**SOFTWARE TESTING :-**

**Fault**: It is a condition that causes the software to fail to perform its required function.

**Error** : Refers to difference between Actual Output and Expected output.

**Failure** : It is the inability of a system or component to perform required function according to its specification.

**IEEE Definitions** 

* **Failure:**External behavior is incorrect
* **Fault:** Discrepancy in code that causes a failure.
* **Error:**Human mistake that caused fault

Error leads to fault and fault leads to failure

**Beta testing**:-

It is the last stage of testing, and normally can involve sending the product to beta test sites outside the company for real-world exposure or offering the product for a free trial download over the Internet.

**Top down Testing:-**

In this approach testing is conducted from main module to sub module. if the sub module is not developed a temporary program called STUB is used for simulate the submodule.

**Bottom up testing :-**

In this approach testing is conducted from sub module to main module, if the main module is not developed a temporary program called DRIVERS is used to simulate the main module.

**Regression Testing:-**

Regression testing is a type of software testing which verifies that software which was previously developed and tested still performs correctly after it was changed or interfaced with other software. Changes may include software enhancements, patches, configuration changes, etc.



**Black box testing:-**

Testing, either functional or non-functional, without reference to the internal structure of the component or system.

**BLACK BOX TESTING TECHNIQUES:-**

**1.Equivalence partitioning**: It is a software test design technique that involves dividing input values into valid and invalid partitions and selecting representative values from each partition as test data.

**2.Boundary Value Analysis:** It is a software test design technique that involves determination of boundaries for input values and selecting values that are at the boundaries and just inside/ outside of the boundaries as test data.

**3.Cause Effect Graphing or Decision Tables:** It is a software test design technique that involves identifying the cases (input conditions) and effects (output conditions), producing a Cause-Effect Graph, and generating test cases accordingly. Based on if, else condition also.

**4.Exploratory Testing:**

**5.Error Guessing:** It is used to find bugs in software application based on tester prior’s experience.

**6.State Transaction:**

**Equivalence partitioning Testing:-**

Equivalence partitioning or equivalence class partitioning (ECP) is a software testing technique that divides the input data of a software unit into partitions of equivalent data from which test cases can be derived. In principle, test cases are designed to cover each partition at least once.

It is a black box testing technique.

**The 3 main White Box Testing Techniques are:**

1. Statement Coverage

2. Branch Coverage(conditional coverage)

3. Path Coverage

**1) Statement coverage:**

Hence “Statement Coverage”, as the name itself suggests, it is the method of validating whether each and every line of the code is executed at least once.

**2) Branch Coverage(conditional coverage):**

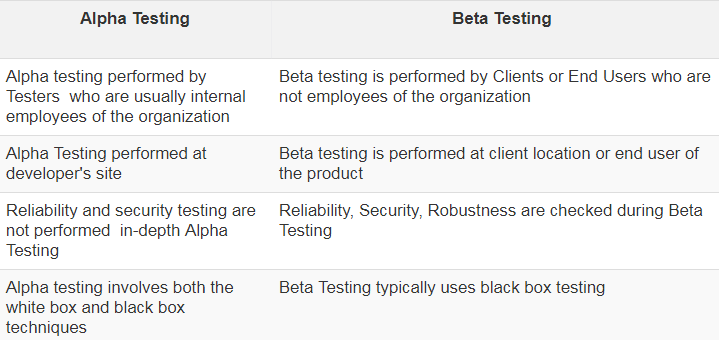
“Branch” in a programming language is like the “IF statements”. An IF statement has two branches: True and False.

we validate whether each branch is executed at least once.

**3) Path Coverage**

Path coverage tests all the paths of the program. This is a comprehensive technique which ensures that all the paths of the program are traversed at least once. Path Coverage is even more powerful than Branch coverage. This technique is useful for testing the complex programs.

**ALPHA AND BETA TESTING:-**

****

**Requirements Analysis**

Requirement analysis is a software engineering task that bridges the gap between system level requirements engineering and software design.

**The Prototyping Model** is a systems development method (SDM) in which a prototype (an early approximation of a final system or product) is built, tested, and then reworked as necessary until an acceptable prototype is finally achieved from which the complete system or product can now be developed.

The prototype model is used when the requirements are unclear. It is used when the customer is unclear about the details of the input, process and the output needs of the software

**Evolutionary prototyping:-**

Build an initial small requirement specifications, code it, then “evolve” the specifications and code as needed.

**RAD model** is Rapid Application Development model. It is a type of incremental model. In RAD model the components or functions are developed in parallel as if they were mini projects. The developments are time boxed, delivered and then assembled into a working prototype.

**The spiral model** is a risk-driven process model generator for software projects. Based on the unique risk patterns of a given project, the spiral model guides a team to adopt elements of one or more process models, such as incremental, waterfall, or evolutionary prototyping.

Assess risks at each step; do most critical action first.

