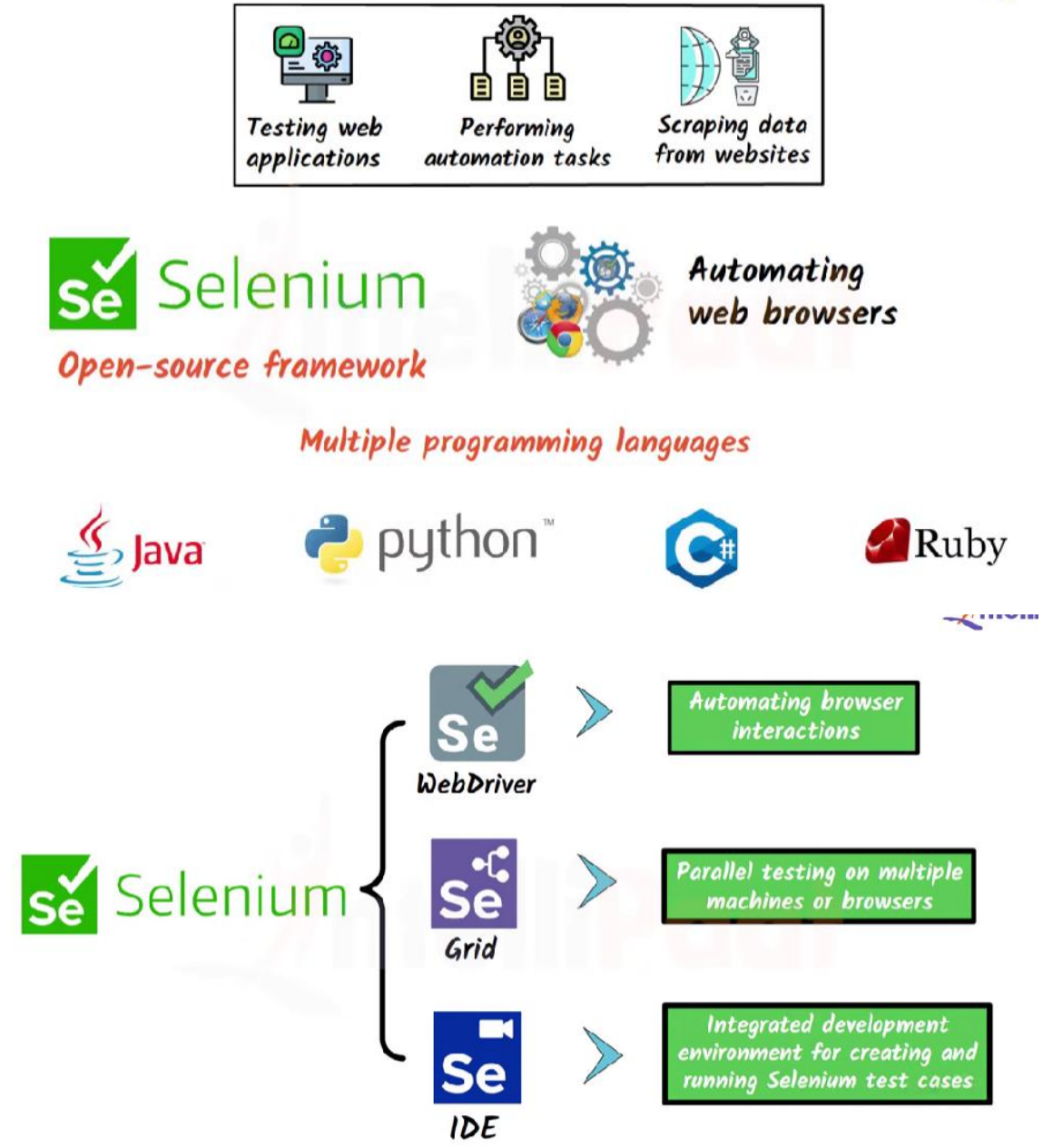
What is Selenium?

Selenium is a popular open-source automation testing tool that automates web applications (like opening browsers, clicking buttons, logging in, etc.).

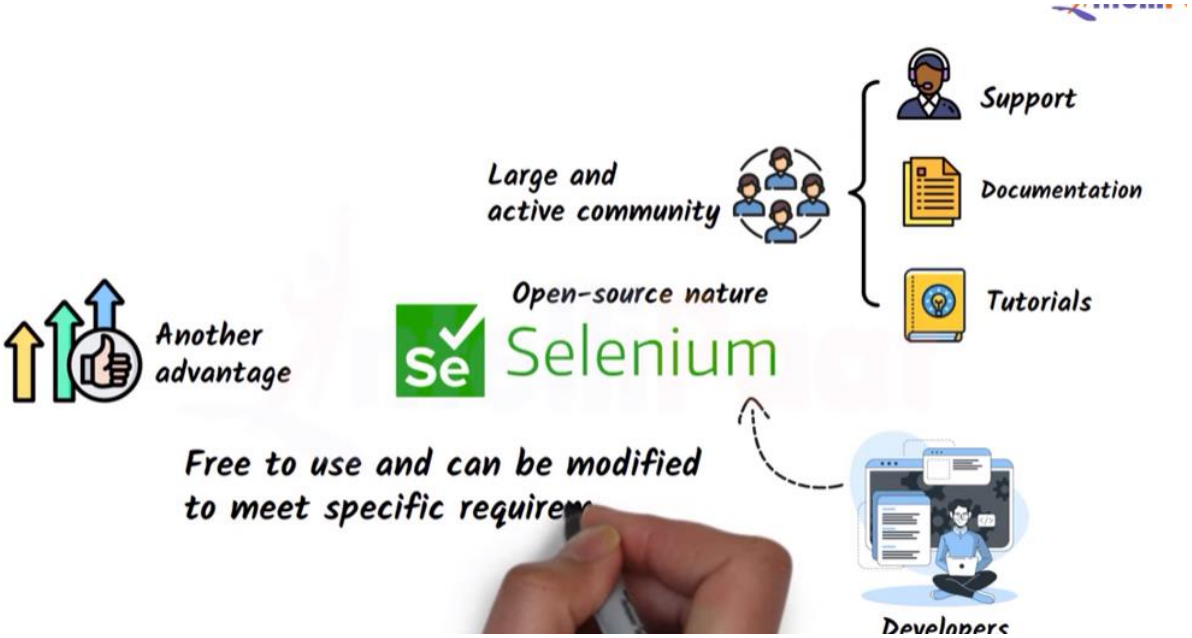
It works with multiple languages – Python, Java, C#, etc. Since you know Python, we'll focus on that.

🧩 Main Components of Selenium

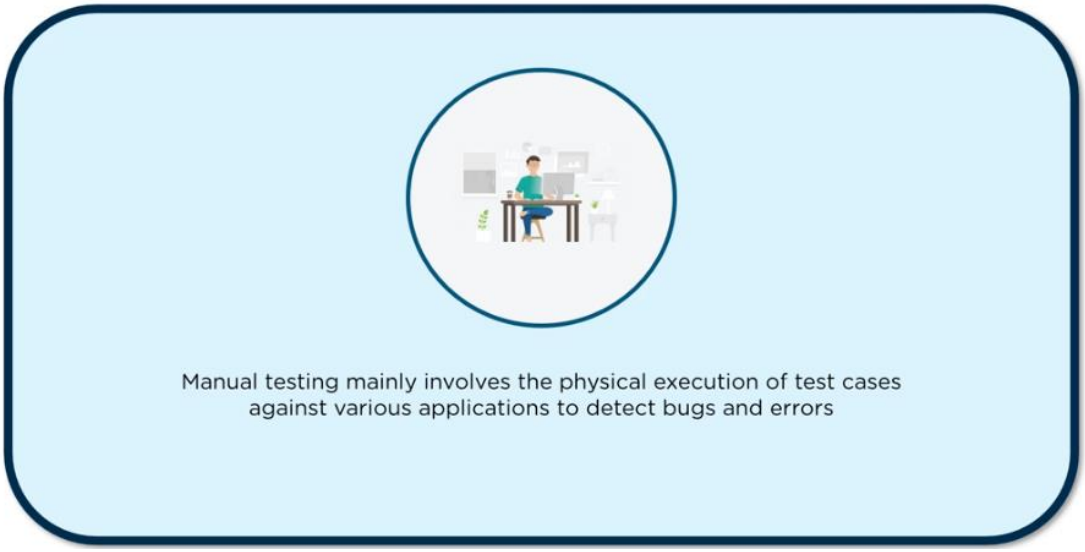
Component	Description	Use
1. Selenium IDE	A browser plugin (Chrome/Firefox) that records user actions and plays them back.	For beginners to record and replay simple test cases.
2. Selenium WebDriver	Core component. Allows you to write custom code (in Python/Java) to automate browser actions.	Real automation testing with full control over the browser.
3. Selenium Grid	Tool to run your tests on multiple machines and browsers in parallel.	Used in large-scale testing (cross-browser, cross-OS).
4. Selenium RC (Deprecated)	Old version, replaced by WebDriver.	❌ Not used anymore. Ignore.


