Software Requirements

Software Requirement Specifications

- The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (aka **requirements document**)

- **SRS** is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements

- The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment.

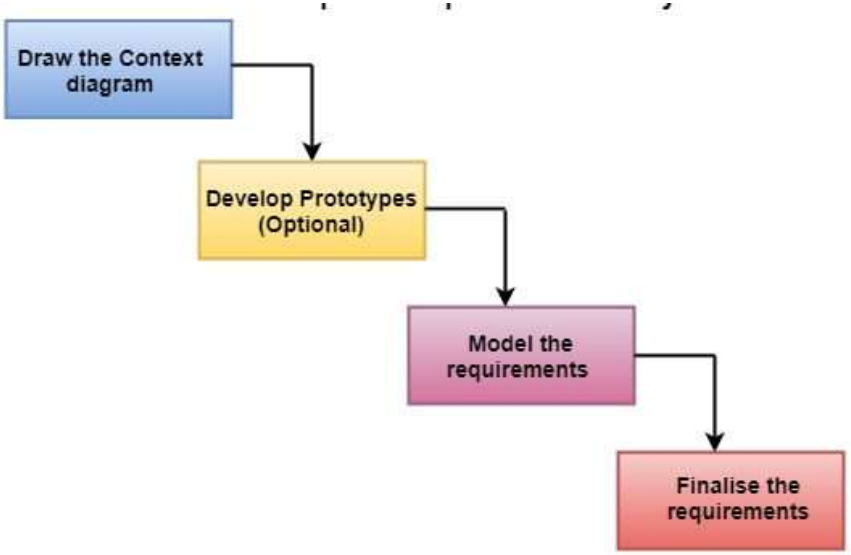
- It serves several goals depending on who is writing it. The client or developer of the system.

Software Requirement Specifications

- Characteristics/Features of good SRS

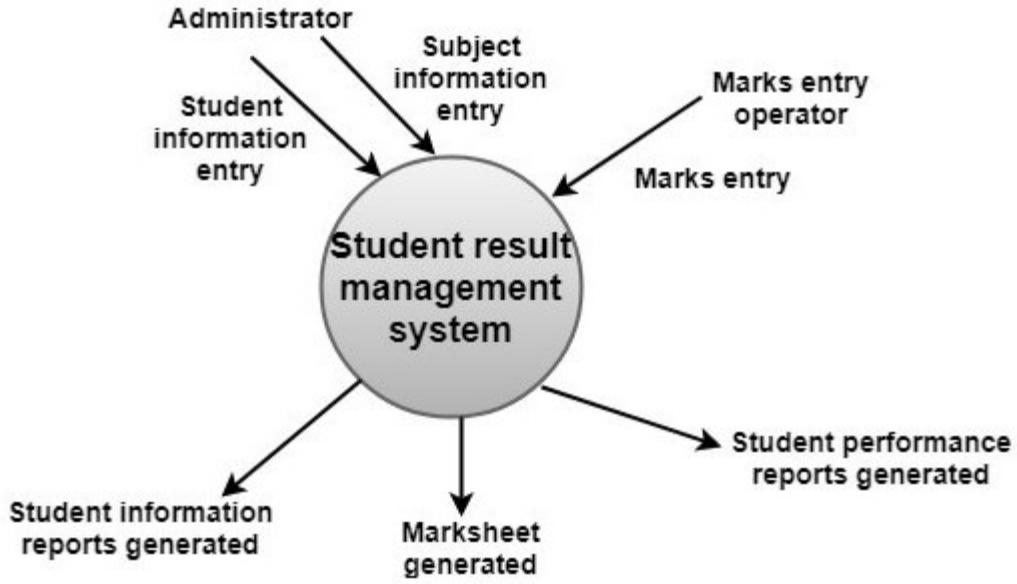
Requirements Analysis

- Requirement analysis is significant and essential activity after elicitation.
- We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements.



Requirements Analysis

Context diagram - The context diagram defines the boundaries and interfaces of the proposed systems with the external world. It identifies the entities outside the proposed system that interact with the system



Requirements Analysis

Prototype (optional): One effective way to find out what the customer wants is to construct a prototype, something that looks and preferably acts as part of the system they say they want.

