Software Testing And Automation

SDLC -

1. SRS - Software Requirement Specification

Business Analyst, Functional Groups , Team Leads

It includes details of software to be developed . User , Processes, screen images, flow diagrams, descriptions, data

2. System Design Phase - detailed

analysis of system and it’s paper design.

Screen Designs - By Graphic Designer

DFD / Flow Diagrams/ UML/ Use Case Diagram /

Database Design - RDBMS / Module Division

Testing Plan

3. Coding Phase - Java, HTML, JSP, MySQL

4. Testing - Manual/Automation

5. Installation Maintenance

1. Manual Testing
2. Basics of Web Development (HTML+CSS+JavaScript)
3. Core Java, SQL/MySQL
4. Testing Automation with Selenium, Jmeter, **Jira**, TestNG, Appium

Testing Phase - match user requirements with the developed software

Software Tester Career Path

Your Software Testing career growth as a software tester (QA Analyst) but will vary from

company to company

1. Quality Assurance Analyst (Fresher) / SDET
2. Sr. QA Analyst (2-3 years' experience)
3. QA Team Coordinator (5-6 years' experience)
4. Test Manager (8-11 years' experience)
5. Senior Test Manager (14+ experience)

TESTING

Course Objectives

- To understand what is testing?
- To understand Software development model.

- To Understand Architectures of software development.
- To learn the features of Software development models.
- To learn major concepts of the testing methodologies.
- To know different approaches to Testing.
- To understand of the types of testing.
- To plan and create test plan
- To execute the test plan.
- To create and manage test cases and defect profiles.
- To build strategies to track testing processes in the bug tracking systems.
- To do document of the test report in the testing enclosure document.
- Differences between Manual and

Automation Testing.

Course Content

1. Manual Testing
2. Basics of Web Development (HTML+CSS+JavaScript)
3. Core Java, Database Design
4. Testing Automation with Selenium

Manual Testing -

White for dev, Black for QA , Monkey Testing
Risk Management

TestManagement tool - JIRA with Addon
JMeter, Postman

Bug Reporting Tool - JIRA, BugZilla, SpiraTest ,

ISTQB Certificate on Manual Testing -

Selenium - Core Java + Tool , Mobile Testing, APPIUM , TestNG
Web development Basic - HTML, CSS , JavaScript

placements@codebetter.in
placements*noan2@#n:4tyn:xdgh e5y7r6y?

Deve Ops Concepts - Git Hub, Docker, Jenking etc….
RPA -

Manual Testing

Module 1: Software Testing Introduction

🆘 What is testing?

Software Testing is a method to check whether the actual software product matches expected requirements and to ensure that software product is **DEFECT free**. It involves execution of software/ system components using manual or automated tools to evaluate one or more properties of interest. The purpose of software testing is to identify errors, gaps or missing requirements in contrast to actual requirements.

⍰ Importance of testing

Software Testing is Important because if there are any bugs or errors in the software, it can be identified early and can be solved before delivery of the software product. Properly tested software product ensures reliability, security and high performance which further results in time saving, cost effectiveness and customer satisfaction.

Testing is important because software bugs could be expensive or even dangerous. Software bugs can potentially cause monetary and human loss. Examples.

- Starbucks was forced to close about 60 percent of stores in the U.S and Canada due to software failure in its POS system. At one point, the store served coffee for free as they were unable to process the transaction.
- Some of Amazon's third-party retailers saw their product price is reduced to 1p due to a software glitch. They were left

with heavy losses.

Here are the benefits of using software testing:

- **Cost-Effective:** It is one of the important advantages of software testing. Testing any IT project on time helps you to save your money for the long term. In case if the bugs caught in the earlier stage of software testing, it costs less to fix.
-
- **Security:** It is the most vulnerable and sensitive benefit of software testing. People are looking for trusted products. It helps in removing risks and problems earlier.
-
- **Product quality:** It is an essential requirement of any software product. Testing ensures a quality product is delivered to customers.
-
- **Customer Satisfaction:** The main aim of any product is to give satisfaction to

their customers. UI/UX Testing ensures the best user experience.

Module 2: Software Development Life Cycle

SDLC (Software Development Life Cycle) defines all the standard phases which are involved during the software development process. SDLC life cycle is a process of developing software through a phased manner in the following order

1. SDLC Phases

? **Requirements** gathering and Analysis Phase.

? **Design phase**. A design for developer(Flow Design Using DFD/ ER, UseCases and Activity, Module Division, Technology Decision, Technology Independent design, DB

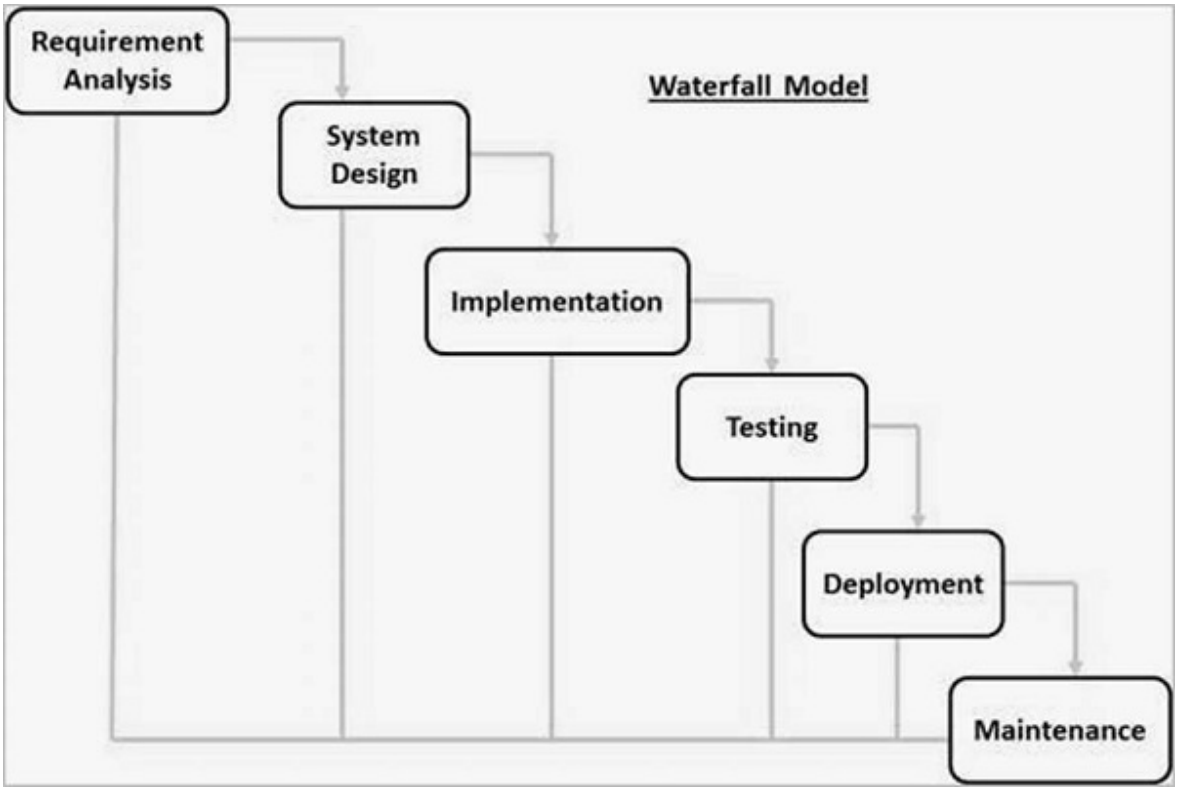
Design)

- **Coding** Phase (MYSQL - DB, Java/AngularJS/HTML).
- **Testing phase.**
- **Release** and Maintenance Phase.

2. SDLC Models

- **Waterfall Model.**

In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.



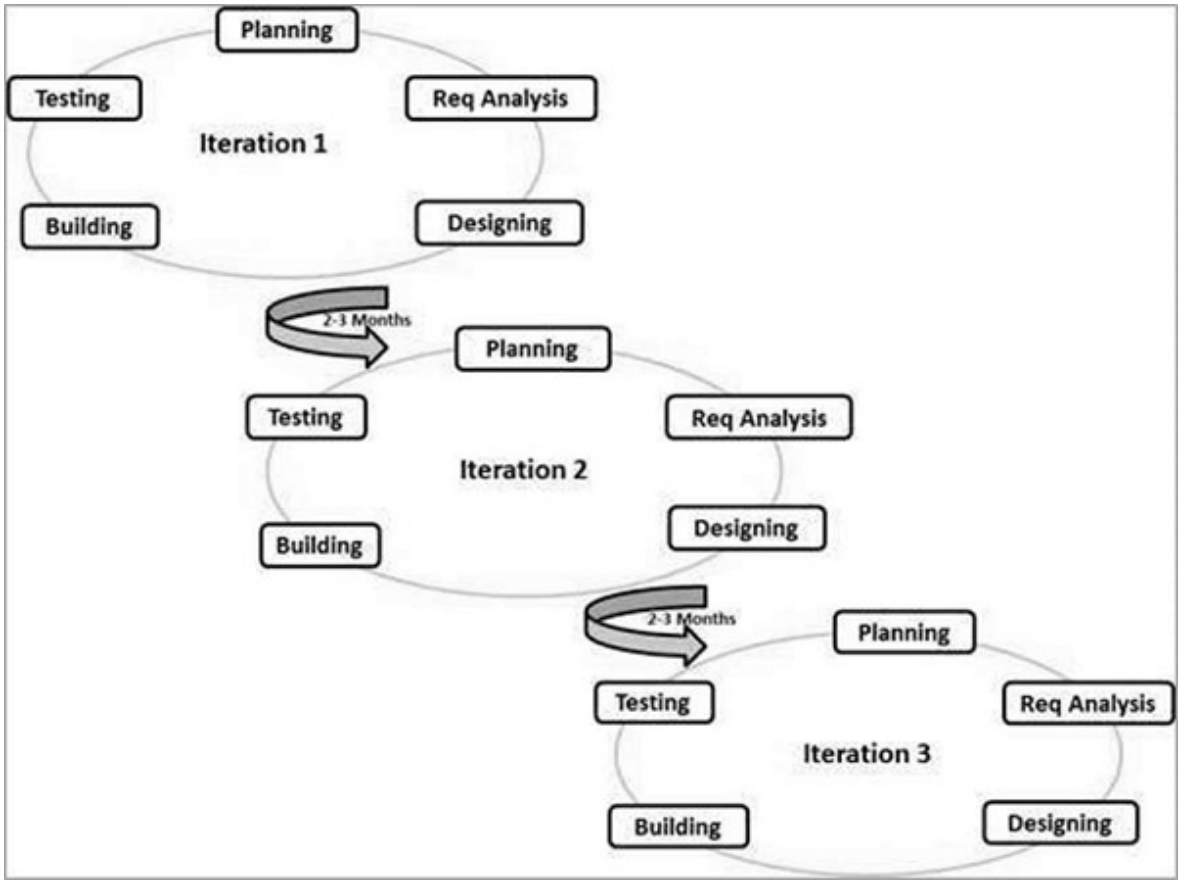
? Agile Model. (Major)

Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. **Agile Methods break the product into small incremental builds. These builds are provided in iterations**. Each iteration typically lasts from about **one to three weeks**. Every iteration involves cross functional teams working simultaneously on various areas like −

- Planning

- Requirements Analysis
- Design
- Coding
- Unit Testing and
- Release and Acceptance Testing

Agile model believes that every project needs to be handled differently and the existing methods need to be tailored to best suit the project requirements. In Agile, the tasks are divided to time boxes (small time frames) to deliver specific features for a release.