Testing web applications
Performing automation tasks
Scraping data from websites
Selenium
Open-source framework
Automating web browsers
Multiple programming languages
Java
python
Ruby
Selenium
WebDriver
Automating browser interactions
Grid
Parallel testing on multiple machines or browsers
IDE
Integrated development environment for creating and running Selenium test cases


Test web applications on different platforms
Multiple browsers
Issues
Bugs
Selenium
Key advantage
Real-world user interactions
Web applications
Compatibility
Risk of issues
End-users


Support
Large and active community
Documentation
Tutorials
Another advantage
Open-source nature
Selenium
Free to use and can be modified to meet specific

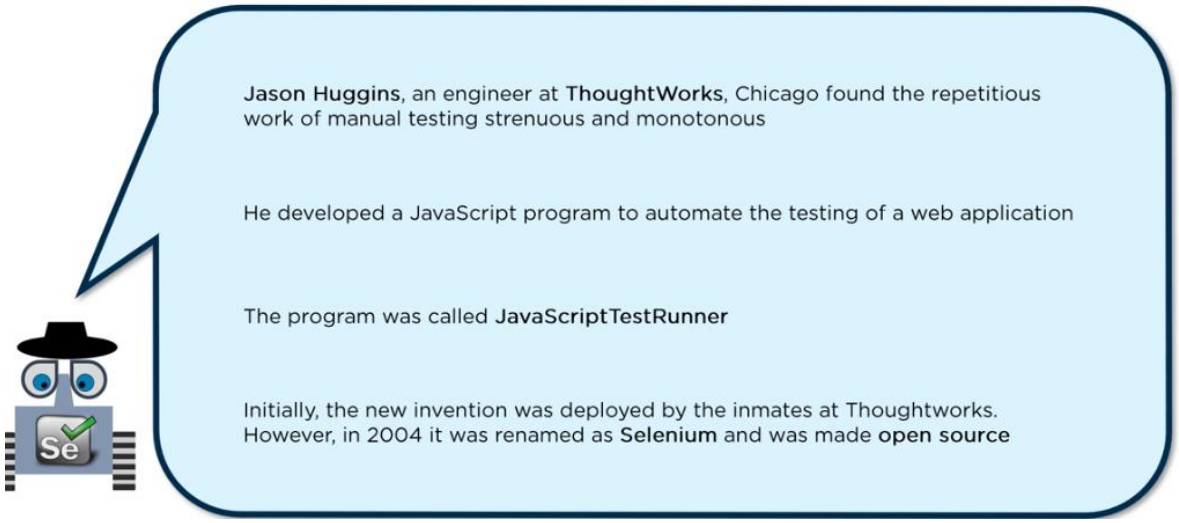
Manual Testing



Manual Testing

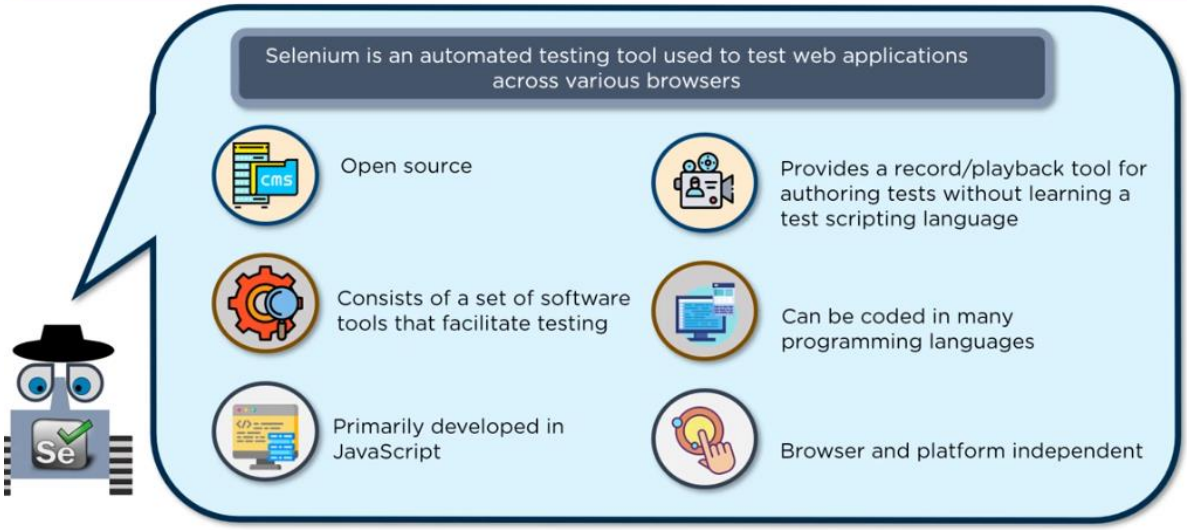
Limitations of Manual Testing

Birth of Selenium



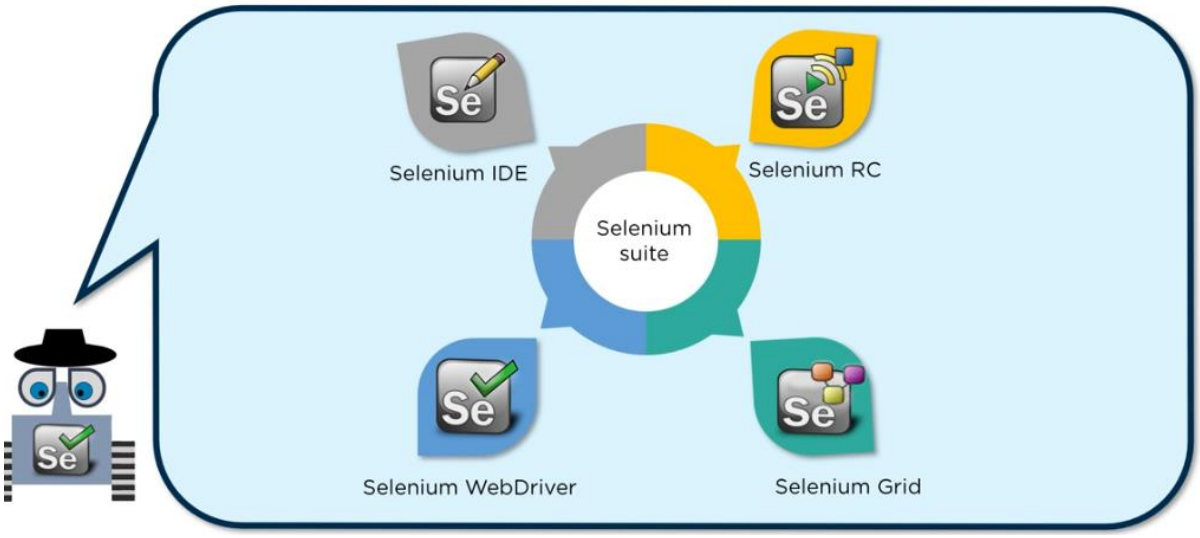
How does Selenium help?

What is Selenium?



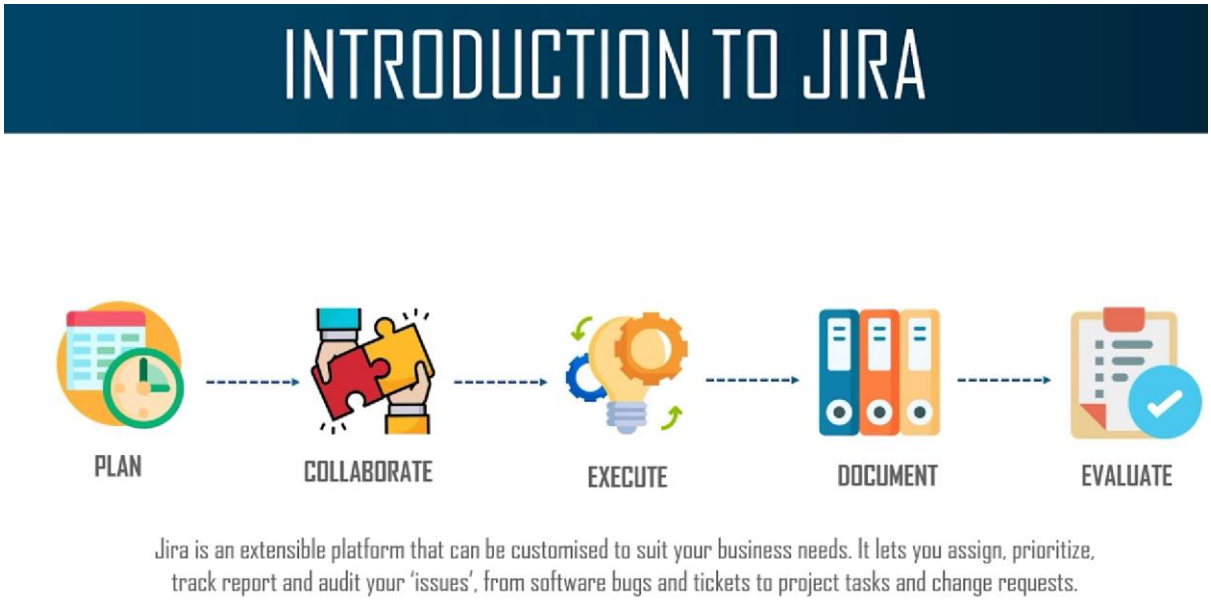
What is Selenium?

