White Box Testing	Black Box Testing
① It is a way of software testing in which the tester has knowledge about the internal structure or the code or the program of the software	① It is a way of software testing in which the internal structure or the program or the code is hidden and nothing is known about it.
② It is mostly done by software developers.	② It is mostly done by software testers.
③ Knowledge of implementation is required.	③ No knowledge of implementation is needed.
④ It is the inner or the internal software testing	④ It can be referred as outer or external software testing
⑤ It is structural test of the software.	⑤ It is functional test of the software.
⑥ This type of testing of software is started after detail design document.	⑥ This testing can be initiated on the basis of requirement specifications document.
⑦ It is mandatory to have knowledge of programming	⑦ No knowledge of programming is required.

(8) It is the logic testing of the software.	(8) It is the behavior testing of the software.
(9) It is generally applicable to the lower levels of software testing.	(9) It is applicable to the higher levels of testing of software.
(10) It is also called as clear box testing.	(10) It is also called as closed testing
(11) It is most time consuming	(11) It is least time consuming
(12) It is suitable for algorithm testing	(12) It is not suitable or preferred for algorithm testing
(13) Data domains along with inner or internal boundaries can be better tested	(13) Can be done by trial and error ways and methods.
(14) Example: By input to check and verify loops	(14) Example: Search something on google by using keywords.
(15) Types of white box testing A. Path Testing B. Loop Testing C. Condition Testing	(15) Types of black box testing A. Functional Testing B. Non-Functional Testing C Regression Testing

Functional Testing	Non-Functional Testing
① It verifies the operations and actions of an application.	① It verifies the behavior of an application.
② It is based on requirements of customer.	② It is based on expectations of customer.
③ It helps to enhance the behavior of the application.	③ It helps to improve the performance of the application
④ Functional testing is easy to execute manually.	④ It is hard to execute non-functional testing manually
⑤ It tests what the product does.	⑤ It describes how the product does.
⑥ Functional testing is based on the business requirements	⑥ Non-Functional testing is based on the performance requirement.
⑦ Examples: A. Unit Testing B. Smoke Testing C. Integration Testing D. Regression Testing	⑦ Examples: A. Performance Testing B. Load Testing C. Stress Testing D. Scalability Testing.

Verification	Validation
① It includes checking documents, design, codes and programs.	① It includes testing and validating the actual product.
② Verification is the static testing	② Validation is the dynamic testing
③ It does not include the execution of the code.	③ It includes the execution of the code.
④ Methods used in verification are reviews, walkthroughs, inspections and desk-checking.	④ Methods used in validation are Black Box Testing, White Box Testing, and Non-Functional Testing
⑤ It checks whether the software conforms to specifications or not.	⑤ It checks whether the software meets the requirements and expectations of a customer or not
⑥ It can find the bugs in the early stage of the development.	⑥ It can only find the bugs that could not be found by the verification process.
⑦ The goal of verification is application and software architecture and specification.	⑦ The goal of validation is an actual product.

(7) Quality Assurance team does verification.	(8) Validation is executed on software code with the help of testing team.
(9) It comes before validation.	(9) It comes after verification.
(10) It consists of checking of documents / files and is performed by human.	(10) It consists of execution of program and is performed by computer.

White Box Testing

- White box testing is a testing technique, that examines the program structure and derives test data from the program logic / code.
- The other names of white box testing are clear box testing, open box testing, logic driven testing or path driven testing.
- White box testing lets testers determine how thoroughly the program needs to be tested, which is often done via adequacy criterion. A test adequacy criterion is a predicate which is used to determine whether a program has been tested enough.
- An example of a white box testing would be in-circuit testing where the interconnections between each component and verified and each internal connection is checked for its working.
- White box testing is mainly used for detecting logic errors, uncovering incorrect programming assumptions
- White box testing is done at low level design and it can be applied at all levels of system development especially unit testing and integration testing.

- Following are the different White Box Testing Techniques

① Statement Coverage :

- The statement coverage strategy aims to design test cases so that every statement in a program is executed at least once.

② Branch Coverage :

- In the branch coverage-based testing strategy, test cases are designed to make each branch condition to assume true and false values in turn.

③ Condition Coverage:

- In this structural testing, test cases are designed to make each component of a composite conditional expression to assume both true and false values.

④ Path Coverage :

- The path coverage-based testing strategy requires us to design test cases such that all linearly independent paths are executed at least once.
- A linearly independent path can be defined in terms of the Control Flow Graph (CFG) of a program.

⑤ Data Flow - Based Testing:

- Data flow-based testing method selects test paths according to the locations of the definitions and uses of different variables in a program.

- Advantages of White Box Testing

1. Forces test developer to reason carefully about implementation.
2. Reveals Errors in "hidden" code
3. Spots the Dead Code or other issues with respect to best programming practices.

- Disadvantages of White Box Testing

1. Expensive as one has to spend both time and money to perform white box testing
2. Every possibility that few lines of code are missed accidentally
3. In-depth knowledge about the programming language is necessary to perform white box testing

Black Box Testing

- Black Box Testing is a method of software testing that examines the functionality of an application based on the specifications. It is also known as specification - based testing. Independent testing team usually performs this type of testing during the software testing life cycle.
- This method of test can be applied to each and every level of software testing such as unit, integration testing and acceptance testing.
- This testing methodology looks at what are the available inputs for an application and what the expected outputs are that should result from each input.
- Black Box Testing is done on completely finished product.
- Black Box Testing is done based on end user perspective. The main importance of black box testing is it handles both valid and invalid inputs from customer's perspective.
- Following are categories of errors that can be detected by using black box testing

1. The performance errors or the behavioural errors.
2. Missing or incorrectly defined functions.
3. Data structures incorrectly defined and external database access errors.
4. Errors occured during initialization and termination
5. Errors occured during interface.

- There are two main approaches for black box testing.

① Equivalence Class Partitioning:

- The possible input values to the program are partitioned into a set of equivalence classes.
- An equivalence class is a set of data of the same type and can be defined by examining the input values.
- Example: If black box testing is applied for a program computing square of the input within the range 0 to 5000 then three equivalence classes can be defined.
  - → Class 1 consists of numbers less than zero.
  - → Class 2 consists of numbers in the range 0 to 5000.
  - → Class 3 consists of numbers greater than 5000.

② Boundary Value Analysis:

- A type of programming error frequently occurs at the boundaries of different equivalence classes of inputs.
- Example: Programmers may improperly use < instead of <= or conversely <= for <.
- Boundary value analysis leads to selection of test cases at the boundaries of the different equivalence classes.

- Types of Black Box Testing

Note: There are many types of black box testing but the following are the prominent ones.

① Functional Testing:

- This black box testing type is related to the functional requirements of a system; it is done by software testers.

② Non-Functional Testing:

- This type of black box testing is not related to testing of specific functionality, but non-functional requirements such as performance, scalability, usability.

③ Regression Testing:

- Regression Testing is done after code fixes, upgrades or any other system maintenance to check the new code has not affected the existing code.

- Advantages of Black Box Testing

① Testers don't need to have the knowledge of programming or implementation.

② Both programmers and Testers are independent of each other.

- Disadvantages of Black Box Testing

① All the boundary conditions are difficult to detect.

② Can be difficult to cover all portions of software implementation.

Path Testing:

- Path Testing is a structural testing method that involves using a source code of a program in order to find every possible executable path.
- It helps to determine all faults lying within a piece of code.
- This method is designed to execute all or selected path through a computer program.

Basis Path Testing:

- The basis path testing is same, but it is based on a White Box Testing method., that defines test cases based on the flows or logical path that can be taken through the program., Basis Path Testing involves execution of all possible blocks in a program and achieves maximum path coverage with least number of test cases. It is a hybrid of branch testing and path testing methods.
- The objective behind basis path testing is that it defines the number of independent paths, thus the number of test cases needed can be defined explicitely. (Maximizes the coverage of each test case).

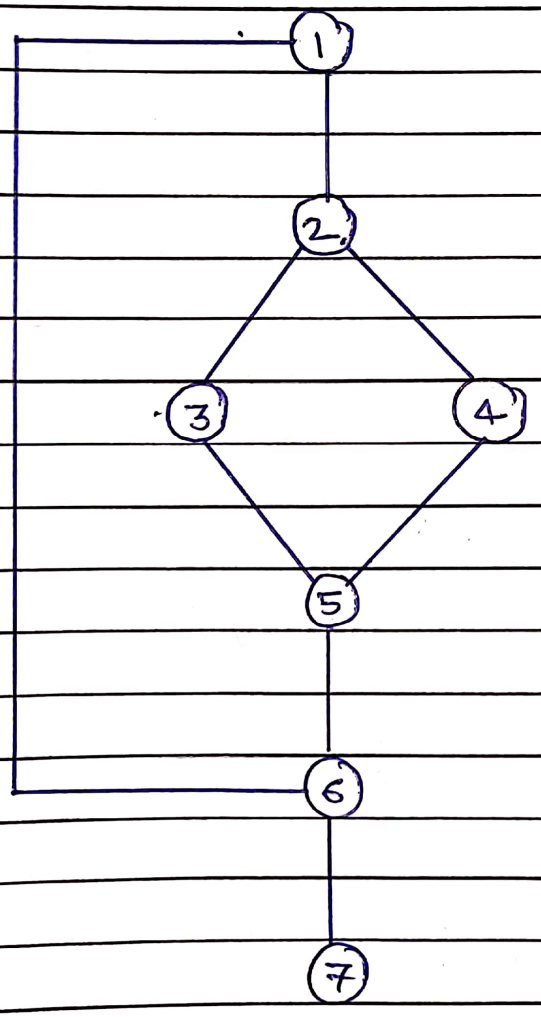
- Example: In the example below, we can see there are few conditional statements that is executed depending on what condition is suffice.

