**Software Engineering Syllabus**

Unit-1

# Basic Principles of Good Software Engineering approach

[Software Engineering](https://www.geeksforgeeks.org/software-engineering-introduction-to-software-engineering/) is the systems engineering approach for software product/application development. It is an engineering branch associated with analyzing user requirements, design, development, testing, and maintenance of software products.

Some basic **principles** of good software engineering are –

1. One of the basic software Engineering principles is Better Requirement analysis which gives a clear vision of the project. At last, a good understanding of user requirements provides value to its users by delivering a good software product that meets users’ requirements.
2. All designs and implementations should be as simple as possible mean the KISS (Keep it Simple, Stupid) principle should be followed. It makes code so simple as a result debugging and further maintenance become simple.
3. Maintaining the vision of the project is the most important thing throughout complete development process for the success of a software project. A clear vision of the project leads to the development of the project in the right way.
4. Software projects include a number of functionalities, all functionalities should be developed in a modular approach so that development will be faster and easier. This modularity makes functions or system components independent.
5. Another specialization of the principle of separation of concerns is Abstraction for suppressing complex things and delivering simplicity to the customer/user means it gives what the actual user needs and hides unnecessary things.
6. Think then Act is a must-required principle for software engineering means before starting developing functionality first it requires to think about application architecture, as good planning on the flow of project development produces better results.
7. Sometimes developer adds up all functionalities together but later find no use of that. So, following the Never add extra principle is important as it implements what actually needed and later implements what are required which saves effort and time.
8. When other developers work with another’s code they should not be surprised and should not waste their time in getting code. So, providing better Documentation at required steps is a good way of developing software projects.
9. Law of Demeter should be followed as it makes classes independent on their functionalities and reduces connections and inter dependability between classes which is called coupling.
10. The developers should develop the project in such a way that it should satisfy the principle of Generality means it should not be limited or restricted to some of cases/functions rather it should be free from unnatural restrictions and should be able to provide service to customers what actually they need or general needs in an extensive manner.
11. Principle of Consistency is important in coding style and designing GUI (Graphical User Interface) as consistent coding style gives an easier reading of code and consistency in GUI makes user learning easier in dealing with interface and in using the software.
12. Never waste time if anything is required and that already exists at that time take the help of Open source and fix it in your own way as per requirement.
13. Performing continuous validation helps in checking software system meets requirement specifications and fulfils its intended purpose which helps in better software quality control.
14. To exit in current technology market trends Using modern programming practices is important to meet users’ requirements in the latest and advanced way.
15. Scalability in Software Engineering should be maintained to grow and manage increased demand for software applications.

### There are several basic principles of good software engineering approach that are commonly followed by software developers and engineers to produce high-quality software. Some of these principles include:

1. **Modularity**: Breaking down the software into smaller, independent, and reusable components or modules. This makes the software easier to understand, test, and maintain.
2. **Abstraction**: Hiding the implementation details of a module or component and exposing only the necessary information. This makes the software more flexible and easier to change.
3. **Encapsulation**: Wrapping the data and functions of a module or component into a single unit, and providing controlled access to that unit. This helps to protect the data and functions from unauthorized access and modification.
4. **DRY principle (Don’t Repeat Yourself):** Avoiding duplication of code and data in the software. This makes the software more maintainable and less error-prone.
5. **KISS principle (Keep It Simple, Stupid)**: Keeping the software design and implementation as simple as possible. This makes the software more understandable, testable, and maintainable.
6. **YAGNI (You Ain’t Gonna Need It):** Avoiding adding unnecessary features or functionality to the software. This helps to keep the software focused on the essential requirements and makes it more maintainable.
7. **SOLID principles:**A set of principles that guide the design of software to make it more maintainable, reusable, and extensible. This includes the Single Responsibility Principle, Open/Closed Principle, Liskov Substitution Principle, Interface Segregation Principle, and Dependency Inversion Principle.
8. **Test-driven development:** Writing automated tests before writing the code, and ensuring that the code passes all tests before it is considered complete. This helps to ensure that the software meets the requirements and specifications.
9. By following these principles, software engineers can develop software that is more reliable, maintainable, and extensible.
10. It’s also important to note that these principles are not mutually exclusive, and often work together to improve the overall quality of the software.

**Software** is the set of instructions in the form of programs to govern the computer system and to process the hardware components. To produce a software product the set of activities is used. This set is called a software process. 