Selenium suite of tools



Ide mainly hai prototyping k liye
rc is remote control , jo ki web driver se replace ho jaata hai
web driver interects with browser directly lekin rc m server ko interect krna hota tha for test case generation and programming interface
selenium grid is used for parallel execution (distributed environment m test case run kr ske)

JIRA SOFTWARE

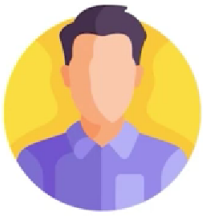


PRODUCT OWNER

Understand Product Requirements
Identifies Objectives for Project
Balances Stakeholder Needs
Maintains Product Backlog
Manages Releases

SCRUM MASTER

Responsible for Coordination
Intermediary Between Product Owner and Dev Team
Helps in Planning and Breaking Down Work
Manages Backlog and Ensures Completion
Provides Transparency



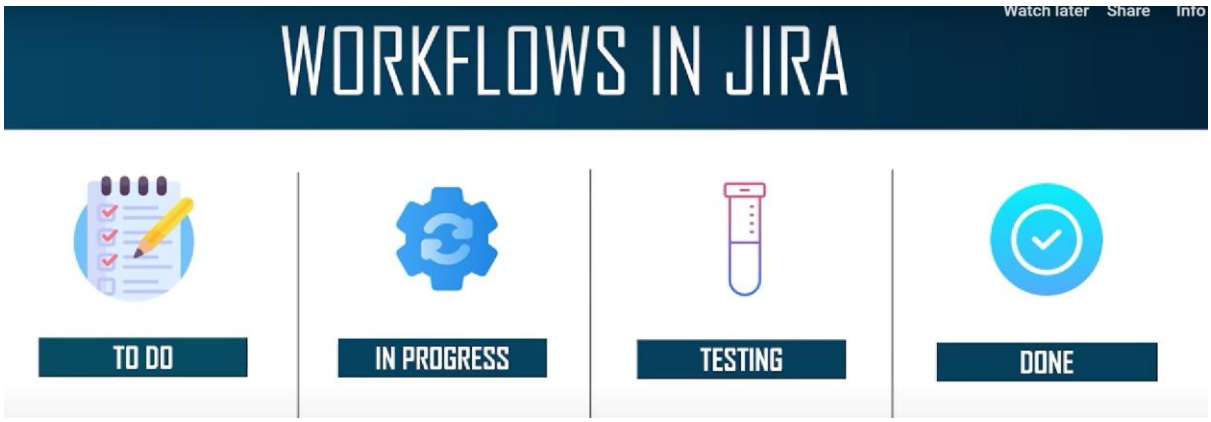
SCRUM TEAM

Consists of Developers and QA Team
Breaks Down Work into Subtasks
Deals with Planning, Implementation, Testing etc.
Ensures Timely Delivery while Adhering to Quality Guidelines
Coordinates with the Scrum Master

WORKFLOWS IN JIRA

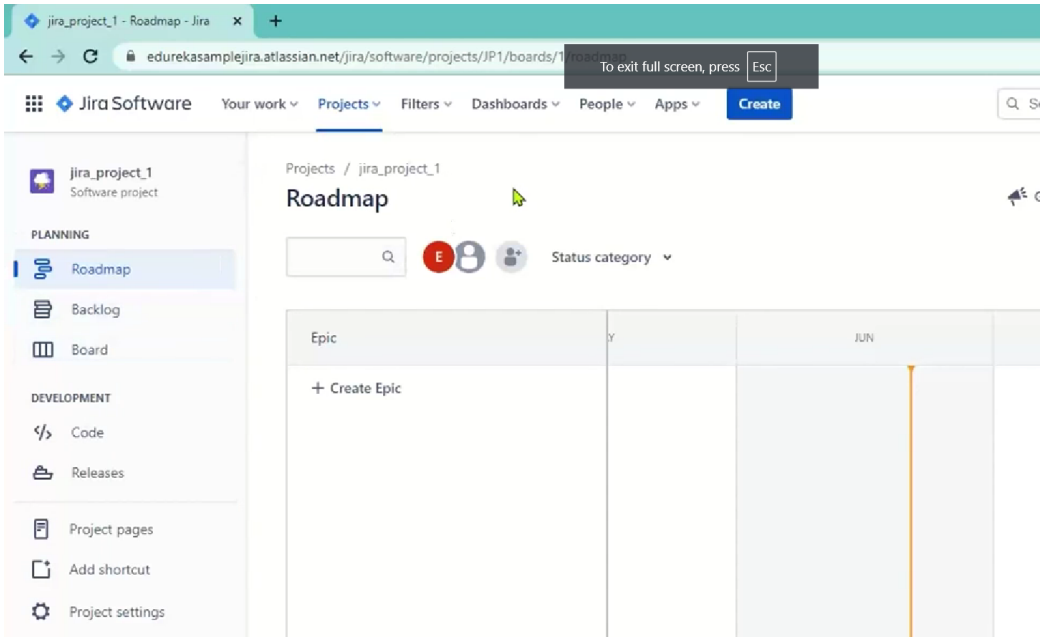
Definition:

A Jira workflow is a set of **statuses and transitions** that an issue moves through during its lifecycle, and typically **represents a process within your organization**