Model the requirements: various graphical representations of the functions, data entities, external entities, and the relationships between them.

The graphical view may help to find incorrect, inconsistent, missing, and superfluous requirements. Data Flow diagram, Entity-Relationship diagram, Data Dictionaries, State-transition diagrams, etc.

Finalise the requirements: The inconsistencies and ambiguities have been identified and corrected. The flow of data amongst various modules has been analyzed. Elicitation and analyze activities have provided better insight into the system.

Data Flow Diagrams

A Data Flow Diagram (DFD) - visual representation of the information flows within a system. The objective of a DFD is to show the scope and boundaries of a system as a whole.

observations about DFDs

1. All names should be unique.
2. DFD is not a flow chart. Arrows in a flow chart represents the order of events; arrows in DFD represents flowing data. A DFD does not involve any order of events.
3. Suppress logical decisions. If we ever have the urge to draw a diamond-shaped box in a DFD, suppress that urge! A diamond-shaped box is used in flow charts to represents decision points with multiple exists paths of which the only one is taken. This implies an ordering of events, which makes no sense in a DFD.
4. Do not become bogged down with details. Defer error conditions and error handling until the end of the analysis.

Data Flow Diagrams

Levels in Data Flow Diagrams (DFD)

0-level DFD-fundamental system model, or context diagram represents the entire software requirement as a single bubble with input and output data denoted by incoming and outgoing arrows.

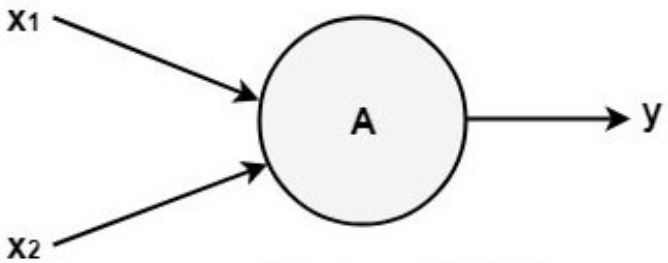


Fig: Level-0 DFD.

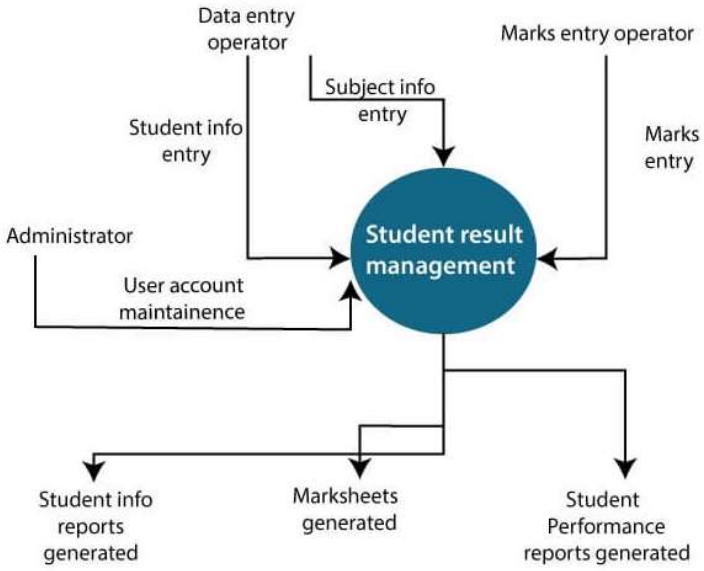


Fig: Level-0 DFD of result management system

Data Flow Diagrams

Levels in Data Flow Diagrams (DFD)

1-level DFD-In 1-level DFD, a context diagram is decomposed into multiple bubbles/processes. main objectives of the system is highlighted and break down of the high-level process of 0-level DFD into sub-processes.

