Manual Testing

**Module 1: Testing Concepts (Theory) What to test?**

* What is software & types of software?
* What is software testing?
* What is software quality?
* Project vs Product
* Why do we need testing?
* Error, bug & failure
* Why the software has bugs?
* SDLC & STLC
* Waterfall Model
* Spiral Model
* V Model
* QA & QC & QE
* Different levels of software testing
* White box & black box testing
* Static testing & Dynamic testing
* Verification & validation
* System testing types
* GUI testing
* Functional & non-functional testing
* Test design techniques
* Re-testing & regression testing
* Exploratory testing
* Ad-hoc testing
* Sanity & smoke testing
* End-to-end testing
* Software Testing Life Cycle
* Use case, test scenario & test case
* Test environment and execution
* Defect reporting
* Test closure
* Test metrics

**Module 2: Testing Project (Practical) How to test?**

* Project introduction
* Understanding functional requirements from FRS
* Creating test scenarios
* Creating test cases
* Test execution
* Bug reporting & tracking
* Test sign off

**Module 3: Agile Process & Jira Tool**

Agile/Scrum process:

* What’s agile
* What’s scrum/scrum team
* What’s sprint
* What is user story
* How to give story points/how to estimate user story
* What’s definition of done and definition of ready
* Different sprint activities
* Sprint planning/backing refinement/sprint review/sprint retrospective

Jira Tool:

* How to install and configure Jira tool
* How to create epic/user stories in Jira
* Creating sprints in Jira
* Backlogs in Jira
* Creating bugs in Jira
* How to write test cases in Jira with Zephyr plugin
* Creating test cycles and execute test cases in Jira

**What is software?**

Software: a collection of computer programs that helps us to perform a task.

**Types of software**

* System software will be used to run the system

Eg: device drivers, operating system, servers, utilities

* Programming software processes the input internally and provides the output

Eg: compilers, debuggers, interpreters

* Application software is the end user will be using the software

Eg: web app, mobile app, desktop app

**What is software testing?**

* A part of software development process.
* An activity to detect and identify the defects in the software.
* The objective of testing is deliver and release the quality of the product customers. It should follow the customer requirements.

**What is software quality?**

* Bug-free: impossible to 100% bug free, but 1-2% is possible or minor bug
* Delivered on time
* Within budgets
* Meets requirements and/or expectation
* Maintainable: after installing the software in customer environment they should able to resolve minor issue and user friendly

**Project vs Product**

Project

* Software app is developed for specific customer based on the requirement
* Only specific customer can be used it
* Eg: banking system

Product

* Software app is developed for multiple customers based on the market requirement
* Can be used by everyone/multiple users
* Eg: Google

**Why do we need testing?**

To release and deliver the quality product and should follow the customer requirements. Ensure the software bug-free and working as the customer requirements or not.

**Error, bug/defect & failure**

* Error: human mistakes made by developer during development
* Bug: informal term for a defect due to coding error
* Defect: flaw in software that causes it to behave unexpectedly related to requirements
* Failure: inability of the software to perform as required

**ERROR 🡪 DEFECT/BUG 🡪 FAILURE**

**Why the software has bugs?**

* Miscommunication or no communication between developers and testers
* Software complexity long time of development, numbers of developers & testers, many modules, have integration with external software
* Programming errors
* Frequently changing requirements
* Lack of skilled testers

**Software Development Life Cycle (SDLC)**

A step by step process used by software industry to design, develop, test and deliver the software to the customers.

P: people

P: process

P: product

**REQUIREMENT ANALYSIS 🡪 DESIGN 🡪 DEVELOPMENT 🡪 TESTING 🡪 MAINTENANCE**

Requirement analysis

* Collect, understand and prepare the requirements from customer

Design

* The designer will design the software; UI, navigation

Development

* The software developer team will start writing codes

Testing

* Before deliver the software to the customer, tester will test the software; functional/non-functional, security

Maintenance

* After deliver the software, maintenance will start

**Waterfall Model – Traditional Model; Long Term Project**

Requirement Analysis

* The project's requirements are gathered and documented
* Understanding the needs of the stakeholders and ensuring that all requirements are clear and complete

System Design

* Based on the requirements, the system's architecture and design are created
* Creating design documents and specifications that serve as a blueprint for the implementation phase

Implementation

* The actual code is written based on the design documents
* Developers build the software by translating design specifications into functional code

Testing

* After implementation, the software is tested to identify and fix defects
* Ensures that the software meets the specified requirements and works as intended

Deployment

* Once testing is complete, the software is deployed to the production environment
* Involves installing the software for end-users and making it operational

Maintenance

* After deployment, the software enters the maintenance phase
* Includes making updates, fixing bugs, and making improvements based on user feedback and changing requirements

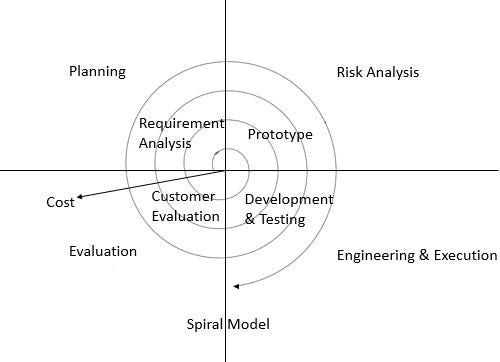
Advantages

* Quality of the product will be good; every phase has detailed and clear documentation
* Since requirements changes are not allowed; chances of finding bugs will be less
* Initial investment is less since the testers are hired at the later stages
* Preferred for small projects where requirements are freezed; every phase has static documentations, cannot modify the documents

Disadvantages

* Requirement changes are not allowed
* If there is defect in requirement phase, that will be continued in later phase; in system design there will be wrong, implementation will be wrong and so on
* Total investment is more because time taking for rework on defect is time consuming which leads to high investment; if find any defects in any phase, have to go back to the previous phase and modify the documents
* Testing will start only after coding/development; only have one test phase

**Spiral Model - Iterative software development process; combine design and prototyping; suitable for product market requirement**



Planning – Requirement Analysis

* Define objectives, identify alternative solutions, and plan the development process for the next iteration
* Gathering requirements and setting goals for the project

Risk Analysis – Prototype

* Identify and analyse potential risks and uncertainties
* Develop strategies to mitigate these risks
* Helps in understanding what might go wrong and how to handle it

Engineering & Execution – Development & Testing (Design & Develop)

* Develop and test the product
* Involves actual coding, testing, and implementation based on the planning and risk analysis done in the previous steps

Evaluation – Customer Evaluation; release to customer v1, after completing then release v2

* Review and evaluate the progress and the product developed so far
* Gather feedback from stakeholders and make necessary adjustments
* Ensures that the project is on track and meets the requirements

Each loop of the spiral represents one iteration, and the product is refined with each cycle, gradually moving closer to the final product.

* Spiral model is iterative model
* Spiral model overcome drawbacks of waterfall model
* We follow spiral model whenever there is dependency on the modules
* In every cycle new software will be released to customer
* Software will be released in multiple versions, it’s called version control model

Advantages

* Testing is done in every cycle, before going to the next cycle
* Customer will get to use the software for every module; customer no needs to wait for the entire process/software; release product every one cycle
* Requirement changes are allowed after every cycle, before going to the next cycle

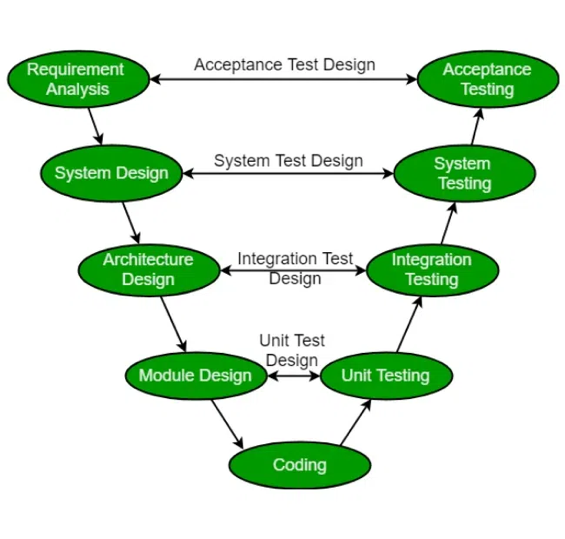
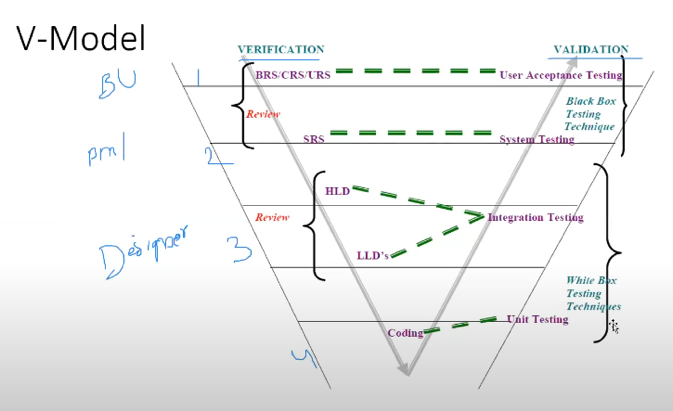
Disadvantages