Agile Manifesto principles −

- **Individuals and interactions** − In Agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.

- **Working software** − Demo working software is considered the best means of communication with the customers to understand their requirements, instead of just depending on documentation.

- **Customer collaboration** − As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.

- **Responding to change** − Agile Development is focused on quick responses to change and continuous development.

? Prototype Model.

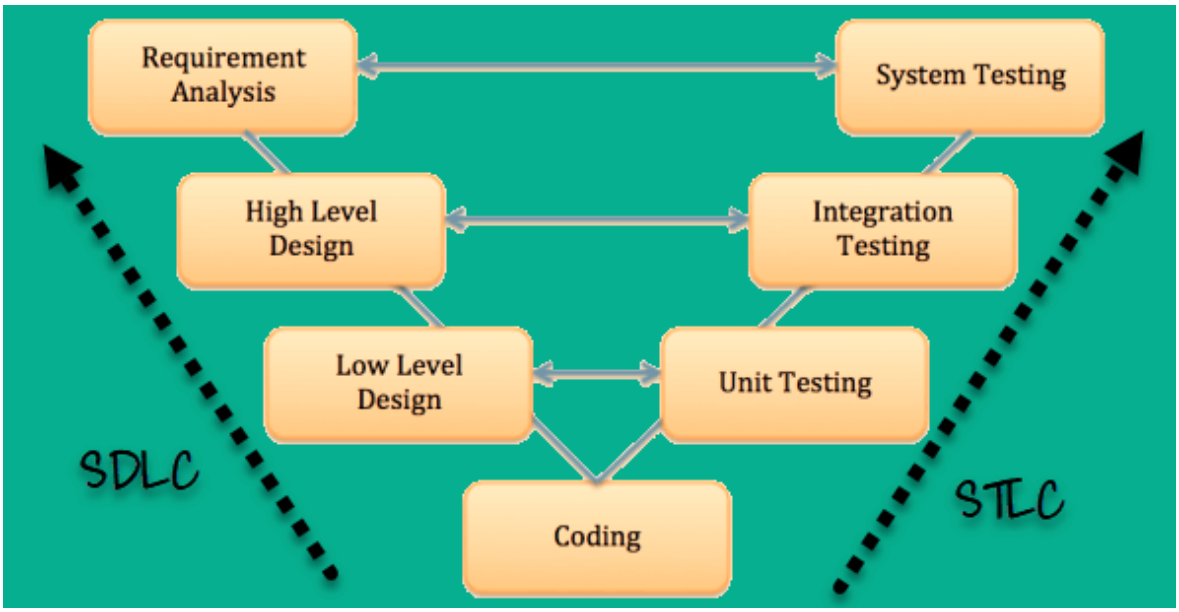
Module 3: Software Testing Methodologies

Software Testing Life Cycle (STLC) is the testing process that is executed in a well-planned manner. In the STLC process, various activities are carried out to improve the quality of the product. However, **STLC phases only deal with testing and detecting errors but not development itself.**

Different companies define different phases in STLC. However, generic Software Test Life Cycle has the following stages.

1. Requirement Analysis(**S**oftware **R**eq. **S**pecification)
2. Test Planning - Design Document
3. Test Case Development - Design Doc and SRS
4. Test Environment Setup - os/browser/ hardware

5. Test Execution & Closure - actual testing and reporting



Project -> Modules -> Scenarios -> Test Cases.

Login/signup/password change
Bus search and book
Payment
Agent
Clerk/Admin

Attendance and HR System for BPO

1. Daily Login Time of employee
2. Attendance recording
3. Daily work target
4. Error in task
5. Manage break time
6. Logoff time
7. WFH
8. Employee records(personal detail, professional detail, shift details)
9. Late login relaxation
10. Leaves 3/month
11. Salary calculation
12. Overtime facility
13. Employee Care Request
14. Employee work observation

Users - Employees, Client, Team Leader, Manager, HR, EC

Module - 1 - Login and Authorization

- Login screen
- forgot password
-Logout
-Profile Screen
- Authorization

Module - 2 Attendance

- Login Time History
- Logoff time recording and History
- Overtime recording screen

- Late Login relaxations - 3
- break times

Module - 3 - Employee records
- Add/Edit/View employee record(personal and professional)
- Salary Calculation
- Employee Shift details, WFH
- Employee care request
- Leave details

Module 4 - EmployeeWork
- assign task to employee and decision target hours

-

Types of Test Documentation:

Types of Testing	Description
Test policy	It is a high-level document which describes principles, methods and all the important testing goals of the **organization**.
Test strategy	A high-level document which identifies the Test Levels (types) to be executed for the **project**.
Test plan	A test plan is a complete planning document which contains the scope, approach, resources, schedule, etc. of testing activities.

Requirements Traceability Matrix	This is a document which connects the requirements # to the test cases #.
Test Scenario	Test scenario is an item or event of a software system which could be verified by one or more Test cases. **Like search and select a product on amazon.in, Login/Registration**
Test case	It is a group of input values, execution preconditions, expected execution postconditions and results. It is developed for a Test Scenario.
Test Data	Test Data is a data which exists before a test is executed. It used to execute the test case.

Defect Report	**Defect** report is a documented report of any flaw in a Software System which fails to perform its expected function.
Test summary report	Test summary report is a high-level document which summarizes testing activities conducted as well as the test result.

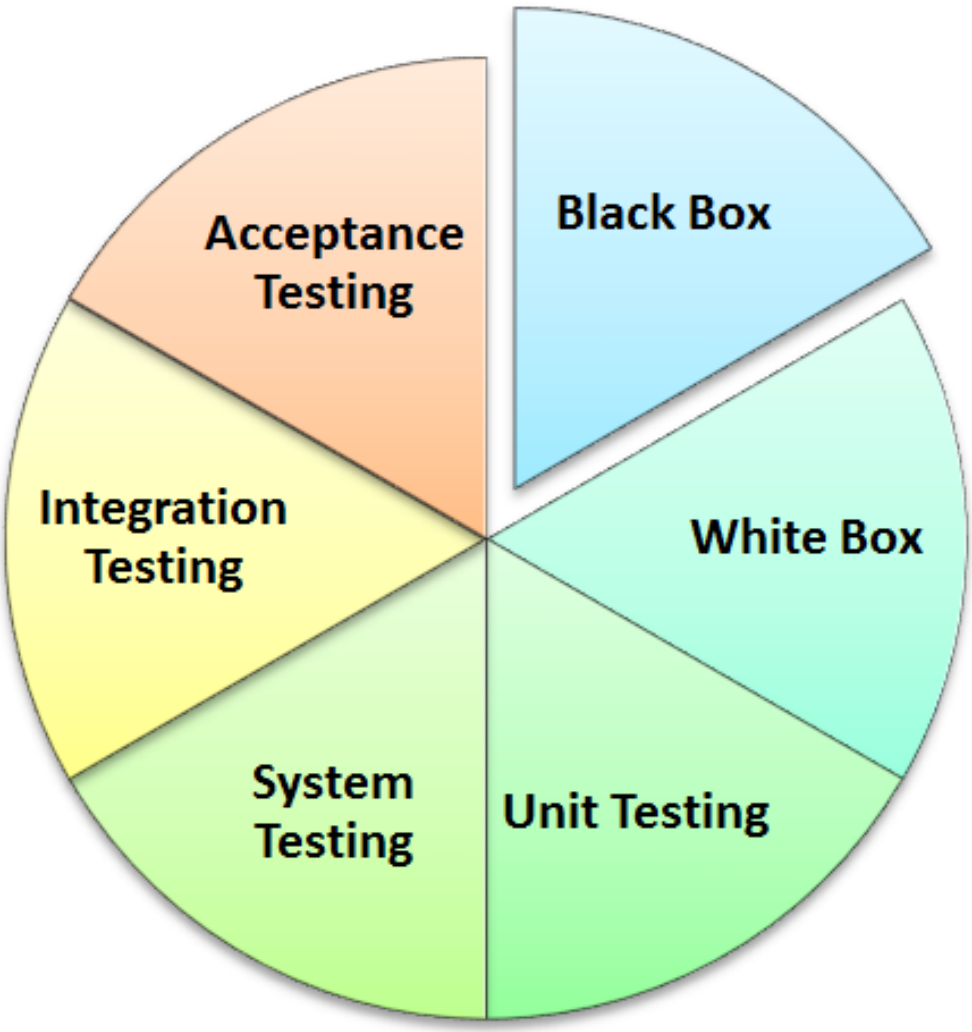
Steps to perform Manual Testing

1. Read and understand the software project documentation/guides. Also, study the Application Under Test (AUT) if available.
2. Draft Test cases that cover all the requirements mentioned in the documentation.

3. Review and baseline the test cases with Team Lead, Client (as applicable)
4. Execute the test cases on the AUT
5. Report bugs.
6. Once bugs are fixed, again execute the failing test cases to verify they pass.

Manual Testing

Manual Testing is a type of software testing in which test cases are executed manually by a tester without using any automated tools. The purpose of Manual Testing is to identify the bugs, issues, and defects in the software application. Manual software testing is the most primitive technique of all testing types and it helps to find critical bugs in the software application.



White Box Testing.

White-box testing is a testing technique which checks the internal functioning of the system. In this method, testing is based on coverage of code statements, branches, paths or conditions. White-Box testing is considered as **low-level** testing. It is also called glass box, transparent box, clear box or code base testing. **The white-box**

Testing method assumes that the path of the logic in a unit or program is known.

⍰ Black Box Testing. (**Major**)

In Black-box testing, a tester doesn't have any information about the internal working of the software system. Black box testing is a **high level** of testing that focuses on the **behavior of the software**. It involves testing from an external or **end-user perspective**. Black box testing can be applied to virtually every level of software testing: unit, integration, system, and acceptance.