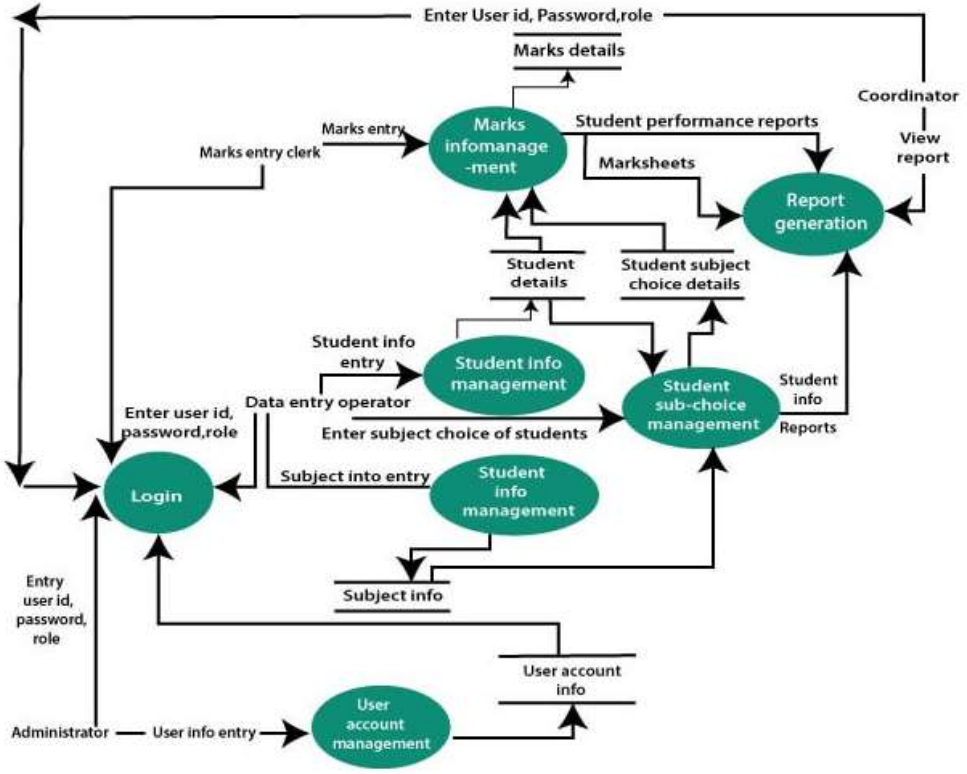
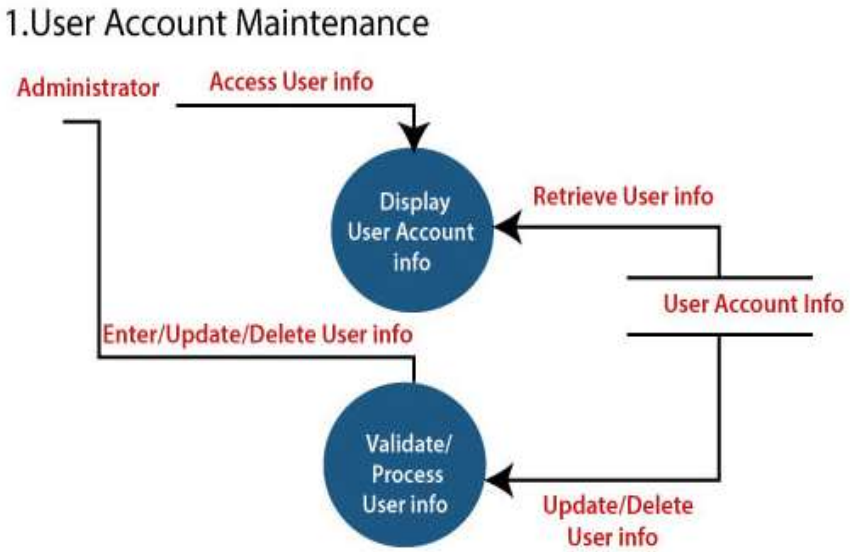


