DAY-2

**1.What is software testing ?**

* Software Testing is a part of software development processes.
* Software Testing is an activity to detect and identify the defects in the software
* The main objective is in the software Testing is to release the Q quality product to the clients.

**2.what is software Quality ?**

* Software quality measures how well the software is designed(quality of design),and how well the software conforms to that design (quality of conformance).
* **Quality ?**

The quality is degree of conformance of a software product to requirements and expectations.(business)

* **Quality software is reasonably ?**
* Bug free
* Delivered on time
* Reach the customer expectation
* Within the budget the work will finish

**3. Product VS Project ?**

* **Product:** The product is that has been developed an application to the multiple customer based on their requirement maintain for the benefit of a user base often to satisfy in the market.
* **Project:** The project is developed on specific customer based on their requirements.

**4. why do we need Testing ?**

* Identifying the defects
* Improving the customer satisfaction
* Reducing the project risks
* Saving Money
* To Ensuring the Quality

**Error:** Issue that occurs the software development process and causes the software to behave differently then expected.

**Bug:** A bug is a problem with a computer program that causes it to behave unexpectedly or incorrectly.

**Failure:** A bug is a problem with a computer program that causes it to behave unexpectedly or incorrectly.

**5. Why the software has bugs Normally ?**

* **Complexity of code**
* **Lack of skilled testing**
* **Mis communication**
* **Requirement changes**
* **Human Error**

**6. Software development life cycle (SDLC) ?**

**Software development life cycle (SDLC) is a structured process that is used to design, develop, and test good-quality software.** SDLC, or software development life cycle, is a methodology that defines the entire procedure of software development step-by-step.

Five phases are their in the SDLC

* **Requirement analysis**
* **Design**
* **Development**
* **Testing**
* **Maintenance**

**7.waterfall model (Advantages and Disadvantages)**

The Waterfall Model is a linear application development model that uses rigid phases: When one phase ends, the next begins. Steps occur in sequence, and, if unmodified, the model does not allow developers to go back to previous steps (hence “waterfall”: Once water falls down, it cannot go back up).

**Advantages of waterfall model :**

**Detailed documentation**

**Well-defined requirements**

**Reduces risk**

**Easy to manage**

**Disadvantages of waterfall model :**

There’s no consideration for error correction

Work on different phases doesn’t overlap

Design flaws, when discovered, often mean starting over from scratch

Delaying the[testing](https://safetyculture.com/topics/non-destructive-testing/) until the end of development is common

**8. why Testing is necessary ?**

**It helps identify errors improve product functionality, and avoid risks.**

**Identify errors**

**Avoid extra costs**

**Avoid risks**

**Customer satisfaction**

**Save money**

**Improve performance**

**9.Software bugs occur when :**

Software bugs are flaws in computer software that result in unexpected outcomes. They can occur when there are errors in the software development process, such as when the software is being designed, coded, or tested.

**Human error**

**Lack of Time**

**Faculty Design**

**Ambiguous requirement**

**The cost of bugs**

The cost of bugs can be significant, and includes direct costs to fix the bugs, as well as indirect costs

**Direct costs**

Fixing bugs later in the development cycle is more expensive than addressing them early on. This is because it requires more time and resources for code rewrites, debugging, and regression testing

**Indirect costs**

* + **Customer support costs**: Buggy products can lead to more customer support requests, which can strain your budget.
  + **Security breaches**: Bugs can introduce security risks, which can lead to data breaches, cyberattacks, and financial theft.
  + **Opportunity costs**: While your team is fixing bugs, they can't focus on other initiatives, which can stifle growth.
  + **Loss of reputation**: Buggy software can indicate to clients that you don't produce high-quality products.
  + **Productivity loss**: Bugs can contribute to worker downtime, disruptions, and delays.

**10.Types of Testings:**

It is a process of test an application to identifying the error or bugs the main goal of Testing is to finding a bugs.

Testings are two types one is **Static** and Second **Dynamic.**

**Static:** It is the software testing that is perform the check defects in the without actually executing the code

**Dynamic Testing :** Dynamic testing refers to analysing code's dynamic behaviour in the software. In this type of testing, you have to give input and get output as per the expectation through executing a test case. You can run the test cases manually or through an automation process, and the software code must be compiled and run for this.