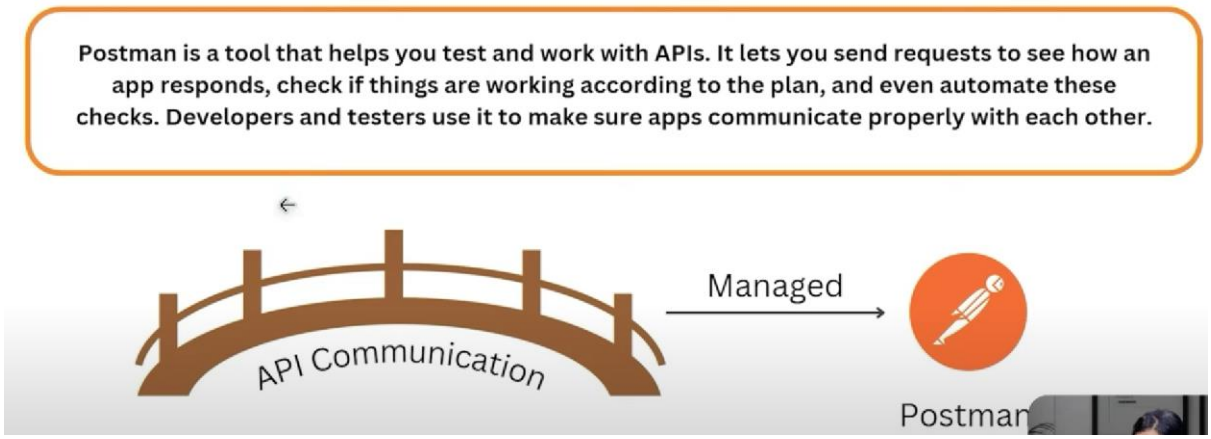


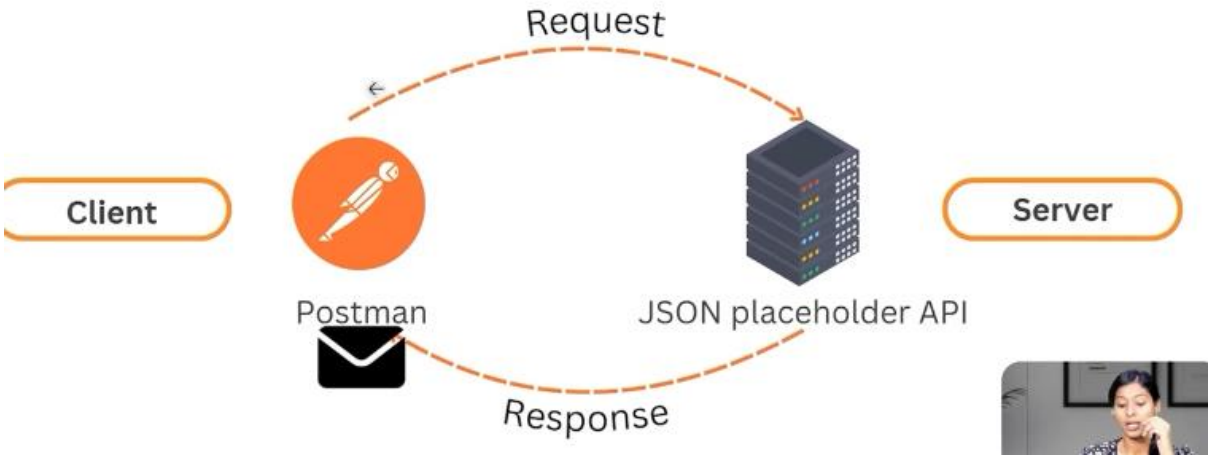
EPIC: WISHLIST

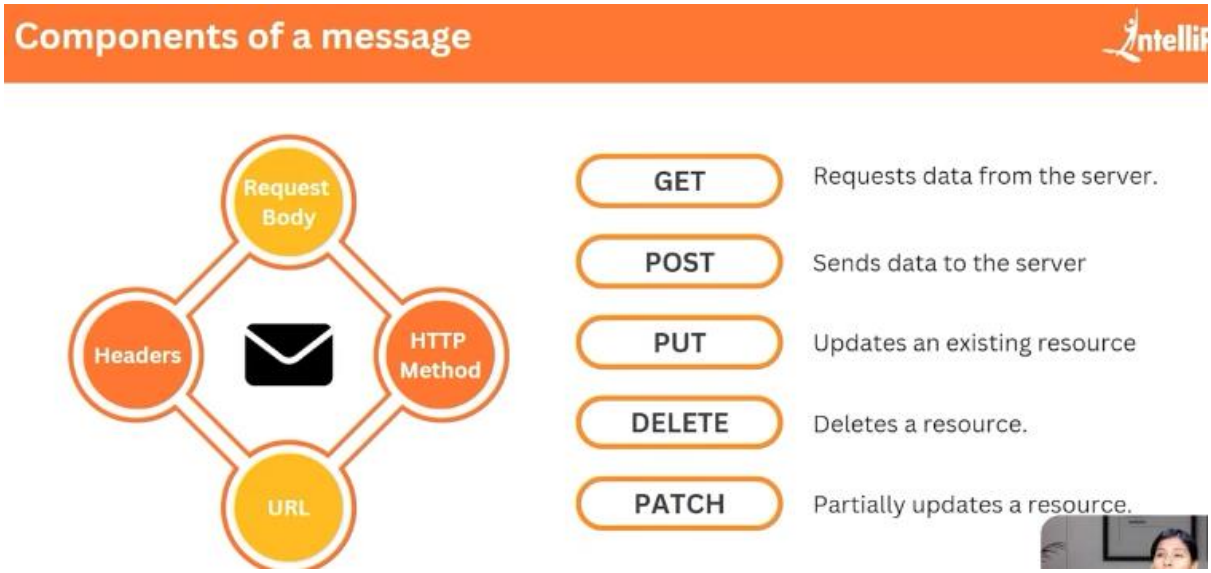
- STORIES:
  - Create Wishlist
  - View Wishlist
- TASKS:
  - Create Wishlist
    - Create Wishlist Button to Add Products to Wishlist
    - Create Wishlist Database
  - View Wishlist
    - Create View Wishlist Button to View Saved Items
    - Create Wishlist Page to View Saved Items

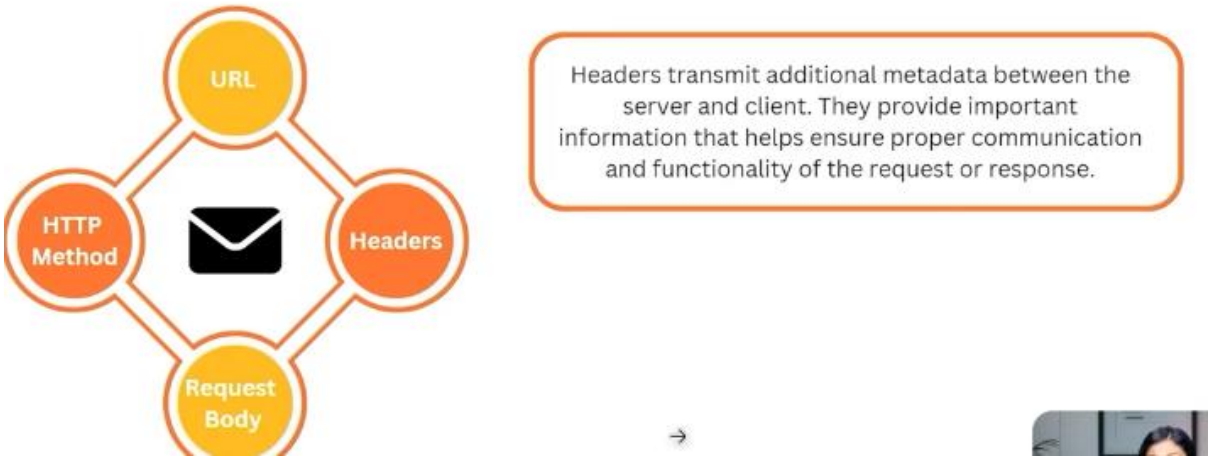


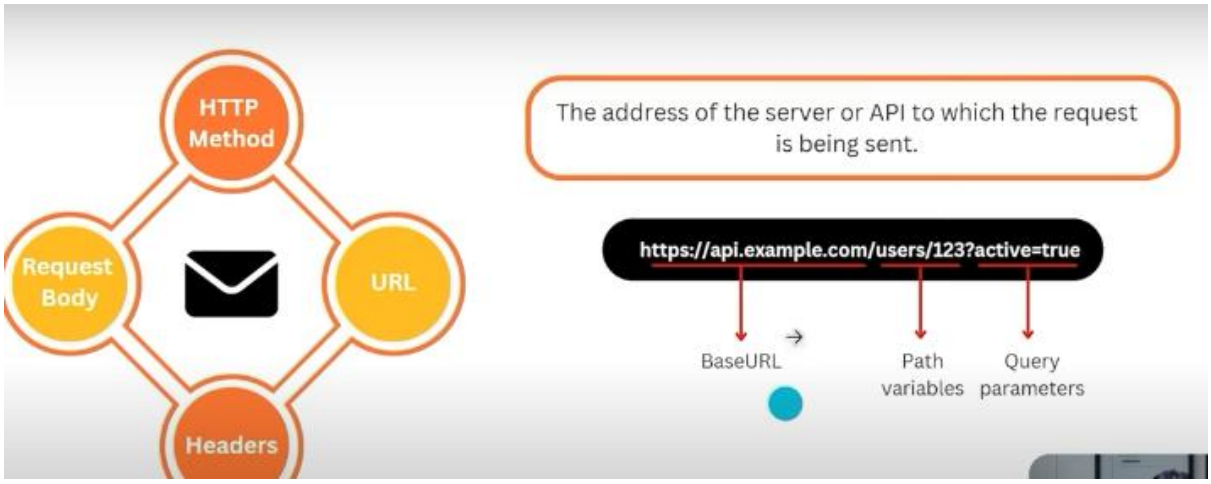
POSTMAN API


Postman is a tool that helps you test and work with APIs. It lets you send requests to see how an app responds, check if things are working according to the plan, and even automate these checks. Developers and testers use it to make sure apps communicate properly with each other.
API Communication
Managed
Postman