* Requirement changes are NOT ALLOWED in between the cycle
* Every cycle of spiral model looks like waterfall model
* There is no testing in requirement and design phase

Prototype model – blue print of the software

* Initial requirements from the customer
* Develop the prototype – sample/blue print of the software
* Show and explain the prototype to the customer
* If the customer satisfied with the prototype, design, coding & testing will start

**V-model – verification & validation model; In every phase, the test will be conducted**



Verification

Business req spec (BRS) /customer req spec (CRS) /user req spec (URS) 🡪 contain business requirement document

Software req spec (SRS)

High level design – main modules

Low level design – sub modules

Static testing – testing the project related documents

Testing techniques (verify the documents):

* Review
* Walkthrough
* Inspection

Dynamic testing – testing the actual software

Validation – testing on the software

\*Unit testing: testing a single module/component of the software; usually developers

\*Integration testing: integrate/make multiple modules into one main module; usually developers

\*White box testing – involve coding/programming knowledge

\*\*System testing

\*\*User acceptance testing

\*\*Black box testing – not involve coding/programming knowledge, verify/testing more on functional

Verification

1. Verification checks whether we are building the right product; building the product correctly or not
2. Focus on documentation
3. Product is not ready
4. Verification (static testing) typically involves:

* Reviews
* Walkthrough
* Inspection

Validation

1. Validation checks whether we are building the product right; the product is right or not according to customer requirement
2. Takes place after verifications are completed & after the coding part completed
3. Focus on actual software
4. Product is ready
5. Validation (dynamic testing) typically involves actual testing:

* Unit testing
* Integration testing
* System testing
* User accept testing

Verification testing the documentation whether following the right process or not; done before software is ready

Validation testing the actual software by giving the input and observing the outputs; done after software is ready

Advantages

Testing is involved in each and every phase

Disadvantages

Documentation is more

Initial investment is more

**Static testing technique**

1. Review: read entire document whether it’s completely written or not, all the contents mentioned in the document, all the contents correct or not

* Conduct on documents to ensure correctness and completeness
* Requirement reviews: review the documents
* Design reviews: review design documents, LLD & HLD
* Code reviews: review the source codes by developer
* Test plan reviews
* Test cases reviews

1. Walkthrough: informal review, do not have specific plan & meetings

* It’s a informal review: not have proper plan/schedule
* Author (who creates the documents) reads the documents or code and discuss with peers
* It’s not pre-planned and can be done whenever required
* Also walkthrough does not have minutes of the meet

1. Inspection: more formal, invite other team members

* Most formal review type
* In which at least 3-8 people will sit in the meeting: reader (write the document), writer (write the issue raised), moderator (organize the meeting) plus concerned
* A proper schedule which will be intimated via email to the concerned developer/tester

**Dynamic testing technique**

Unit testing

Integration testing

System testing

User acceptance testing

**Quality Assurance (QA), Quality Control (QC) & Quality Engineer (QE)**

1. QA

* QA is process related
* QA involves in the whole SDLC process; requirement analysis, design, coding, testing, deployment and maintenance
* QA focuses on building in quality
* QA is preventing defects; prevent from having the defects in the future
* QA is process oriented
* QA for entire life cycle

1. QC

* QC is the actual testing of the software
* QC only involves in testing
* QC focuses on testing for quality
* QC is detecting defects; bugs are there, try to find out
* QC is product oriented; produce quality product, finding bugs
* QC for testing part in SDLC

1. QE writes code for automation testing

**Level of testing: step by step of testing, what’s next to be tested**

1. Unit testing – white box testing

* Test small/single module/component
* A single component/module of a software
* Conduct on a single program/module
* White box testing technique; internal logic of the program
* Conducted by the developers
* Unit testing technique:
* Basis path testing: every line of the program/code should be executed at least once
* Control structure testing:

1. Conditional coverage: if-else statement
2. Loops coverage: iteration statement

* Mutation testing: repetition testing of multiple set of data

1. Integration testing – white box testing

* Combine multiple modules/components into main component
* Perform between 2/more modules
* Focuses on checking data communication between multiple
* White box testing technique
* Types of integration testing
* Incremental integration testing: incrementally adding gradually the modules & testing the data flow/communication between the modules
* 2 approaches:

1. Top down: incrementally adding the modules and testing the data flow between the modules, and ensure the module added is the child of previous module (parent module 🡪 child/sub module)
2. Bottom up: incrementally adding the modules and testing the data flow between the modules, and ensure the module added is the parent of previous module (child/sub module 🡨 parent module)
3. Sandwich/hybrid approach: combination of top-down & bottom-up approach

* Non-incremental integration testing: adding all the modules in a single shot and test the data flow between modules
* Drawbacks of non-incremental integration testing:

1. Might miss data flow between some of the modules
2. If find any defect, we cannot understand the root cause of the defects
3. System testing – black box testing

* Test overall functionality of the software whether following the user requirements or not
* Testing overall functionality of the application with respective client requirements (feature functionalities)
* It’s a black box testing technique
* This testing is conducted by testing team
* After completion of component and integration level testing, system testing will kick off
* Before conducting system testing, we should know the customer requirements
* Focuses on below aspects:
* Graphical user interface (GUI) testing: images, text boxes align or not, look and feel of the app the colours and fonts
* Functional testing: test the functionality of the app; the flow/navigation
* Non-functional testing: security testing, compatibility testing, performance testing; speed of the app
* Usability testing: verify user manual (step by step to use the app) to users, how user friendly the app is

1. User acceptance testing – black box testing

* User and tester will conduct UAT
* User uses the app; whether they satisfied the app or not
* After completion of system testing, UAT team conducts acceptance testing in 2 levels
* Alpha testing: user tests the app at the development environment
* Beta testing: user tests and install the app at the their own environment and do some basic testing

**System Testing**

1. GUI testing – front-end

* Graphical user interface (GUI): a process of testing the UI of an app
* Includes all the elements such as menus, checkbox, buttons, colours, fonts, sizes, icons, contents and images
* GUI testing checklist (look and feel of the app)
* Size, position, width, height of the elements
* Error messages that are getting displayed
* Different sections of the screen
* Font whether it’s readable or not
* Different screen resolutions with the help of zooming in and zooming out
* The alignment of the texts and other elements like icons, buttons are in proper place or not
* Colours of the fonts
* Image has good clarity or not
* The alignment of the images
* The spelling
* User must not get frustrated while using the system interface
* The interface is attractive or not
* The scrollbars according to the size of the page if any
* The disabled fields if any
* The size of the images
* The headings whether it’s properly aligned or not
* The colour of the hyperlink
* UI elements like button, textbox, text area, check box, radio buttons, drop downs, links, etc

1. Usability testing – easiness of the app

* This testing validates app provided context sensitive help or not to the user – help menu/user manual
* Check how easily the end-users are able to understand and operate the app

1. Functional testing

* Test the behaviour of the app; whether working according to user requirements or not
* About how the app’s features should work
* Eg: only 8 characters allowed to input in the text field starting with uppercase
* Types:
* Object properties testing: everything in elements/attributes/properties. Text field is enable/disable. Radio button only can choose one. Check the properties of objects present on the app, eg: enable, disable, visible, focus attributes
* Database testing – back-end part: DB – storage area. Checking DB operations w.r.t user operations, focused on DML – SELECT, INSERT, UPDATE & DELETE. Table & column level validations (column type, column length, number of columns). Relation between the tables (normalization). Functions, procedures, triggers, indexes, views etc.
* Error handling: verify the error messages while performing incorrect actions on the app. Error messages should be readable and user understandable/simple language, eg: Invalid user
* Calculations/manipulations testing: financial/banking app. Focused on the output of the calculations part. Check and verify the calculations, working properly or not
* Links existence & links execution (web app) – where exactly the links are placed – links existence. Links are navigating to proper page or not – links execution. Types: 1) internal link – click on the link that will navigate to the same page in different section. 2) external link - navigate to the different page. 3) broken link – link is there but there’s no action, not navigating to any pages
* Cookies & sessions (web app) – cookies: temporary collected data files created by browser while browsing the data through internet. Sessions – time slots created by the server. Session will be expired after some time if you’re idle from some time. Test by browsing the page and leave a few mins & try to do some actions, it should give expiry session/re-login (usually banking app).