**Water Fall Modal:-**

It is a linear-sequential life cycle model. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of software development model is basically used for the project which is small and there are no uncertain requirements

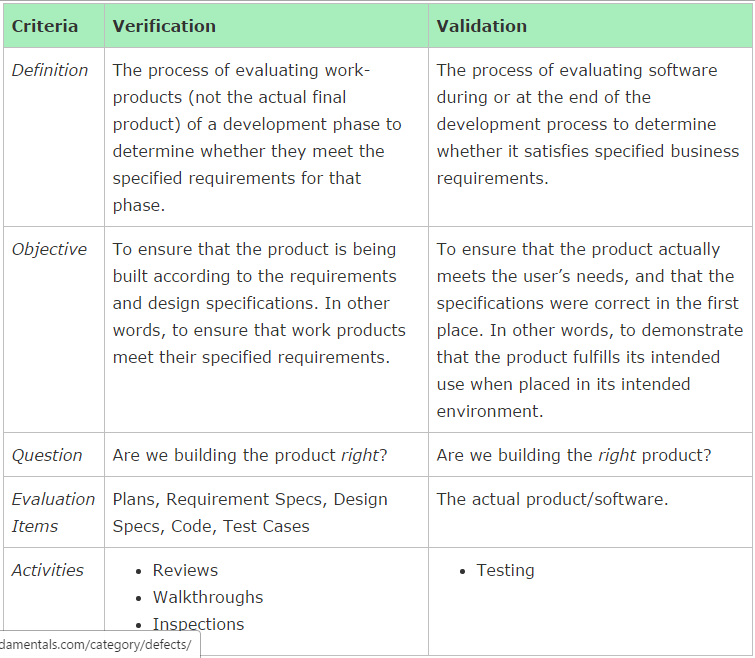
**ITERATIVE MODEL(Iterative Enhancement Modal):-**

It has 3 phases.

**SCRUM MODEL :-**

-Scrum is a subset of Agile.

-Time boxed delivery means delivered in short period of time. That is within 2 week not more than a month.



**Regression Testing:-**

It is a maintenance testing. Testing the already existing code and functionality with the new features added.

**FUNCTION POINT ANALYSIS**

In function point analysis, the number of complexity adjustment factors is 14

VAF = 0.65 + [ (Ci) / 100] .i = is from 1 to 14 representing each GSC.

FP = UAF \* VAF

GSC = General System Characteristic

UAF = Unadjusted Function Point

VAF = Value Adjustment Equation (VAF)

Ci = degree of influence for each General System Characteristic

**CYCLOMATIC COMPLEXITY**

As a software developer, we perform coding in any programming language. Once we are done with the coding part, we do assume that our part is over. But, that's not true in a real scenario. Our job does not end here. We need to look back to our code and take care of the below points.

1. Keep your code simple to understand.

2. Less complexity while using If…else statement or Switch case or any conditional statement.

3. Is this code manageable in future for any kind of changes or new development?

- In General, Cyclomatic Complexity tells how complex your code is.

- It is a quantitative measure of the number of linearly independent paths through a program's source code.

|  |  |  |
| --- | --- | --- |
| **Score** | **Cyclomatic** | **Risk Type** |
| 1 to 10 | Simple | Not much risk |
| 11 to 20 | Complex | Low risk |
| 21 to 50 | Too complex | Medium risk, attention |
| More than 50 | Too complex, | Can't test , high risk |

**Cyclomatic complexity = E - N + P**

where,

E = number of edges in the flow graph.

N = number of nodes in the flow graph.

P = number of nodes that have exit points

The complexity of the program can be defined as –

**V(G) = E - N + 2**

Where,

E - Number of edges

N - Number of Nodes

**V (G) = P + 1**

Where P = Number of predicate nodes (node that contains condition)

The easiest way is to sum the number of binary decision statements (e.g. if, while, for, etc.) and add 1 to it.

**Risk Exposure**,

RE, determined using:

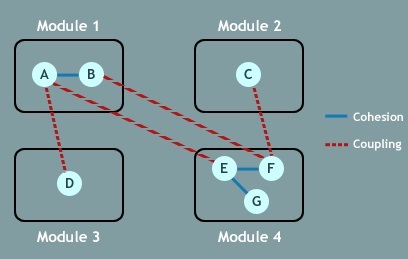
RE = P x C

P is the probability of occurrence for a risk

C is the the cost to the project should the risk occur.

C = Average Componenent 100 LOC \* Amount \* Components to be developed

**COUPLING AND COHESION:-**



|  |  |
| --- | --- |
| **Cohesion** | **Coupling** |
| within module. | Coupling is the indication of the relationships between modules. |
| Cohesion shows the module’s relative functional strength. | Coupling shows the relative independence among the modules. |
| Cohesion is a degree (quality) to which a component / module focuses on the single thing. | Coupling is a degree to which a component / module is connected to the other modules. |
| While designing you should strive for ***high cohesion*** i.e. a cohesive component/ module focus on a single task (i.e., single-mindedness) with little interaction with other modules of the system. | While designing you should strive for ***low coupling***i.e. dependency between modules should be less. Cohesion is the indication of the relationship |
| Cohesion is the kind of natural extension of data hiding for example, class having all members visible with a package having default visibility. | Making private fields, private methods and non public classes provides loose coupling. |
| Cohesion is Intra – Module Concept. | Coupling is Inter -Module Concept. |

**REFACTORING:-**

"Refactoring is the process of changing a software system in such a way that it does not alter the external behavior of the code yet improves its internal structure." -- MartinFowler

Changing the code only not the external behaviour or adding new feature.