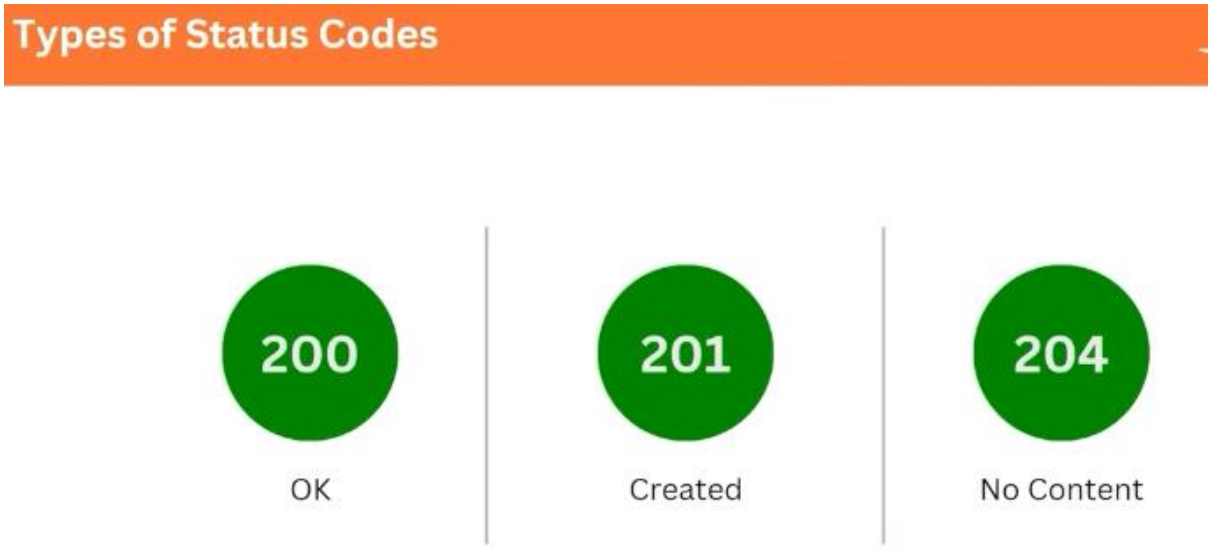

Request
Client
Server
Postman
JSON placeholder API
Response



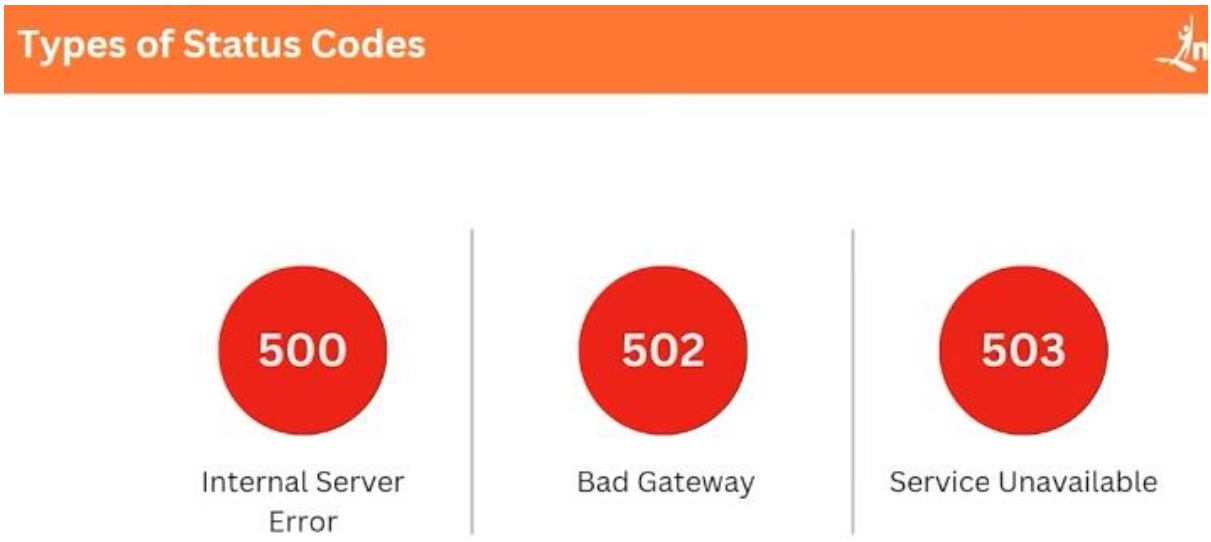




Request body is for when we want any new thing to add or update it to something than we have to mention the details of what we are giving from our side


Types of Status Codes
200
OK
201
Created
204
No Content


Types of Status Codes
400
Bad Request
401
Unauthorized
403
Forbidden
404
Not Found
405
Method Not Allowed
409
Conflict


Types of Status Codes
500
Internal Server Error
502
Bad Gateway
503
Service Unavailable

Questionaries

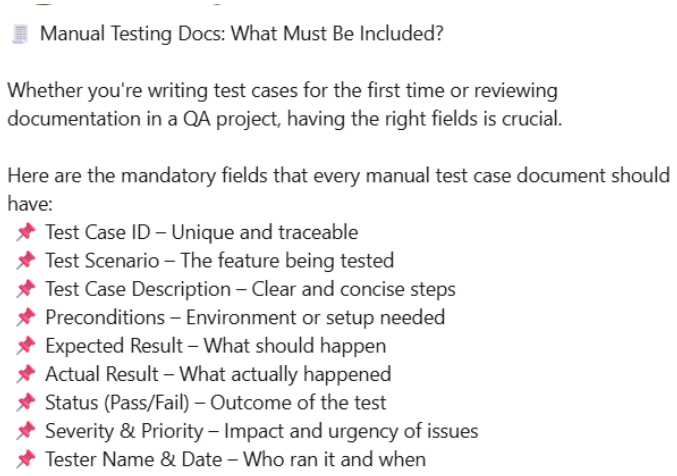
1. What is embedded system testing?

Embedded system testing involves the evaluation of both software and hardware components to ensure the absence of defects in the final product. It verifies if the embedded hardware and software meet the business requirements, encompassing functional and non-functional aspects of the system.

2. What is a Test Case?

Test Case is a detailed document that shows how to check if a part of the software works correctly. It lists the steps to follow, the information needed, and what the result should be. Test cases help make sure the software does what it's supposed to by providing clear instructions. It includes:

- *Test Case ID: A unique name or number to identify the test case.*
- *Title/Description: What the test case is meant to check.*
- *Preconditions: What needs to be set up before running the test.*
- *Test Steps: The actions to perform during the test.*
- *Test Data: The specific information needed for the test.*
- *Expected Result: What should happen if the software is working right.*
- *Actual Result: What actually happens when the test is run.*
- *Status: Whether the test passed or failed.*
- *Postconditions: What should be true after the test.*


Manual Testing Docs: What Must Be Included?

Whether you're writing test cases for the first time or reviewing documentation in a QA project, having the right fields is crucial.

Here are the mandatory fields that every manual test case document should have:

- 📌 Test Case ID – Unique and traceable
- 📌 Test Scenario – The feature being tested
- 📌 Test Case Description – Clear and concise steps
- 📌 Preconditions – Environment or setup needed
- 📌 Expected Result – What should happen
- 📌 Actual Result – What actually happened
- 📌 Status (Pass/Fail) – Outcome of the test
- 📌 Severity & Priority – Impact and urgency of issues
- 📌 Tester Name & Date – Who ran it and when

3. What is a test plan, and what does it include?

Test Plan is a document that consists of all future testing-related activities. It is prepared at the project level and in general, it defines work products to be tested, how they will be tested, and test type distribution among the testers. Before starting testing there will be a test manager who will be preparing a test plan.

Test Plan includes test objectives, scope, test types, tools, and timelines.

4. What is Quality Assurance(QA)?

Software Quality Assurance is a process that works parallel to Software Development. It focuses on improving the process of development of software so that problems can be prevented before they become major issues. Software Quality Assurance is a kind of Umbrella activity that is applied throughout the software process.

The five major quality assurance functions are-

- **Technology Transfer**: This involves getting the product design document as well as trial and error data and its evaluation.
- **Documentation**: This function controls the distribution and archiving of the documents. For incorporating any change in the document, a proper change control procedure needs to be adopted.
- **Validation**: In this function, the validation master plan for the system, and approval of test criteria for validating the product and process are also set.
- **Assuring Quality Products**: This function aims at checking and assuring the quality of the software product.
- **Quality Improvement Plans**: In this function, quality improvement plans are generated to check whether the software product meets the requirements.

4. What is a traceability matrix?

A traceability matrix also known as Requirement Traceability Matrix(RTM) is a document that is used in the development of the software application to trace

the requirements proposed by the client to the system being built. Some of the features of RTM are-

- It is prepared before test execution to make sure that all the requirements are covered in the test cases.
- The test engineers will prepare RTM for the respective assigned modules.
- These will be submitted to the Test Lead.
- The Test Lead will verify the specific module RTM to see if all the requirements of that module are mapped to corresponding test cases and in the end, will consolidate all the RTMs and prepare one necessary RTM document.
- The main purpose of RTM is to check that all requirements are checked via test cases.

5. What is the Severity and Priority in Software Testing?

Severity is defined as the extent to which a particular defect can create an impact on the software. Severity is a parameter to denote the implication and the impact of the defect on the functionality of the software. and Priority is defined as a parameter that decides the order in which a defect should be fixed. Defects having a higher priority should be fixed first.

6. What is a User Story?

The requirements in the development of software and products constantly change and according to those changes we must implement proper features here we use the concept of user story.

Some of the features of the user story are-

- It describes the type of user, what the user wants, and why.
- It helps to create a simplified description of a requirement.
- They are often recorded on index cards, on Post-it notes, or in project management software.
- User stories may be written by various stakeholders such as clients, users, managers, or development team members.
- They provide a user-focused framework for daily work.

7. What is a Test Environment?

The test environment is the hardware and software set up for the testing teams to run test cases. This test environment setup varies from product to product and its configuration completely depends on the application under test requirement. The easiest way to organize a test environment is through automation.

- The test environment is used by the testing teams to test the software, identify the bugs, and find a possible fix for the bugs.
- A test environment is used to check the quality of the software and the impact of the application before release.

8. What is Cookie Testing?

Cookie Testing is the Type of Software Testing that checks the cookie created in the web browser.. A cookie stores user information that can be used to track users' website navigation and can be used to communicate between different web pages. It is very important to keep a check on the cookies, and how they are written and saved in the system to avoid any security threats.