1. Non-functional testing – focused on customer satisfaction/expectation

* Performance testing (web app) – speed of the app
* Load testing: gradually increasing load or number of users till certain numbers, then check the speed of the app. Check until the server breaks down, say 100 users, the server breaks down
* Stress testing: suddenly increasing/decreasing the load on the app and check the speed of app. Suddenly increasing 50 users, suddenly decreasing 20 users
* Volume (size) testing: user sending some data; check how much data is able to handle by the app
* Security testing: verify how secure the app. Authentication – verify users are valid or not. Authorisation/access control – permission of the valid user only able to access certain features.
* Recovery testing: check the system changes to abnormal state to normal state.
* Compatibility testing
* Forward compatibility – newest upgraded version, the previous version should still compatible; inspects a software product on the latest versions of OS
* Backward compatibility – the same machine is compatible with the newest version; inspects a product behaviour in older version of OS
* Hardware compatibility – configuration testing; software working/support on multiple platforms or not – windows, linux, ram
* Configuration testing
* Installation testing – check screens are clear to understand/simple or not. Check for uninstallation – should be removed completely. Screens navigation
* Sanitation/garbage testing – remove unwanted features that doesn’t follow the user requirements – it’s considered as bugs and need to be raised it. If any app provides extra features/functionality then, we consider them as a bug

|  |  |
| --- | --- |
| **Functional Testing** | **Non-Functional Testing** |
| Validates functionality of software | Verify the performance, security and reliability of the software |
| Describes what software does | Describes how software works |
| Focus on user requirement | Focus on user expectation |
| Take place before non-functional testing | Take place after finishing functional testing |

**Software Testing Terminology**

1. Regression testing

* Testing conducts on modified builds to make sure there will not be impacted on the existing functionality because of changes like adding/deleting/modifying features
* Unit regression testing
* Testing only the changes/modifications done by the developer – test only the impacted module
* Regional regression testing
* Testing the modified module along with the impacted modules – test the changes along with the impacted module
* Impact analysis meeting conducts to identify impacted modules with QA & QE
* Full regression
* Testing the main feature & remaining part of the app due to many changes/modification of the multiple modules done by the developers – not focusing only on the impacted modules
* Eg: developer has done changes in many modules, instead of identifying impacted modules, we perform one round of full regression

1. Re-testing – testing again and again; make sure the bug has corrected

* Whatever we have reported in Build 1, we will verify in Build 2 – whatever bugs reported in the previous build (compiled version of software and ready to be tested and deployed) whether fixed or not we will verify in the upcoming builds
* Whenever the developer fixed a bug, test will test the bug fix
* Tester close the bug if it worked otherwise re-open and send to developer
* To ensure that the defects which were found and posted in the earlier build were fixed or not in the current build
* Examples:
* Build 1.0 was released, test team found some defects (Defect ID 1.0.1, 1.0.2) and posted
* Build 1.1 was released, now testing the defects 1.0.1 and 1.0.2 in this build is re-testing

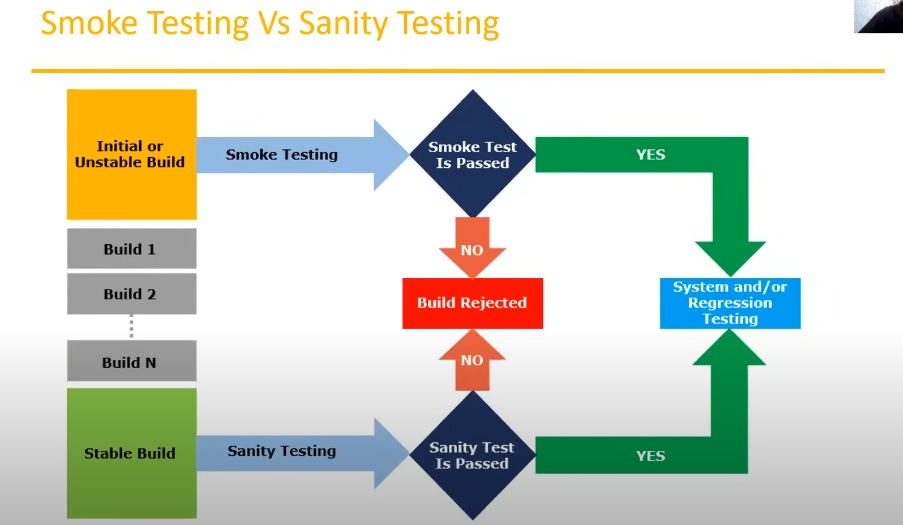
Regression testing vs Re-testing

Example

* An app under test has 3 modules; Admin, Purchase and Finance
* Admin 🡪 Purchase 🡪 Finance
* Finance module depends on Purchase module
* If a tester found a bug on Purchase module and posted. Once the bug is fixed, the tester needs to do **Re-testing** to verify whether the bug related to the **Purchase** is fixed or not, and also tester needs to do **Regression Testing** to test **Finance** module which depends on the Purchase module
* Re-testing Purchase only testing the bug fixed, while Regression Testing verify the bugs and impacted module

1. Sanity & smoke testing – comes into the picture after build released

|  |  |
| --- | --- |
| **Smoke Testing – basic functionality testing, build verification testing, navigation, check for missing files** | **Sanity Testing – done after smoke testing becomes stable after multiple build versions** |
| Done to make sure the build we received from the development team is testable/stable or not | Done during the release phase to check for the main functionalities of the app without going deeper |
| Performed by both developer and tester | Performed by tester alone |
| Build may be either stable or unstable | Build is relatively stable |
| It’s done on initial builds | It’s done on stable builds |
| A part of basic testing | A part of regression testing |
| Usually it’s done every time there is a new build release | It’s planned when there is no enough time to do in-depth testing |



1. Exploratory testing – exploring the functionalities of the software without having a requirement document

* We have to explore the app, understand completely and test it
* Understand the app, identify all possible scenarios, document it then use it for testing
* We do exploratory testing when the app ready but there’s no requirement
* Test engineer will do exploratory testing when there’s no requirement
* Drawbacks:
* Might misunderstand any feature as a bug OR any bug as feature since you do not have requirement; confuse to differentiate it’s a bug or it’s a feature
* Time consuming because there’s no requirement documentation to refer to
* If there’s any bug in app, you will never know about it

1. Ad-hoc testing

* Testing app functionalities randomly without any test cases or any business requirement document
* An informal testing type with an aim to break the system
* Tester should have previous experience/knowledge of app even though he doesn’t have requirements/test cases
* It’s usually an unplanned activity
* No documentation, no test design, no test cases

1. Monkey/gorilla testing

* Testing app randomly without any test cases or any business requirement document
* Ad-hoc testing is an informal testing type with an aim to break the system
* Tester doesn’t have knowledge of app
* Suitable for gaming app

|  |  |  |
| --- | --- | --- |
| **Exploratory Testing** | **Ad-hoc Testing** | **Monkey Testing** |
| No documentation | No documentation | No documentation |
| No plan | No plan | No plan |
| Informal testing | Informal testing | Informal testing |
| Tester doesn’t know app functionality | Tester should know app functionality | Tester doesn’t know app functionality |
| Random testing | Random testing | Random testing |
| Intention is to learn/explore functionality of app | Intention is to break the app/find out corner defects | Intention is to break the app/find out corner defects |
| Any app which is new to tester | Any app | Gaming app |