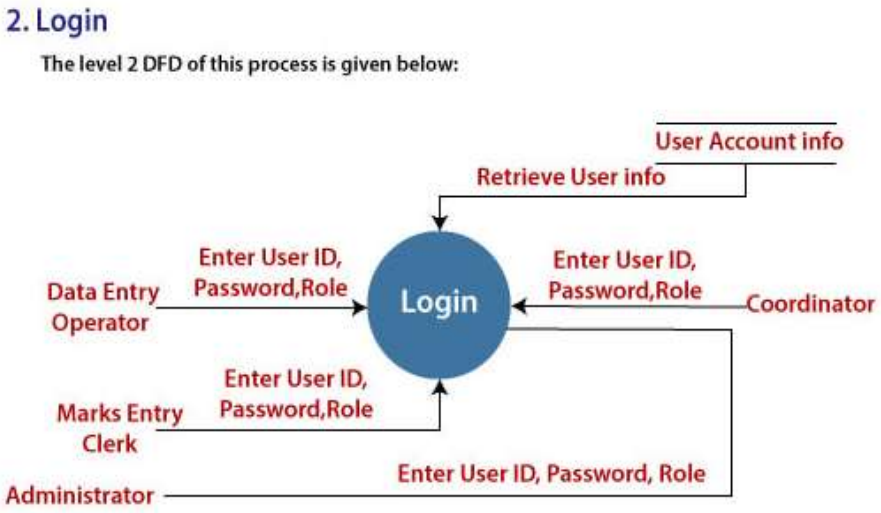
Fig: Level-1 DFD of result management system

Data Flow Diagrams

Levels in Data Flow Diagrams (DFD)

2-level DFD- used to project or record the specific/necessary detail about the system's functioning.

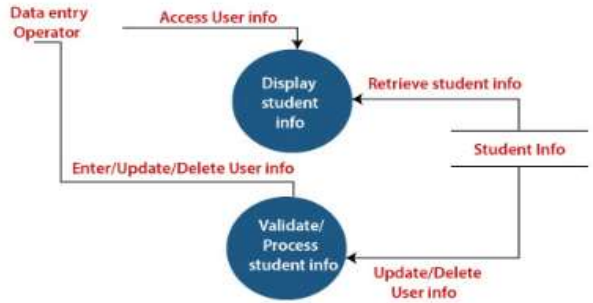




Data Flow Diagrams

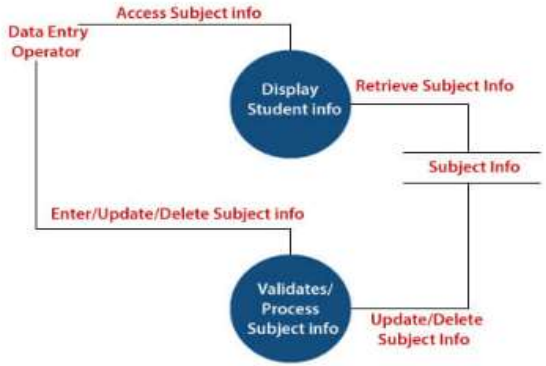
Levels in Data Flow Diagrams (DFD) -2-level DFD

3. Student Information Management



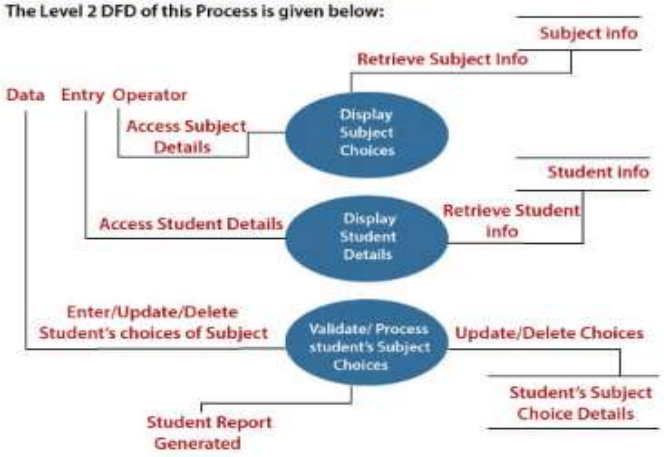
4. Subject Information Management

The level 2 DFD of this process is given below:



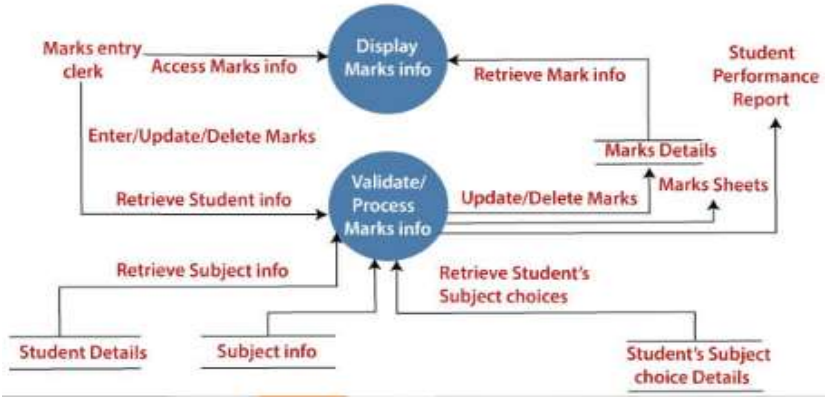
5. Student's Subject Choice Management

The Level 2 DFD of this Process is given below:



6. Marks Information Managment

The Level 2 DFD of this Process is given below:



Entity-Relationship Diagrams

ER-modeling is a data modeling method used in software engineering to produce a conceptual data model of an information system.

Purpose of ERD

- The database analyst gains a better understanding of the data to be contained in the database through the step of constructing the ERD.
- The ERD serves as a documentation tool.
- The ERD is used to connect the logical structure of the database to users. In particular, the ERD effectively communicates the logic of the database to users.

Entity-Relationship Diagrams

Components of an ER Diagrams

1. Entities - real-world object, either animate or inanimate, that can be merely identifiable

   entity set- collection of related types of entities.

2. Attributes – properties of an entity

   - Key attribute -Composite attribute -Derived attribute

   - Single-valued attribute -Multi-valued attribute

3. Relationships- association among entities

   Relationships set- relationships of a similar type

   Degree of the relationship - The number of participating entities in a relationship

   Unary (degree1)

   Binary (degree2)

   Ternary (degree3)

- Cardinality – one-to-one, one-to-many, many-to-one, many-to-many

Software Design

Software Design

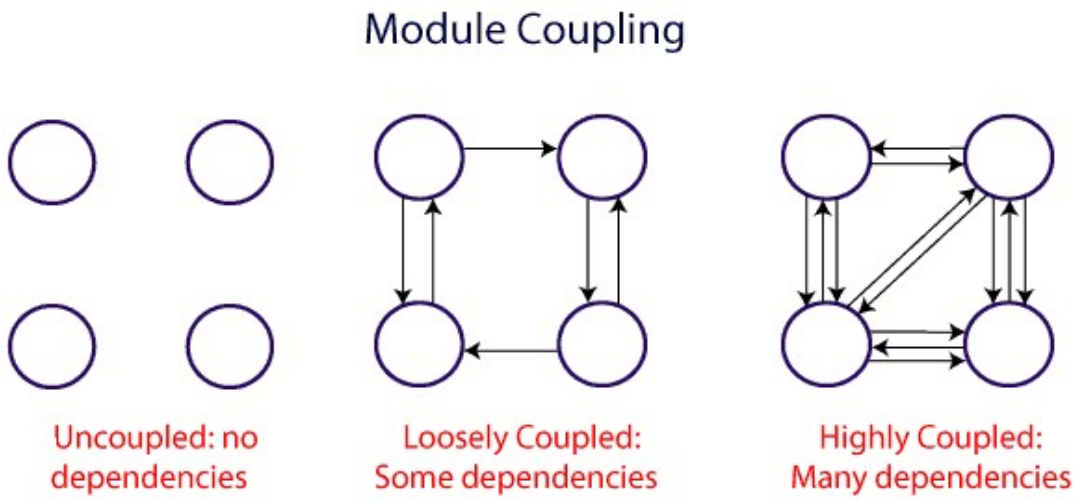
Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation.