Here, there are 3 path or condition that need to be tested to get the output,

→ Path 1 : 1, 2, 3, 4, 5, 6, 7

→ Path 2 : 1, 2, 4, 5, 6, 7

→ Path 3 : 1, 6, 7



1. If A = 50
2. THEN IF B > C
3. THEN A = B
4. ELSE A = C
5. ENDIF
6. ENDIF
7. Print A

Steps for Basis Path Testing

- The basic steps involved in basis path testing include

1. Draw a control graph (to determine different program paths).
2. Calculate cyclomatic complexity (metrics to determine the number of independent paths.)
3. Find a basis set of paths.
4. Generate test cases to exercise each path.

Benefits of basis path testing

1. It helps to reduce the redundant tests.
2. It focuses attention on program logic
3. It helps facilitates analytical versus arbitrary case design.
4. Test cases which exercise basis set will execute every statement in a program at least once.

Software Configuration Management (SCM)

- SCM is a software engineering discipline consisting of standard processes and techniques often used by organizations to manage the changes introduced to its software products.
- SCM helps in identifying individual elements and configurations, tracking changes and version selection.
- SCM is also known as software Control Management.
- SCM aims to control changes introduced to large complex software systems through reliable version selection and version control.
- Processes involved in SCM are:
  1. Identification and Establishment
  2. Version Control
  3. Change Control
  4. Configuration auditing
  5. Reporting
- Participants of SCM Process:
  1. Configuration Manager
  2. Developer
  3. Auditor
  4. Project Manager
  5. User
- Software Configuration Management tools:
  1. Concurrency Management
  2. Version Control
  3. Synchronization.

Popular tools: Git, Team Foundation Server, Ansible, etc.

Version Control

- Software Version Control is a system or tool that captures the changes to a source code element: files, folders, images or binaries.
- A version control system (also known as Revision Control System) is a repository of files, often the files for the source code of computer programs, with monitored access. Every change made to the source is tracked, along with who made the change, why they made it and references to problems fixed or enhancements introduced by the change.
- Version control systems are essential for any form of distributed, collaborative development. Whether it is the history of a wiki page or large software development project, the ability to track each change as it was made, and to reverse changes when necessary can make all the difference between a well managed and well controlled process and an uncontrolled 'first come, first served' system.
  It can also serve as a mechanism for due diligence for software projects.
- Combines procedures and tools to manage the different versions of configuration objects created during the software process.

- Version control systems require the following capabilities.

① Project Repository:
- stores all relevant configuration objects.

② Version Management Capability:
- Stores all versions of a configuration object (enables any version to be built from past versions).

③ Make facility:
- Enables collection of all relevant configuration objects and construct a specific software version.

④ Issues (bug) Tracking Capability:
- Enables team to record and track status of outstanding issues for each configuration object.

- Uses a system modelling approach (template - includes component hierarchy and component build order, construction rules, verification rules).

Change Control

- Change control is a systematic approach to managing all changes made to a product or system. The purpose is to ensure that no necessary changes are made, that all changes are documented, that services are not unnecessarily disrupted and that resources are used efficiently.
- Here's an example of a six-step-process for a software change request.

① Documenting the change request:

- When the client requests the change, that request is categorized and recorded, along with informal assessments of the importance of that change and the difficulty of implementing it.

② Formal Assessment:

- The justification for the change and risks and benefits of making / not making the change are evaluated.
- If the change request is accepted, a development team will be assigned.
- If the change request is rejected, that fact is documented and communicated to the client.

③ Planning:

- The team responsible for the change creates a detailed plan for its design and implementation, as well as a plan for rolling back the change should it be deemed unsuccessful.

④ Designing and Testing:

- The team designs the program for the software change and tests it.
- If the change is deemed successful, the team requests approval and a date for implementation.

⑤ Implementation and Review:

- The team implements the program and stakeholders review the change.

⑥ Final Assessment:

- If the client is satisfied that the change was implemented satisfactorily, the change request is closed.
- If the client is not satisfied, the project is reassessed and steps may be repeated.

Cyclomatic Complexity

- Cyclomatic complexity is a software metric (measurement)
- It was developed by Thomas J. McCabe, Sr. in 1976 and used to indicate the complexity of a program.
- It is quantitative measure of the number of linearly independent paths through a program's source code.
- Cyclomatic complexity is computed using the control flow graph of the program: the nodes of the graphs correspond to indivisible groups of commands of a program, and a directed edge connects two nodes if the second command might be executed immediately after the first command.

How cyclomatic complexity is calculated.

- Cyclomatic complexity may also be applied to individual functions, modules, methods or classes within a program.
- The cyclomatic complexity of a section of source code is the count of the number of linearly independent paths through the source code.

For instance, if the source code contained no decision points such as IF statements or FOR loops, the complexity would be 1, since there is only a single path through the code. IF the code had a single IF statement containing a single condition there would be two paths through the code, one path where the IF statement is evaluated as TRUE and one path where the IF statement is evaluated as False.

- Mathematically, the cyclomatic complexity of a structured program is defined with reference to a directed graph containing the basic blocks of the program, with an edge between two basic blocks if the control may pass from the first to the second (the control flow graph of the program).
- The complexity is then defined as :

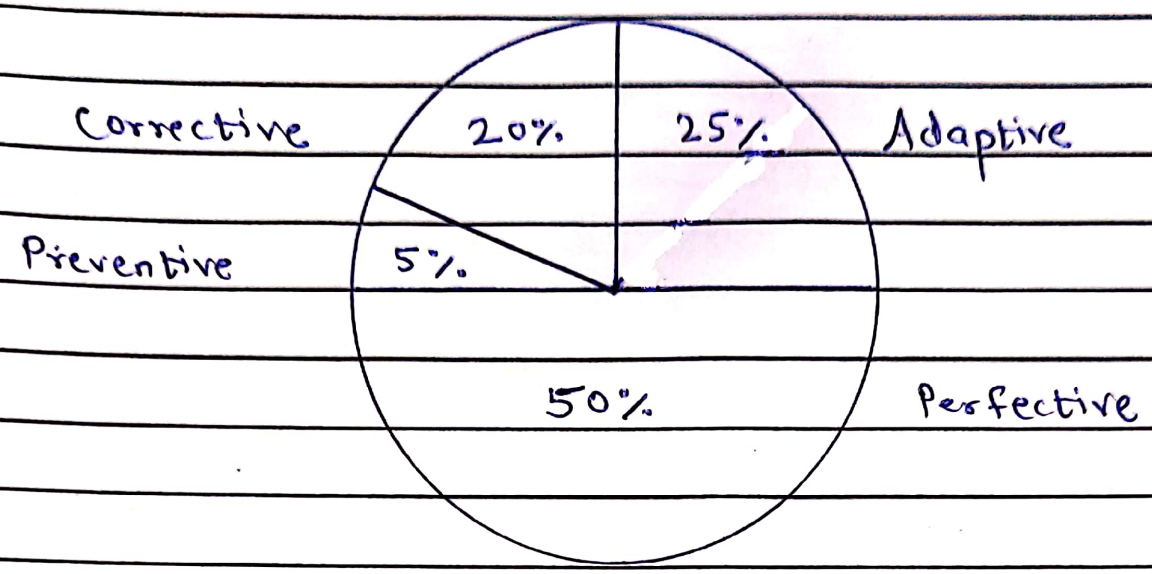
$$M = E - N + P$$

where,

M = Cyclomatic Complexity
E = The number of edges of the graph
N = The number of nodes of the graph
P = The number of connected components.

- Cyclomatic Complexity = Total number of regions in the flow graph.

Types of Software Maintenance



Types of Maintenance

Following are the four types of maintenance.

① Corrective Maintenance :

- The corrective maintenance is the type of maintenance in which the errors are fixed when it is observed during the use of software.
- In this the errors may be caused due to faulty software design, incorrect logic and improper coding. All these errors are called as residual errors and they prevent the final specification.
- In case of system failure, the corrective maintenance approach is initiated to locate the original specification. Corrective maintenance normally used for more than 20% of all the maintenance activities.

② Adaptive Maintenance:

- The adaptive maintenance means the implementation of the modification in the system. The changes may be of hardware or the operating system environments.
- The examples of adaptive maintenance are:
  1. Business rules
  2. Work Patterns
  3. Government policies that have substantial impact on the software.
- Adaptive maintenance normally used for more than 25% of all the maintenance activities.