1. Positive testing

* Testing the app with valid inputs
* Check whether an app behaves as expected with positive inputs
* Eg: enter only numbers – 99999 🡪 positive testing
* The text box in an app which can accept only numbers, entering values up to 9999 will be acceptable by the system and any other values apart from this should not be acceptable. To do positive testing, set the valid input values from 0 to 99999 and check whether the system is accepting the values

1. Negative testing

* Testing the app with invalid inputs
* Check whether an app behaves as expected with negative inputs
* Eg: enter only numbers – abcdef 🡪 negative testing
* Negative testing can be performed by entering characters A to Z from a to z. Either software system should not accept the values or else it should throw an error message for these invalid data inputs

Positive vs Negative Test Cases

* Requirement
* A text box is listed as a feature and in functional requirement spec it is mentioned as textbox accepts 6-20 characters and only alphabets
* Positive test cases
* Textbox accepts 6 characters
* Textbox accepts up to 20 characters length
* Textbox accepts any value in between 6-20 characters length
* Textbox accepts all alphabets
* Negative test cases
* Textbox should not accept less than 6 characters
* Textbox should not accept characters more than 20 characters
* Textbox should not accept special characters
* Textbox should not accept numerical

1. End-to-end testing

* Testing the overall functionalities of the system including the data integration among all the modules
* Eg: Login 🡪 Add new customer 🡪 Edit customer 🡪 Delete customer 🡪 Logout

1. Globalisation and localisation testing

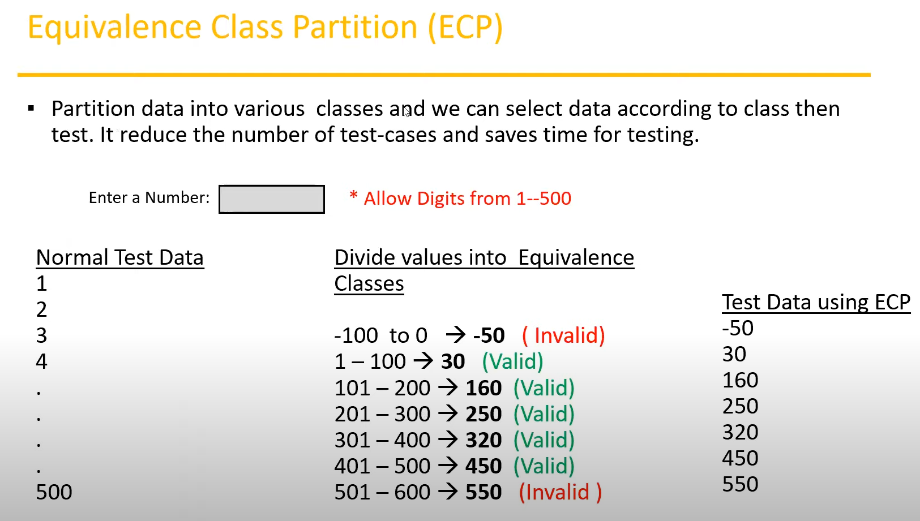
* Globalisation/Internationalisation (I18N) testing – supports app globally or not
* Perform to ensure the system/software app can run in any cultural or local environment
* Different aspects of the software app are tested to ensure that it supports every language and different attributes
* It tests the different currency formats, mobile number formats and address formats are supported by the app
* Eg: facebook supports many of the languages and it can be accessed by people of different countries, hence it’s globalised product
* Localisation testing – supports local community/language or not
* Perform to check system/software app for a specific geographical and cultural environment
* Localised product only supports the specific kind of language and is usable only in specific region
* It tests the specific currency format, mobile number format and address format is working properly or not
* Eg: baidu supports only the Chinese language and can be accessed only by people of a few countries, hence it’s a localised product

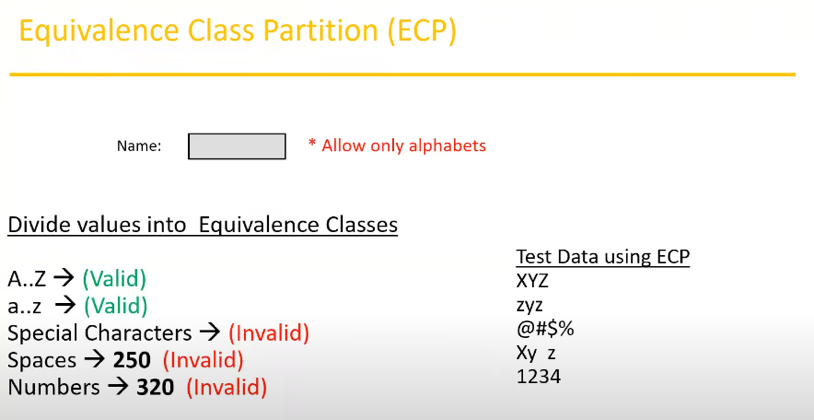
Test design techniques

* Used to prepare data for testing
* Reduce data: data used for testing
* More coverage: whatever data we prepare it should be covered in all the scenarios/functionalities
* Eg: valid username with valid password, invalid username with invalid password
* Eg: Age text field, must in range 18-30 years allowed. Need to test < 18 and > 30 as well as negative testing – this’s where we need to use the test design technique so that we don’t have to test each of the number/ages
* Test design techniques helps to design better cases
* Reduce the number of test cases to be executed
* Types:

1. Equivalence Class Partitioning (ECP)

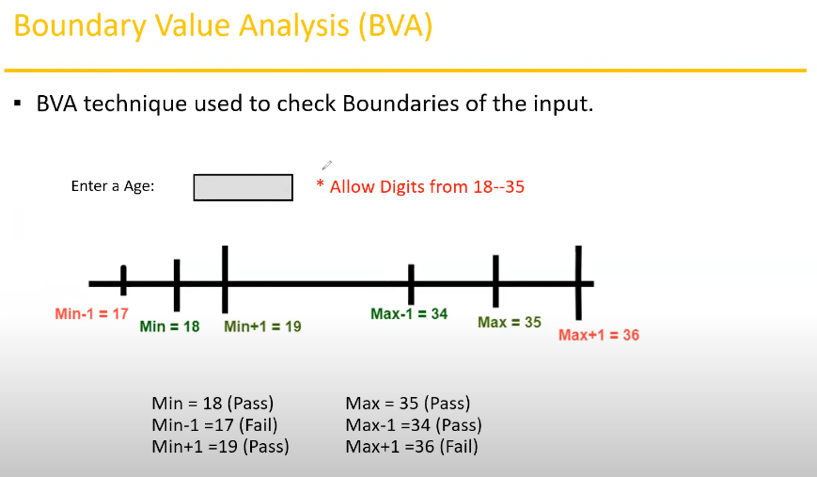
* Partition data into various classes and we can select data according to the class, then test. It reduces the number of test cases and saves time for testing
* Value check
* Classify/divide/partition the data in to multiple class and choose the value from the classes created
* Just take a single value in between the range





1. Boundary Value Analysis (BVA)

* Used to check boundaries of the input
* Enter value below 18 and above 35
* Only test the boundaries
* Min
* Min – 1
* Min + 1
* Max
* Max – 1
* Max + 1



**ECP & BVA will be used in Input Domain Testing**

* Verify whatever the inputs we provide whether it’s correct or not
* The value will be verified in the text box/input fields

1. Decision Table Based Technique

* Cause and effect table
* More conditions and corresponding actions
* If we have more number of conditions and actions then we use Decision Table
* Eg: transfer money online to an account which is already added and approved
* Conditions to transfer money – YES or No, TRUE or FALSE:

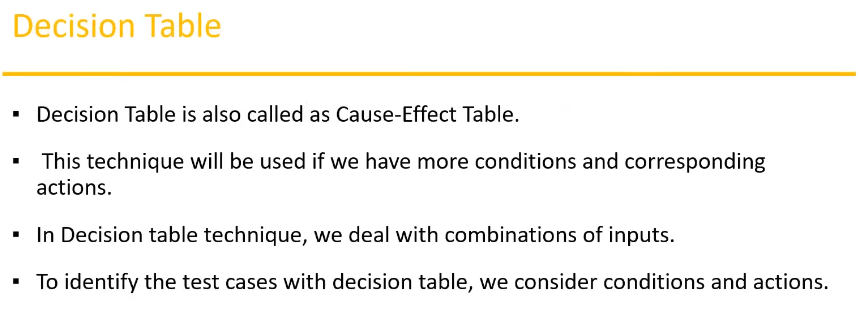
1. Account already approved
2. One time password (OTP) matched
3. Sufficient money in the account