Software design principles are concerned with providing means to handle the complexity of the design process effectively

- Problem Portioning
- Abstraction
- Modularity (read up- advantage and disadvantage)
- Top-Down and Bottom-Up Strategy

Coupling and Cohesion

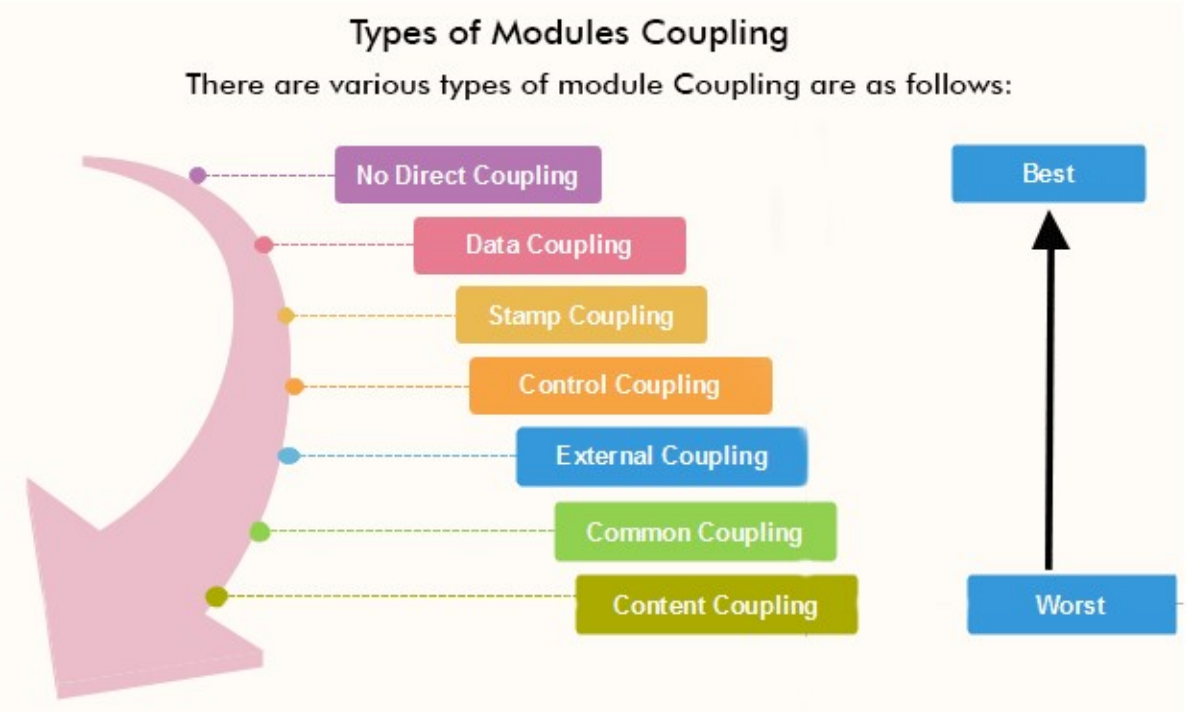
Coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other.



A good design is the one that has low coupling.

Coupling and Cohesion

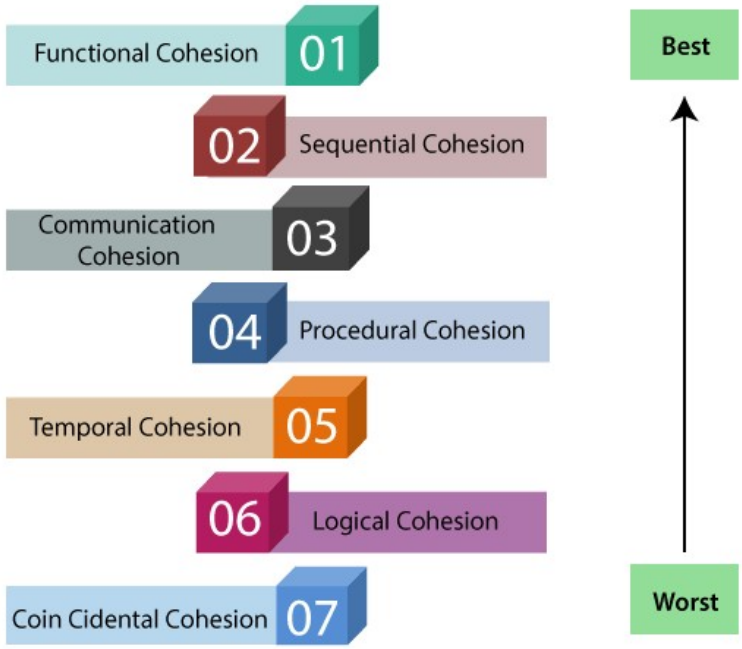
Coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other.