③ Perfective Maintenance

- Perfective maintenance deals with the modified and changed user requirements. The functional enhancements are taken into consideration in perfective maintenance.
- In perfective maintenance, the function and efficiency of the code is continuously improved.
- Following are the examples of perfective maintenance:
  1. Modifying the payroll program to add new union settlement.
  2. Adding a new report in the sales analysis system.
- Perfective maintenance normally used for more than 50% of all the maintenance activities and it is the largest among all.

④ Preventive Maintenance:

- Preventive maintenance is used to prevent the possible errors to occur. Thus in this activity, the complexity is minimized and the quality of the program is enhanced.
- Adaptive maintenance normally used for 5% of all the maintenance activities and it is the smallest among all.

Software Reverse Engineering

- The reverse engineering is the discipline of software engineering, where the knowledge and design information is extracted from the source code or it is reproduced.
- The reverse engineering is a process where the system is analyzed at higher level of abstraction.
- It is also called as going backward through all the development cycles.
- Following are some important purposes of Reverse Engineering
  1. Security auditing
  2. Enabling Additional features
  3. Used as learning tools.
  4. Developing compatible products cheaper than that are currently available in the market.
- Following are three important parameters to be considered for a reverse engineering process.

1. Abstraction Level:

- In the abstraction level of a reverse engineering process the design information is extracted from the source code.
- It is the highest level in the reverse engineering process.
- It is always expected that the abstraction level for any reverse engineering process must be high.
- When the level of abstraction is high, then it becomes easy for the software developer to understand the program.

② Completeness:

- The completeness is nothing but the details available through the abstraction level of reverse engineering process.
- For example, from the given source code it is very easy to develop the complete procedural design.

③ Directionality:

- The directionality of a reverse engineering process is a method of extracting information from the source code and making it available to the software developer.
- The software developers can use this information during the maintenance activity.

The reverse engineering process include the following steps

① The raw or dirty source code obtained from the abstraction level is taken as input.

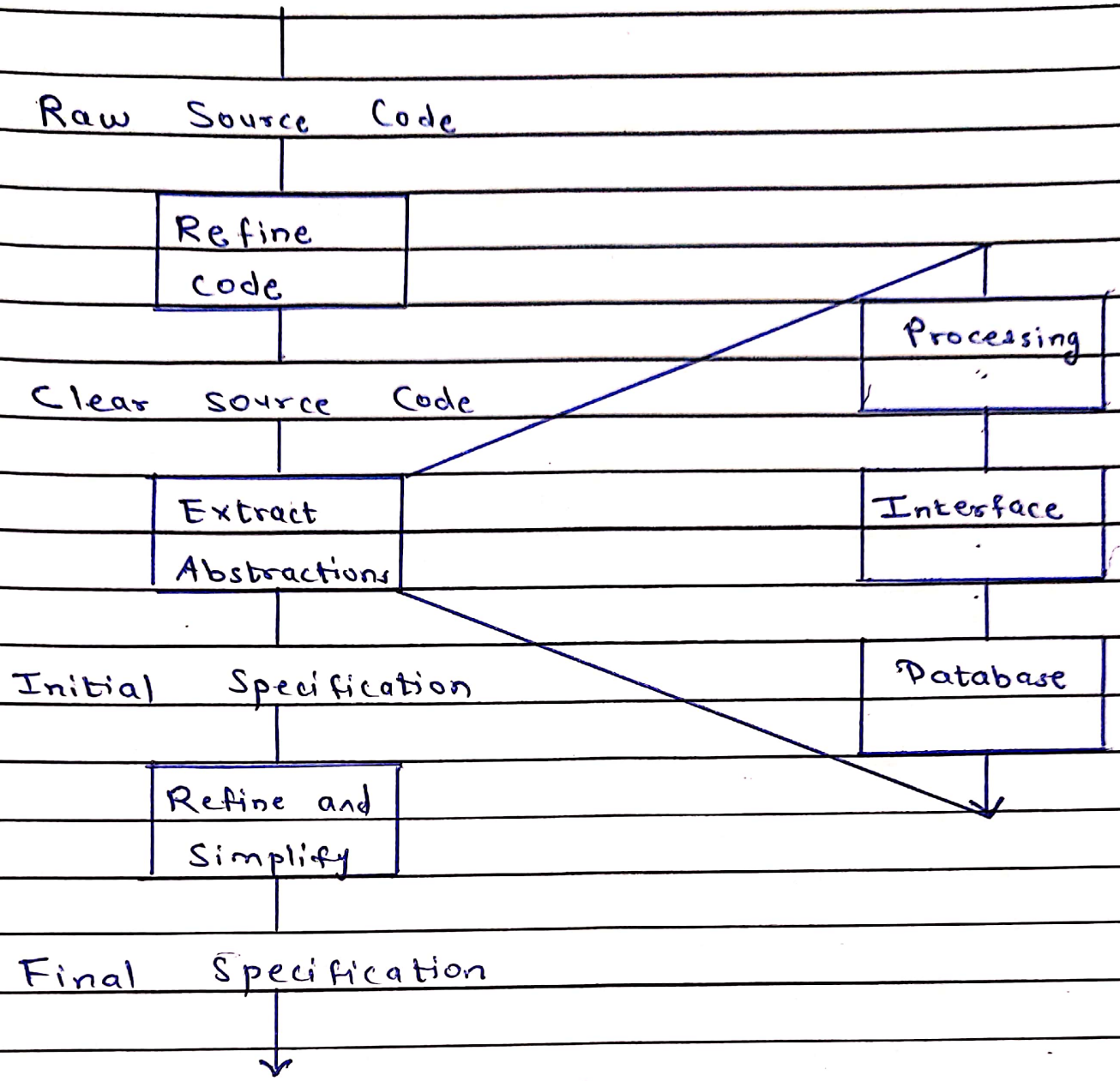
② This code is refined or restructured. (Clean code is obtained).

③ From the clean code, the abstraction is extracted.

④ From this initial specification, the code is refined and simplified.

⑤ Now, we get the final specification.

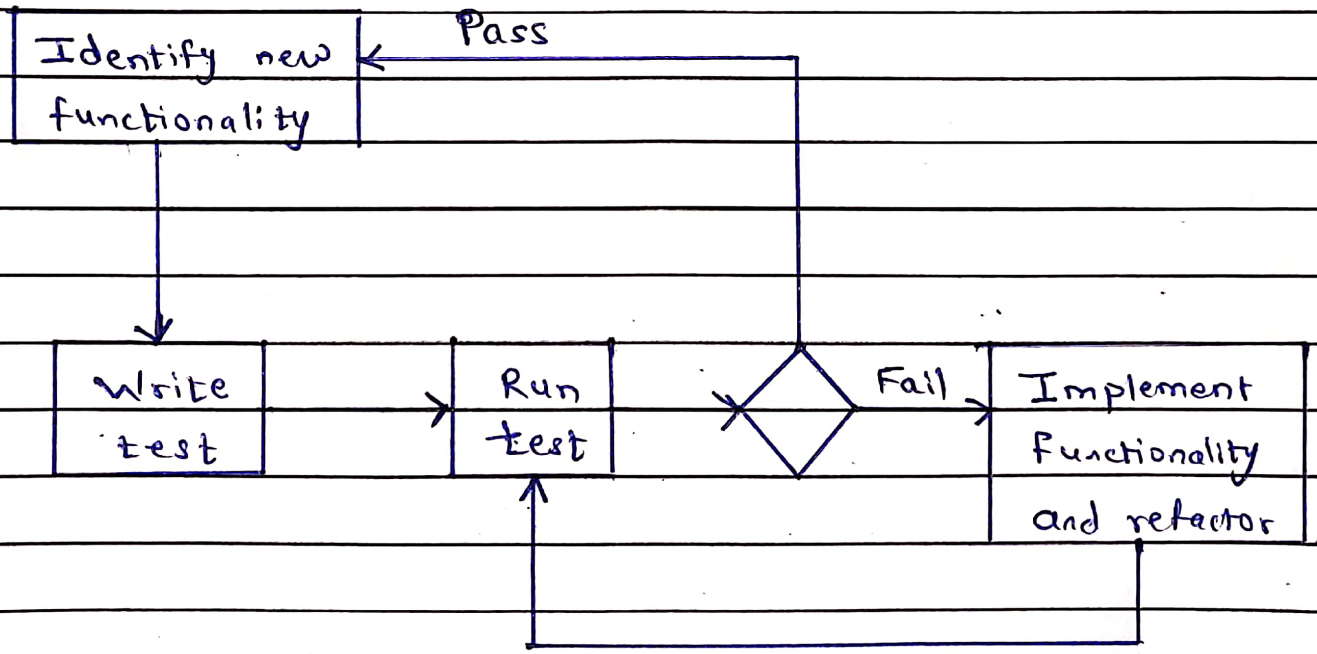
Thus the final specification obtained from the reverse engineering process, is used as final specification by the software developers.