MAIN DIFFERENCE

- In Black Box, testing is done without the knowledge of the internal structure of program or application whereas in White Box, testing is done with knowledge of the internal structure of program.
- Black Box test doesn’t require programming knowledge whereas the White Box test requires programming

knowledge.

- Black Box testing has the main goal to test the behavior of the software whereas White Box testing has the main goal to test the internal operation of the system.
- Black Box testing is focused on external or end-user perspective whereas White Box testing is focused on code structure, conditions, paths and branches.
- Black Box test provides low granularity reports whereas the White Box test provides high granularity reports.
- Black Box testing is a not time-consuming process whereas White Box testing is a time-consuming process.

⍰ Grey Box Testing.

In Software Engineering, Gray Box Testing gives the ability to test both sides of an application, presentation layer as well as the code part. It is primarily useful in Integration Testing and Penetration Testing.

To perform Gray box testing, it is not necessary that the tester has the access to

the source code. A test is designed based on the knowledge of algorithm, architectures, internal states, or other high -level descriptions of the program behavior.

Example. - While testing websites feature like links or orphan links, if tester encounters any problem with these links, then he can make the changes straightaway in HTML code and can check in real time.

Blood Bank Search

1.Blood Bank – About->Name, address, city, contact, email, username, password, services, contact person, Registration number

2.Donor and Blood Entry - Donor name, Nationality, Occupation(Optional) Donor address, donor city, email id, contact no., blood group, Temperature, Height, Volume of blood, Bag No(unique ID), collection date, expiry date , Place of Donation, Blood Bank name where blood is going to be collected.

3.Admin task - Can add Blood Bank Master Data->
Admin information -> Id, Name, Gender, (optional)User type(Staff or main Admin), Email , contact number, username, password.

4. Blood bag Receiver – Name, Address, City, Email id, contact no. , Another contact no. for emergency calling, Blood group, Quantity needed, Allotted Bag no.(Blood Bag), [Hospital name(Optional) where treatment is going on] , Request no.,

Request date., Request status(Urgent, open, closed,)

5. Family details of Receiver- Number of members, their names, Blood groups, contact numbers, Address.

6. Complaints and Feedback-> Complaint no. , Complaint date, Complain Detail.

7. Alerts to donor for next donation-> Alert template , Notification date, sent to Donor name,

Module -

1 - **Login/Logout -** Admin, Bloodbank can login and logout

Screen - 1. Login

2. Logout option on home page of user.

2. - **BloodBank Modul**e - Bloodbank can add donor information and Family details along with details of blood collected.

Screen - 1. Add new Donor Entry

2. List All Donors

3. Edit Donor Info

4. Add Receiver Info

5. Edit Receiver Info

6. List all receiver

7. Black list frequent receivers

3. - **Search and Complaint -** General user can provide complaint or feedback and admin can view and respond.

Screen

1. Search blood bank by city and blood group.

2. Search Result Screen(blood

bank list)

3. View selected blood bank details
4. Entry screen for complain
5. View all complain (by admin)
6. View selected complain (admin)

4. **Alert** - System can timely notify regular donors about next donation

Screen 1. To select donors and send message

2. Screen to set common message

5. **Admin** - Manage blood bank information

Screen

1. Add new Blood bank detail
2. List All Bloodbank
3. Edit Blood bank Info
4. Report generation

School App Discussion

Admin side-

Add notice - by class/section/all - title , desc, date, file(optional)

Homework - date , subject, description, class , section , file (optional)

Multilingual - English / Hindi App side

Date wise Attendance - for section add by class teacher - super admin. / Absent/Leave/Present

Student Details - name, scholar_number, class, section, address, city, father name, mother name, primary contact, secondary contact, primary_email

Student Achievement - Student, Title, desc , file(optional)

Class wise fee records - master data for classes , fee date, fee head , fee amount, session(e.g. 2020-21)

Admin should be able to add records for particular student.
Fee Records - student, fee head , pay date, amount ,

Staff detail - first name, last name, designation, subject, staff_type, dept, joining date

Events and Gallery - event title, date, description, photos

Bus Stops - Stops code, Stop name
Bus Routes - Bus No , Bus stop code , order , stop_arrival_time

StudentBusAllocation - student, Bus No , drop_stop, pick_stop .

Test - subject, class, test_date, max marks, teacher , test_type (monthly, qtly, half yearly, third qtly, yearly)
TestResult - student , test, marks , grade

//=====================
App Side
E Learning. - Material Link class wise
Class Meetings - to share zoom/google meet - subject, class , date, time , link
Notification of Circular and notices
Attendance Records for Parents to view.

Result -

Test - subject, class, date, max marks, teacher , type (monthly, qtly, half qrly, third qtly, yearly)
TestResult - student , test, marks , grade

Screen Priority -

App - Main Menu

App Modules Division -

1. Login Authentication and Authorization
    - Login, Logout, User Create, User rights,

2. Fee Management
    - Fee records, fee payment, fee display, fee titles, summary and reports

3. Student Test
      - Test Master Records, Manage Student Test Data

4. Bus Management
     - Bust Stops, Route Records, Bus Order, Bus Allocation

5. Events and Gallery
     - Manage Multiple events and their data

Test Case Development - Excel Sheet

1. Project Name - BloodBankSearch

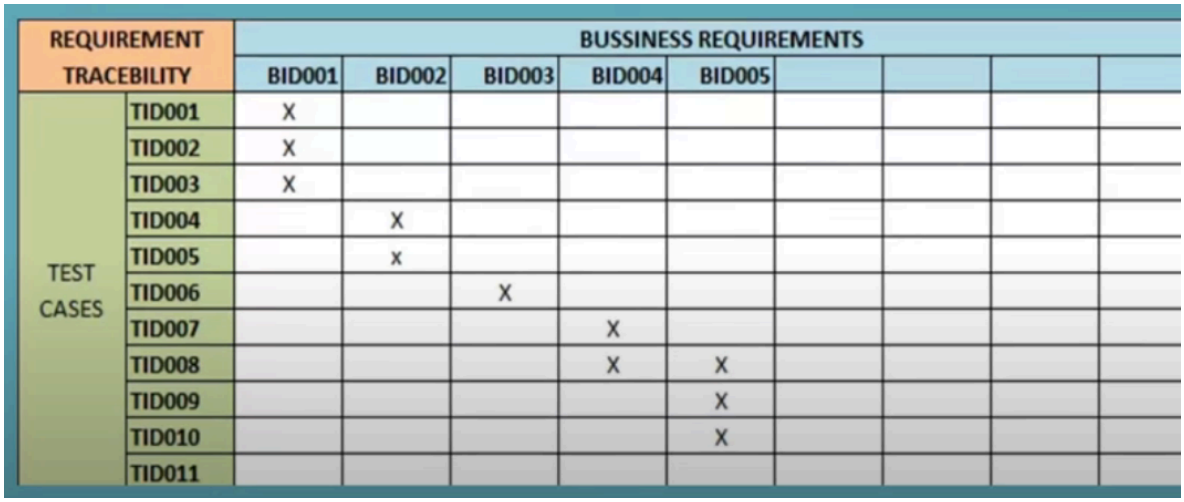
2. Module Name - Search and complain
3. Scenario Name - Search
4. Create By - Hemant
5. Create Date - 30-07-21
6. Reviewed By - Amit
7. Review Date -
—Cycles Run / pass count - 2
8.Test ID (multiple entries)
9. Test Desc(multiple entries)
10. Input Data
11. Steps
12. Pre Condition
13. Post Conditions
14. Expected Result
15. Actual Result
16. Test Status(pass/fail)
17. Executed By
18. Execute Date
19. Comments

- Scenario - Test Login Page of Your

App
- Scenario - Add Student Marks

Requirement Traceability Matrix (RTM)

Requirement Traceability Matrix (RTM) is a document that maps and traces user requirement with test cases.
The main purpose of Requirement Traceability Matrix is to validate that all requirements are checked via test cases such that no functionality is unchecked during Software testing.



REQUIREMENT TRACEBILITY		BUSSINESS REQUIREMENTS								
		BID001	BID002	BID003	BID004	BID005				
TEST CASES	TID001	X								
	TID002	X								
	TID003	X								
	TID004		X							
	TID005		x							
	TID006			X						
	TID007				X					
	TID008				X	X				
	TID009					X				
	TID010					X				
	TID011									

Alternate way of creating RTM

Req No	Req Desc	Testcase ID	Status
123	Login to the application	TC01,TC02,TC03	TC01-Pass TC02-Pass
345	Ticket Creation	TC04,TC05,TC06, TC07,TC08,TC09 TC010	TC04-Pass TC05-Pass TC06-Pass TC06-Fail TC07-No Run
456	Search Ticket	TC011,TC012, TC013,TC014	TC011-Pass TC012-Fail TC013-Pass TC014-No Run

Module 4: Test Case Design Techniques

In this module you learn design test cases in such a way that we get the maximum coverage using an optimal set of test cases. Focus on highlighting the various Methods and Techniques in designing test cases for both Black Box Testing and White Box testing.

VERIFICATION	VALIDATION
As per IEEE-STD-610: The process of evaluating the software to determine whether the products of a given development phase satisfy the conditions imposed at the beginning of that phase.	As per IEEE-STD-610: The process of evaluating the software during or at the end of the development process to determine whether it satisfies specified requirements
It is the process, to ensure whether we are developing the product accordingly or not.	It is the process, to validate the product which we have developed is right or not.
In simple words, Verification is verifying the documents	In simple words, Validation is to validate the actual and expected output of the software
Activities involved here are Inspections, Reviews, Walkthroughs	Activities involved in this is Testing the software application
It's a Low-Level Activity.	It's a High Level Activity.
Verification is a static method of checking documents and files.	Validation is a dynamic process of testing the real product.
Am I building a product right?	Am I building a right product?

Static Techniques:

Static Testing is a software testing technique which is used to check defects in software application without executing the code. Static testing is done to avoid errors at an early stage of development as it is easier to identify the errors and solve the errors

Static Testing Techniques

- Informal Reviews
- Walkthroughs
- Technical Reviews
- Inspections
- Static Analysis
  - Data Flow
  - Control Flow
-

In Static Testing, following things are tested

- Unit Test Cases
- Business Requirements Document (BRD)
- Use Cases
- System/Functional Requirements
- Prototype
- Prototype Specification Document
- DB Fields Dictionary Spreadsheet
- Test Data
- Traceability Matrix Document
- User Manual/Training Guides/Documentation
- Test Plan Strategy Document/Test Cases
- Automation/Performance Test Scripts

During the Review process four types of participants that take part in testing are:

- **Moderator**: Performs entry check, follow up

on rework, coaching team member, schedule the meeting.