* Actions performed – Approve/Deny/Success/Error Message:

1. Transfer money
2. Show a message as insufficient amount
3. Block the transaction in case of suspicious transaction

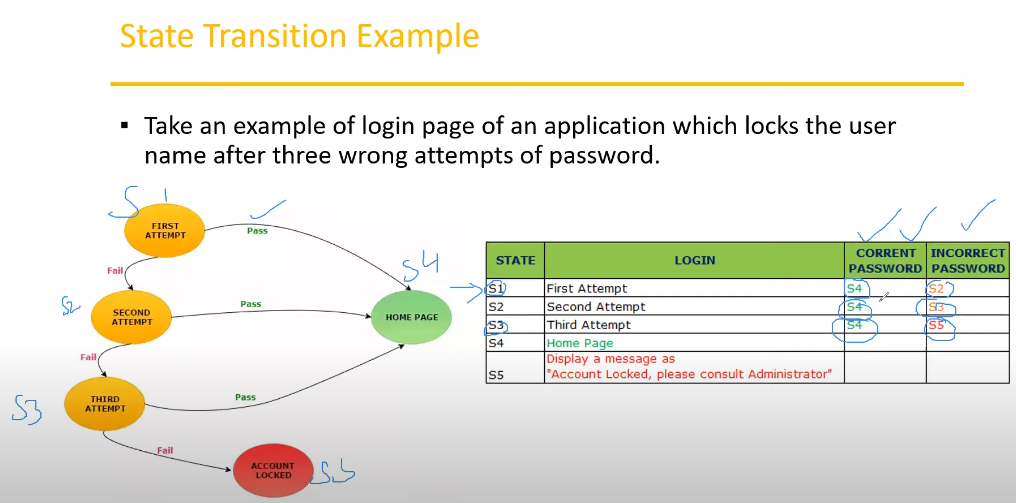
* Example

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | Valid TC | Negative TC | | | Invalid |
|  |  | TC1 | TC2 | TC3 | TC4 | TC5 |
| Condition 1 | Account already approved | TRUE | TRUE | TRUE | TRUE | FALSE |
| Condition 2 | OTP matched | TRUE | TRUE | FALSE | FALSE | X |
| Condition 3 | Sufficient money in the account | TRUE | FALSE | TRUE | FALSE | X |
| Action 1 | Transfer money | Execute |  |  |  |  |
| Action 2 | Show message ‘Insufficient Amount’ |  | Execute |  |  |  |
| Action 3 | Block the transaction in case of suspicious transaction |  |  | Execute | Execute | X |



1. State Transition – changing from one state to another state; multiple states/attempts

* In state transition technique changes in input conditions change the state of the app
* Allows the tester to test the behaviour of an AUT
* Tester can perform this action by entering various input conditions in a sequence
* Testing team provides positive + negative input test values for evaluating the system behaviour
* Provide valid input 🡪 Homepage
* Provide invalid input 🡪 next state



1. Error Guessing

* Used to find bugs in software app based on tester’s prior experience
* Don’t follow any specific rules
* It depends on tester analytical skill and experience
* Eg: submitting a form without entering values & entering invalid values such as entering alphabets in the numeric field

**Software Development Life Cycle (SDLC)** – The whole process of development

Requirement Analysis 🡪 Design 🡪 Code 🡪 Test 🡪 Deploy 🡪 Maintenance

**Software Testing Development Cycle (STLC)**

1. Requirement Analysis
2. Test Planning
3. Test Design
4. Test Execution
5. Defect/Bug Reporting
6. Test Closure

Requirement Analysis

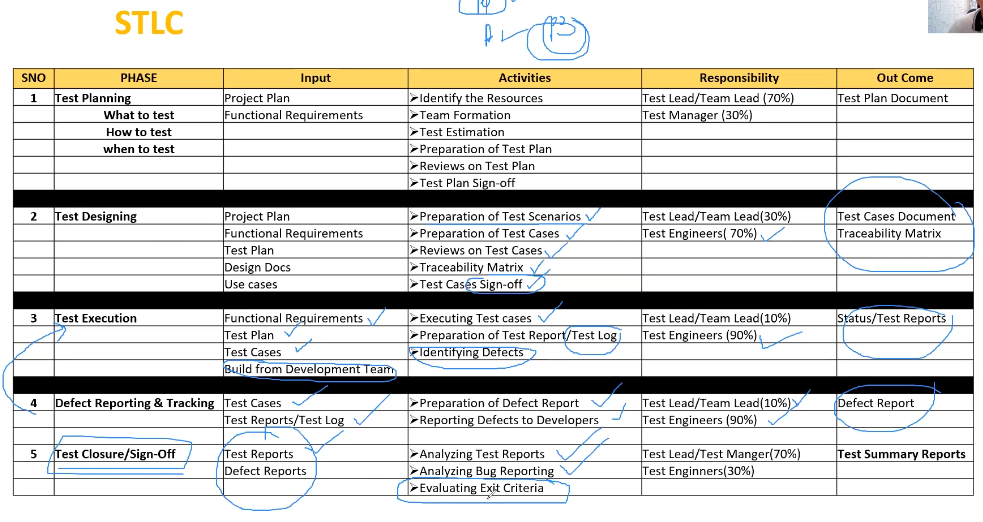
Test Planning

Test Case Development

Environment Setup

Test Execution

Test Cycle Closure



Test plan contents

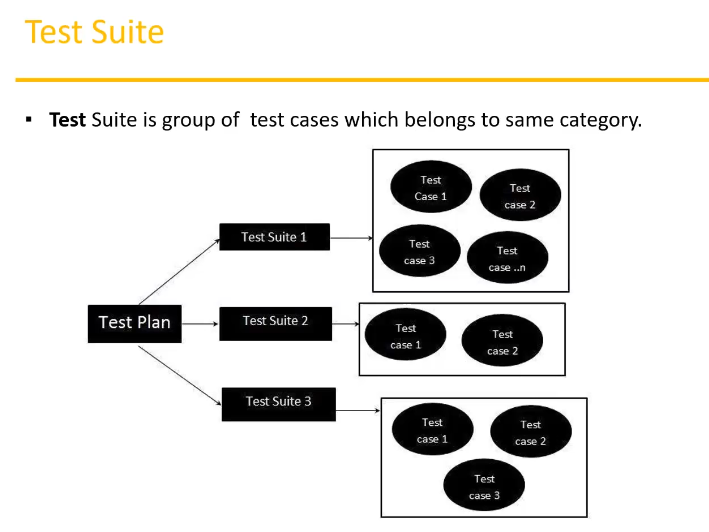
* A document that describes the test scope, test strategy, objectives, schedule, deliverables and resources required to perform testing for a software product
* Template contents:

1. Overview
2. Scope\* - what to test, what not going to test

* Inclusions
* Test environments
* Exclusions

1. Test strategy – manual/automation/what technique to use\*
2. Defect reporting procedure
3. Roles/responsibilities
4. Test schedule\* - process of defining the timeline for the testing
5. Test deliverables\* – test plan, test cases, defect report, test execution report
6. Pricing
7. Entry and exit criteria
8. Suspension and resumption criteria
9. Tools
10. Risks and mitigations – hardware/devices/people more to backups
11. Approvals

|  |  |  |
| --- | --- | --- |
| **Use Case** | **Test Scenario** | **Test Case** |
| * Describes the requirement – referred to FRS * Contains 3 items:  1. **Actor**: the **user** which can be a single person/group of people, interacting with a process – perform the **action** 2. **Action**: to reach the final outcome - **process** 3. **Goal/outcome**: the successful user outcome  * A picture/data flow diagram to understand the requirement | * A possible area to be tested – **what to test** * Derive the test scenarios from use cases | * Step by step actions to be performed to validate functionality of Application Under Test (AUT) – **how to test** * Contains test steps, expected result & actual result |
| * Describes functional requirement prepared by Business Analyst (BA) |  | * Describe test steps/procedures prepared by Test Engineer |
|  | Check the functionality of login button | * TC1: Click the button without entering username and password * TC2: Click the button only entering username * TC3: Click the button while entering wrong username and wrong number |