A Reverse Engineering Process

Test Driven Development (TDD)

- Test Driven Development (TDD) is a development approach in that testing and coding are combined together.
- Each of the increment is coded and at the same time it is tested too.
- Until and Unless the last increment (versions) is not fully tested, the next increment is not coded.
- The TDD was introduced as a part of agile software development process. For example: Extreme Programming.



A sample Test Driven Development.

Following are steps in the Test Driven Development process.

1. Identify the functionality of the increment
2. Write the proper test for the functionality identified and implement it.
3. Run the test written for the functionality.
4. Refactor the code to improve it.
5. After executing all the test, move to the next set of functionalities.

- This may be implemented by using automated testing environment. eg JUnit environment that supports Java Programming Testing

Advantages of Test Driven Development:

1. Code Coverage:
   - All the segments of code must have an associated test. Therefore the errors are detected early in the coding process itself.

2. Regression Testing:
   - It is used to find that the program has not introduced any new errors after conducting the test.

3. Simplified Debugging:
   - The TDD process normally avoids the automated testing tools due to simplified debugging process.

4. System Documentation:
   - The test plan itself acts as ~~an~~ document. After reading the test, one can easily understand the code.

Steps for creating a maintenance log

- A log file, in a computing context, is the automatically produced and time-stamped documentation of events relevant to a particular system.
- Virtually all software applications and systems produce log files.
- For maintenance activities in software applications, log files have its own significance.
- Steps for creating a maintenance log file:

1. Make all entries of data exist in the database for the application under maintenance in the maintenance log file before start of maintenance activity.
2. Perform routine maintenance and make entry of each activity in the log file.
3. Keep electronic record of interactions that have occured during maintenance activities.
4. Check database state and compare it with the database state before the maintenance initiated.
5. Restore the data to its original state for any inconsistent state.

Software Maintenance

- Software maintenance is the process of modifying a software product after it has been delivered to the customer.
- The main purpose of software maintenance is to modify and update software application after delivery to correct faults and to improve performance.

Need for Maintenance

- Software maintenance must be performed in order to:

1. Correct faults.
2. Improve the design.
3. Implement enhancements
4. Interface with other systems
5. Accomodate programs so that different hardware, software, system features and telecommunication facilities can be used.
6. Migrate legacy software
7. Retire software

Types of software maintenance

1. Corrective maintenance
2. Adaptive maintenance
3. Perfective Maintenance
4. Preventive Maintenance

Software Design

- It is a process to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.
- It deals with representing the client's requirement as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable using programming language.

Fundamental of software design concepts:

1. Abstraction
2. Modularity
3. Architecture
4. Refinement
5. Pattern
6. Information Hiding
7. Refactoring

Fundamental of software design concepts:

① Abstraction:

- Hide irrelevant data.
- Abstraction simply means to hide the details to reduce complexity and increases efficiency or quality.
- Different levels of abstraction are necessary and must be applied at each stage of the design process. So that any error that is present can be removed to increase the efficiency of the software solution and to refine the software solation. The solution should be described in broadways that covers a wide range of different things at a higher level of abstraction and more detailed description of a solution of software should be given at the lower level of abstraction.

② Modularity:

- Subdivide the system
- Modularity simply means to divide the system or project into smaller parts to reduce the complexity of the system or project.
- In the same way, modularity in design means to subdivide a system into smaller parts so that these parts can be created independently and then use these parts in different systems to perform different functions.
- Modularity in design has now became a trend and is also important.

③ Architecture:

- Design a structure of something.
- Architecture simply means a technique to design a structure of something.
- Architecture in designing software is a concept that focuses on various elements and the data of the structure.
- These components interact with each other and use the data of the structure in architecture.

④ Refinement:

- Removes impurities.
- Refinement simply means to refine something to remove any impurities if present and increase the quality.
- The refinement concept of software design is actually a process of developing or presenting the software or system in a detailed manner. that means to elaborate a system or software.

⑤ Pattern:

- A repeated form
- A pattern simply means a repeated form or design in which the same shape is repeated several times to form a pattern
- The pattern in the design process means the repetition of the solution to a common recurring problem within a certain context.

⑥ Information Hiding :

- Hide the information
- Information hiding simply means to hide the information so that it cannot be accessed by an unwanted party.
- In software design, information hiding is achieved by designing the modules in a manner that the information gathered or contained in one module is hidden and it can't be accessed by any other modules.

⑦ Refactoring :

- Reconstruct something.
- Refactoring simply means to reconstruct something in such a way that it does not affect the behavior or any other features.
- Refactoring in software design means to reconstruct the design to reduce complexity and simplify it without affecting the behavior or its functions.
- Fowler has defined refactoring as "the process of changing the software system in a way that it won't affect the behavior of the design and improves the internal structure."

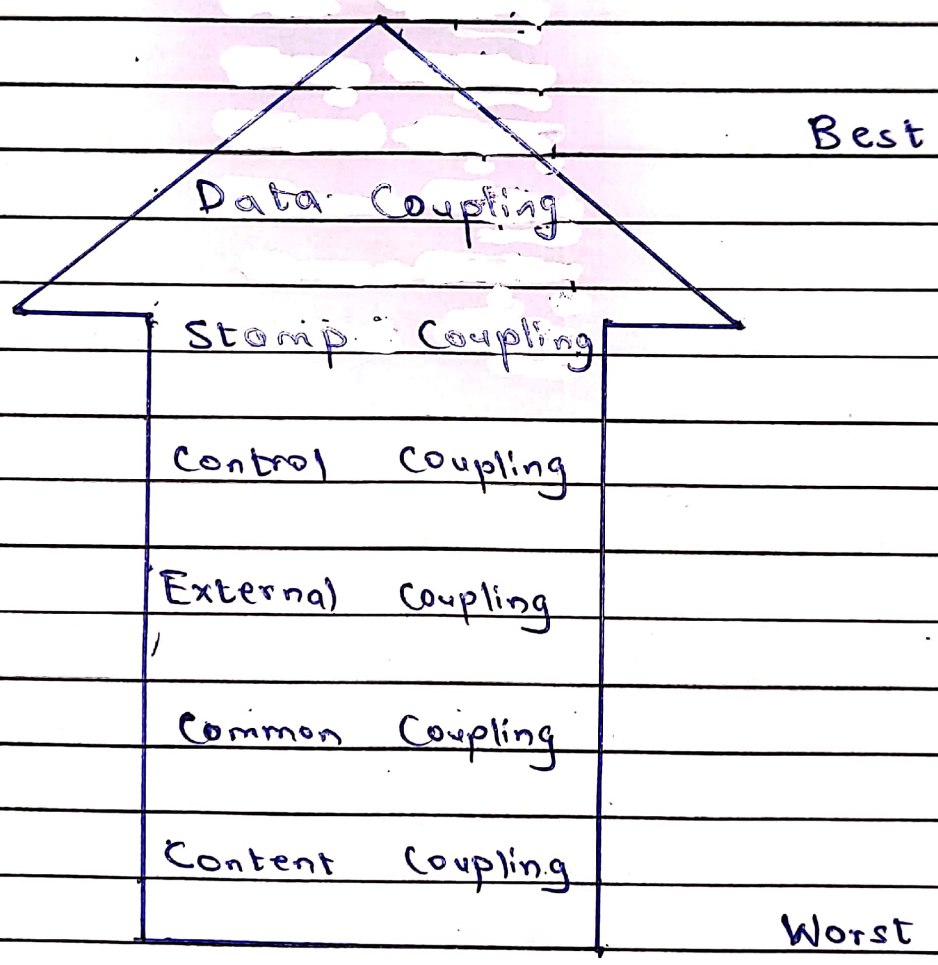
Basis of comparison	Cohesion	Coupling
Description	Cohesion represents the degree to which a part of a code base forms a logically single atomic unit.	Coupling represents the degree to which a single unit is independent from others. (Coupling is the number of connections between two or more units.)
Concept	It is an intra-module concept.	It is an inter-module concept.
What does it depicts?	It depicts the module's relative functional strength	It depicts the relative independence among modules
What does it entail?	High cohesion is about keeping parts of a code base that are related to each other in a single place.	Low coupling is about separating unrelated parts of the code base as much as possible.
Types / Classes	① Coincidental Cohesion ② Logic Cohesion ③ Temporal Cohesion ④ Procedural Cohesion ⑤ Communication Cohesion ⑥ Sequential Cohesion ⑦ Functional Cohesion	① Data Coupling ② Control Coupling ③ Stamp Coupling ④ Common Coupling ⑤ Content Coupling

STUDENT SIGNATURE: Amey

Manipulation	Cohesion is a kind of natural extension of data hiding. For example: The class having all members visible with a package having default visibility	Making private fields, private methods and non-public classes provides loose coupling
Designing	While designing, you need to strive for high cohesion i.e. focus on a single task with little interaction with other modules of the system.	While designing, you need to strive for low coupling i.e., dependence between modules should be less.
Increase in Coupling v/s Cohesion	Increase in cohesion is good for software.	Increase in coupling is avoided for software.
How best software is achieved?	High cohesion gives the best software	Loose coupling gives the best software
Possibility	It is possible to create fully cohesive code without introducing unnecessary coupling.	It is impossible to achieve full decoupling without damaging cohesion.