In the Dynamic testing we have two different types their White box and block Box Testing

**White Box testing**

* **Unit Testing: It is an individual component of software tested it is focus on the smallest unit of software design.**
* **Integration Testing: This Testing is individual unit s are combinined and tested.**
* **Top down**
* **Bottom up**
* **Sandwich**
* **Big-bang**

**Block box Testing**

* **System Testing**
* **User acceptance Testing**
* **Alpha**
* **Beta**

**11. Levels of Testing:**

* **Unit testing : It is an individual component of software tested it is focus on the smallest unit of software design**
* **Integration Testing: This Testing is individual unit s are combinined and tested.**
* **System Testing: In this system testing we can Test whole application tested by the tester.**
* **Acceptance Testing:** Acceptance testing is a quality assurance (QA) process that verifies if a software application or product meets user needs and business requirements before it's released. It's a crucial final step in the software development lifecycle.

Unit testing and Integration Testing it combines to White Box testing (Internal logic Developer)

Software Testing and user Acceptance Testing is combines to Block box Testing (no need code only functional tester)

**12. Testing types:**

* Functional Testing**:** Functional testing is a type of testing that seeks to establish whether each application feature works as per the software requirements.It uses the block box testing.
* Non Functional Testing:

Load Testing

Reliability

The readiness of a system

Usability testing

checking how many people can simultaneously check out of a shopping basket.

* Black Box Testing ( without code High level) test is without having any knowledge.
* White box testing (with code low level) monitoring internal structure check internal logic done by developer.
* Smoke testing: Smoke testing is a type of software testing that evaluates the basic functionality and stability of a software application after a new build or code change.

**QC → Build s/w → QA → Testing**

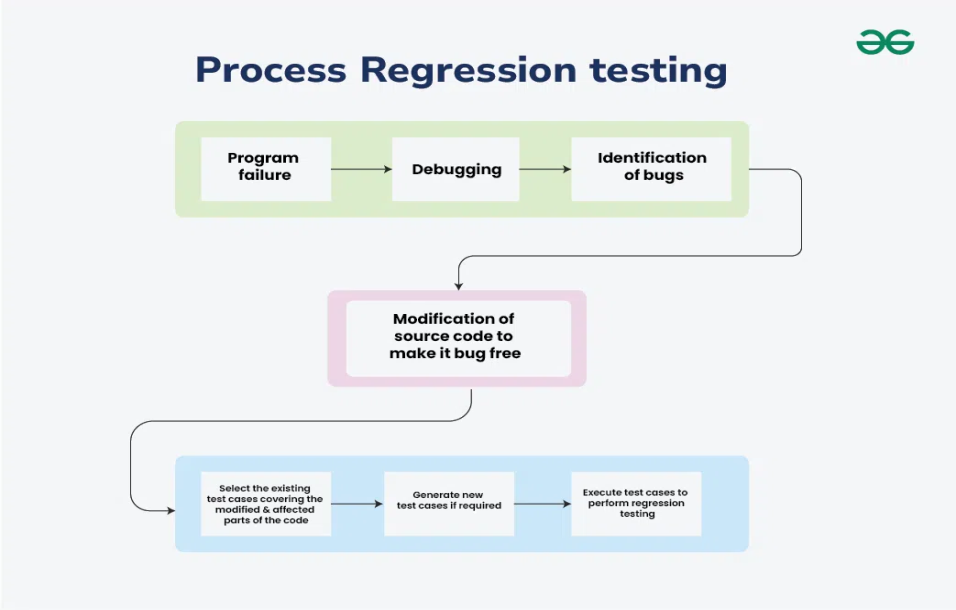
**Compulsory Testing → smoke testing**

**Sanity Testing**

**Retesting**

**20/11/2024**

**Regression testing: Regression testing is like a software testing quality check up after any changes are made. Then our program executed fails in the previously designed the source code. After the failure in the source code is identify or debug in the code. appropriate modifications are made.** Then appropriate test cases are selected from the already existing test suite which covers all the modified and affected parts of the source code. We can add new test cases if required. In the end, regression testing is performed using the selected test cases.



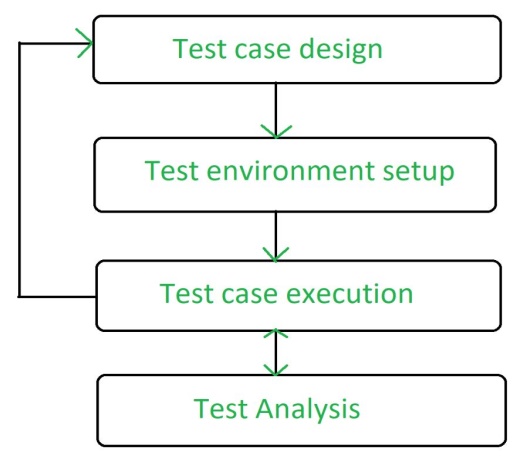
**Smoke testing:**

Smoke testing is a software testing method that verifies the basic functionality of a software application after a new build or code change.