- **Author**: Takes responsibility for fixing the defect found and improves the quality of the document
- **Scribe**: It does the logging of the defect during a review and attends the review meeting
- **Reviewer**: Check material for defects and inspects
- **Manager**: Decide on the execution of reviews and ensures the review process objectives are met.

Types of defects which can be easier to find during static testing are:

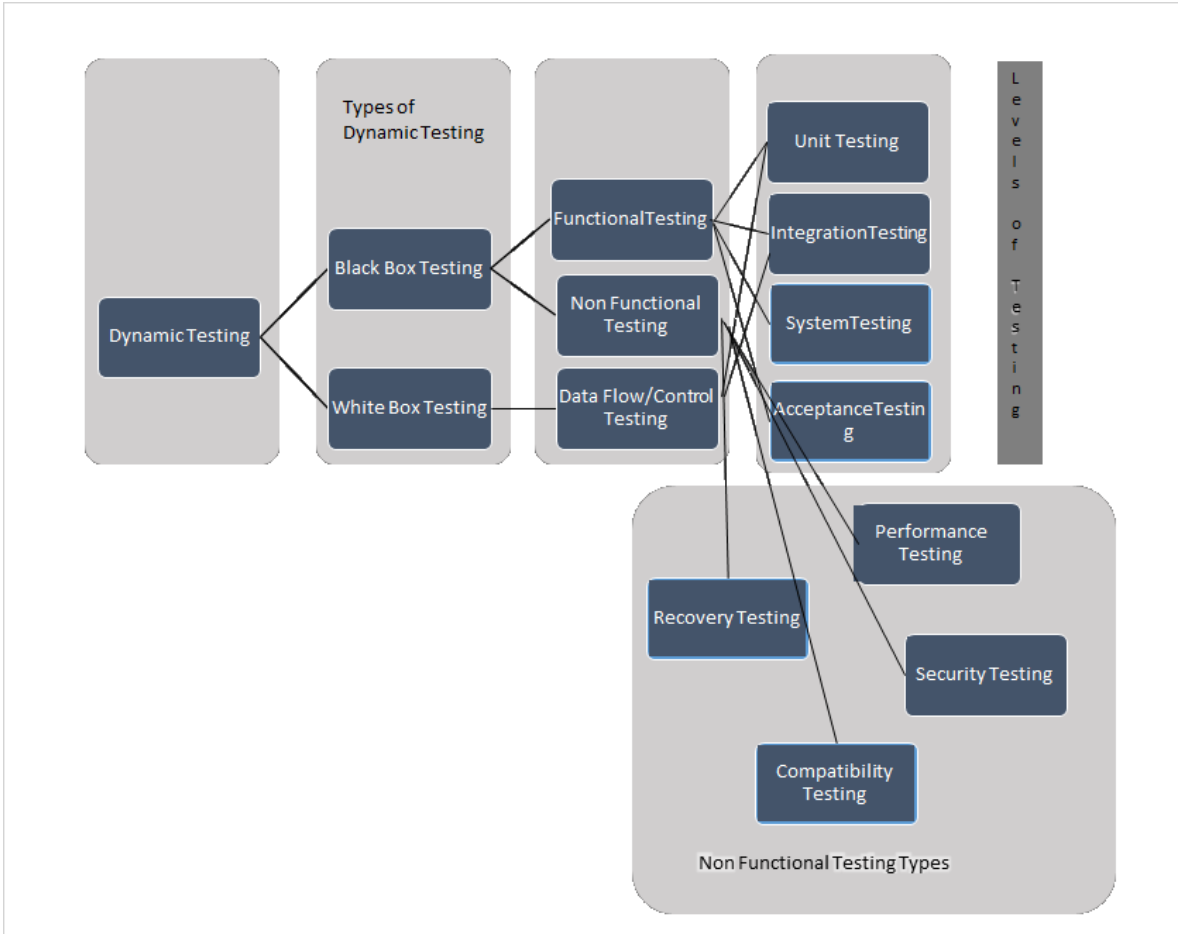
- Deviations from standards
- Non-maintainable code
- Design defects
- Missing requirements
- Inconsistent interface specifications

The various activities for performing Static Testing are:

1. **Use Cases Requirements Validation:** It validates that all the end-user actions are identified, as well as any input and output associated with them. The more detailed and thorough the use cases are, the more accurate and comprehensive the test cases can be.

2. **Functional Requirements Validation**: It ensures that the Functional Requirements identify all necessary elements. It also looks at the database functionality, interface listings, and hardware, software, and network requirements.
3. **Architecture Review**: All business level process like server locations, network diagrams, protocol definitions, load balancing, database accessibility, test equipment, etc.
4. **Prototype/Screen Mockup Validation**: This stage includes validation of requirements and use cases.
5. **Field Dictionary Validation**: Every field in the UI is defined well enough to create field level validation test cases. Fields are check for min/ max length, list values, error messages, etc.

Dynamic Techniques: The main aim of the Dynamic tests is to ensure that software works properly during and after the installation of the software ensuring a stable application without any major flaws.



WhiteBox or Structural Techniques

- [ ] Statement Coverage Testing

- [ ] Branch Coverage Testing

- [ ] Path Coverage Testing

- [?] Conditional Coverage Testing

- [?] Loop Coverage Testing

Black Box Techniques

Functional Testing:

Functional testing is performed to verify that all the features developed are according to the functional specifications, and it is performed by executing the functional test cases written by the QA team, in functional testing phase, system is tested by providing input, verifying the output and comparing the actual results with the expected results.

There are different Levels of Functional Testing out of which the most important are

- **Unit Testing** – Generally Unit is a small piece of code which is testable, Unit Testing is performed at individual unit of software and is performed by developer

- **Integration Testing** - Integration Testing is the testing which is performed after Unit

Testing and is performed by combining all the individual units which are testable and is performed either by developers or testers

- **System Testing** - System Testing is a performed to ensure whether the system performs as per the requirements and is generally performed when the complete system is ready, it is performed by testers when the Build or code is released to QA team

- **Acceptance Testing** - Acceptance testing is performed to verify whether the system has met the business requirements and is ready to use or ready for deployment and is generally performed by the end users.

Non- Functional Testing: Non-Functional testing is a testing technique which does not focus on functional aspects and mainly concentrates on the nonfunctional attributes of the system such as memory leaks, performance or robustness of the system. Non-Functional testing is performed at all test levels.

There are many Non-Functional Testing Techniques out of which the most important are

- **Performance Testing** – Performance Testing is performed to check whether the response time of the system is normal as per

the requirements under the desired network load.

- **Recovery Testing** - Recovery testing is a method to verify on how well a system is able to recover from crashes and hardware failures.
- **Compatibility Testing** – Compatibility testing is performed to verify how the system behaves across different environments.
- **Security testing** – Security testing is performed to verify the robustness of the application, i.e to ensure that only the authorizes users/roles are accessing the system
- **Usability testing** – Usability testing is a method to verify the usability of the system by the end users to verify on how comfortable the users are with the system.

Defect Severity vs. Priority

Severity is a parameter to denote the impact of a particular defect on the software.

Priority is a parameter to decide the order in which defects should be fixed.

Severity means how **severe** defect is affecting the functionality.

Priority means how fast defect has to be fixed.

Types of Severity

- **Critical**: This defect indicates complete shut-down of the process, nothing can proceed further
- **Major**: It is a highly severe defect and collapses the system. However, certain parts of the system remain functional
- **Medium**: It causes some undesirable behavior, but the system is still functional
- **Low**: It won't cause any major break-down of the system
-

Types of Priority

- **Low:** The Defect is an irritant but repair can be done once the more serious Defect has been fixed
- **Medium:** During the normal course of the development activities defect should be resolved. It can wait until a new version is created
- **High:** The defect must be resolved as soon as possible as it affects the system severely and

cannot be used until it is fixed

• Priority is associated with scheduling	• Severity is associated with functionality or standards
• Priority indicates how soon the bug should be fixed	• Severity indicates the seriousness of the defect on the product functionality
• Priority of defects is decided in consultation with the manager/client	• QA engineer determines the severity level of the defect

Examples

- **A very low severity with a high priority:** A logo error for any shipment website, can be of low severity as it not going to affect the functionality of the website but can be of high priority as you don't want any further shipment to proceed with the wrong logo.
- **A very high severity with a low priority:** Likewise, for flight operating website,

a defect in reservation functionality may be of high severity but can be a low priority as it can be scheduled to release in a next cycle.

Positive Test Cases

Positive test cases test valid inputs to verify the software is doing what it's supposed to do. The main purpose of this is to make sure the system isn't throwing errors when it's not supposed to. In general, positive testing ensures the system meets the requirements under positive scenarios and day-to-day use.

Examples of Positive Test Cases

Let's consider a password creation field that should accept 5-10 characters and include only letters and/or numbers. Here are some positive test case examples:

Password field should accept 5 characters
Password field should accept 10 characters
Password field should accept all numbers
Password field should accept all letters
Password field should accept a combination of letters & numbers

Negative Test Cases

Negative test cases test invalid inputs and verify that the software is *not* doing what it is not supposed to do. The intent of Negative testing is to ensure the system validates against invalid inputs by throwing errors or otherwise not

allowing the system to perform in a certain away. Let's consider the same password creation field example above and look at some examples of negative test cases.

Examples of Negative Test Cases

- Password field should not accept 1-4 characters
- Password field should not accept any characters outside of numbers or letter (!@#.. etc)
- Password field should not accept 11 characters

In each of these tests, the system should not accept the input and it should result in some form of validation. The system should handle these validations gracefully, whether through error messages or other UI approaches to validation.

Destructive Test Cases

The purpose of destructive test cases is to test what the system can handle until it breaks or "destructs". Load testing and script injections are common approaches to destructive testing. Here are some examples of destructive testing.

Examples of Destructive Test Cases

- Applying a significant load to the system which results in a failure
- Injecting JavaScript into a web form with malicious intent
- Fast-clicking in attempt to break a webpage

⍰ Boundary Value Analysis
Testing technique

Boundary value analysis is based on testing at the boundaries between partitions. It includes maximum, minimum, inside or outside boundaries, typical values and error values.