9. What is Bottom-up Testing?

Bottom-up Testing is a type of integration testing that tests the lowest components of a code base first. In these, low-level modules are tested first, and then the high-level modules are tested. The driver is a temporary module that is used to simulate the caller module for a module to be tested, to call the interface procedures of the module to be tested, and to report the results.

What is Dynamic Testing?

Dynamic testing is a software testing technique where the dynamic behavior of the code is checked. The purpose of this type of testing is to check and analyze the software behavior with dynamic variables and find the weak areas in the software runtime environment.

- In this type of testing, the software must be compiled and run.
- Giving the input values and checking if the output is as expected by executing specific test cases.

10. What is Risk-Based Testing?

In Risk-Based Testing, we use the risk items identified during risk analysis, along with the level of risk associated with each risk item to guide the testing. It is a type of software testing technique that is primarily based on the probability of the risk. Risk-based testing involves the following steps:

1. Accessing the risk based on software quality.
2. Frequency of use.
3. Criticality of Business.
4. Possible areas with defects, etc.

11. What is Fuzz Testing?

Fuzz Testing is a Software Testing technique that uses invalid, unexpected, or random data as input and then checks for exceptions such as crashes and potential memory leaks. It is an automated testing technique that is performed to describe the system testing processes involving randomized or distributed approaches.

Objectives:

- To check the vulnerability of the system or software application.
- To detect security faults and defects.
- To determine the defects in effective cost.

12. What is Test Harness?

Test Harness is a collection of stubs, drivers, and other supporting tools that are required to automate test execution. It allows for the automation of tests.

- It contains all the information that is required to compile and run a test. For example, test cases, stubs, drivers, Source files under tests, etc.
- It helps to enhance the quality of the software components and applications.
- It helps developers to measure code coverage at a code level.

13. What is Concurrency Testing?

Concurrency Testing also known as Multi-user Testing checks the software performance when multiple users are logged into the system and perform actions simultaneously.

- It helps to monitor the effect on the system when multiple users are performing the same action at the same time.
- It helps to monitor the system for deadlocking, locking, and single-threaded code.

14. What is Defect Age?

Defect Age is defined as the time difference between the defect detected date and the current date provided the defect is still in the open state. It is divided into two parameters:

- Defect Phase: The defect phase is the numerical value that is assigned to a defect occurring at any phase, depending upon the degree of risk involved in the defect.
- Defect Age Time: Defect age time is a way to determine the difference between the date of defect detection and the time till when the defect is open or has been resolved.

15. What is Un-Installation Testing?

Uninstallation testing is a type of software testing that is performed to make sure that all the components of the application are removed during the process or not.

16. What is Equivalence Class Partition (ECP)?

Equivalence Class Partitioning is a black-box software testing technique that divides the input data into partitions of equivalent data from which test cases can be derived.

- In this approach, test cases are designed to cover each partition at least once.
- It divides the input data of software into different equivalence data classes.

17. What is Software Configuration Management?

Software Configuration Management (SCM) is a process to manage, organize, and control the changes in the code, document, and other entities during the Software Development Life Cycle (SDLC).

- It is commonly used in software development groups in which several developers are concurrently working on a common set of files.
- SCM is designed to avoid the problem of sharing files in a multiuser environment.

18. What is a Test Script?

Test scripts are step-by-step instructions containing information about the system transactions that should be performed to validate the application or system under test.

- These are the programs that run tests on the software product/ application.
- The tester has to write and run test scripts to validate if the application's outcome meets the business requirements.

19. What is Test Bed?

Test Bed is a platform for conducting rigorous, transparent, and replicable testing that consists of specific hardware, software, operating system, network configuration, software configuration, etc.

20. What is Sanity Testing?

Sanity Test also known as Surface Testing is a type of software testing that is performed to make sure that the code changes made are working properly without any bugs.

- This type of testing is done on the stable build of the software.
- It is a subset of Regression testing.
- Sanity testing is usually done after the software product has passed the Smoke test.
- The focus of this type of testing is to validate the functionality of the application and not detailed testing.

21. What is Test Strategy?

Test Strategy is a document that outlines the testing technique that is used in the Software Development Life Cycle and guides QA teams to define Test Coverage and Testing scope.

22. What is Test Scenario?

Test Scenario is a detailed document that covers end to end functionality of a software application that can be tested. It is also known as Test Possibility or Test Condition. In this, the testers need to put themselves in the place of the user as they test the software application from the user's point of view.

23. What is Code Coverage?

Code Coverage is a software testing metric that is used to determine how much of the code is tested. This helps in assessing the quality of the test suite and analyzing how comprehensively a software is verified. It is one form of white box testing that finds the areas of the program not exercised by a set of test cases.

24. Explain the role of testing in software development.

Testing plays a vital role in software development. The techniques used for software testing differ from one company to another. Below are some of the important roles played by testing in software development-

- Examine the code for discovering problems and errors early in the system.
- Testing evaluates the capabilities of the program over the entire product.
- Testing helps in reviewing requirements and design as well as executing the code.
- Testing is important in measuring the system functionality and quality of the product.
- Testing plays a vital role in lowering the maintenance cost of the software product.
- It helps in providing interaction between developers and users.

25. What is a bug report?

During testing, the tester records all the observations, and other information useful for the developers or the management. This test record is known as a bug report. The bug report helps the team members in the following aspects:

- Understand the problem.
- Steps to reproduce the problem.
- The environment under which the defect or bug happens.
- The resolution is when the developer fixes the bug.

The few bits of information the bug report should contain are:

1. Defect/Bug Name: A short headline describing the defect. It should be specific and accurate.
2. Defect/Bug ID: Unique identification number for the defect.
3. Defect Description: Detailed description of the bug including the information of the module in which it was detected. It contains a detailed summary including the severity, priority, expected results vs actual output, etc.
4. Severity: This describes the impact of the defect on the application under test.
5. Priority: This is related to how urgent it is to fix the defect. Priority can be High/ Medium/ Low based on the impact urgency at which the defect should be fixed.
6. Reported By: Name/ ID of the tester who reported the bug.
7. Reported On: Date when the defect is raised.
8. Steps: These include detailed steps along with screenshots with which the developer can reproduce the same defect.
9. Status: New/ Open/ Active
10. Fixed By: Name/ ID of the developer who fixed the defect.
11. Data Closed- Date when the defect is closed.

26. What are the different types of testing metrics?

1. Process Metrics: These metrics are essential for the improvement and maintenance of the process in SDLC. These are used to improve the process efficiency of the SDLC.
2. Product Metrics: These define the size, performance, quality, and complexity of the product. It deals with the quality of the software product and thus can help developers to enhance the software product quality.
3. Project Metrics: These define the overall quality of the project. These can be used to measure the efficiency of the project team or any testing tools being used by the team members.

27. What is a test case? Can you write one?

Answer:
A test case is a set of conditions or variables under which a tester determines whether an application is working as expected.

Example Test Case (Login Functionality):

- Test Case ID: TC001
- Objective: Validate successful login
- Steps: Enter valid username and password → Click login
- Expected Result: User is redirected to the dashboard

28. What are exit and entry criteria in system testing?

Answer:

- Entry Criteria: Conditions that must be met before testing begins (e.g., test environment is set up, code is complete).
- Exit Criteria: Conditions to stop testing (e.g., all test cases executed, no high-severity defects remain).

29. How do you prioritize test cases in system testing?

Answer:

- Based on business impact, risk, frequently used functionalities, and critical path scenarios.
- Focus on high-risk modules first (e.g., payment systems, authentication)

30. How do you handle failed test cases?

Answer:

- Log the defect in a tracking tool (e.g., JIRA, Bugzilla)
- Provide steps to reproduce
- Work with developers for resolution
- Retest after the fix

31. What challenges do you face in system testing?

Answer:

- Incomplete requirements
- Test environment issues
- Time constraints
- Managing large and complex test data
- Integration with external systems

32. How do you ensure test coverage?

Answer:

- Mapping test cases to requirements using RTM (Requirement Traceability Matrix)
- Using code coverage tools if white-box testing is involved

33. What is smoke testing and sanity testing?

Answer:

- Smoke Testing: Quick check to see if the system is stable enough for further testing.
- Sanity Testing: Focused testing to check specific functionalities after minor changes.

34. How do you measure the success of system testing?

Answer:

- % of passed test cases
- Number of defects detected
- Defect leakage rate
- Meeting deadlines and coverage goals
- Stability of the system in UAT/production

35. What would you do if you find a major bug close to the release date?

Answer:

- Report it immediately
- Assess severity and business impact
- Collaborate with the team to see if it can be fixed quickly
- Suggest alternatives like a hotfix or delaying the release (based on priority)

36. What is End-to-End Testing and how is it related to System Testing?

Answer:
End-to-End (E2E) testing ensures the complete flow of an application works from start to finish, mimicking real user scenarios. System testing focuses on verifying the entire system, and E2E is a type of system testing that includes integration with external interfaces (like payment gateways, third-party APIs, etc.).