**Static Testing :**

Static Testing also known as Verification testing or Non-execution testing is a type of Software Testing. The method that performed in the stage of development process to avoid errors and easy to find the sources of failures and fixed easily.

**Dynamic Testing:**



Dynamic Testing that is performed to analyze the dynamic behavior of the code. It includes the testing of the software for the input values and output values that are analyzed. It is performed to describe the dynamic behavior of code. It refers to the observation of the physical response from the system to variables that are not constant and change with time.

**Alpha Testing**:

Alpha Testing a type of software testing performed to identify bugs before releasing the product to real users or the public. Alpha Testing is one of the UAT . This is referred to as Alpha Testing only because it is done early on, near the end of the Development of the software.

**Beta testing:**

Beta testing is the process of testing a software product or service in a real-world environment before its official release. It is an essential step in the software development lifecycle as it helps identify bugs and errors that may have been missed during the development process.

**Installation testing :**

Installation testing is a type of software Testing  that verifies the software’s installation process and ensures that the software is installed correctly on various systems and configurations. It is a critical aspect of the software testing process and helps ensure that the software can be easily installed, configured, and uninstalled without any issues.

**Usability Testing : checking application for user friendliness**

**Monkey Testing : used for game testing, used for random input**

**To check the application or system will crash**

**Portablity Testing : Developed application Shoud support multiple enviorment**

**Forced error Testing : to check valid error message will display**

**Exploratory Testing : When test engineer does not have idea of functional testing then he is learning through**

**exploring application**

**End to End Testing : We can check all internal componant for successful response**

**Internal componant like Client , Network, Server Database etc are working fine**

**Means Testing internal component**

**Security Testing : Checking Secuirity of application**

**Relablity Testing : The Developed application Should Support Longer Duration i.e. Stablity**

**Audit : it is independent evolution of software .**

**Inspection : it is formal evolution of software**

**Concurrency Testing : multiuser Testing**

**Debugging : executing program line by line for finding errors.**

**Some of the most popular SDLC models are:**

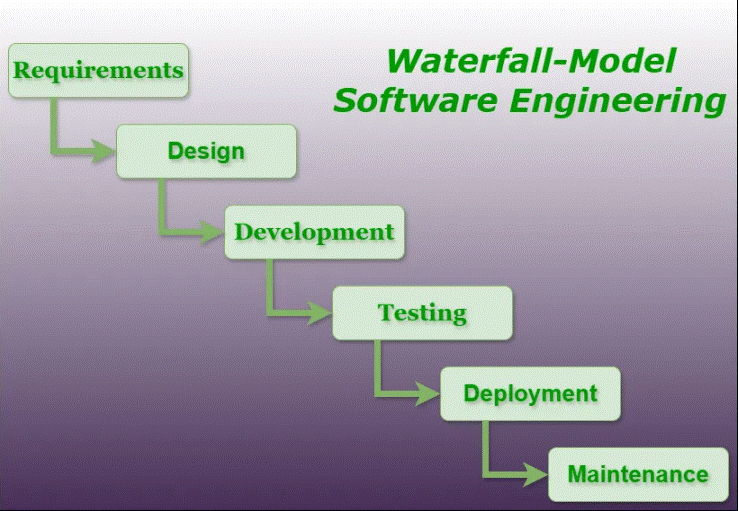
**\* V-Shaped Model**

**\* Incremental Life Cycle Model**

**\* Spiral Model**

**\* Waterfall Model:**

The Waterfall model follows a linear and sequential approach to software development. Each phase in the development process must be completed before moving on to the next one, resembling the downward flow of a waterfall. The model is highly structured, making it easy to understand and use.

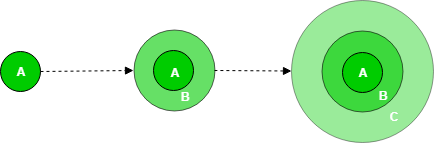


**V shanped-model:**

V-model is a type of [SDLC model](https://www.geeksforgeeks.org/top-8-software-development-models-used-in-industry/?ref=lbp) where the process executes sequentially in a V-shape. It is also known as the Verification and Validation model. It is based on the association of a testing phase for each corresponding development stage. The development of each step is directly associated with the testing phase. The next phase starts only after completion of the previous phase i.e., for each development activity, there is a testing activity corresponding to it.