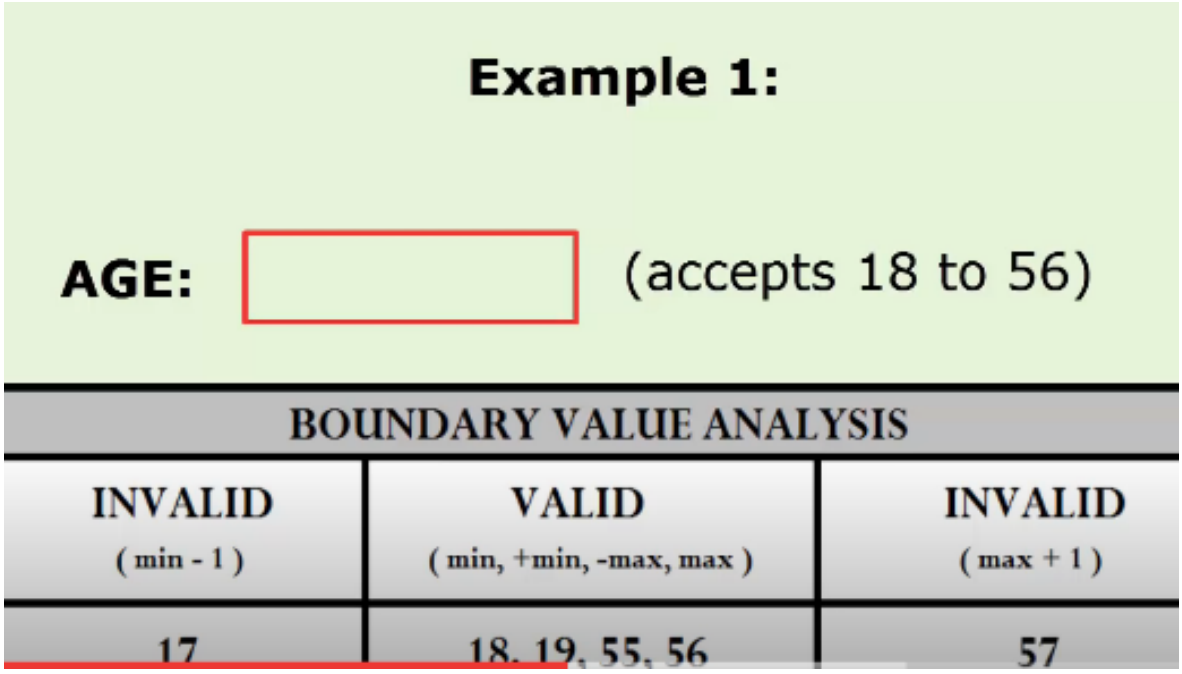
It is generally seen that a large number of errors occur at the boundaries of the defined input values rather than the center. It is also known as BVA and gives a selection of test cases which exercise bounding values.

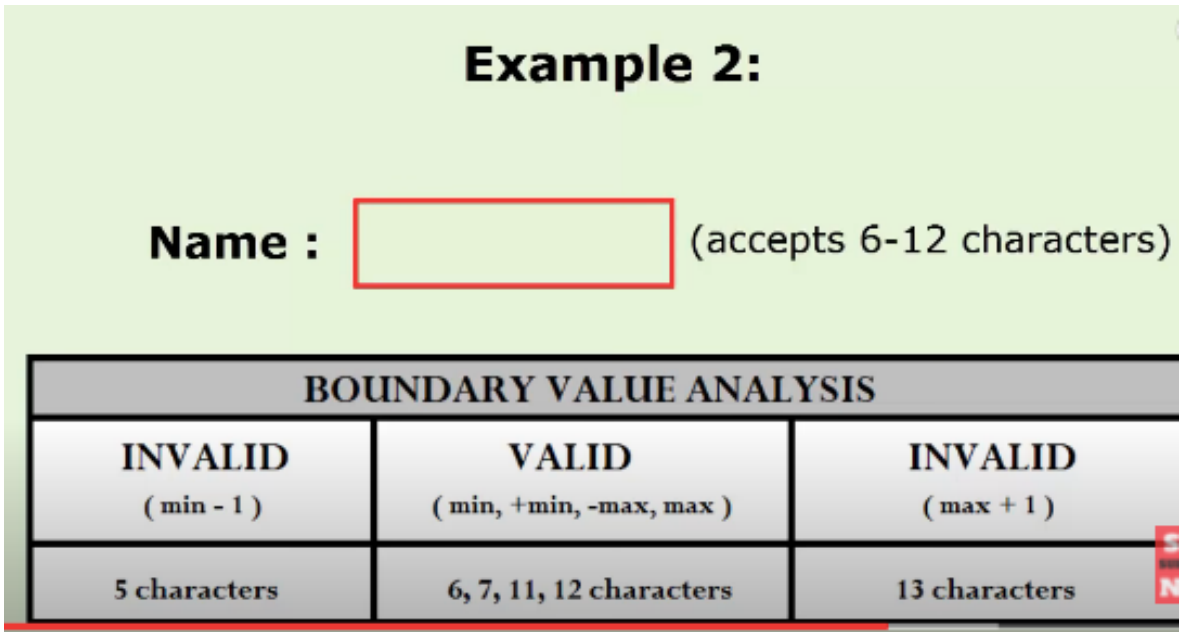
Guidelines for Boundary Value analysis

- If an input condition is restricted between values x and y, then the test cases should be designed with values x and y as well as values which are above and below x and y.
- If an input condition is a large number of values, the test case should be developed which need to exercise the minimum and maximum numbers. Here, values above and below the minimum and maximum values are also tested.

```
Input condition is valid between
```

```
1 to 10
   Boundary values 0,1,2 and
9,10,11
```





⍰ Equivalence Class Partition

Equivalent Class Partitioning allows you to divide set of test condition into a partition which should be considered the same. This software testing method divides the input domain of a program into classes of data from which test cases should be designed.

The concept behind this technique is that test case of a representative value of each class is equal to a test of any other value of the same class. It allows you to Identify valid as well as invalid equivalence classes.

Example:

Input conditions are valid between

```
1 to 10 and 20 to 30
```

Hence there are five equivalence classes

```
--- to 0 (invalid)
1 to 10 (valid)
11 to 19 (invalid)
20 to 30 (valid)
31 to --- (invalid)
```

You select values from each class, i.e.,

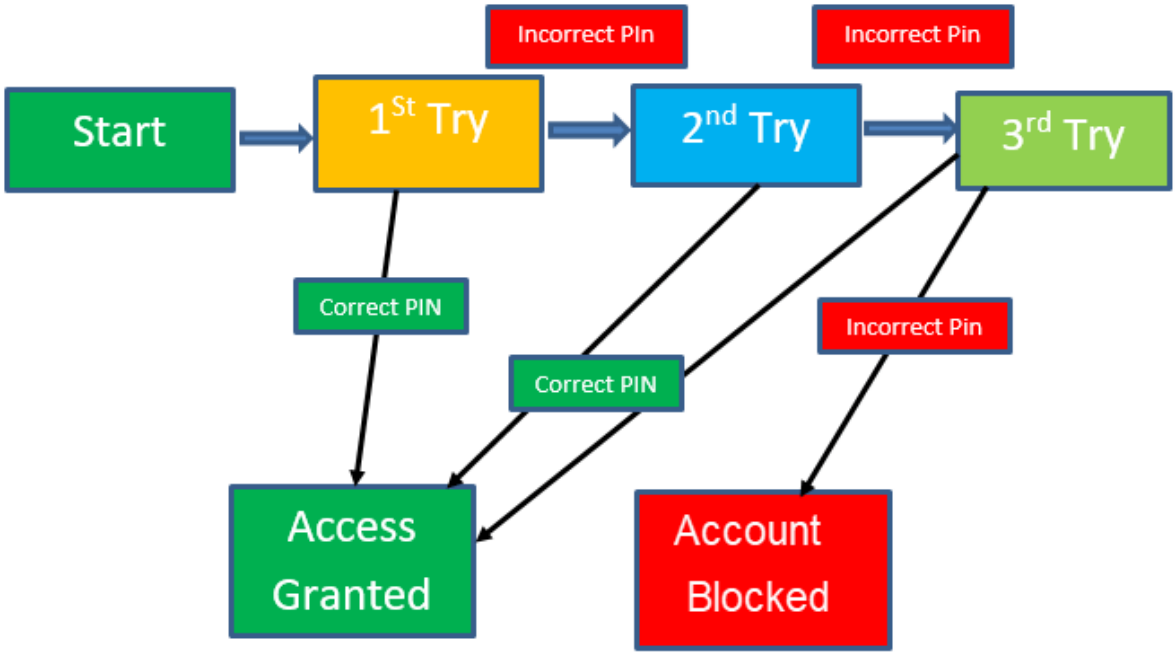
```
-2, 15, 45. – Negative Test
5, 27 - Positive Test
```

? State Transition Technique

In State Transition technique changes in input conditions change the state of the Application Under Test (AUT). This testing technique allows the tester to test the behavior of an AUT. The tester can perform this action by entering various input conditions in a sequence.

Example:

In the following example, if the user enters a **valid password in any of the first three attempts** the user will be able to log in successfully. If the user enters the invalid password in the first or second try, the user will be prompted to re-enter the password. When the user enters password incorrectly 3rd time, the action has taken, and the account will be blocked.



State Transition Table

	Correct PIN	Incorrect PIN
S1) Start	S5	S2
S2) 1st attempt	S5	S3
S3) 2nd attempt	S5	S4
S4) 3rd attempt	S5	S6
S5) Access Granted	-	-
S6) Account blocked	-	-

In the above-given table when the user enters the correct PIN, the state is transitioned to Access granted. And if the user enters an incorrect password, he or she is moved to next state. If he does the same 3rd time, he will reach the account blocked state.

⍰ Decision Table

This software testing technique is used for functions which respond to a combination of inputs or events. **For example, a submit button should be enabled if the user has entered all required fields.**

The first task is to identify functionalities where the output depends on a combination of inputs. If there are large input set of combinations, then divide it into smaller subsets which are helpful for managing a decision table.

For every function, you need to create a table and list down all types of combinations of inputs and its respective outputs. This helps to identify a condition that is overlooked by the tester.

- List the inputs in rows
- Enter all the rules in the column
- Fill the table with the different combination of inputs
- In the last row, note down the output against the input combination.

Example: A submit button in a contact form is enabled only when all the inputs are entered by the end user.

	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5	Rule 6	Rule 7	Rule 8
Input								
Name	F	T	F	T	F	T	F	T
Email	F	F	T	T	F	F	T	T
Message	F	F	F	F	T	T	T	T
Output								
Submit	F	F	F	F	F	F	F	T

Conditions	Rule 1	Rule 2	Rule 3	Rule 4	Rule 5
Credit status = accepted	–	Y	–	–	Y
Credit status <> accepted	–	–	–	Y	–
Credit limit = 0	–	Y	–	–	N
Credit limit > 0	–	–	–	–	Y
Credit limit = blank/unknown	Y	–	–	N	N
Next review date = blank/unknown	–	–	Y	N	N
Next review date = valid date	–	–	–	Y	Y
Outcomes					
Run credit rules				X	X
Fail credit processing	X	X	X		

Field Validation Table (FVT)

Generally, each and every field in the application needs to be validated thoroughly to ensure or identify defects which may get unnoticed in the fields. This technique is very much useful in

identifying those underlying flaws in the fields.