Test Cases

* A set of step by step or actions executed to validate particular feature or functionality of software app
* Test case contents:
* Test Case ID
* Test Case Title
* Description
* Pre-Condition
* Priority (P0 – very important; smoke/sanity TC, P1 – regression TC, P2 – functional TC, P3 – UI related) – order
* Requirement ID
* Steps/Actions
* Expected Result
* Actual Result
* Test Data

Requirement Traceability Matrix (RTM)

* Describes the mapping of requirement’s with the test cases
* Main purpose: see all test cases are covered so that no functionality should miss while doing software testing
* Parameters include:
* Requirement ID
* Requirement Description
* Test Case ID

Sample RTM

|  |  |  |  |
| --- | --- | --- | --- |
| Requirement No. | Requirement Description | Test Case ID | Status |
| 123 | Login to the app | TC01, TC02, TC03 | TC01 – Pass  TC02 – Pass |
| 345 | Ticket creation | TC04, TC05, TC06, TC07, TC08, TC09, TC010 | TC04 – Pass  TC05 – Pass  TC06 – Pass  TC07 – Fail  TC08 – No Run |
| 456 | Search ticket | TC011, TC012, TC013, TC014 | TC011 – Pass  TC012 – Fail  TC013 – Pass  TC014 – No Run |

Test Environment

* A platform specially build for test case execution on the software app
* Created by integrating the required software and hardware along with proper network config – what kind of machine user using, how much the RAM, network speed, version, OS, database
* Stimulates production/real time environment
* Another name: Test Bed – software and hardware environment created to perform testing

Test Execution

* Test team will carry out the testing based on the test plans and the test cased prepared
* Entry criteria: test cases – approved, test data – if necessary & test plan
* Activities:
* Test cases are executed based on the test planning
* Status of test cases are marked; passed, failed, blocked, run etc
* Documentation of test results and log defects for failed cases is done
* All blocked and failed test cases are assigned as bug ids
* Re-testing once the defects are fixed
* Defects are tracked till closure
* Deliverables: provides defect and test execution report with completed results

Guidelines for test execution

* The build being deployed to the QA environment is the most important part of the test execution cycle – the app is ready to be tested
* Test execution is done in QA environment – must be same as user environment
* Test execution happens in multiple cycles – many bugs will be found
* Test execution phase consists executing the test cases & test scripts (automation)

Defects/bugs/issues

* Any mismatched functionality found in an app between expected and actual result
* During test execution, test engineers are reporting mismatches as defects to developers through templates/tools
* Defect reporting tools (only for defect management):
* Clear quest
* DevTrack
* Jira
* Quality center
* Bug Jilla etc

Defect report contents

1. Defect ID: unique identification number for the defect
2. Defect description: detailed description of the defect including information about the module in which defect was found
3. Version (build version): version of the app in which defect was found
4. Steps: detailed steps along with screenshots with which the developer can reproduce the defects
5. Date raised: date when the defect is raised
6. Reference: where you provide reference to the documents; requirements, design. Architecture or may be even screenshots of the error to help understand the defect
7. Detected by: name/ID of the tester who raised the defect
8. Status: status of the defect
9. Fixed by: name/ID of the developer who fixed it
10. Date closed: date when the defect is closed
11. Severity: the impact of the defect on the app
12. Priority: defect fixing urgency

Defect classification

1. Severity (critical/major/minor/low/trivial)

* Blocker
* Critical
* Major
* Minor

1. Priority (high/medium/low)

* P1
* P2
* P3

Defect severity

* Describes the seriousness of defect & how much impact on business workflow
* Defect severity can be categorised into 4 classes:

1. Blocker (show stopper): nothing can proceed further

* Eg: app crashed, login not worked

1. Critical: main/basic/major functionality is not working, customer business workflow is broken, they cannot proceed further. Core functionality and security

* Eg: fund transfer is not working in net banking
* Eg: ordering product in e-commerce app is not working

1. Major: cause some undesirable behaviour, but the feature/app is still functional. Significant functionality and UX

* Eg: after sending email there’s no confirmation message
* Eg: after booking cab there’s no confirmation
* Eg: incorrect calculation

1. Minor: won’t cause any major break down of the system. Affects less critical aspects of the system/UX

* Eg: look & feel issues, spelling/typos, alignment of the UI

Defect priority

* Describes the importance/urgency (related to time) of the defect to be fixed/resolved
* Defect priority states the order in which a defect should be fixed
* Classes:

1. P0 – high: defect must be resolved immediately as it affects the system severely and cannot be used until it’s fixed
2. P1 – medium: it can wait until a new version/build is created
3. P2 – low: can be fixed at a later release/time

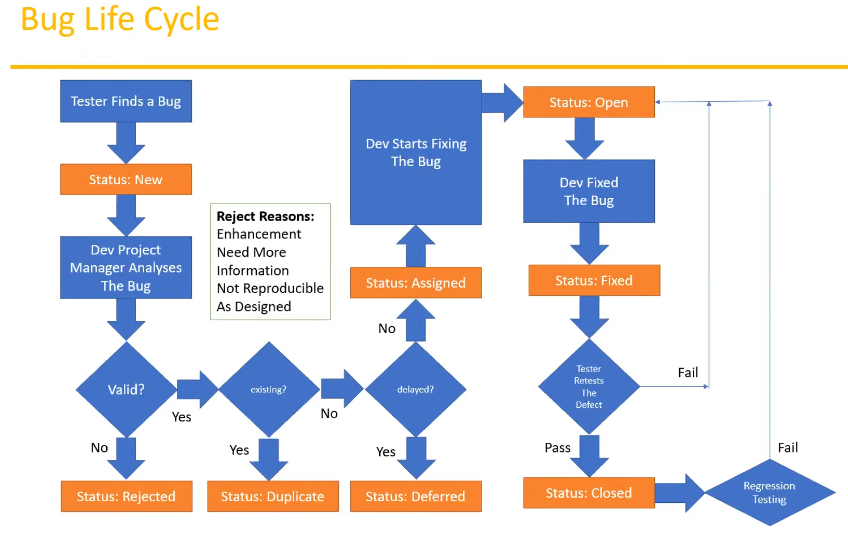
High severity, priority & low severity, priority defects

|  |  |  |  |
| --- | --- | --- | --- |
|  | Priority | | |
| Severity |  | High | Low |
| High | Login us taking to the blank page | About Us link going to blank page |
| Issue with login functionality; user not able to login into the app | App crashing in some very corner case |
|  | Web page not found when user clicks on a link; user doesn’t visit that page generally |
| Low | After user is logged into app, he can see Home page, but there’s spelling typos/mistakes in Home page (obvious because it’s in the main page) | User opens Contact page, but email ID has spelling mistake (not too obvious) |
| Slight change in logo color/spelling mistake in company name | A spelling mistake in a page not frequently navigated by users |
|  | Any cosmetic (imperfections of GUI; font, sizes, color schema) /spelling issues which is within a paragraph or in the page |

Defect resolution

* After receiving the defect report from the testing team, development team conducts a review meeting to fix defects, then they send a resolution type to the testing team for further communication
* Resolution types:
* Accept – accept and ready to be fixed
* Reject – won’t be fixed; reasons: 1) upcoming enhancement, 2) need more info, 3) not reproducible, 4) as designed
* Duplicate – raise the defect earlier
* Enhancement – new feature for upcoming release
* Need more information – provide more info; screenshots, backlogs
* Not reproducible – found defect in tester environment, but not found in developer environment
* Fixed – defect fixed
* As designed – developer says it not a defect, it’s the actual functionality, it should be like that

Bug life cycle – the states of the defects



Test cycle closure – stop the testing, app free from bugs; at least minor bugs

* Activities
* Evaluate cycle completion criteria based on time, test coverage, cost, software, critical business objectives, quality
* Prepare test metrics based on the above parameters
* Document the learning out the project
* Prepare test summary report
* Qualitative and quantitative reporting of quality of the work product to the customer
* Test result analysis to find out the defect distribution by type & severity
* Deliverables
* Test closure report; test summary report, test plan, test scenario, test cases, test execution, defect reports
* Test metrics