**DIFFERENT TYPES OF COUPLING:-**

**Data Coupling**

Two modules are data coupled if they communicate by passing parameters. This has been told to you as a "good design principle" since day one of your programming instruction.

Diagram

**Stamp Coupling**

Two modules are stamp coupled if they communicate via a passed data structure that contains more information than necessary for them to perform their functions.

**Control Coupling**

Two modules are control coupled if they communicate using at least one "control flag".

**Common Coupling**

Two modules are common coupled if they both share the same global data area. Another design principle you have been taught since day one: don't use global data.

**Content Coupling**

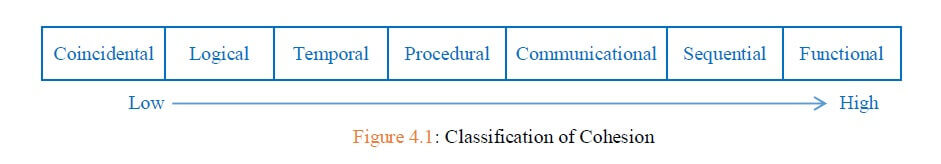
Two modules are content coupled if:

one module changes a statement in another (Lisp was famous for this ability)

one module references or alters data contained inside another module

one module branches into another module

**DIFFERENT TYPE OF COHESION:**



**CMM's Five Maturity Levels of Software Processes**

1.At the **initial level**, processes are disorganized, even chaotic. Success is likely to depend on individual efforts, and is not considered to be repeatable, because processes would not be sufficiently defined and documented to allow them to be replicated.

2.At the **repeatable level**, basic project management techniques are established, and successes could be repeated, because the requisite processes would have been made established, defined, and documented.

3.At the **defined level,** an organization has developed its own standard software process through greater attention to documentation, standardization, and integration.

4.At the **managed level**, an organization monitors and controls its own processes through data collection and analysis.

5.At the **optimizing level**, processes are constantly being improved through monitoring feedback from current processes and introducing innovative processes to better serve the organization's particular needs.

**Functional requirements are** those which are related to the technical functionality of the system.

**Non-Functional requirement** is a requirement that specifies criteria that can be used to judge the operation of a system in particular conditions, rather than specific behaviors.

For example if you consider a shopping site, adding items to cart, browsing different items, applying offers and deals and successfully placing orders comes under functional requirements.

Whereas performance of the system in peak hours, time taken for the system to retrieve data from DB, security of the user data, ability of the system to handle if large number of users login comes under non functional requirements.

**Some of the more typical functional requirements include:-**

Business Rules, Transaction corrections, adjustments and cancellations, Administrative functions, Authentication, Authorization levels, Audit Tracking, External Interfaces, Certification Requirements, Reporting Requirements, Historical Data, Legal or Regulatory Requirements.

**Some typical non-functional requirements are**:

Performance – for example Response Time, Throughput, Utilization, Static Volumetric Scalability, Capacity, Availability, Reliability, Recoverability, Maintainability, Serviceability, Security, Regulatory, Manageability, Environmental, Data Integrity,Usability, Interoperability

**Type of software maintenance:-**

Adaptive, corrective, perfective and preventive are the four types of software maintenance.

**Corrective maintenance:-**

It is concerned with fixing errors that are observed when the software is in use.

**Adaptive Maintenance:-**

The modification of the software to match changes in the ever changing environment, falls under adaptive category of software maintenance

Modification of a software product performed after delivery to keep a software product usable in a changed or changing environment.

**Perfective maintenance:-**

Perfective maintenance: Modification of a software product after delivery to improve performance or maintainability. It is concerned with the change in the software that occurs while adding new functionalities in the software.

**Preventive maintenance**

It involves implementing changes to prevent the occurrence of errors. The distribution of types of maintenance by type and by percentage of time consumed. This includes modifications and updations to prevent future problems of the software

**Quality assurance as well as Quality control Factor:-**

**Reliability**

Measure if product is reliable enough to sustain in any condition. Should give consistently correct results.

Product reliability is measured in terms of working of project under different working environment and different conditions.

**Maintainability**

Different versions of the product should be easy to maintain. For development its should be easy to add code to existing system, should be easy to upgrade for new features and new technologies time to time.

**Usability**

This can be measured in terms of ease of use. Application should be user friendly. Should be easy to learn. Navigation should be simple.

**Reverse Engineering:-**

Designers then do reverse engineering by looking at the code and try to get the design. With design in hand, they try to conclude the specifications. Thus, going in reverse from code to system specification.

**UMBRELLLA ACTIVITY:-**

umbrella activities that persist across the entire software process. These umbrella activities include:

software project management, formal technical reviews, software quality assurance

• software configuration management

• reusability management

• measurement

• document preparation and production

• risk management