Data Type	Valid Inputs	Invalid Inputs
Integers or Numbers	• Only Numbers • Less than the limit (N) • Enter the value within the limit (N + 1)/2	• More than the limit (N + 1) • Numbers with precision • Numbers in Exponential Form • Negative Integers • Only Alphabets • Numbers + Alphabets • Numbers + Special Characters • Unicode Characters e.g. U+0000, U+0001

String	• Only Alphabets • Only Numbers • Only Special Characters • Numbers + Alphabets • Numbers + Special Characters • Alphabets + Special Characters • Less than the limit (N) • Enter the value within the limit (N + 1)/2	• More than the limit (N + 1) • Unicode Characters e.g. U+0000, U+0001

Date	• Check that whether date picker is present or not • Check that date field is non editable • Ensure that, upon right clicking on the date field, paste option should be **disabled** & copy option should be **enabled** • Ensure that, upon clicking on date in the calendar, it should be displayed in the date field • Select a leap year and verify the days in February month • Select a non leap year and verify the days in February month • Ensure that, calendar is having provision to select any year, month (combo box, Drop down list, links etc.) • Ensure that, clear button is present in the date picker to remove the selected date

Application specific table should also be created with the application specific fields and the other columns. This mainly helps to validate each and every field in the application and clearly specifies where the defect lies and on which test data.

S. No	User Story # or Requirement ID	TC ID	Feature	Section	Mandatory (Y/N)	Field Type	Data Type	Field Size	Default Value	Is it Editable?	Is Special Characters Allowed?	Results
1	12345	10	Login	User ID	Y	Text	Alphanumeric	14	NA	Y	N	Pass
				Password	Y	Text	Alphanumeric	10	NA	Y	Y	Pass
				Submit	Y	Button	NA	NA	NA	NA	NA	Pass
				Cancel	N	Button	NA	NA	NA	NA	NA	Pass

General Test Scenarios

1. All mandatory fields should be validated and indicated by an asterisk (*) symbol.
2. Validation error messages should be displayed properly in the correct position.
3. All error messages should be displayed in the same CSS style (**For Example,** using red color)
4. General confirmation messages should be displayed using CSS style other than error messages style (**For Example,** using green color)
5. Tooltips text should be meaningful.
6. Drop-down fields should have the first entry as blank or text like ‘Select’.

7. ‘Delete functionality’ for any record on a page should ask for a confirmation.
8. Select/deselect all records option should be provided if page supports record add/delete/ update functionality
9. Amount values should be displayed with correct currency symbols.
10. Default page sorting should be provided.
11. Reset button functionality should set default values for all fields.
12. All numeric values should be formatted properly.
13. Input fields should be checked for the max field value. Input values greater than the specified max limit should not be accepted or stored in the database.
14. Check all input fields for special characters.
15. Field labels should be standard e.g. field accepting user’s first name should be labeled properly as ‘First Name’.
16. Check page sorting functionality after add/edit/ delete operations on any record.
17. Check for timeout functionality. Timeout values should be configurable. Check application behavior after the operation timeout.
18. Check cookies used in an application.
19. Check if downloadable files are pointing to the correct file paths.
20. All resource keys should be configurable in

config files or database instead of hard coding.
21. Standard conventions should be followed throughout for naming resource keys.
22. Validate markup for all web pages (validate HTML and CSS for syntax errors) to make sure it is compliant with the standards.
23. Application crash or unavailable pages should be redirected to the error page.
24. Check the text on all pages for spelling and grammatical errors.
25. Check numeric input fields with character input values. A proper validation message should appear.
26. Check for negative numbers if allowed for numeric fields.
27. Check the number of fields with decimal number values.
28. Check the functionality of buttons available on all pages.
29. The user should not be able to submit a page twice by pressing the submit button in quick succession.
30. Divide by zero errors should be handled for any calculations.
31. Input data with the first and last position blank should be handled correctly.

GUI And Usability Test Scenarios

1. All fields on a page (**For Example,** text box, radio options, drop-down lists) should be aligned properly.
2. Numeric values should be justified correctly unless specified otherwise.
3. Enough space should be provided between field labels, columns, rows, error messages, etc.
4. The scrollbar should be enabled only when necessary.
5. Font size, style, and color for headline, description text, labels, infield data, and grid info should be standard as specified in SRS.
6. The description text box should be multi-lined.
7. Disabled fields should be greyed out and users should not be able to set focus on these fields.
8. Upon click of an input text field, the mouse arrow pointer should get changed to the cursor.
9. The user should not be able to type in drop-down select lists.
10. Information filled by users should remain intact when there is an error message on page submit. The user should be able to submit the form again by correcting the errors.
11. Check if proper field labels are used in error messages.
12. Drop-down field values should be displayed in defined sort order.
13. Tab and Shift+Tab order should work properly.
14. Default radio options should be pre-selected

on the page load.
15. Field-specific and page-level help messages should be available.
16. Check if the correct fields are highlighted in case of errors.
17. Check if the drop-down list options are readable and not truncated due to field size limits.
18. All buttons on a page should be accessible by keyboard shortcuts and the user should be able to perform all operations using a keyboard.
19. Check all pages for broken images.
20. Check all pages for broken links.
21. All pages should have a title.
22. Confirmation messages should be displayed before performing any update or delete operation.
23. Hourglass should be displayed when the application is busy.
24. Page text should be left-justified.
25. The user should be able to select only one radio option and any combination for checkboxes.

Test Scenarios For Filter Criteria

1. The user should be able to filter results using all parameters on the page.
2. Refine search functionality should load the search page with all user-selected search parameters.
3. When there are at least one filter criteria

required to perform the search operation, make sure the proper error message is displayed when the user submits the page without selecting any filter criteria.
4. When at least one filter criteria selection is not compulsory, the user should be able to submit the page and the default search criteria should get used to query results.
5. Proper validation messages should be displayed for all invalid values for filter criteria.

Database Testing Test Scenarios

1. Check if correct data is getting saved in the database upon a successful page submit.
2. Check values for columns that are not accepting null values.
3. Check for data integrity. Data should be stored in single or multiple tables based on the design.
4. Index names should be given as per the standards e.g.
IND_<Tablename>_<ColumnName>
5. Tables should have a primary key column.
6. Table columns should have description information available (except for audit columns like created date, created by, etc.)
7. For every database add/update operation log should be added.

8. Required table indexes should be created.
9. Check if data is committed to the database only when the operation is successfully completed.
10. Data should be rolled back in case of failed transactions.
11. Database name should be given as per the application type i.e. test, UAT, sandbox, live (though this is not a standard it is helpful for database maintenance)
12. Database logical names should be given according to the database name (again this is not standard but helpful for DB maintenance).
13. Stored procedures should not be named with a prefix “sp”
14. Check if values for table audit columns (like created date, created by, updated, updated by, is deleted, deleted data, deleted by, etc.) are populated properly.
15. Check if input data is not truncated while saving. Field length shown to the user on the page and in database schema should be the same.
16. Check numeric fields with minimum, maximum, and float values.
17. Check numeric fields with negative values (for both acceptance and non-acceptance).
18. Check if the radio button and drop-down list options are saved correctly in the database.
19. Check if the database fields are designed with the correct data type and data length.

20. Check if all the table constraints like a Primary key, Foreign key, etc. are implemented correctly.
21. Test stored procedures and triggers with sample input data.
22. Input field leading and trailing spaces should be truncated before committing data to the database.
23. Null values should not be allowed for the Primary key column.

Test Scenarios For Image Upload Functionality

(Also applicable for other file upload functionality)
1. Check for uploaded image path.
2. Check image upload and change functionality.
3. Check image upload functionality with image files of different extensions (**For Example,** JPEG, PNG, BMP, etc.)
4. Check image upload functionality with images having space or any other allowed special character in the file name.
5. Check duplicate name image upload.
6. Check image upload with image size greater than the max allowed size. The Proper error message should be displayed.
7. Check image upload functionality with file types other than images (**For Example,** txt, doc, pdf, exe, etc.). A proper error message should be

displayed.
8. Check if images of specified height and width (if defined) are accepted otherwise rejected.
9. The image upload progress bar should appear for large size images.
10. Check if the cancel button functionality is working in between the upload process.
11. Check if file selection dialog shows only supported files listed.
12. Check multiple images upload functionality.
13. Check image quality after upload. Image quality should not be changed after upload.
14. Check if the user is able to use/view the uploaded images.

Performance Testing Test Scenarios

1. Check if the page load time is within the acceptable range.
2. Check the page load on slow connections.
3. Check the response time for any action under a light, normal, moderate, and heavy load conditions.
4. Check the performance of database stored procedures and triggers.
5. Check the database query execution time.
6. Check for load testing of the application.
7. Check for the Stress testing of the application.
8. Check CPU and memory usage under peak

load conditions.

Security Testing Test Scenarios

1. Check for SQL injection attacks.
2. Secure pages should use the HTTPS protocol.
3. Page crash should not reveal application or server info. The error page should be displayed for this.
4. Escape special characters in the input.
5. Error messages should not reveal any sensitive information.
6. All credentials should be transferred over an encrypted channel.
7. Test password security and password policy enforcement.
8. Check application logout functionality.
9. Check for Brute Force Attacks.
10. Cookie information should be stored in encrypted format only.
11. Check session cookie duration and session termination after timeout or logout.
11. Session tokens should be transmitted over a secured channel.
13. The password should not be stored in cookies.
14. Test for Denial of Service attacks.
15. Test for memory leakage.
16. Test unauthorized application access by manipulating variable values in the browser

address bar.
17. Test file extension handing so that exe files are not uploaded and executed on the server.
18. Sensitive fields like passwords and credit card information should not have to autocomplete enabled.
19. File upload functionality should use file type restrictions and also anti-virus for scanning uploaded files.
20. Check if directory listing is prohibited.
21. Passwords and other sensitive fields should be masked while typing.
22. Check if forgot password functionality is secured with features like temporary password expiry after specified hours and security question is asked before changing or requesting a new password.
23. Verify CAPTCHA functionality.
24. Check if important events are logged in log files.
25. Check if access privileges are implemented correctly.

Bug Reporting

How to write a Good Bug Report?

Defect ID: Add a Defect ID using a naming convention followed by your team. The Defect ID will be generated automatically in case of defect management tool.

Title/Summary: Title should be short and simple. It should contain specific terms related to the actual issue. Be specific while writing the title.

Good: "Uploading a JPEG file (Profile Picture) in the Registration Page crashes the system".

Bad: "System crashes".