Coupling

- Coupling is the measure of the degree of interdependence between the modules.
- A good software will have low coupling



Types of coupling:

① Data coupling:

- If the dependency between the modules is based on the fact that they communicate by passing only data, then the modules are said to be data coupled.
- In data coupling, the components are independent to each other and communicating through data.
- Module communications don't contain tramp data.
- Example: customer billing system

② Stamp coupling:

- In stamp coupling, the complete data structure is passed from one module to another module
- Therefore, it involves tramp data.
- It may be necessary due to efficiency factors - this choice made by the insightful designer, not a lazy programmer.

③ Control coupling:

- If the modules communicated by passing control information, then they are said to be control coupled.
- It can be bad if parameters indicate completely different behavior and good if parameters allow factoring and reuse of functionality
- Example: sort function that takes comparison function as an argument.

④ External coupling:

- In external coupling, the modules depend on other modules, external to the software being developed or to a particular type of hardware.
- Example: Protocol, external file, device format, etc.

⑤ Common coupling:

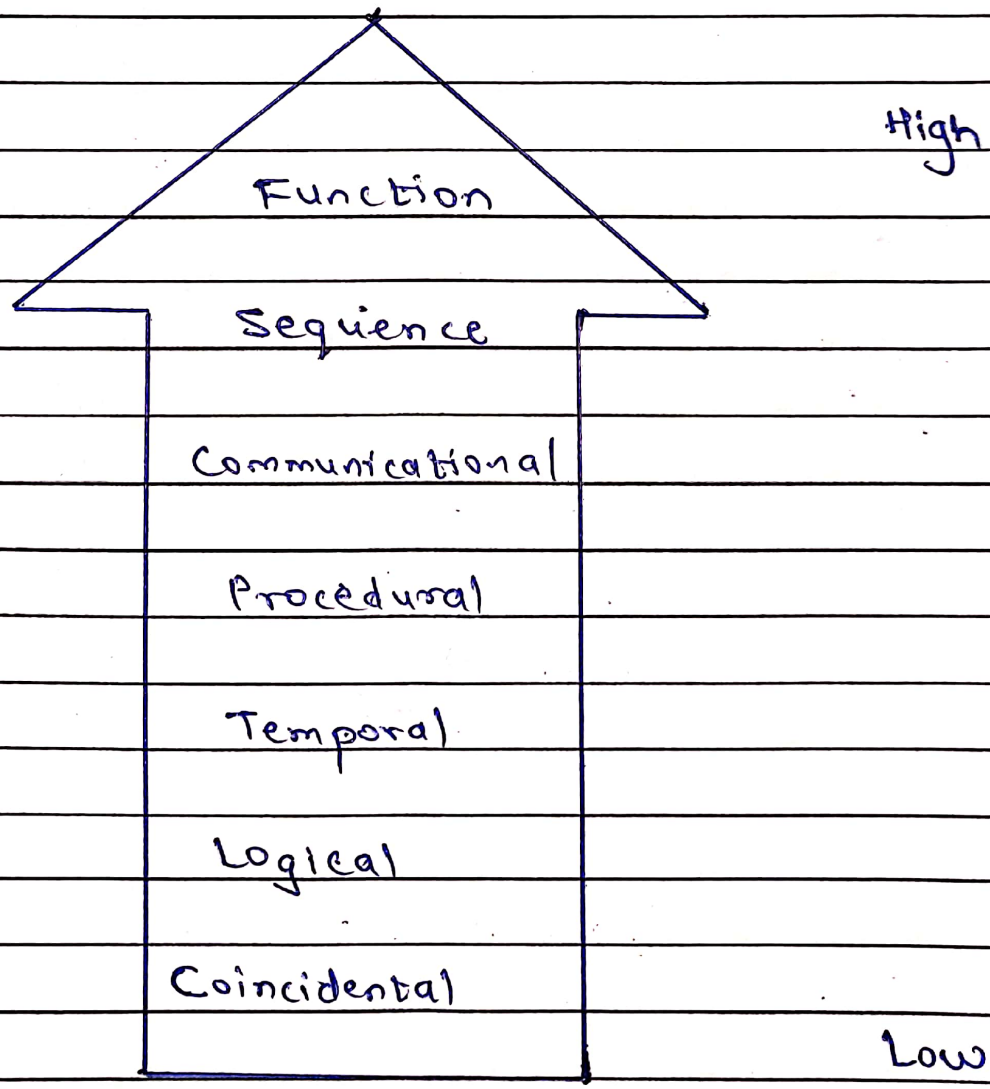
- The modules have shared data such as global data structures.
- The changes in global data means tracking back to all modules which access that data to evaluate the effect of the change.
- So it has got disadvantages like difficulty in reusing modules, reduced ability to control data accesses and reduce maintainability.

⑥ Content coupling:

- In a content coupling, one module can modify the data of another module or control flow is passed from one module to the other module.
- This is the worst form of coupling and should be avoided.

Cohesion

- Cohesion is a measure of the degree to which the elements of the module are functionally related.
- It is the degree to which all elements directed towards performing a single task are contained in the component.
- Basically, cohesion is the internal glue that keeps the module together.
- A good software design will have high cohesion.



Types of Cohesion

① Functional Cohesion :

- Every essential element for a single computation is contained in the component.
- A functional cohesion performs the task and functions.
- It is an ideal situation.

② Sequential Cohesion

- An elements outputs some data that becomes the input for other element, i.e., data flow between the parts. It occurs naturally in functional programming languages.

③ Communicational Cohesion:

- Two elements operate on the same input data or contribute towards the same output data
- Example: update record int the database and send it to the printer.

④ Procedural Cohesion:

- Elements of procedural cohesion ensures the order of execution.
- Actions are still weakly connected and unlikely to the reusable.
- Examples: Calculate student GPA, print student record, Calculate cumulative GPA, print cumulative GPA.

⑤ Temporal cohesion:

- The elements are related by their timing involved.
- A module connected with temporal cohesion all the tasks must be executed in the same time-span.
- This cohesion contains the code for initializing all the parts of the system.
- Lots of different activities occur, all at init time.

⑥ Logical cohesion:

- The elements are logically related and not functionally.
- Example: A component reads inputs from tape, disk, and network.
  All the code for these functions is in the same component.
  Operations are related but the functions are significantly different.

⑦ Coincidental cohesion:

- The elements are not related (unrelated).
- The elements have no conceptual relationship other than location in source code.
- It is accidental and worst form of cohesion.
- Example: Print next line and reverse the characters of a string in a single component.

Agile Methodology

- The agile process, light weight methods are people-based rather than plan-based methods.
- The agile process forces the development team to focus on software itself rather than design and documentation.
- The agile process believes in iterative method.
- The aim of agile process is to deliver the working model of software quickly to the customer.
- Example: Extreme programming is the best known of agile process.

Agile Methodology Example - Extreme Programming

Extreme Programming (XP):

- Extreme programming (XP) is one of the best known agile process.
- The extreme programming approach was suggested by Kent Beck in 2000.
- Extreme programming is explained as follows:

① Customer specifies and prioritize the system requirements. Customer becomes one of the important members of development team. The developer and customer together prepare a story-card in which customer needs are mentioned.

② The developer team then aims to implement the scenarios in story-card.

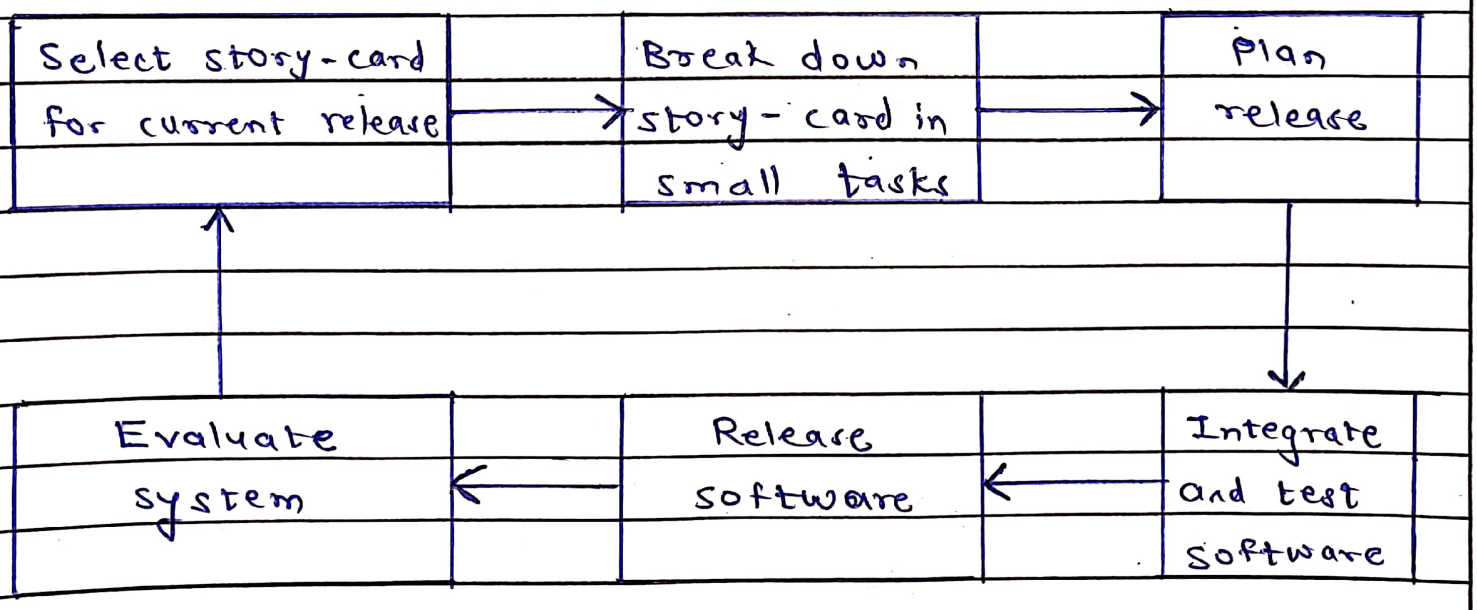
③ After developing the story-card the development team breaks down the total work in small tasks. The efforts and the estimated resources required for these tasks are estimated.