Coupling and Cohesion

cohesion defines the degree to which the elements of a module belong together.

cohesion measures the strength of relationships between pieces of functionality within a given module.



Coupling and Cohesion

Difference between coupling and cohesion

Coupling	Cohesion
Coupling is also called Inter-Module Binding.	Cohesion is also called Intra-Module Binding.
Coupling shows the relationships between modules.	Cohesion shows the relationship within the module.
Coupling shows the relative **independence** between the modules.	Cohesion shows the module's relative **functional** strength.
While creating, you should aim for low coupling, i.e., dependency among modules should be less.	While creating you should aim for high cohesion, i.e., a cohesive component/ module focuses on a single function (i.e., single-mindedness) with little interaction with other modules of the system.
In coupling, modules are linked to the other modules.	In cohesion, the module focuses on a single thing.

Design

Functional Design

Object Oriented Design

User Interface Design

Coding

Coding

The coding is the process of transforming the design of a system into a computer language format.

This coding phase of software development is concerned with translating design specification into the source code.

Coding Standards and Guidelines

- Proper and consistent indentation
- Inline comments
- Rules for limiting the use of global
- Naming conventions
- Error return conventions and exception handling system
- Spacing
- Length of any function should not be too long

Programming Style

Programming style refers to the technique used in writing the source code for a computer program

The goal of good programming style is to provide understandable, straightforward, elegant code.

The programming style used in a various program may be derived from the coding standards or code conventions of a company or other computing organization, as well as the preferences of the actual programmer.

Programming Style

Some general rules or guidelines in respect of programming style:

- Clarity and simplicity of Expression.
- Naming style should not be cryptic and non-representative.
- Single entry and single exit constructs are desirable
- Information secure in the data structures should be hidden from the rest of the system where possible
- Deep nesting of loops and conditions should be avoided as it greatly harm the static and dynamic behavior of a program
- Make heavy use of user-defined data types like enum, class, structure, and union. They make code easy to write and understand
- The module size should be uniform. If the module size is too large, it is not generally functionally cohesive. If the module size is too small, it leads to unnecessary overheads.

Software Maintenance

Software Maintenance

The goal is to modify and update software application after delivery to correct errors and to improve performance.

Software is a model of the real world. When the real world changes, the software require alteration wherever possible.

Need for Maintenance

- Correct errors
- Change in user requirement with time
- Changing hardware/software requirements
- To improve system efficiency
- To optimize the code to run faster
- To modify the components
- To reduce any unwanted side effects.

Types of Software Maintenance

1. Corrective Maintenance: aims to correct any remaining errors regardless of where they may cause specifications, design, coding, testing, and documentation, etc.