Reporter Name: Name of the one who found the defect (Usually tester's name but sometimes it might be Developer, Business Analyst, Subject Matter Expert (SME), Customer)

Defect Reported Date: The date on which you have found the bug.

Who Detected: Specify the designation of the one who found the defect.
Example: QA, Developer, Business Analyst, SME, Customer

How to write a Good Bug Report?

How Detected: Specify on how you have detected such as *while doing Testing* or *while doing Review* or *while giving Walkthrough* etc.,

Project Name: Mention the name of the project

Release/Build Version: On which release this issue occurs. Mention the build version details clearly.

Defect/Enhancement: If the system is not behaving as intended then you need to specify it as a Defect. If its just a request for a new feature then you must specify it as Enhancement.

Environment: You must mention the details of Operation Systems, Browser Details and any other related to the test environment in which you have encountered the bug.

Example: Windows 8, Chrome 48

Priority: It defines how soon the bug should be fixed. Usually, the priority of the bug is set by the Managers. Based on the priority, developers could understand how soon it must be fixed and set the order in which a bug should be resolved.

How to write a Good Bug Report?

Status: Specify the status of the bug. If you just found a bug and about to post it then the status will be "New". In the course of bug fixing, the status of the bug will change.

Example:
1. New
2. Assigned
3. Open
4. Fixed
5. Test
6. Verified
7. Closed
8. Reopen
9. Duplicate
10. Deferred
11. Rejected
12. Cannot be fixed
13. Not Reproducible
14. Need more information

How to write a Good Bug Report?

Description: In the description section, you must briefly explain what you have done before facing the bug.

Steps To Reproduce: You should describe how to reproduce the bug in step by step manner. Easy to follow steps give room to the developers to fix the issue without any chaos. These steps should describe the bug well enough and allows developers to understand and act on the bug without discussing to the one who wrote the bug report. Start with "opening the application", include "pre requisites" and write till the step which "causes the bug".

Good:
i. Open URL "Your URL"
ii. Click on "Registration Page"
iii. Upload "JPEG" file in the profile photo field

Bad: Upload a file in the registration page.

How to write a Good Bug Report?

Expected Result: What is the expected output from the application when you make an action which causes failure.

Good: A message should display "Profile picture uploaded successfully"

Bad: System should accept the profile picture.

Actual Result: What is the expected output from the application when you make an action which causes failure.

Good: "Uploading a JPEG file (Profile Picture) in the Registration Page crashes the system".

Bad: System is not accepting profile picture.

Attachments: Attach the screenshots which you had captured when you faced the bug. It helps the developers to see the bug which you have faced.

SAMPLE BUG REPORT

Bug Name: Application crash on clicking the SAVE button while creating a new user.
Bug ID: (It will be automatically created by the BUG Tracking tool once you save this bug)
Area Path: USERS menu > New Users
Build Number: Version Number 5.0.1
Severity: HIGH (High/Medium/Low) or 1
Priority: HIGH (High/Medium/Low) or 1
Assigned to: Developer-X
Reported By: Your Name
Reported On: Date
Reason: Defect
Status: New/Open/Active (Depends on the Tool you are using)
Environment: Windows 2003/SQL Server 2005

Description:
Application crash on clicking the SAVE button while creating a new
the user, hence unable to create a new user in the application.

Steps To Reproduce:
1) Login into the Application
2) Navigate to the Users Menu > New User
3) Filled all the user information fields
4) Clicked on the ‘Save’ button
5) Seen an error page “ORA1090 Exception: Insert values Error…”

6) See the attached logs for more information (Attach more logs related to bug..IF any)
7) And also see the attached screenshot of the error page.

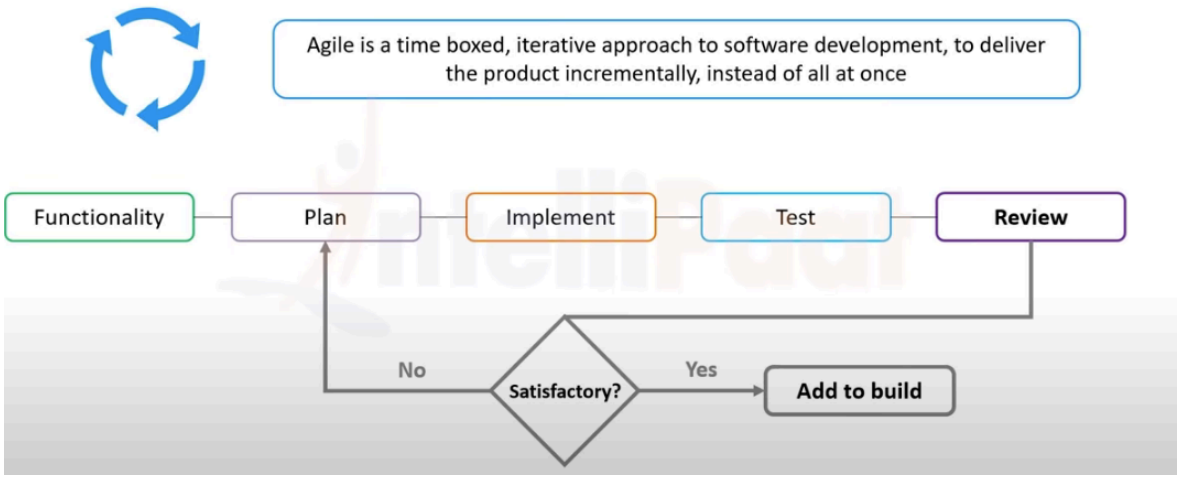
Expected Result: On clicking SAVE button, should be prompted to a success message “New User has been created successfully”.

(Attach ‘application crash’ screenshot. IF any)

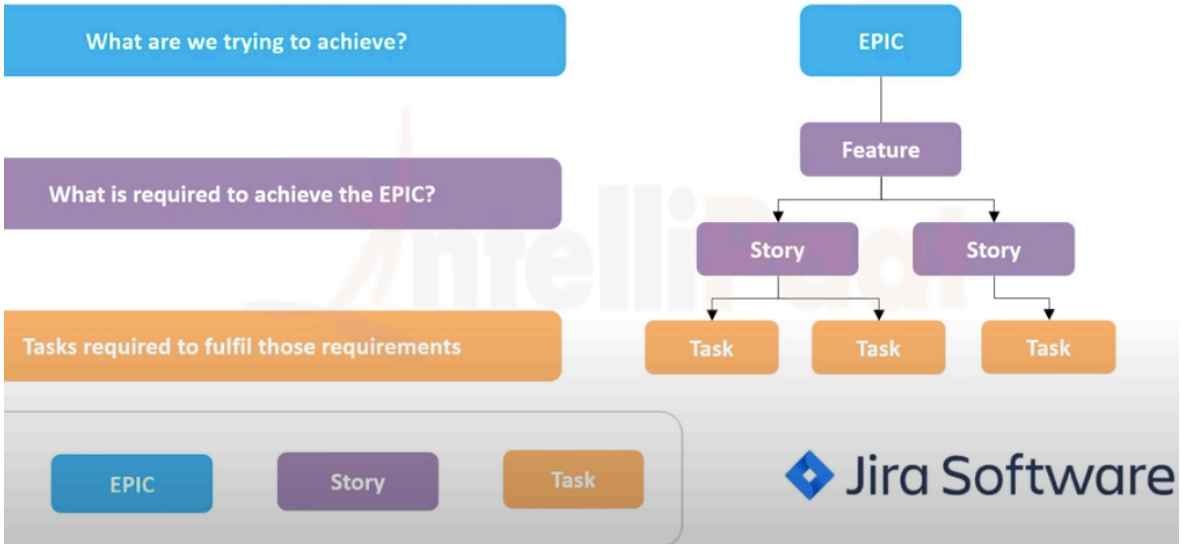
Save the Defect/Bug in the BUG TRACKING TOOL. You will get a Bug id, which you can use for further bug reference.
Default ‘New Bug’ mail will go to the respective developer and the default module owner (Team leader or manager) for further action.

Agile Methodology and Scrum

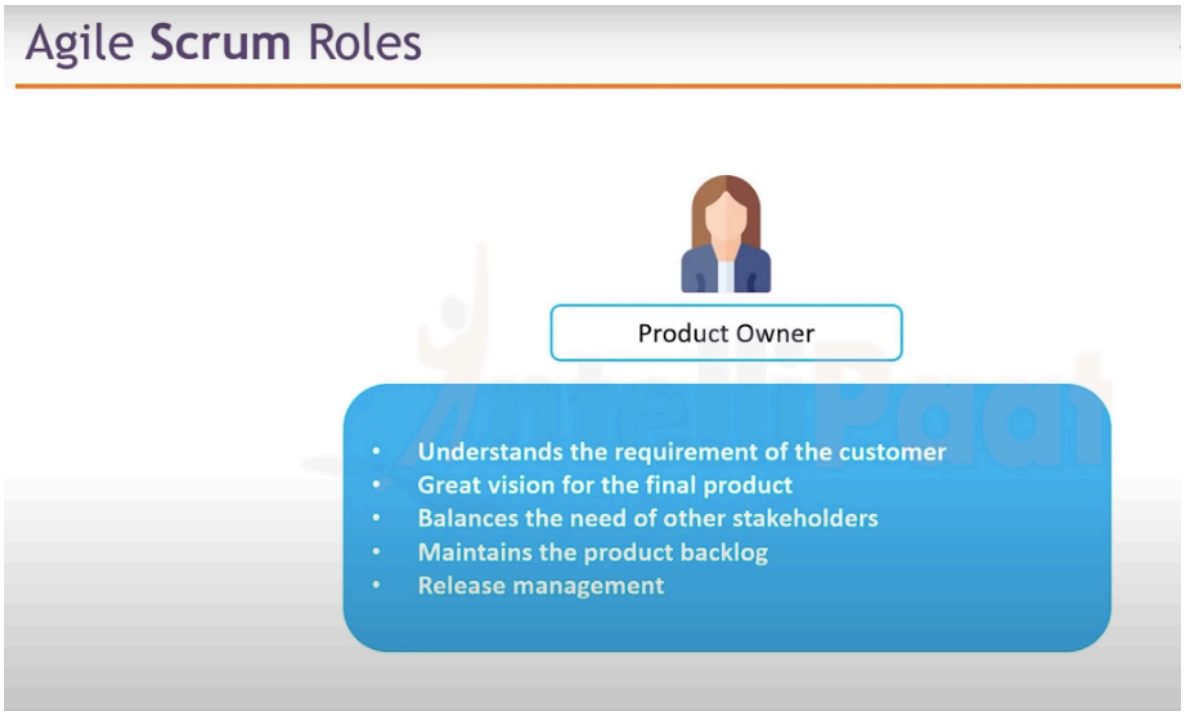
1.



2. Agile Terminology .



3.