④ The customer prioritize the stories for implementation. If the requirement changes then sometimes unimplemented stories have to be discarded. Then release the complete software in small and frequent releases.

⑤ For accommodating, new changes, new story-card must be developed.

⑥ Evaluate the system along with customer.

- Process is demonstrated by the following figure.



Extreme programming Release Cycle

Principles of Agile methodology:

① Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.

② Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.

③ Deliver working software frequently, from a couple of weeks to a couple of months, with a preferences to a shorter timescale.

④ Business people and developers must work together daily throughout the project.

⑤ Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.

⑥ The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.

⑦ Working software is the primary measure of progress.

(8) Agile processes promote sustainable development. The sponsors, developers and users should be able to maintain a constant pace indefinitely.

(9) Continuous attention to technical excellence and good design enhances agility.

(10) Simplicity - the art of maximizing the amount of work not done - is essential.

(11) The best architectures, requirements and designs emerge from self - organizing teams.

(12) At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

Waterfall Model	Spiral Model
① Waterfall model works in sequential method.	① Spiral model works in evolutionary method.
② In waterfall model, errors or risks are identified and rectified after the completion of stages.	② In spiral model, errors or risks are identified and rectified earlier.
③ Waterfall model is adopted by customers.	③ Spiral model is adopted by developers.
④ Waterfall model is applicable for small projects.	④ Spiral model is used for large projects
⑤ In waterfall model, requirements and early stage planning is necessary.	⑤ In spiral model, requirements and early stage planning is necessary if required.
⑥ Flexibility to change in waterfall model is difficult.	⑥ Flexibility to change in spiral model is not difficult.
⑦ There is high amount risk	⑦ There is low amount risk
⑧ It is comparatively inexpensive	⑧ It is very expensive.

STUDENT SIGNATURE:

Software Requirement

- The software requirements are description of features and functionalities of the target system.
- Requirements convey the expectations of users from the software product.
- The requirements can be obvious or hidden, known or unknown, expected or unexpected from client's point of view.
- A condition or capability needed by a user to solve a problem or achieve an object.
- A condition or capability that must be met or possesed by a system or system component to satisfy a contract, standard, specification or other formally imposed documents

Types of requirement:

1. Functional requirement
2. Non - functional requirement
3. Domain requirement.

Types of requirements:

① Functional requirements:

- These are the requirements that the end user specifically demands as basic facilities that the system should offer.
- All these functionalities need to be necessarily incorporated into the system as a part of the contract.
- These are represented or stated in the form of input to be given to the system, the operation performed and the output expected.
- They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.
- Example: Banking system

  Functional requirements:

  ① User Bank Account Creation.

  ② Withdrawal of money

  ③ Cheque clearance.

② Non - functional requirements:

- These are basically the quality constraints that the system must be satisfy according to the project contract.
- The priority or extent to which these functions are implemented varies from one project to the other.
- They basically deal with issues like:
  1. Portability
  2. Security
  3. Maintainability
  4. Reliability
  5. Scalability
  6. Performance
  7. Reusability
  8. Flexibility

- Example: ATM system

  Non - functional requirements:

  1. System must be available for 24*7.

③ Domain requirements:

- These are the requirements which are characteristics of a particular category or domain of projects.
- The basic functions that systems of a specific domain must necessarily exhibit come under this category.
- Example: Academic system

  Domain requirements:

  ① In an academic software that maintains record of a school or college, the functionality of being able to access the list of faculty and list of students of each grade is a domain requirement.

Following are the requirement modelling strategies:

① Flow - oriented modelling

② Class - based modelling

① Flow oriented modelling:

- It shows how data objects are transformed by processing the function.
- The flow oriented elements are:

① Data flow model:

- It is graphical technique. It is used to represent information flow.
- The data objects are flowing within the software and transformed by processing the elements.
- The data objects are represented by labeled arrows. Transformation are represented by circles called as bubbles.
- DFD shown in a hierarchical fashion. The DFD is split into different levels. It is also called as Context level diagram.

② Control flow model:

- Large class applications require a control flow modelling
- The application creates control information instead of reports.
- The application process the information in specific time.
- An event is implemented as a boolean value.
- Example: the boolean values are true or false, On or off, 1 or 0.

③ Control specification:

- A short term for control specification is CSPEC.
- It represents the behavior of the system.
- The state diagram in CSPEC is a sequential specification of the behavior.
- The state diagram includes states, transitions, events and activities.
- State diagram shows the transition from one state to another state if particular event has occured.

④ Process specification:

- A short term for process specification is PSPEC.
- The process specification is used to describe all flow model processes.
- The content of process specification consists narrative text, Program Design Language (PDL) of the process algorithm, mathematical equations, tables or UML activity diagram.

② Class based modelling:

- Class based modelling represents the object: The system manipulates the operation.
- The elements of the class based models consist of classes and object, attributes; operations, class - responsibility - collaborator (CRS) models.

Eliciting Requirements Engineering

- Requirements elicitation is the practice of collecting the requirements of a system from users, customers and other stakeholders.
- The practice is sometimes referred to as requirement gathering
- The term elicitation is used in books and reasearch to raise the fact that good requirements cannot just be collected from the customer, as would be indicated by the name requirements gathering.
- Requirements elicitation is non-trivial because you can never be sure you get all requirements from the user and customer by just asking them what the system should do or not do.
- Requirement elicitation practices include interviews, questionnaires, user observation, workshops, brainstorming, use cases, role playing and prototyping.
- Before requirements can be analyzed, modeled or specified they must be gathered through an elicitation process.
- Requirements elicitation is a part of the requirements engineering process, usually followed by analysis and specification of the requirement

Requirement Gathering Techniques

- The following are some of the well-known requirements gathering techniques:

① Brainstorming:

- Brainstorming is used in requirement gathering to get as many ideas as possible from group of people.
- Generally used to identify possible solutions to problems and clarify details of opportunities.

② Document Analysis:

- Reviewing the documentation of an existing system can help when creating AS-IS process document, as well as driving gap analysis for scoping of migration projects
- In an ideal world, we would even be reviewing the requirements that drove creation of the existing system
  - a starting point for documenting current requirements
- Nuggets of information are often burried in existing documents that helps us ask questions as part of validating requirement completeness.

③ Focus Group:

- A focus group is a gathering of people who are representative of the users or customers of a product to get feedback.
- The feedback can be gathered about needs/ opportunities/ problems to identify requirements or can be gathered to validate and refine already elicited requirements.
- This form of market research is distinct from brainstorming in that it is a managed process with specific participants.

④ Interface Analysis:

- Interface for a software product can be human or machine.
- Integration with external systems and devices is just another interface.
- User centric design approaches are very effective at making sure that we create usable software.
- Interface analysis - reviewing the touch points with other external systems is important to make sure we don't overlook requirements that aren't immediately visible to users.

⑤ Interview:

- Interviews of stakeholders and users are critical to creating the great software.
- Without understanding the goals and expectations of the users and stakeholders, we are very unlikely to satisfy them.
- We also have to recognize the perspective of each interviewee, so that, we can properly weigh and address their inputs.
- Listening is the skill that helps a great analyst to get more value from an interview than an average analyst.

⑥ Observation:

- By observing users, an analyst can identify a process flow, steps, pain points and opportunities for improvement.
- Observations can be passive or active (asking questions while observing).
- Passive observation is better for getting feedback on a prototype (to refine requirements), where active observation is more effective at getting an understanding of an existing business process. Either approach can be used.