**Software Development:** In this process, designing, programming, documenting, testing, and bug fixing is done.

**Components of Software:**   
There are three components of the software: These are: Program, Documentation, and Operating Procedures.

1. **Program –**A computer program is a list of instructions that tell a computer what to do.
2. **Documentation –** Source information about the product contained in design documents, detailed code comments, etc.
3. **Operating Procedures –**Set of step-by-step instructions compiled by an organization to help workers carry out complex routine operations.
4. **Code**: the instructions that a computer executes in order to perform a specific task or set of tasks.
5. **Data**: the information that the software uses or manipulates.
6. **User**interface: the means by which the user interacts with the software, such as buttons, menus, and text fields.
7. **Libraries**: pre-written code that can be reused by the software to perform common tasks.
8. **Documentation**: information that explains how to use and maintain the software, such as user manuals and technical guides.
9. **Test cases**: a set of inputs, execution conditions, and expected outputs that are used to test the software for correctness and reliability.
10. **Configuration** **files**: files that contain settings and parameters that are used to configure the software to run in a specific environment.
11. **Build and deployment** scripts: scripts or tools that are used to build, package, and deploy the software to different environments.
12. **Metadata**: information about the software, such as version numbers, authors, and copyright information.

All these components are important for software development, testing and deployment.

There are four basic key process activities:

1. **Software Specifications –**   
   In this process, detailed description of a software system to be developed with its functional and non-functional requirements.
2. **Software Development –**  
   In this process, designing, programming, documenting, testing, and bug fixing is done.
3. **Software Validation –**  
   In this process, evaluation software product is done to ensure that the software meets the business requirements as well as the end user needs.
4. **Software Evolution –**   
   It is a process of developing software initially, then timely updating it for various reasons.

**Software Crisis:**

1. **Size and Cost –**  
   Day to day growing complexity and expectation out of software. Software are more expensive and more complex.
2. **Quality –**  
   Software products must have good quality.
3. **Delayed Delivery –**  
   Software takes longer than the estimated time to develop, which in turn leads to cost shooting up.
4. The term “software crisis” refers to a set of problems that were faced by the software industry in the 1960s and 1970s, such as:
5. High costs and long development times: software projects were taking much longer and costing much more than expected.
6. **Low quality**: software was often delivered late, with bugs and other defects that made it difficult to use.
7. **Lack of standardization**: there were no established best practices or standards for software development, making it difficult to compare and improve different approaches.
8. **Lack of tools and methodologies**: there were few tools and methodologies available to help with software development, making it a difficult and time-consuming process.
9. These problems led to a growing realization that the traditional approaches to software development were not effective and needed to be improved. This led to the development of new software development methodologies, such as the Waterfall and Agile methodologies, as well as the creation of new tools and technologies to support software development.

However, even today, software crisis could be seen in some form or the other, like for example software projects going over budget, schedule and not meeting the requirement.

Why is Software Engineering required?

Software Engineering is required due to the following reasons:

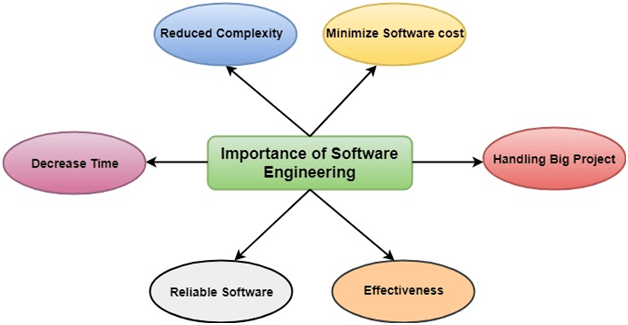
* To manage Large software
* For more Scalability
* Cost Management
* To manage the dynamic nature of software
* For better quality Management

Need of Software Engineering

The necessity of software engineering appears because of a higher rate of progress in user requirements and the environment on which the program is working.

* **Huge Programming:**It is simpler to manufacture a wall than to a house or building, similarly, as the measure of programming become extensive engineering has to step to give it a scientific process.
* **Adaptability:**If the software procedure were not based on scientific and engineering ideas, it would be simpler to re-create new software than to scale an existing one.
* **Cost:**As the hardware industry has demonstrated its skills and huge manufacturing has let down the cost of computer and electronic hardware. But the cost of programming remains high if the proper process is not adapted.
* **Dynamic Nature:**The continually growing and adapting nature of programming hugely depends upon the environment in which the client works. If the quality of the software is continually changing, new upgrades need to be done in the existing one.
* **Quality Management:** Better procedure of software development provides a better and quality software product.