**Incremental Life Cycle Model:**

First, a simple working system implementing only a few basic features is built and then that is delivered to the customer. Then thereafter many successive iterations/ versions are implemented and delivered to the customer until the desired system is released.



A, B, and C are modules of Software Products that are incrementally developed and delivered.

Spiral Model:

The Spiral Model is a Software development life cycle  model that provides a systematic and iterative approach to software development. In its diagrammatic representation, looks like a spiral with many loops. The exact number of loops of the spiral is unknown and can vary from project to project. Each loop of the spiral is called a **phase**of the software development process.

**Software development life cycle (SDLC) :**

**SDLC is a structured process that is used to design, develop, and test good-quality software.** SDLC, or software development life cycle, is a methodology that defines the entire procedure of software development step-by-step. The **goal of the SDLC life cycle model**is to deliver high-quality, maintainable software that meets the user’s requirements. SDLC in software engineering models outlines the plan for each stage so that each stage of the software development model can perform its task efficiently to deliver the software at a low cost within a given time frame that meets users requirements. In this article we will see Software Development Life Cycle (SDLC) in detail.



Validation → Actually testing software → done after software **V shaped Model** : verification and validation

Verification → done before development → check we are doing correct ?

Verifying document >> because no software ready

Review | walkthrough | inspection

Static Testing = verifying doc

ready → done Right ?

Product is ready just check ok or not

Dynamic Testing = unit | integration | system | user acceptance

Verification = before s/w = static

Validation = after s/w = dynamic

Verification → done before development → check we are doing correct ?

Verifying document >> because no software ready

Review | walkthrough | inspection

Static Testing = verifying doc

Validation → Actually testing software → done after software ready → done Right ?

Product is ready just check ok or not

Dynamic Testing = unit | integration | system | user acceptance

Verification = before s/w = static

Validation = after s/w = dynamic

**Spiral model:**

1. Spiral model is a iterative model it is a drawback of water fall model

2. we follow spiral model whenever dependency on other Modules.

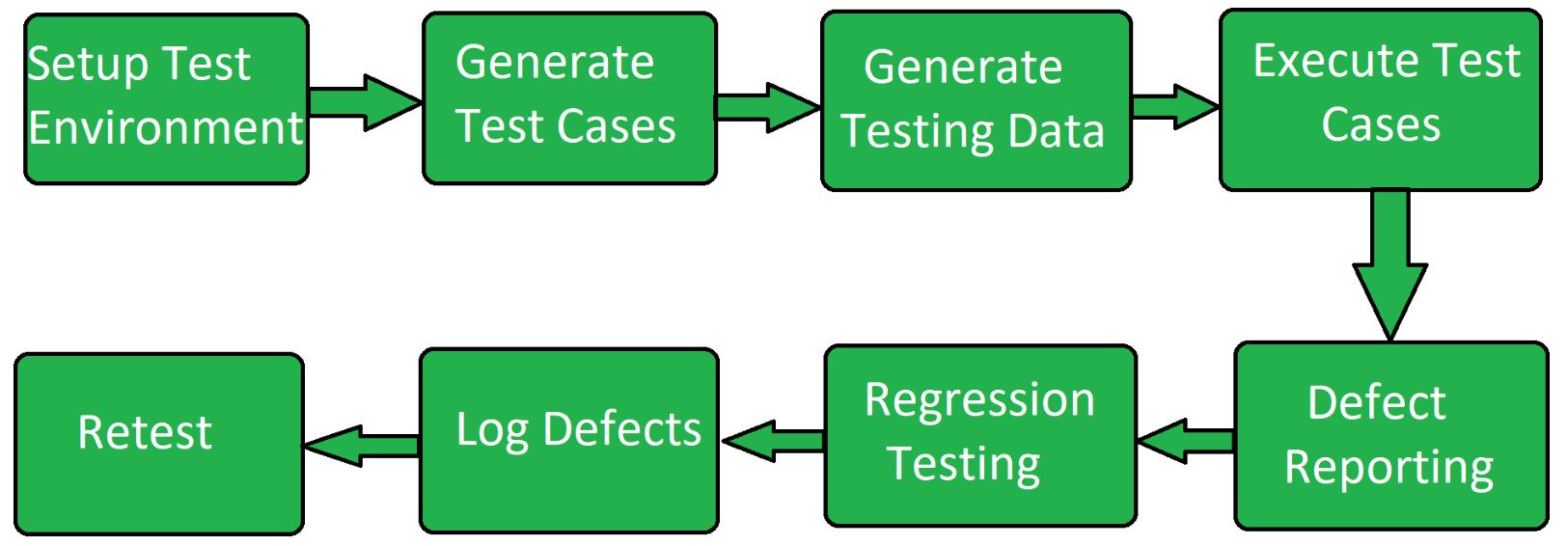
3. In every cycle new software will be release to customer.