Test metrics – we have to measure/track our tests; to know our progress

1. No of requirements
2. Average no of test cases written per requirement
3. Total no of test cases written for all requirement
4. Total no of test cases executed
5. No of test cases passed
6. No of test cases failed
7. No of test cases blocked
8. No of test cases unexecuted
9. Total no of defects identified
10. Critical defects count
11. Higher defects count
12. Medium defects count
13. Low defects count
14. Customer defects
15. No of defects found in UAT

Calculations of the test metrics

* % of test cases executed
* (No of test cases executed / Total no of test cases written) \* 100
* % of test cases not executed
* (No of test cases not executed / Total no of test cases written) \* 100
* % of test cases passed
* (No of test cases passed / Total no of test cases executed) \* 100
* % of test cases failed
* (No of test cases failed / Total no of test cases executed) \* 100
* % of test cases blocked
* (No of test cases blocked / Total no of test cases executed) \* 100
* Defect density: number of defects identified per requirements/s
* No of defects found / size (no of requirements)
* Defect removal efficiency (DRE)
* (A / A+B) \* 100
* (Fixed defects / (Fixed defects + Missed defects)) \* 100
* A – defects identified during testing = fixed defects
* B – defects identified by the customer = missed defects
* Defect leakage – cannot identify
* (No of leakage found in UAT / No of defects found in testing) \* 100
* Defect rejection ratio
* (No of defect rejected / Total no of defects raised) \* 100
* Defect age
* Fixed date – Reported date
* Customer satisfaction
* No of complaints per period of time

QA/Testing Activities

* Understanding the requirements and functional specifications on the app
* Identifying required test scenarios – what area to test
* Designing test cases to validate app – write multiple test cases
* Setting up test environment – Test Bed
* Execute test cases to validate app – test execution
* Log test results – how many test cases pass/fail; test report
* Defect reporting and tracking – keep on looking defects until case closed
* Re-test fixed defects of previous build
* Perform various types of testing in app – perform smoke, sanity, functional, non-functional
* Report to Test Lead about the status of assigned tasks
* Participated in regular team meetings
* Creating automation scripts
* Provide recommendations on whether or not the app/system is ready for production

7 principles of software testing

1. Start software testing at early stages; means from the beginning when you get the requirements – verify the test scenario/test cases
2. Test the software in order to find the defects
3. Highly impossible to give the bug free software to the customer – maybe after 2 3 months we just found the undefine defects
4. Should not do exhaustive testing; means we should not use same type of data for testing every time – should use different data for the same test case
5. Testing is context based (types of app – web/mobile/desktop); means deciding what types of testing should be conducted based on type of app
6. We should follow the concept of pesticide paradox; means if you’re executing same cases for longer run, they won’t be find any defects. We have to keep update test cases in every cycle/release in order to find more defects – keep on upgrading test cases every time
7. We should follow defects clustering; means some of the modules contains most of the defects. By experience, we can identify such risky modules 80% of the problems are found in 20% of the modules

**Manual Testing Project**

1. Project introduction
2. Understand & explore the functionality
3. Estimation - agile
4. Test plan – scope of the testing; what to test/not to test, schedule, tool to use, type of testing, when to test, features to test
5. Write test scenarios
6. Write test cases & reviews
7. Environment setup & build deployment
8. Test execution
9. Bug reporting & tracking
10. Sanity testing, re-testing & regression testing
11. Test sign off

Project (develop for specific customer) VS Product (develop for multiple customer)

Project introduction

* Product based
* E-commerce product/app

e-commerce done by customer

1. Login
2. Search for products/items
3. Add products to cart
4. Do payment
5. Product will be delivered
6. Return the product

Front-end – public access over internet; internet app can be access through internet

Back-end – can be accessed by admin within the company; intranet can be used only in the organisation

Customer opencart

Mockup screens – how the screen will be in real app

Test plan

Inclusions - features to be tested/not to be tested

Environment - which environment to test

Exclusions - what are the features not going to be tested

Test strategy - what type of testing; manual testing/automation

Defect reporting procedure

Roles/responsibilities - name and responsibilities; PM, test engineer etc

Test schedule - when to start to finish; date

Test deliverables - documents; test plan, test cases, test execution, defect report

Pricing - PM

Entry & exit criteria - when to start testing and exit the from testing

Suspension & resumption criteria - when to start testing or resume the testing at certain time

Tools - MS Excel/automation tools/bug reporting tool

Risks & mitigations - back up resources; problem & solution

Approvals

Ramp up - adjust the resources; increase time

Priority

P0 - higher priority

P1 - basic sanity & functionality testing  
P2 -

P3 -

P4 - low priority; related to UI

Release notes: developer provides it along with the app deployed, the installation steps & procedures, what the builds contains/features & bug fixed

If test cases not done yet the test execution won’t start off

If the smoke testing (the build) fails – reject the build and wait for the new build; the test execution won’t start off also

Documents:

1. Functional requirement spec – FRS
2. Test plan
3. Test scenarios
4. Test cases
5. Requirement traceability matrix – RTM
6. Bug report

After build deployed, the build still unstable 🡪 smoke testing

After build already stable 🡪 sanity testing

Developer will fix the reported bugs in the upcoming build 🡪 re-testing

Other existing functionalities should not be impacted after fixing/updating the bugs/functionalities, execute the same test cases again and again 🡪 regression testing

**Agile Testing & Jira Tool**

* Agile model/agile methodology/agile process – iterative (repeating the same process again and again) & incremental (keep adding new modules/features on the existing app) process/approach
* Deliver the software with some features implemented to the customer – not yet fully done for all features; don’t want customer to wait for a long time
* Agile principle:

1. Customer no need to wait for longer time till the app is complete software developed with all the features – will implement only 10/100 features first and release the app as early as we can
2. We develop, test and release a piece of software to the customer with few numbers of features
3. Can accept/accommodate the requirement changes request by the customer – customer gives 100 requirements, then customer requests for change/add some new requirements

* Advantages:
* There will be good communication between customer, business analyst, developers & testers
* Requirement changes are allowed in any stages of development/can accommodate requirement changes in the middle of development
* Releases will be very fast – weekly
* Customer no need to wait for long time
* Good communication between team
* It’s very easy model to adopt
* Disadvantages:
* Less focus on design and documentation since we deliver software very faster
* Agile – defined process/approach model with a principle which how the process should be, Scrum – framework of the work to develop and test the software and finally deliver the software to the customer; to follow the principle of agile – Scrum is the framework to help us
* Scrum – framework through which we build a software product by following agile principles
* Scrum includes a group of people called Scrum Team – usually contains 5-9 members
* Scrum team:

1. Product owner

* Define/write the features of the product
* Prioritize features according to market value
* Adjust features and priority every iteration as needed
* Accept/reject work results

1. Scrum master – a leader

* The main role is facilitating (make an action or process easy or easier) and driving the agile process