⑦ Prototyping:

- Prototyping is a relative modern technique for gathering requirements.
- In this approach, you gather preliminary requirements that you used to build an initial version of the solution - a prototype.
- You show this to a client, who then gives you additional requirements.
- You change the application and cycle around with the client again.
- This repetitive process continues until the product meets the critical mass of business needs or for an agreed number of iterations.

⑧ Requirement Workshops:

- Workshops can be very effective for gathering requirements.
- More structured than a brainstorming session, involved parties collaborate to document requirements.
- One way to capture the collaboration is with creation of domain-model artifacts (like static diagrams, activity diagrams).
- A workshop will be more effective with two analysts than with one.

(9) Reverse Engineering:

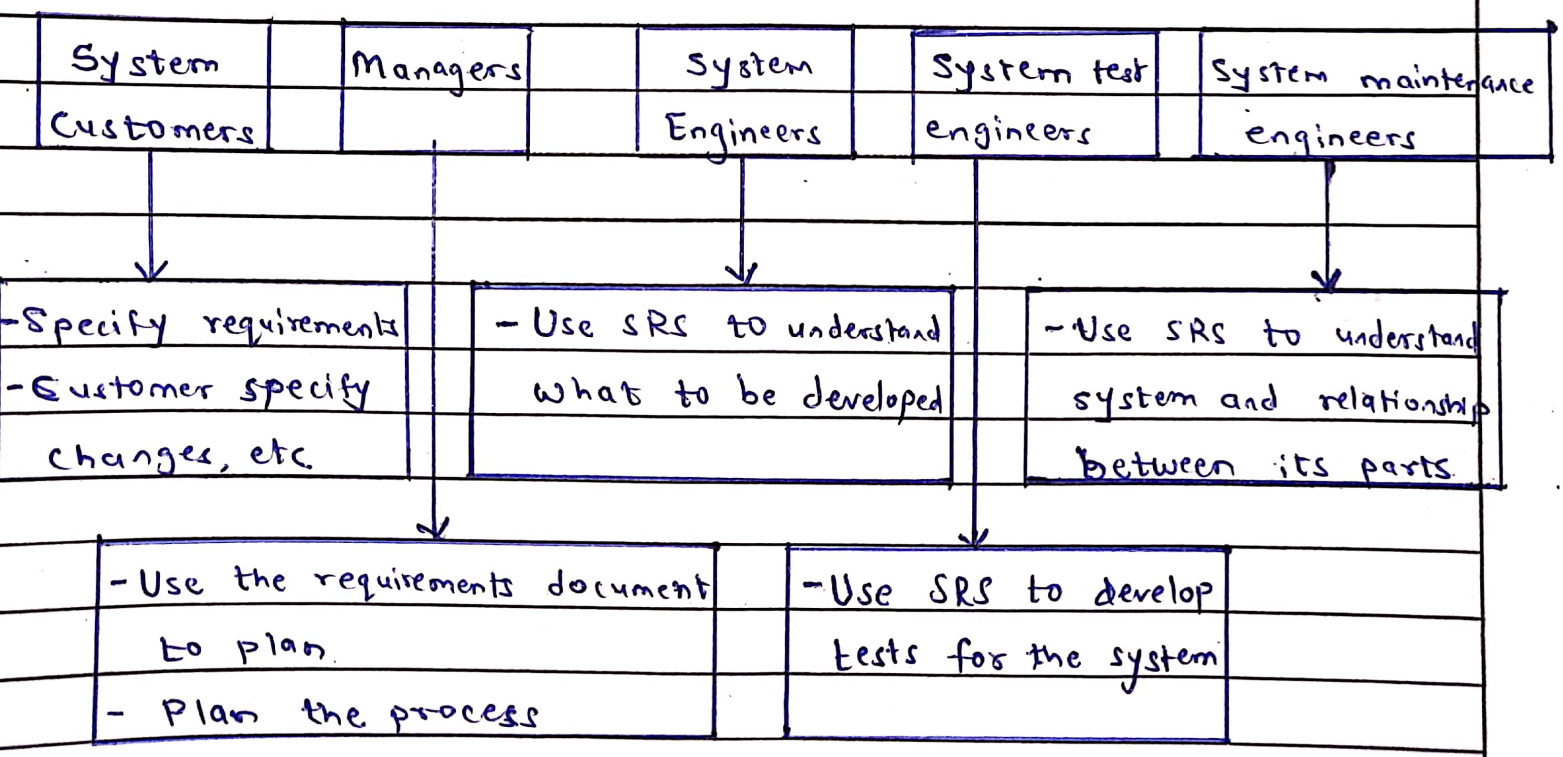
- When a migration project does not have access to sufficient documentation of the existing system, reverse engineering will identify what the system does.
- It will not identify what the system should do and will not identify when the system does the wrong thing.

(10) Survey / Questionnaires

- When collecting information from many people - too many to interview with budget and time contraints - a survey or questionnaire can be used.
- The survey can force users to select from choices, rate something ("Agree Strongly, agree...") or have open ended questions allowing free-form responses.
- Survey design is hard - questions can bias the respondents.

Software Requirement Specifications (SRS)

- A software requirement specifications (SRS) is a comprehesive description of the intended purpose and environment for software under development.
- The SRS fully describes what the software will do and how it will be expected to perform.
- A SRS is a document that describes the nature of a project, software or application.
- In simple words, SRS document is a manual of a project provided it is prepared before you kick-start a project / application.
- This document is also known by the names SRS report, software document.
- SRS includes both system requirement and user requirement.



Users of SRS

Characteristics of a good SRS document:

① Correctness:

- User review is used to ensure the correctness of requirements stated in the SRS.
- SRS is said to be correct if it covers all the requirements that are actually expected from the system.

② Completeness:

- Completeness of SRS indicates every sense of completion including the numbering of all the pages, resolving the to be determined parts to as much extent as possible as well as covering all the functional and non-functional requirements properly.

③ Consistency:

- Requirements in SRS are said to be consistent if there are no conflicts between any set of requirements
- Examples of conflict include differences in terminologies used at separate places, logical conflicts like time period of report generation, etc.

④ Unambiguousness:

- A SRS is said to be unambiguous if all the requirements stated have only 1 interpretation.
- Some of the ways to prevent unambiguousness include the use of modelling techniques like ER diagrams, proper reviews and buddy checks, etc.

⑤ Ranking for importance and stability :

- There should a criterion to classify the requirements as less or more important or more specifically as desirable or essential.
- An identifier mark can be used with every requirement to indicate its rank or stability.

⑥ Modifiability :

- SRS should be made as modifiable as possible and should be capable of easily accepting changes to the system to some extent.
- Modifications should be properly indexed and cross-referenced.

⑦ Verifiability :

- A SRS is verifiable if there exists a specific technique to quantifiably measure the extent to which every requirement is met. by the system.

⑧ Traceability :

- One should be able to trace a requirement to design component and then to code segment in the program. Similarly, one should be able to trace a requirement to the corresponding test cases.

(9) Design independence:

- There should be an option to choose from multiple design alternatives for the final system.
- More specifically, the SRS should not include any implementation details.

(10) Testability:

- A SRS should be written in such a way that it is easy to generate test cases and test plans for the document.

(11) Understandable: by the customer.:

- An end user maybe an expert in his/her specific domain but might not be an expert in computer science. Hence, the use of formal notations and symbols should be avoided to as much extent as possible. The language should be kept easy and clear.

(12) Right level of abstraction:

- If the SRS is written for the requirements phase, the details should be explained explicitely.
- Wheras for a feasibility study, fewer details can be used.
- Hence, the level of abstraction varies according to the purpose of the SRS.

Capability Maturity Model (CMM)

- Capability Maturity Model is a bench mark for measuring the maturity of an organization's software process.
- It is a methodology used to develop and refine an Organization's software development process.
- CMM can be used to assess an organization against a scale of five process maturity levels based on certain Key Process Areas (KPA).
- It describes the maturity of the company based upon the project the company is dealing with the clients
- Each level ranks the organization according to its standardization of processes in the subject area being assessed.

- A maturity model provides:

1. A place to start
2. The benefit of a community's prior experiences.
3. A common language and a shared vision.
4. A framework for prioritizing actions.
5. A way to define what improvement means for your organization.

- Maturity levels:

1. Initial
2. Managed
3. Defined
4. Quantitatively Managed
5. Optimizing.

- The software Engineering Institute (SEI) Capability Maturity Model (CMM) specifies an increasing series of levels of a software development organization.
- The higher the level, the better the software process, Hence reaching each level is an expensive and time-consuming process.

Levels of CMM :-

⑤ Optimization - Defect Prevention - Test Process Optimization - Quality Control	→ Optimizing	
④ Measured - Test measurement - Software Quality Evaluation - Advanced Peer Reviews	→ Managed	
③ Defined - Test Organization - Test lifecycle and integration - Non-functional testing.	→ Defined	
② Managed - Test Planning - Test Monitoring and control - Test Environment	→ Repeatable	
① Initial	→ Initial.	