4. software will be released in multiple versions so that is called as version control Model.

V shaped Model : verification and validation.

**System Testing: (Actual Testing)**

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



System Testing : GUI Testing | Usability Testing | Functional Testing | non functional

**1. GUI Testing :**

:- Testing GUI application , user interface testing

:- such as menus , check boxes , icon, images

:- not functional , just look and feel

:- check size and position of element

:- image quality, spelling check , alignment

:- Fonts are understanding or not

**2. Usability Testing :**

:- check the easiness of application

:- helping messages are display if user confuse

:- check user friendly application or not ?

**3. Functional Testing :**

:- check behaviour of application

:- check database testing (work with database ok ?)

:- error handling . display error message ok ?

:- calculation and manipulation

Eg. 5+5 = 15 ( user requirement ) => we follow this

5+5 = 10 ( math calculation)

:- check text box disable or enable as user requirement

:- Check database operation DML table, column , records etc

Checking database operation

Black box testing + white box testing = Gray box Testing

**4. Non Functional testing :**

:- once functional testing done i.e. s/w work user requirement then do non

functional testing

:- performance testing

Load testing – gradually increase the load

Stress testing – suddenly increase the load (Eg. Online filling form)

Volume testing – how much data handle

:- security of software

:- recovery of application

: - Compatibility testing – work with all platform

User form

UI Testing

Black box

Database

White box

End to End Testing:

:- testing overall application after including all module

Eg : login → add customer

→ delete and edit customer

→ logoutTestitg all function i.e. add delete edit and logout

**Test Case Design Technique:**

A test case is a defined format for [software testing](https://www.geeksforgeeks.org/software-testing-basics/)required to check if a particular application/software is working or not. A test case consists of a certain set of conditions that need to be checked to test an application or software i.e. in more simple terms when conditions are checked it checks if the resultant output meets with the expected output or not. A test case consists of various parameters such as ID, condition, steps, input, expected result, result, status, and remarks.

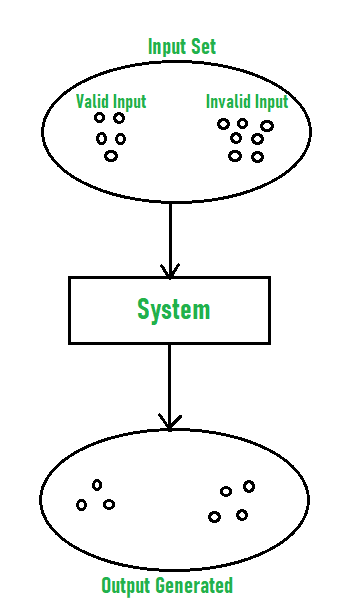
* **Before development:**Test cases could be written before the actual coding as that would help to identify the requirement of the product/software and carry out the test later when the product/software gets developed.
* **After development:**Test cases are also written directly after coming up with a product/software or after developing the feature but before the launching of a product/software as needed to test the working of that particular feature.
* **During development:**Test cases are sometimes written during the development time, parallelly. so whenever a part of the module/software gets developed it gets tested as well.

So, test cases are written in such cases, as test cases help in further development and make sure that we are meeting all the needed requirements.

Equivalence Class Partitioning (ECP

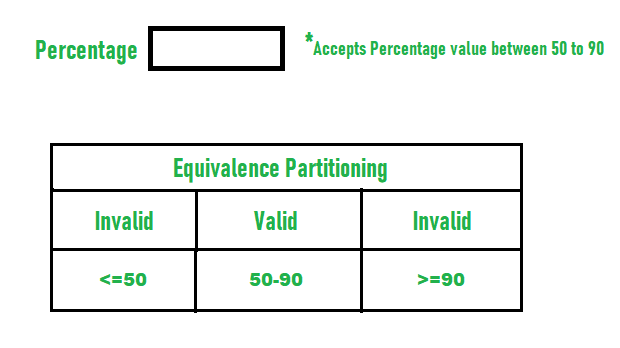
Equivalence Class Partitioning (ECP) is a software testing technique that groups input data into partitions based on similar characteristics. The goal of ECP is to reduce the number of test cases needed to test a software system while still ensuring effective test coverage

* If the range condition is given as an input, then one valid and two invalid equivalence classes are defined.
* If a specific value is given as input, then one valid and two invalid equivalence classes are defined.
* If a member of set is given as an input, then one valid and one invalid equivalence class is defined.
* If Boolean no. is given as an input condition, then one valid and one invalid equivalence class is defined.