Importance of Software Engineering



**The importance of Software engineering is as follows:**

1. **Reduces complexity:** Big software is always complicated and challenging to progress. Software engineering has a great solution to reduce the complication of any project. Software engineering divides big problems into various small issues. And then start solving each small issue one by one. All these small problems are solved independently to each other.
2. **To minimize software cost:** Software needs a lot of hard work and software engineers are highly paid experts. A lot of manpower is required to develop software with a large number of codes. But in software engineering, programmers project everything and decrease all those things that are not needed. In turn, the cost for software productions becomes less as compared to any software that does not use software engineering method.
3. **To decrease time:** Anything that is not made according to the project always wastes time. And if you are making great software, then you may need to run many codes to get the definitive running code. This is a very time-consuming procedure, and if it is not well handled, then this can take a lot of time. So, if you are making your software according to the software engineering method, then it will decrease a lot of time.
4. **Handling big projects:** Big projects are not done in a couple of days, and they need lots of patience, planning, and management. And to invest six and seven months of any company, it requires heaps of planning, direction, testing, and maintenance. No one can say that he has given four months of a company to the task, and the project is still in its first stage. Because the company has provided many resources to the plan and it should be completed. So, to handle a big project without any problem, the company has to go for a software engineering method.
5. **Reliable software:** Software should be secure, means if you have delivered the software, then it should work for at least it’s given time or subscription. And if any bugs come in the software, the company is responsible for solving all these bugs. Because in software engineering, testing and maintenance are given, so there is no worry of its reliability.
6. **Effectiveness:** Effectiveness comes if anything has made according to the standards. Software standards are the big target of companies to make it more effective. So, Software becomes more effective in the act with the help of software engineering.

The Software Process Model

A software process model is a specified definition of a software process, which is presented from a particular perspective. Models, by their nature, are a simplification, so a software process model is an abstraction of the actual process, which is being described. Process models may contain activities, which are part of the software process, software product, and the roles of people involved in software engineering. Some examples of the types of software process models that may be produced are:

1. **A workflow model:** This shows the series of activities in the process along with their inputs, outputs and dependencies. The activities in this model perform human actions.
2. **2. A dataflow or activity model:** This represents the process as a set of activities, each of which carries out some data transformations. It shows how the input to the process, such as a specification is converted to an output such as a design. The activities here may be at a lower level than activities in a workflow model. They may perform transformations carried out by people or by computers.
3. **3. A role/action model:** This means the roles of the people involved in the software process and the activities for which they are responsible.

# **Software Development Life Cycle (SDLC)**

A software life cycle model (also termed process model) is a pictorial and diagrammatic representation of the software life cycle. A life cycle model represents all the methods required to make a software product transit through its life cycle stages. It also captures the structure in which these methods are to be undertaken.

In other words, a life cycle model maps the various activities performed on a software product from its inception to retirement. Different life cycle models may plan the necessary development activities to phases in different ways. Thus, no element which life cycle model is followed, the essential activities are contained in all life cycle models though the action may be carried out in distinct orders in different life cycle models. During any life cycle stage, more than one activity may also be carried out.

## Need of SDLC

The development team must determine a suitable life cycle model for a particular plan and then observe to it.

Without using an exact life cycle model, the development of a software product would not be in a systematic and disciplined manner. When a team is developing a software product, there must be a clear understanding among team representative about when and what to do. Otherwise, it would point to chaos and project failure. This problem can be defined by using an example. Suppose a software development issue is divided into various parts and the parts are assigned to the team members. From then on, suppose the team representative is allowed the freedom to develop the roles assigned to them in whatever way they like. It is possible that one representative might start writing the code for his part, another might choose to prepare the test documents first, and some other engineer might begin with the design phase of the roles assigned to him. This would be one of the perfect methods for project failure.

SDLC Cycle represents the process of developing software. SDLC framework includes the following steps:



## The stages of SDLC are as follows:

**Stage1: Planning and requirement analysis**

Requirement Analysis is the most important and necessary stage in SDLC.

The senior members of the team perform it with inputs from all the stakeholders and domain experts or SMEs in the industry.

Planning for the quality assurance requirements and identifications of the risks associated with the projects is also done at this stage.

Business analyst and Project organizer set up a meeting with the client to gather all the data like what the customer wants to build, who will be the end user, what is the objective of the product. Before creating a product, a core understanding or knowledge of the product is very necessary.