4.

Agile **Scrum** Roles

Scrum Master

- Responsible for gluing everything together
- Sometimes act as intermediatory between the product owner and dev team
- Helps in planning and breaking down work
- Manages backlog, ensures completion
- Ensures transparency

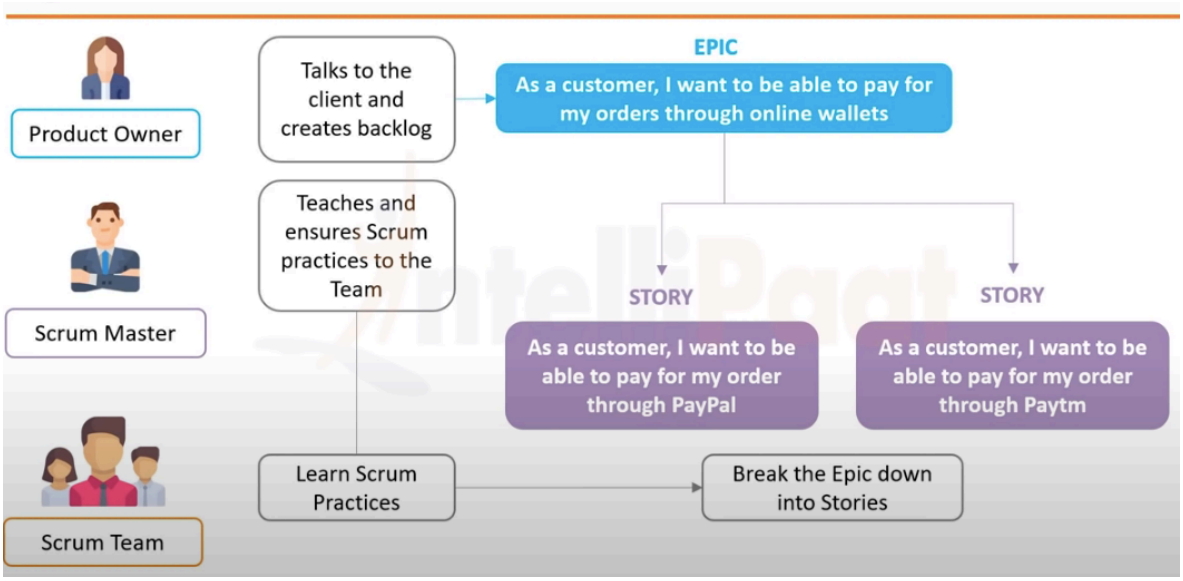
5.

Agile **Scrum** Roles

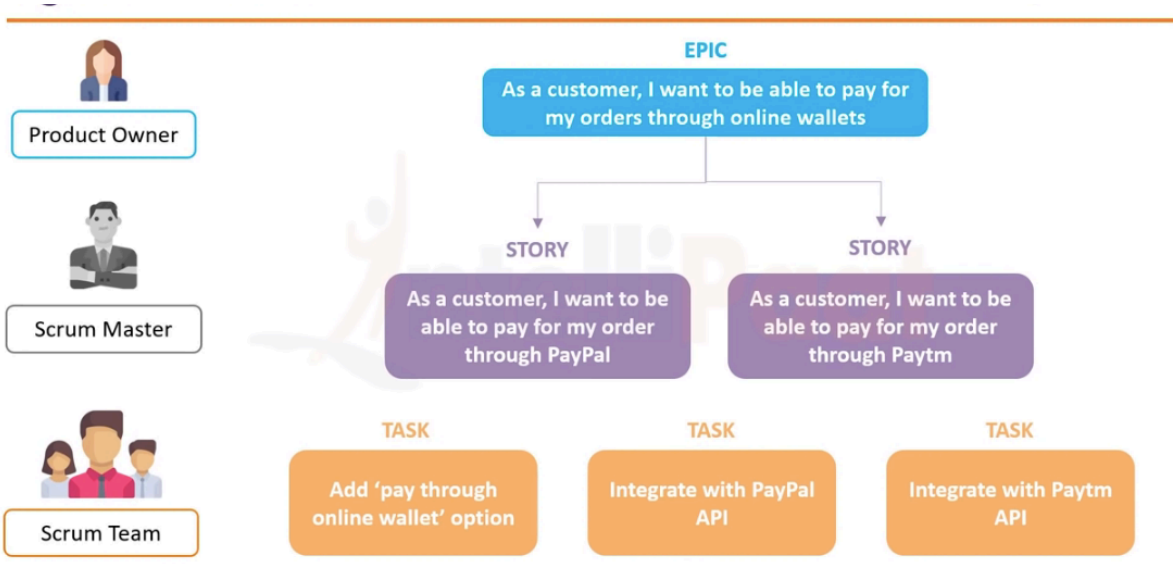
Scrum Team

- Consists of Software Developers & Quality Assurance
- Break down the work into subtasks
- Deal with the planning, implementation, testing and preliminary reviewing of goals
- Ensure timely delivery, with quality assurance
- Communicate their progress with the Scrum master

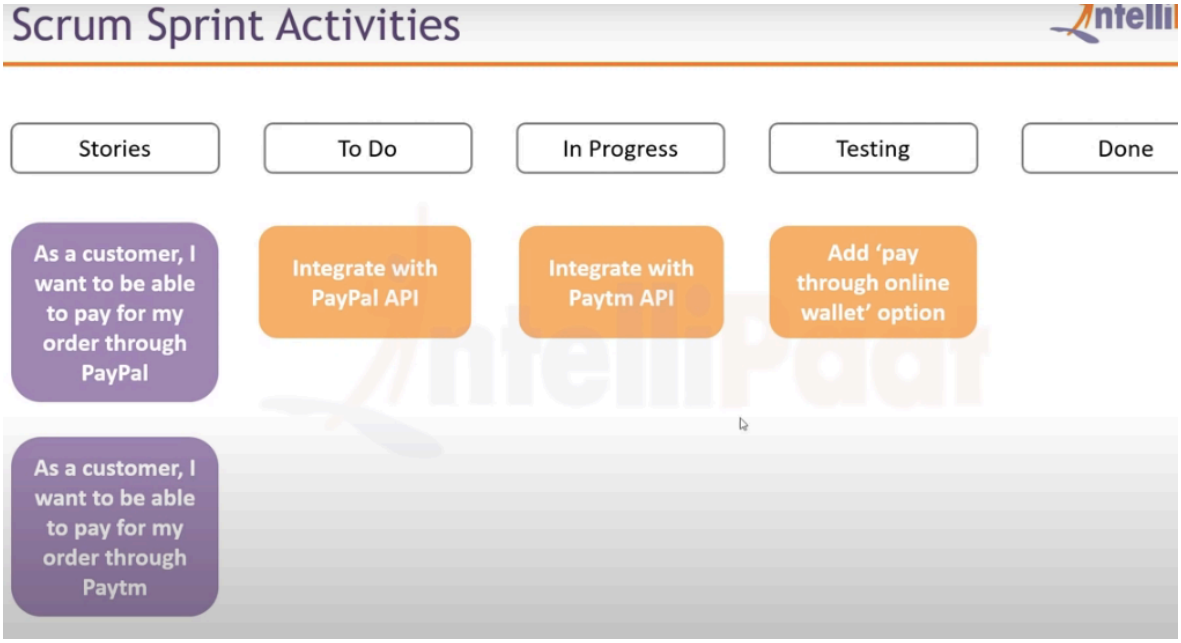
6.


Product Owner
Scrum Master
Scrum Team
Talks to the client and creates backlog
Teaches and ensures Scrum practices to the Team
Learn Scrum Practices
EPIC
As a customer, I want to be able to pay for my orders through online wallets
STORY
As a customer, I want to be able to pay for my order through PayPal
STORY
As a customer, I want to be able to pay for my order through Paytm
Break the Epic down into Stories

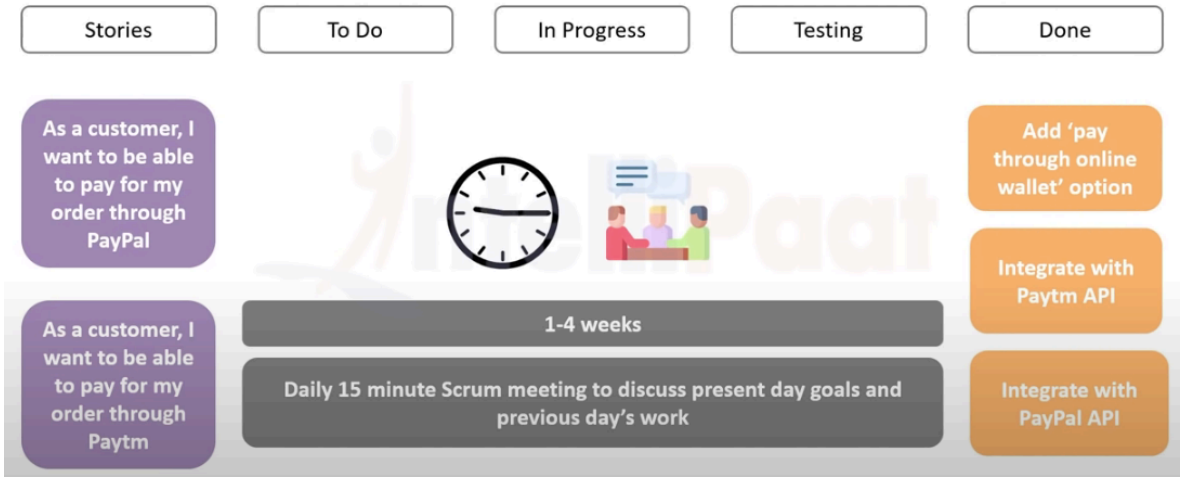
7.


Product Owner
Scrum Master
Scrum Team
EPIC
As a customer, I want to be able to pay for my orders through online wallets
STORY
As a customer, I want to be able to pay for my order through PayPal
STORY
As a customer, I want to be able to pay for my order through Paytm
TASK
Add 'pay through online wallet' option
TASK
Integrate with PayPal API
TASK
Integrate with Paytm API

8.


Scrum Sprint Activities
Stories
To Do
In Progress
Testing
Done
As a customer, I want to be able to pay for my order through PayPal
Integrate with PayPal API
Integrate with Paytm API
Add 'pay through online wallet' option
As a customer, I want to be able to pay for my order through Paytm

9.


Stories
To Do
In Progress
Testing
Done
As a customer, I want to be able to pay for my order through PayPal
As a customer, I want to be able to pay for my order through Paytm
1-4 weeks
Daily 15 minute Scrum meeting to discuss present day goals and previous day's work
Add 'pay through online wallet' option
Integrate with Paytm API
Integrate with PayPal API