Let us consider an example of any college admission process. There is a college that gives admissions to students based upon their percentage.

Consider percentage field that will accept percentage only between 50 to 90 %, more and even less than not be accepted, and application will redirect user to an error page. If percentage entered by user is less than 50 %or more than 90 %, that equivalence partitioning method will show an invalid percentage. If percentage entered is between 50 to 90 %, then equivalence partitioning method will show valid percentage. 



### Boundary Value Analysis( BVA):

### Boundary Value Analysisis based on testing the boundary values of valid and invalid partitions. The behavior at the edge of the equivalence partition is more likely to be incorrect than the behavior within the partition, so boundaries are an area where testing is likely to yield defects.

It checks for the input values near the boundary that have a higher chance of error. Every partition has its maximum and minimum values and these maximum and minimum values are the boundary values of a partition.

In simple terms boundary value Analysis is like testing the edge cases of our software where most of the time it will get broke so it is important to do BVA before deploying the code.

* boundary value for a valid partition is a valid boundary value.
* A boundary value for an invalid partition is an invalid boundary value.
* For each variable we check-
  + Minimum value.
  + Just above the minimum.
  + Nominal Value.
  + Just below Max value.
  + Max value.

**Example:**Consider a system that accepts ages from 18 to 56.

| **Boundary Value Analysis(Age accepts 18 to 56)** | | |
| --- | --- | --- |
| Invalid  (min-1) | Valid  (min, min + 1, nominal, max – 1, max) | Invalid  (max + 1) |
| 17 | 18, 19, 37, 55, 56 | 57 |

**Valid Test cases:**Valid test cases for the above can be any value entered greater than 17 and less than 57.

* Enter the value- 18.
* Enter the value- 19.
* Enter the value- 37.
* Enter the value- 55.
* Enter the value- 56.

**Invalid Testcases:**When any value less than 18 and greater than 56 is entered.

* Enter the value- 17.
* Enter the value- 57.

**Single Fault Assumption:**When more than one variable for the same application is checked then one can use a single fault assumption. Holding all but one variable to the extreme value and allowing the remaining variable to take the extreme value. For n variable to be checked:

**Maximum of 4n+1 test cases**

* boundary value for a valid partition is a valid boundary value.
* A boundary value for an invalid partition is an invalid boundary value.
* For each variable we check-
  + Minimum value.
  + Just above the minimum.
  + Nominal Value.
  + Just below Max value.
  + Max value.

**Example:**Consider a system that accepts ages from 18 to 56.

| **Boundary Value Analysis(Age accepts 18 to 56)** | | |
| --- | --- | --- |
| Invalid  (min-1) | Valid  (min, min + 1, nominal, max – 1, max) | Invalid  (max + 1) |
| 17 | 18, 19, 37, 55, 56 | 57 |

**Valid Test cases:**Valid test cases for the above can be any value entered greater than 17 and less than 57.

* Enter the value- 18.
* Enter the value- 19.
* Enter the value- 37.
* Enter the value- 55.
* Enter the value- 56.

**Invalid Testcases:**When any value less than 18 and greater than 56 is entered.

* Enter the value- 17.
* Enter the value- 57.

**Single Fault Assumption:**When more than one variable for the same application is checked then one can use a single fault assumption. Holding all but one variable to the extreme value and allowing the remaining variable to take the extreme value. For n variable to be checked:

***Maximum of 4n+1 test cases***

**Design table technique:**

this technique is used if we have more conditions and based on condition we have to perform the action