37. What if requirements keep changing during system testing?

Answer:

- Use Agile testing practices
- Ensure change control process is in place
- Revisit and update test cases
- Communicate proactively with the team to assess test impact
- Use tools like TestRail or ALM for traceability and updates

38. How do you handle test data in system testing?

Answer:

- Use mock/test databases
- Generate test data using scripts or tools
- Anonymize real data if needed
- Use data-driven testing to test multiple input combinations (e.g., with Selenium + Excel/CSV)

39. What is Test Environment? How do you set it up?

Answer:
A test environment is a setup of software and hardware on which the testing team performs tests. It includes:

- OS
- Databases
- Servers
- Network configurations
- Tools
  Setting it up involves coordination with DevOps or IT and replicating production-like scenarios.

40. What is the difference between positive and negative testing?

Answer:

- Positive Testing: Validates the system works with valid input (e.g., correct credentials)
- Negative Testing: Tests how system handles invalid input (e.g., wrong password)

41. What is the difference between Load Testing and Stress Testing?

Answer:

- Load Testing: Tests system performance under expected user load
- Stress Testing: Tests system behavior under extreme conditions (beyond limit) to find the breaking point

42. What is Exploratory Testing?

Answer:
Unscripted, intuitive testing based on the tester's experience and creativity.
Useful when:

- Requirements are unclear
- Quick feedback is needed
- Time is limited

43. What is Compatibility Testing?

Answer:
Testing an application across different:

- Browsers (Chrome, Firefox, Safari)
- Devices (Mobile, Tablet, Desktop)
- OS (Windows, macOS, Linux)
- Networks (3G, 4G, Wi-Fi)

44. What is Localization and Internationalization Testing?

Answer:

- Internationalization (I18N): Testing if the app can support multiple languages/formats
- Localization (L10N): Testing if the app behaves correctly in a specific language/region (e.g., Hindi calendar, ₹ symbol)

45. What is the difference between Verification and Validation?

Answer:

- Verification: Are we building the product right? (Static – reviews, walkthroughs)
- Validation: Are we building the right product? (Dynamic – testing)

46. What is monkey testing?

Answer:
Random, unstructured testing without predefined test cases. Used to:

- Break the system
- Find unexpected crashes
- Test robustness

47. When would you use manual testing over automation?

Answer:

- When testing usability/UI/UX
- For short-term or one-time test cases
- When automation ROI is low
- When the application is in early development and UI is unstable

48. What is Test Coverage and how is it measured?

Answer:
Test Coverage is the percentage of requirements or code covered by test cases.
Measured by:

- Requirements coverage
- Code/path coverage (if applicable)
- Number of executed test cases vs. total planned

49. What is Volume Testing and how is it different from Load Testing?

Answer:

- Volume Testing: Tests system performance with large data volumes (e.g., DB entries)
- Load Testing: Tests system behavior under expected concurrent users or load

50. How do you test APIs during system testing?

Answer:

- Use tools like Postman, SoapUI, or JMeter
- Validate response status, payload, schema, headers
- Perform integration and boundary tests
- Use automation for regression and performance

51. How do you ensure the quality of a test case?

Answer:

- Clear, concise, and traceable to a requirement
- Includes expected output
- Covers positive and negative scenarios
- Reusable and independent
- Peer-reviewed and regularly updated

52. What if you discover a defect in production missed during system testing?

Answer:

- Perform RCA (Root Cause Analysis)
- Check why it wasn't caught (missing test case, data, scenario)
- Add regression test
- Improve processes or environment setup

ROOT CAUSE ANALYSIS

What is Root Cause Analysis (RCA) in Software Testing?

- Root Cause Analysis (RCA) is a structured process used to identify the underlying cause(s) of a defect or problem in a system. Instead of just fixing the symptom (the bug), RCA helps to find and eliminate the actual source, so that the issue doesn't occur again.

Why is RCA Important in System Testing?

- Prevents recurrence of critical issues
- Improves product quality over time
- Helps teams learn from mistakes
- Reduces cost and time of fixing future bugs

How to Perform Root Cause Analysis? (Step-by-Step)

1. Identify the Problem Clearly

- What happened?
- When and where did it happen?
- Who found it?
- What are the symptoms?

Example: "User gets logged out automatically while submitting a form after 5 minutes of inactivity."

---

2. Collect Relevant Data

- Logs, screenshots, error messages
- Version history and environment details
- User flow and test scenarios
- Related bug reports

---

3. Identify Possible Causes

Ask:

- Was there a recent code change?
- Was the defect introduced in integration?
- Was the requirement misunderstood?
- Was a test case missed?

Use brainstorming, code review, and communication with devs or business analysts.

---

4. Use RCA Techniques

👉 Common RCA Techniques:

Technique	Description	Example
5 Whys	Ask "Why?" 5 times until the root cause is found	Why logout? → Session expired → Timeout config → Server config wrong → Dev missed config file
Fishbone (Ishikawa) Diagram	Categorizes potential causes into areas like People, Process, Tools, etc.	Useful for complex bugs involving multiple teams
Fault Tree Analysis	Top-down approach to find cause of system failure	Useful in hardware+software integrations
Pareto Analysis (80/20 Rule)	Focus on top 20% of causes that lead to 80% of problems	Helps prioritize high-impact fixes
Change Analysis	Identify what changed recently that could have caused the issue	Common in post-deployment bugs

---

5. Identify the Root Cause

Using the above methods, pinpoint the first event or failure that triggered the bug.

---

6. Define Corrective Actions

- Fix the code or logic issue
- Update test cases or coverage
- Improve documentation or review processes
- Train the team if knowledge gap exists

---

7. Preventive Measures

- Add the bug to regression suite
- Automate test case if possible
- Implement code review or design checks
- Add monitoring or alerts in production

🧪 RCA Example (Simple Case)

Issue: App crashes when uploading large files
RCA using 5 Whys:

1. Why did the app crash? → Out of memory
2. Why out of memory? → File size exceeded limit
3. Why wasn't limit enforced? → No validation on client side
4. Why no validation? → Requirement was missed in design
5. Why missed? → Lack of communication between design and dev team

Root Cause: Missing client-side validation due to unclear design handoff

Example scenarios

1. Login System (Authentication)

Scenario: Verify login functionality with valid and invalid credentials.

✅ **Test Cases:**

- Login with correct username and password → success
- Login with wrong password → failure message
- Login with empty fields → validation error
- Login after password reset → check new password works

🧪 *Tests backend API, frontend validation, and DB changes*

2. User Registration

Scenario: Register a new user and validate confirmation.

✅ **Test Cases:**

- Submit all valid data → success message and email sent
- Submit with an existing email → error shown
- Submit with empty fields → form validation triggers
- Database has correct new user entry

🧪 *Tests integration between frontend → backend → DB → email system*

3. Search Functionality

Scenario: Test search bar in an e-commerce or job portal site.

✅ **Test Cases:**

- Search for an existing item → item appears in result
- Search with no results → "No results found"
- Use special characters or SQL injection → safe handling

🧪 *Covers UI, search algorithm, and input sanitation*

4. Payment Gateway Integration

Scenario: Purchase an item using a credit/debit card.

✅ **Test Cases:**

- Valid card details → successful payment, confirmation email
- Expired card → error message
- Duplicate payment attempt → only one transaction saved
- Check payment details in the database and transaction log

🧪 *Tests entire purchase system + payment API + response validation*

5. File Upload/Download

Scenario: Upload a document (e.g., resume) and download a PDF report.

✅ **Test Cases:**

- Upload supported file types (.pdf, .docx)
- Upload unsupported format → validation error
- File exceeds size limit → error shown
- Downloaded file is not corrupted and matches content

🧪 *Tests file system, validation logic, and server handling*

6. Email Notification System

Scenario: Email gets triggered after successful registration or booking.

✅ **Test Cases:**

- Check email received after a successful booking
- Email contains correct user info and formatted content
- Invalid email address → log error, no crash

🧪 *Verifies background jobs, email server config, and content templating*

7. Multi-User System (Admin/User Roles)

Scenario: System has user types with different access levels.

✅ **Test Cases:**

- Admin logs in → sees dashboard with edit/delete rights
- Normal user → only view rights
- Unauthorized access → access denied page

🧪 *Tests authentication + role-based access control (RBAC)*

8. Database Integration

Scenario: A form submission inserts data into a DB.

✅ **Test Cases:**

- Submit form → data correctly saved in DB
- Duplicate entries prevented (unique key check)
- DB rollback on failure → data consistency maintained

🧪 *Validates back-end logic, SQL queries, and data integrity*

9. Mobile App Scenarios

Scenario: User signs in and receives push notification.