1. Developer team

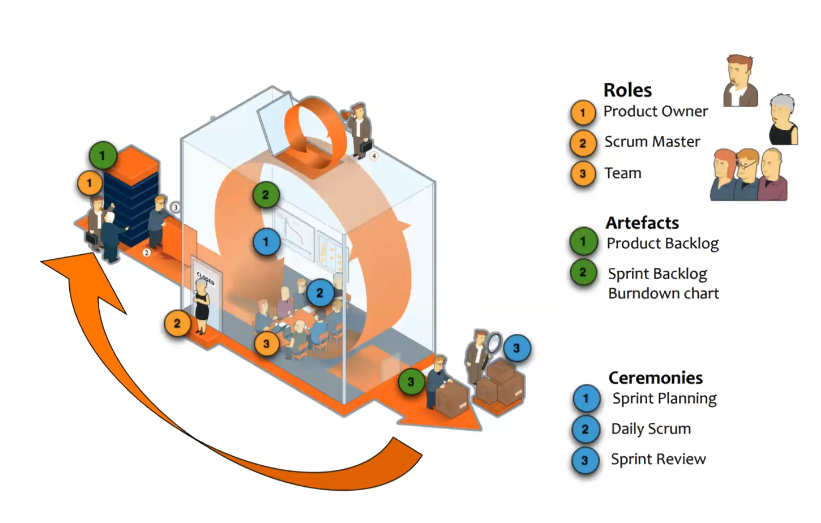
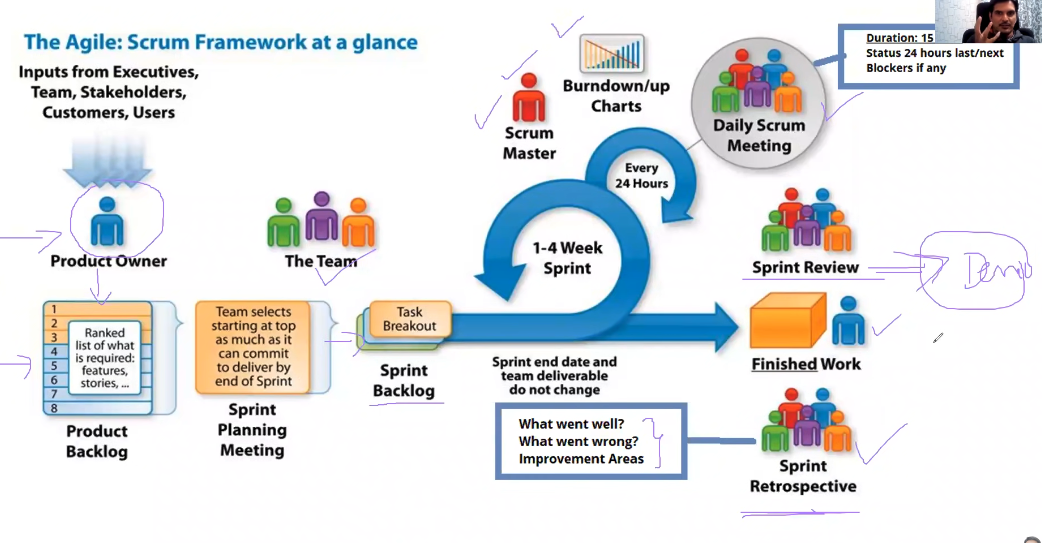
* Develop the software

1. QA team

* Test the software

Scrum terminology

* User story: a feature/module in a software/small requirements
* Epic: collection of user stories/huge requirements
* Product backlog: contains list of user stories, prepared by product owner – document defined the features of the product
* Sprint/iteration: period of time to complete the user stories (development and testing), decided by the product owner and team usually 2-4 weeks of time – first sprint, second sprint
* Sprint planning meeting: meeting conducts with the team to define what can be delivered in the sprint and duration – during; team will decide what user story will be tested and released to the customer at the particular sprint – how many user stories in the backlog, how many stories need to be developed and tested during the sprint and the sprint duration
* Sprint backlog: list of committed stories by developer/QA for specific sprint – developer & QA will choose a few numbers of stories to develop and test at the particular sprint – subset of product backlog
* Scrum meeting: meeting conducted by Scrum Master every day for 15 minutes – called as scrum call/stand up meeting; share task status; task done yesterday, task plan today, task for tomorrow, blocker/issue – what did you do yesterday, what you will do today, are there any issues/blockers in your way?
* Sprint retrospective meeting: will be conducted only once after finishing every sprint. Conducts meeting after completion of sprint. The entire team, including both the Scrum Master and the product owner should participate – recap; what went wrong/well with the previous sprint, what any implements need to be in the next sprint
* Story point: rough estimation of user stories, will be given by developer & QA in the form of Fibonacci series – sprint backlog; developer & QA will choose the stories based on story point
* Fibonacci series: 0 1 1 2 3 5 8…
* 1 story point = 1 hour/1 day (6 hours) – depends on the company
* Login story 🡪 develop – 5 hours, QA – 3 hours = 8 hours/1 day
* Burndown chart: shows how much work remaining in the sprint, maintained by the Scrum Master daily – how much works plan, how much works completed, how much the remaining works need to be completed



* Scrum board – keep tracking all the activities need to be done during sprint

1. Stories
2. To do
3. In progress
4. Testing
5. Done

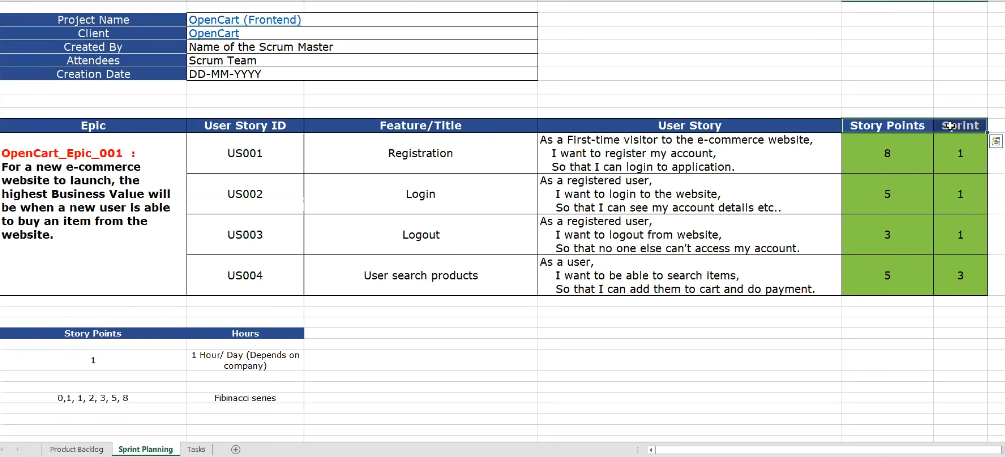
Definition of Ready (DoR) & Definition of Done (DoD)

|  |  |
| --- | --- |
| **Definition of Ready – ready for development and testing** | **Definition of Done – development and testing activities completed** |
| User story is clear | Code produced (all ‘to do’ items in code completed) |
| User story is testable | Code commented, checked in and run against current version in source control |
| User story is feasible - possibility that something can be made, done, or achieved | Peer reviewed (or produced with pair programming) and meeting development standards |
| User story clearly defined | Builds without errors |
| User story acceptance criteria defined | Unit tests written and passing |
| User story dependencies identified | Deployed to system test environment and passed system tests |
| User story sized by development team | Passed user acceptance testing and signed off as meeting requirements |
| Scrum team accept UX artifacts | Any build/deployment/configuration changes are implemented/documented/communicated |
| Performance criteria identified where appropriate | Relevant documentation/diagrams produced and/or updated |
| Team has a good idea what it will mean to demo the user story | Remaining hours for task set to zero and task closed |

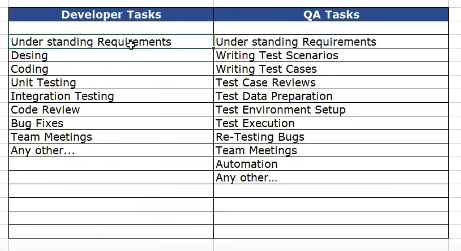
Sample of product backlog



Sprint planning



Task



**Tools**

* Test management tool: tracking task in management tool; define requirements, write test scenarios/cases, update test cased, report bug, close the requirement
* Defect/bug tracking tool: only for defect or bug reporting
* Automation tool
* Agile tools – for tracking agile task: jira, versionone, teamcity

Jira – Agile Management Tool

Jira workflow:

* Track agile activities
* Creation of object
* Backlog of epic and stories
* Create sprint
* Start sprint
* During sprint – developer and tester
* Close the story

Agile scrum activities

1. How to create project in Jira
2. How to add users/people in Jira
3. How to create backlog for epic
4. How to create stories in Jira & add story points
5. Creating sprint in Jira
6. Adding user stories to sprint
7. Start sprint
8. Add subtask
9. Sprint life cycle – to do, in progress, done

Backlog (product owner creates the backlog by taking the req by the customer) 🡪 Epic 🡪 Stories

Create sprint 🡪 Add stories to the sprint 🡪 Start sprint 🡪 Add tasks for every story

Test Management Activities – install Zephyr plugin

1. Write test cases
2. Create test cycle – regression test cycle/functional test cycle/sanity test cycle
3. Update test cases passed/failed/blocked
4. Report bugs
5. Reports

* Create test cycles
* Add test cases to cycle
* Execute/update test cases
* Reporting bugs
* Reports in Zephyr
* RTM