eg. transfer money from account to account

**Condition :**

1. Account no has to approved 🗸

2. OTP Matched 🗸

3. Sufficient money in account 🗸

If condition are ok then do action

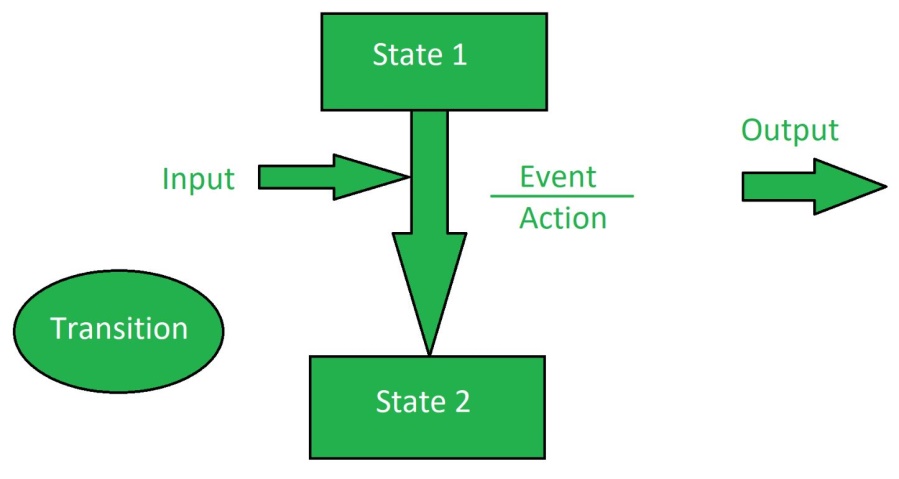
Action :

1. Transfer money

2. Show message insufficient money (if any)

3. Block if any suspicious activity

**State Transition Technique:**



State transition testing is a software testing technique that focuses on how a system transitions between states based on events or inputs. It's a process-oriented technique that's part of black box testing.

Ex: to login page of an application whitch locks the user name after 3 wrong attemptsof the passwords.

|  |  |
| --- | --- |
|  |  |

**Test case Scenario** → simply the name of test what to test (name of Test)

Test case → how to test i.e. step

Group of steps that is to be execuated to check functionality

Eg. Test scenario = check functionality of login button

Test case = TC01 , TC02 , TC03 , ... etc

Test suite → group of test case

Test case group

Test suite 1

Test suite 2

TC01, TC02, TC03

TC01, TC02, TC03

Test case document

:- Test case Id

:- Test case Title

:- Description

: Precondition

:- Priority

:- Request id

:- Steps/ Action

:- Excepted result

:- Actual result

:- Test Data

Requirement Traceblity matrix : (RTM)

:- Trace how many Test case are execuated or covered

: in simple keep track of test cases

## Test Metrics:

Test metrics are essential in determining the software’s quality and performance. Developers may use the right software testing metrics to improve their productivity

## Formula for Test Metrics:

To get the percentage execution status of the test cases, the following formula can be used:

***Percentage test cases executed = (No of test cases executed /  Total no of test cases written) x 100***

**Example of test matrics:**

| **S No.** | **Testing Metric** | **Data retrieved during test case development** |
| --- | --- | --- |
| 1 | No. of requirements | 5 |
| 2 | The average number of test cases written per requirement | 40 |
| 3 | Total no. of Test cases written for all requirements | 200 |
| 4 | Total no. of Test cases executed | 164 |
| 5 | No. of Test cases passed | 100 |
| 6 | No. of Test cases failed | 60 |
| 7 | No. of Test cases blocked | 4 |
| 8 | No. of Test cases unexecuted | 36 |
| 9 | Total no. of defects identified | 20 |
| 10 | Defects accepted as valid by the dev team | 15 |
| 11 | Defects deferred for future releases | 5 |
| 12 | Defects fixed | 12 |

***1. Percentage test cases executed =*** *(No of test cases executed /  Total no of test cases written) x 100   
= (164 / 200) x 100   
= 82*

***2. Test Case Effectiveness =*** *(Number of defects detected / Number of test cases run) x 100   
= (20 / 164) x 100   
= 12.2*

***3. Failed Test Cases Percentage =*** *(Total number of failed test cases / Total number of  tests executed) x 100   
= (60 / 164) \* 100   
= 36.59*

***4. Blocked Test Cases Percentage =*** *(Total number of blocked tests / Total number of  tests executed) x 100   
= (4 / 164) \* 100   
= 2.44*

***5. Fixed Defects Percentage =*** *(Total number of flaws fixed / Number of defects reported) x 100   
= (12 / 20) \* 100   
= 60*

***6. Accepted Defects Percentage =*** *(Defects Accepted as Valid by Dev Team / Total Defects Reported) x 100   
= (15 / 20) \* 100   
= 75*

***7. Defects Deferred Percentage =*** *(Defects deferred for future releases / Total Defects Reported) x 100   
= (5 / 20) \* 100   
= 25*

**Test Case Execution:**

Executing test case based on test plan

Mark status Pass | Fail | Blocked

Reports defects in bug report

**Defect Reporting Tool :**

ClearQuest :- only bug report

Devtrack : only bug report

Jira → test management tool (track each activity)

Bugzilla → test management tool (track each activity)

**Defect Report Details :**

:- Defect id

:- Defect Version

:- Step : details of step along with developer what to do

:- date

: - detected by

:- status

:- fixed by in process | fixed

: severity – impact

:- priority – high| medium | low

**Severity Of defect :-**

Blocker | Critical | Major | Minor

Seriousness of application

Testing engineer decides the severity level of the defect.