✅ **Test Cases:**

- Test login/signup on different screen sizes (responsive UI)
- App goes offline during action → proper error message
- Push notification received on successful action

🧪 *Validates network handling, UI scaling, and OS-specific behavior*

10. System Load Scenario

Scenario: 500 users login simultaneously.

✅ **Test Cases:**

- Server handles load without crashing
- API response time < 2 seconds
- Monitor CPU/RAM usage on server

🧪 *This would involve performance testing using tools like JMeter, but is part of full system validation*

Scenario 11: Password Reset Workflow

📌 **Objective: Ensure the full *"Forgot Password"* flow works securely and reliably.**

✅ **Test Cases:**

- Submit registered email → receive reset link
- Submit unregistered email → error message
- Click expired/invalid link → show error, prevent reset
- Successfully change password → login with new password works
- Database reflects updated password (hashed)

Scenario 12: Session Timeout & Auto-Logout

📌 **Objective: Check session management and security.**

✅ **Test Cases:**

- User logs in → remains active until timeout (e.g., 15 mins)
- After timeout → automatically logged out, redirected to login
- Open multiple tabs → timeout should apply globally
- Perform action just before timeout → session extends (if designed)
- Logout manually → session destroyed

SITUATION–BASED QUESTIONS :

1. Situation:

A developer insists their code is bug-free after unit testing. However, you encounter a bug during system testing.
Question: How would you handle this situation?

Answer:
I would document the bug with clear steps to reproduce, include screenshots/logs, and share it respectfully with the developer. I'd explain that unit testing covers isolated components, while system testing checks the integration of all modules. I'd work collaboratively to resolve the issue instead of assigning blame, ensuring the goal remains delivering a quality product.

2. Situation:

You're testing a login feature. After entering the correct credentials, it takes unusually long to log in.
Question: What steps will you take to analyze and report this?

Answer:
First, I would replicate the issue across different browsers and network conditions to rule out external factors. Then, I'd check the backend logs or performance metrics (if available). I'd raise a performance bug with response time metrics, system specs, and steps to reproduce. I'd also suggest load or stress testing to identify potential bottlenecks.

3. Situation:

The application works fine on the desktop but shows layout issues on mobile devices.
Question: How would you proceed to verify and report the problem?

Answer:
I would test the application across multiple screen sizes and resolutions using responsive testing tools or real devices. I'd document the visual discrepancies with screenshots, CSS breakpoints, and device specs. In the bug report, I'd categorize it under UI/UX issues and suggest responsive design improvements or media queries to address the issue.

4. Situation:

A test case fails intermittently. It passes sometimes and fails other times without code changes.
Question: How would you analyze this?

Answer:
Intermittent failures often point to synchronization issues, environment instability, or data dependency. I'd check for race conditions, timing delays, or backend response inconsistencies. I might introduce waits or mocks where necessary. I'd add logs or debug statements to trace execution flow and share detailed observations in the defect report to aid root cause analysis.

5. Situation:

You're assigned to test a module without documentation.
Question: How would you begin testing?

Answer:
I'd start with exploratory testing to understand the functionality. I'd talk to developers, business analysts, or product owners to gather context. I'd analyze the UI, database interactions, and API calls (if any). Based on this, I'd create test scenarios, validate edge cases, and document my findings. I'd also recommend creating proper documentation for future reference.

6. Situation:

A customer reports a critical bug in production that was missed during testing.
Question: What would you do?

Answer:
First, I'd gather as much detail as possible about the production issue. I'd verify if the issue can be reproduced in a test environment. Then, I'd perform root cause analysis to understand why the bug escaped detection—whether it was due to missing test coverage, incorrect assumptions, or environment differences. I'd update test cases, include regression tests, and suggest improvements in the testing process or test data coverage to prevent recurrence.

7. Situation:

You're testing an e-commerce checkout feature, and payments are randomly failing.
Question: How would you test and analyze this?

Answer:
I'd identify patterns in failures — such as specific payment methods, timeouts, or server errors. I'd test different scenarios like invalid card details, network loss, or concurrent transactions. I'd also use logs to inspect API response codes and analyze integration with the payment gateway. The issue would be reported with complete transaction details, environment, and failure rates.

8. Situation:

Your test automation suite is taking too long to run, affecting deployment timelines.
Question: What improvements can you suggest?

Answer:
I'd identify and eliminate redundant test cases, prioritize high-risk areas, and consider parallel execution to reduce run time. I'd also categorize tests into smoke, regression, and sanity to run only what's needed at each stage. Additionally, optimizing locators, reducing sleep time, and using lightweight frameworks can further speed up execution.

9. Situation:

The product is nearing release, but a major feature hasn't been tested due to delays from the development team.
Question: What would you do?

Answer:
I'd escalate the risk to stakeholders and suggest a contingency plan like deferring the feature to a later release or performing rapid exploratory and risk-based testing once it's delivered. I'd prioritize core functionalities, perform boundary testing, and document what has and hasn't been tested to ensure transparency and mitigate quality risks.

10. Situation:

Your manager asks you to test a third-party tool integration in a limited time.
Question: How would you approach it efficiently?

Answer:
I'd begin by understanding the integration points and identifying critical use cases. I'd focus on testing data flow, authentication, error handling, and edge cases. I'd check compatibility and API contracts if applicable. I'd use tools like Postman or test stubs for faster validation and ensure high-priority scenarios are covered under time constraints.

11. Situation:
You're assigned to test a module, but the requirements are ambiguous or incomplete.
Question: How would you proceed?

Answer:
I'd engage stakeholders (BA, developers, client) for clarification. Meanwhile, I'd analyze existing features, perform exploratory testing, and create assumption-based test cases, marking them clearly for review. I'd advocate for a quick requirement walkthrough to avoid downstream defects.

12. Situation:
A developer disagrees with your reported bug and says, "It's working as designed."
Question: What would you do?

Answer:
I'd revisit the requirements and validate the behavior against them. If ambiguity exists, I'd involve the product owner or BA to clarify. My goal would be not to win an argument but to ensure the end-user gets the correct functionality.

13. Situation:
Testing is happening in a staging environment that's significantly different from production.
Question: How would you ensure valid results?

Answer:
I'd note the differences (data volume, integrations, security settings), and raise it as a risk. I'd request a performance and data configuration closer to production. I might simulate production-like load using tools and raise priority bugs based on likely real-world impact.

14. Situation:
You're told to complete testing in 1 day for a 3-day estimated module.
Question: How would you prioritize?

Answer:
I'd focus on **risk-based testing**: covering critical paths, customer-facing features, and high-defect areas first. I'd skip low-risk or cosmetic issues and keep stakeholders informed of what has/hasn't been tested. I'd suggest deferring less important cases or testing them in post-release hotfix rounds.

15. Situation:
You realize halfway through testing that your test data was incorrect.
Question: What steps would you take?

Answer:
I'd stop and isolate all impacted test cases. I'd regenerate accurate data and re-execute critical tests. I'd document the affected test areas and communicate possible gaps to the team. I might also suggest automation/data reset utilities to prevent recurrence.

16. Situation:
You're testing a role-based access control feature and a user gets access to unauthorized modules.
Question: How do you investigate and report this?

Answer:
I'd check the role mappings, session control, and any cached permissions. I'd try with fresh logins, different users, and confirm the access flow. I'd raise it as a **high severity bug**, include logs/screenshots, and recommend reviewing authorization logic and session tokens.

17. Situation:
A bug that was fixed in QA shows up again in UAT.
Question: What would be your analysis?

Answer:
I'd check for deployment mismatch, environment differences, or regression test failures. I'd verify if the correct build/version was tested and deployed. I'd recommend improving release note validation and suggest sanity testing post-deployment to catch such mismatches early.

18. Situation:
You notice that manual testing is taking too long for repetitive tasks.
Question: What would you propose?

Answer:
I'd propose using automation for repetitive regression tasks. I'd evaluate tools like Selenium, Postman for APIs, or JMeter for performance. I'd create a PoC, show time savings, and advocate for test automation in CI/CD pipelines if possible.

19. Situation:
After release, the client identifies a scenario that wasn't covered in your test cases.
Question: How would you handle it?

Answer:
I'd first acknowledge the oversight, then analyze why it wasn't covered—missing requirement, lack of domain understanding, or test design gap. I'd update the test case library, create a checklist to avoid similar misses, and possibly enhance peer reviews or exploratory test sessions.

20. Situation:
An integrated third-party system goes down during your testing.
Question: What would you do?

Answer:
I'd log the incident and switch to testing non-dependent features or use mocks/simulators if available. I'd work with the dev/Ops team to ensure service monitoring is in place and raise a risk note about the dependency affecting overall test cycles.

21. Situation:
You join a project where no formal test documentation exists.
Question: What would you suggest?

Answer:
I'd start by creating a **test case repository** using tools like TestLink, Zephyr, or Excel. I'd propose setting up test plans, traceability matrices, and a bug tracking process. I'd also recommend a version control system for test assets and test coverage metrics for future efficiency.