Levels of CMM

① Level one : Initial

- The software process is characterized as inconsistent, and occasionally even chaotic.
- Defined processes and standard practices that exist are abondoned during a crisis.
- Success of the organization majorly depends on an individual effort, talent and heroics.
- The heros eventually moves on to other organizations taking their wealth of knowledge or lessons learnt with them

② Level two : Repeatable

- This level of Software Development Organization has a basic and consistent project management processes to track cost, schedule and functionality.
- The process is in place to repeat the ealier successes on projects with similar applications.
- Program management is a key characteristic of a level two organization.

③ Level three : Defined

- The software process for both management and engineering activities are documented, standardized and integrated into a standard software process for the entire organization and all projects across the organization use as approved, tailored version of the organization's standard software process for developing, testing and maintaining the application.

④ Level four : Managed

- Management can effectively control the software development effort using precise measurements.
- At this level, organization set a quantitative quality goal for both software process and software maintenance.
- At this maturity level, the performance of processes is controlled using statistical and other quantitative techniques and is quantitatively predictable.

⑤ Level Five : Optimizing

- The key characteristic of this level is focusing on continually improving process performance through both incremental and innovative technological improvements.
- At this level, changes to the process are to improve the process performance and at the same time maintaining statistical probability to achieve the established quantitative process-improvement objectives.

Graphical User Interface:

- User interface design creates an effective communication medium between a human and a computer.
- Software engineer designs the user interface by applying an interactive process.

Features of good user interface:

① Increased efficiency:

- If the system fits the way its users work and if it has a good ergonomic design, users can perform their task efficiently.
- They do not lose time struggling with the functionality and its appearance on the screen.

② Improved productivity:

- A good interface does not distract the user but rather allows him to concentrate on the task to be done.

③ Reduced errors:

- Many so-called 'human errors' can be attributed to poor user interface quality.
- Avoiding inconsistencies, ambiguities and so on reduces errors.

④ Reduced training:

- A poor user interface hampers learning. A well-designed user interface encourages its users to create proper models and reinforces learning, thus reducing training time.

⑤ Improved acceptance :

- User prefer systems whose interface is well-designed
- Such systems make information easy to find and provide the information in a form which is easy to use.

Formal Technical Review (FTR)

- Formal Technical Review is a software Quality Assurance activity performed by software engineer.

Objectives of FTR:

1. FTR is useful to uncover error in logic, function and implementation for any representation of the software.
2. The purpose of FTR is to ensure that software meets specified requirements.
3. It is also ensure that software is represented according to predefined standards.
4. It helps to review the uniformity in software development process
5. It makes the project more manageable

- The purpose of FTR is to enable junior engineer to observer the analysis, design, coding and testing approach more closely.
- Each FTR is conducted as meeting and is considered successful only if it is properly planned, controlled and attended.

Steps in FTR:

1. The review meeting
2. Review reporting and record keeping
3. Review guidelines.

FTR	Walkthrough
① A formal technical review is a software quality assurance activity performed by software engineers.	① It is led by the author. Author guides the participants through the document according to his thought process to achieve a common understanding and to gather feedback.
② In a review, a work product is examined for defects by individuals other than the person who produced it.	② In a walkthrough, the producer describes the product and asks for comments from the participants.
③ A work product is any important deliverable created during the requirements, design, coding or testing phase of software development.	③ Walkthroughs are usually used to examine source code as opposed to design and requirements documents. The participants do a step by step simulation of the code.
④ It is often performed as a peer review without management participation.	④ A walkthrough is especially useful for higher level documents such as requirement specification.
⑤ The FTR is actually a class of review that includes walkthroughs, inspections, round-robin reviews and other small group technical assessments of software.	⑤ Walkthrough is a type of Formal Technical Review.

Object point

- Application composition estimation model allows one to estimate the cost, effort at the stage one of the COCOMO II model.
- In this model size is estimated first using object points.
- Object points are easy to identify and count.
- Object points define screen, report, third generation (3GL) modules as object.
- Object point estimation is a new size estimation technique but it is well suited in application composition sector.

COCOMO - II Model

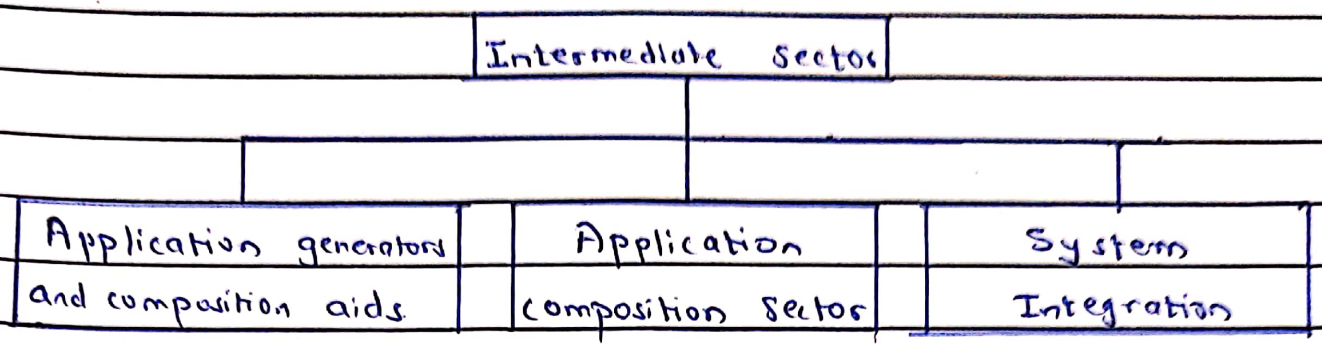
- COCOMO - II is the revised version of the original COCOMO (Constructive Cost Model) and is developed at University of Southern California.
- It is the model that allows one to estimate the cost, effort and schedule when planning a new software development activity.
- It consists of three sub-models.

	Application generators and composition aids	
End User Programming	Application Composition	Infrastructure
	System Integration	

① End User Programming:

- Application generators are used in this sub-model.
- End user write the code by using these application generators.
- Example: Spreadsheets, report generator, etc.

② Intermediate Sector:



① Application generators and composition aids:

- This category will create largely prepackaged capabilities for user programming.
- Their product will have many reusable components.
- Typical firms operating in this sector are Microsoft, Lotus, Oracle, IBM, Borland, Novell.

② Application Composition Sector:

- This category is too diversified and to be handled by prepackaged solutions.
- It includes GUI, Databases, domain specific components such as financial, medical or industrial process control packages.

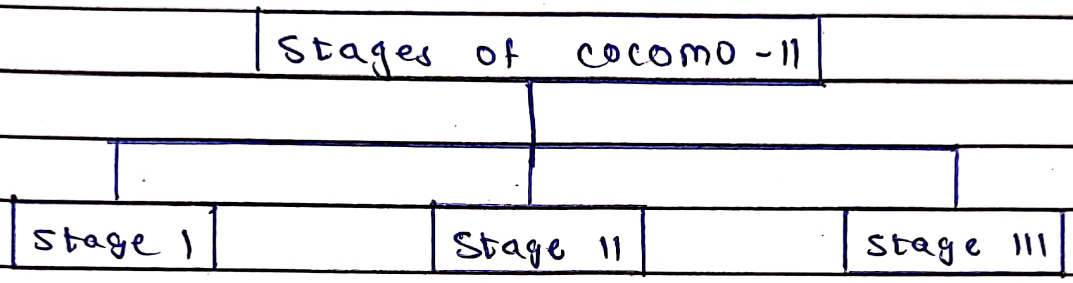
③ System Integration:

- This category deals with large scale and highly embedded systems.

③ Infrastructure Sector :

- This category provides infrastructure for the software development like Operating system, Database management system, User Interface management system, Networking system, etc.

Stages of COCOMO - II :



① Stage - I :

- It supports estimation of prototyping.
- For this it uses Application Composition Estimation model.
- It is used for prototyping stage of application generator and system integration.

② Stage - II :

- It supports estimation in the early design stage of the project, when we less know about it.
- For this it uses Early Stage Estimation model
- It is used in early design stage of application generators, infrastructure, system integration

③ Stage - III :

- It supports estimation in the post architecture stage of a project.
- For this it uses post Architecture Estimation model.
- It is used after completion of the detailed architecture of application generator, infrastructure.

STUDENT SIGNATURE: Amey

Cocomo Model (Constructive Cost Model)

- Cocomo is one of the most widely used software estimation models in the world.
- The model is developed in 1981 by Barry Boehm to give estimation of number of man-months it will take to develop a software product.
- Cocomo predicts the efforts and schedule of software product based on size of software.
- Cocomo has three different models that reflects complexity.

1. Basic model
2. Intermediate model
3. Detailed model

- Similarly, there are three classes of software projects.

1. Organic mode:
   - In this mode, relatively simple, small software projects with a small team are handled.
   - Such team should have good application experience to less rigid requirements

2. Semi-detached Projects:
   - In this class, intermediate projects in which team with mixed experience level are handled.
   - Such projects may have mix of rigid and less than rigid requirements.

③ Embedded Projects:

- In this class, project with tight hardware, software and operational constraints are handled.