Blocker : this defect show application not processed

Critical : main function not working

Major : some undesirable behaviour eg. Email sent but msg not display

Minor : look and feel

Priority of defect: - High | Medium | Low

importance of defect

On which priority defect will be solved or fixed

P0 - High fixed immediately in same version

P1 - Medium fixed in next release

P2- Low next version

**Manual Testing project**: E commerce

:- Project introduction

:- Understanding and explore the functionality

:- Test Plan

:- Writing test scenario

:- Writing test cases

:- Environment setup and build and development

:- Test execution

:- Bug reporting and tracking

:- Sanity testing , smoke testing , regression testing

:- test sign off

**E commerce project**

- Login

- Search for product and item

- Add them to cart

- Do payment

- Product will be delivered

- Return the product

-Etc

# Software Requirement Specification (SRS) Format:

In order to form a good SRS, here you will see some points that can be used and should be considered to form a structure of good Software Requirements Specification (SRS). These are below mentioned in the table.

**Software Requirement Specification (SRS) Format** as the name suggests, is a complete specification and description of requirements of the software that need to be fulfilled for the successful development of the software system. These requirements can be functional as well as non-functional depending upon the type of requirement. The interaction between different customers and contractors is done because it is necessary to fully understand the needs of customers

Depending upon information gathered after interaction, SRS is developed which describes requirements of software that may include changes and modifications that is needed to be done to increase quality of product and to satisfy customer’s demand.

**Introduction**

* **Purpose of this Document –** At first, main aim of why this document is necessary and what’s purpose of document is explained and described.
* **Scope of this document –** In this, overall working and main objective of document and what value it will provide to customer is described and explained. It also includes a description of development cost and time required.
* **Overview –** In this, description of product is explained. It’s simply summary or overall review of product.

**General description**

In this, general functions of product which includes objective of user, a user characteristic, features, benefits, about why its importance is mentioned. It also describes features of user community.

**Functional Requirements**

In this, possible outcome of software system which includes effects due to operation of program is fully explained. All functional requirements which may include calculations, data processing, etc. are placed in a ranked order. Functional requirements specify the expected behavior of the system-which outputs should be produced from the given inputs. They describe the relationship between the input and output of the system. For each functional requirement, detailed description all the data inputs and their source, the units of measure, and the range of valid inputs must be specified.

**Interface Requirements**

In this, software interfaces which mean how software program communicates with each other or users either in form of any language, code, or message are fully described and explained. Examples can be shared memory, data streams, etc.

**Performance Requirements**

In this, how a software system performs desired functions under specific condition is explained. It also explains required time, required memory, maximum error rate, etc. The performance requirements part of an SRS specifies the performance constraints on the software system. All the requirements relating to the performance characteristics of the system must be clearly specified. There are two types of performance requirements: static and dynamic. Static requirements are those that do not impose constraint on the execution characteristics of the system. Dynamic requirements specify constraints on the execution behaviour of the system.

**Design Constraints**

In this, constraints which simply means limitation or restriction are specified and explained for design team. Examples may include use of a particular algorithm, hardware and software limitations, etc. There are a number of factors in the client’s environment that may restrict the choices of a designer leading to design constraints such factors include standards that must be followed resource limits, operating environment, reliability and security requirements and policies that may have an impact on the design of the system. An SRS should identify and specify all such constraints.

**Non-Functional Attributes**

In this, non-functional attributes are explained that are required by software system for better performance. An example may include Security, Portability, Reliability, Reusability, Application compatibility, Data integrity, Scalability capacity, etc.

**Preliminary Schedule and Budget**

In this, initial version and budget of project plan are explained which include overall time duration required and overall cost required for development of project.

**Appendices**

In this, additional information like references from where information is gathered, definitions of some specific terms, acronyms, abbreviations, etc. are given and explained.

**Uses of SRS document**

* Development team require it for developing product according to the need.
* Test plans are generated by testing group based on the describe external behaviour.
* Maintenance and support staff need it to understand what the software product is supposed to do.
* Project manager base their plans and estimates of schedule, effort and resources on it.
* customer rely on it to know that product they can expect.
* As a contract between developer and customer.
* in documentation purpose.