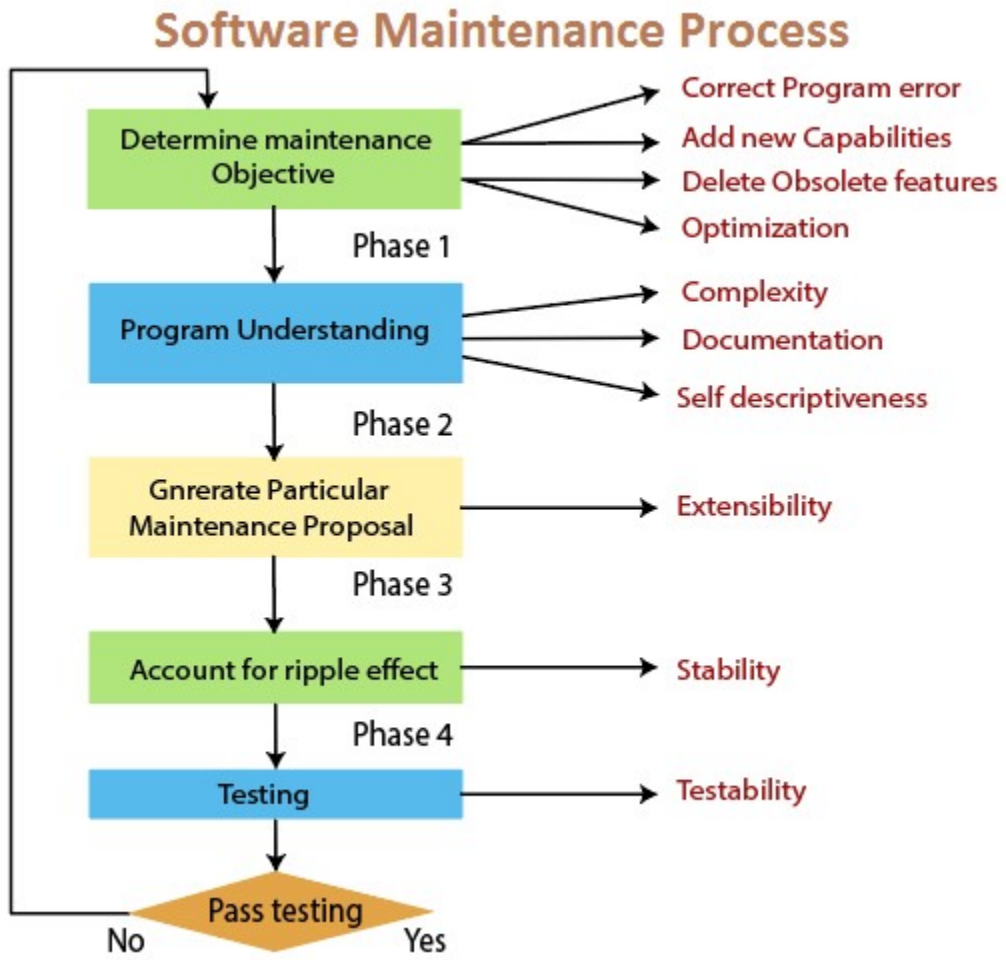
2. Adaptive Maintenance: contains modifying the software to match changes in the ever-changing environment.

3. Preventive Maintenance: process by which we prevent our system from being obsolete. It involves reengineering & reverse engineering in which an old system with old technology is re-engineered using new technology.

4. Perfective Maintenance: It defines improving processing efficiency or performance or restricting the software to enhance changeability.

Causes of Software Maintenance Problems

- Lack of Traceability
- Lack of Code Comments
- Obsolete Legacy Systems



Software Quality

- Is a set of activities for ensuring quality in software engineering process.
- It ensures that developed software meets and complies with defined specifications.
- It accounts for a large amount of development time.
- It guarantees a level of quality for the end client.
- It helps development team identify problems early by

testing

Software Quality

Software quality product is defined in term of its fitness of purpose. A quality product does precisely what the users want it to do.

Example: Consider a functionally correct software product. That is, it performs all tasks as specified in the SRS document. But, has an almost unusable user interface. Even though it may be functionally right, we cannot consider it to be a quality product.

Modern view of a quality associated with a software product

- Portability
- Usability
- Reusability
- Correctness
- Maintainability

Software Testing

Software Testing

- The intent of software testing is to uncover software bugs/errors/defects.
- The goal of software testing is to demonstrate to the developer and the customer that the software meets its requirements.
- Software testing can only show the presence of errors, not their absence.
- Software testing is part of a broader process of software verification and validation.

Software Testing Process

- Test strategy and test plan: planning to execute tests; selecting test approach; processes to follow; selecting tools and techniques; documenting plan, setting up test environment; identifying risks and dependencies; scheduling test.
- Test design: designing test suite(collection of test cases) in line with the requirements specified.
- Test execution: application of tests at various stages of development.
- Test closure: 100% requirements coverage; a large amount of test pass; all critical defects discovered to be fixed.

Stages of Software Testing

- **Development Testing:** the system is tested during development to discover bugs and defects. Involves system designers and programmers .
- **Release Testing:** The aim of release testing is to check that the system meets the requirements of system stakeholders. Involves a separate testing team. Tests the complete version.
- **User Testing:** where users or potential users of a system test the system in their own environment.

Methods of Software Testing

- **Manual testing:** a tester runs the program with some test data and compares the results to their expectations.
- **Automated testing:** the tests are encoded in a program that is run each time the system under development is to be tested.
- Automated testing executes faster than manual testing, especially when it involves re-running tests.
- In practice, the testing process usually involves a mixture of manual and automated testing.