**For Example**, A client wants to have an application which concerns money transactions. In this method, the requirement has to be precise like what kind of operations will be done, how it will be done, in which currency it will be done, etc.

Once the required function is done, an analysis is complete with auditing the feasibility of the growth of a product. In case of any ambiguity, a signal is set up for further discussion.

Once the requirement is understood, the SRS (Software Requirement Specification) document is created. The developers should thoroughly follow this document and also should be reviewed by the customer for future reference.

**Stage2: Defining Requirements**

Once the requirement analysis is done, the next stage is to certainly represent and document the software requirements and get them accepted from the project stakeholders.

This is accomplished through "SRS"- Software Requirement Specification document which contains all the product requirements to be constructed and developed during the project life cycle.

**Stage3: Designing the Software**

The next phase is about to bring down all the knowledge of requirements, analysis, and design of the software project. This phase is the product of the last two, like inputs from the customer and requirement gathering.

**Stage4: Developing the project**

In this phase of SDLC, the actual development begins, and the programming is built. The implementation of design begins concerning writing code. Developers have to follow the coding guidelines described by their management and programming tools like compilers, interpreters, debuggers, etc. are used to develop and implement the code.

**Stage5: Testing**

After the code is generated, it is tested against the requirements to make sure that the products are solving the needs addressed and gathered during the requirements stage.

During this stage, unit testing, integration testing, system testing, acceptance testing is done.

**Stage6: Deployment**

Once the software is certified, and no bugs or errors are stated, then it is deployed.

Then based on the assessment, the software may be released as it is or with suggested enhancement in the object segment.

After the software is deployed, then its maintenance begins.

**Stage7: Maintenance**

Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

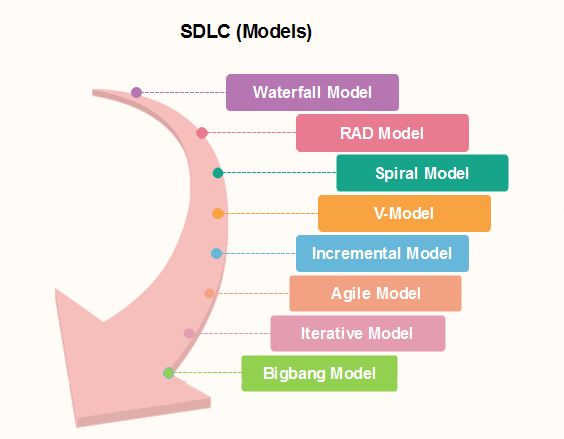
This procedure where the care is taken for the developed product is known as maintenance.

# **SDLC Models**

Software Development life cycle (SDLC) is a spiritual model used in project management that defines the stages include in an information system development project, from an initial feasibility study to the maintenance of the completed application.

There are different software development life cycle models specify and design, which are followed during the software development phase. These models are also called "**Software Development Process Models**." Each process model follows a series of phase unique to its type to ensure success in the step of software development.

**Here, are some important phases of SDLC life cycle:**



### [**Waterfall Model**](https://www.javatpoint.com/software-engineering-waterfall-model)

The waterfall is a universally accepted SDLC model. In this method, the whole process of software development is divided into various phases.

The waterfall model is a continuous software development model in which development is seen as flowing steadily downwards (like a waterfall) through the steps of requirements analysis, design, implementation, testing (validation), integration, and maintenance.

Linear ordering of activities has some significant consequences. First, to identify the end of a phase and the beginning of the next, some certification techniques have to be employed at the end of each step. Some verification and validation usually do this mean that will ensure that the output of the stage is consistent with its input (which is the output of the previous step), and that the output of the stage is consistent with the overall requirements of the system.

### [**RAD Model**](https://www.javatpoint.com/software-engineering-rapid-application-development-model)

RAD or Rapid Application Development process is an adoption of the waterfall model; it targets developing software in a short period. The RAD model is based on the concept that a better system can be developed in lesser time by using focus groups to gather system requirements.

* Business Modeling
* Data Modeling
* Process Modeling
* Application Generation
* Testing and Turnover

### [**Spiral Model**](https://www.javatpoint.com/software-engineering-spiral-model)

The spiral model is a **risk-driven process model**. This SDLC model helps the group to adopt elements of one or more process models like a waterfall, incremental, waterfall, etc. The spiral technique is a combination of rapid prototyping and concurrency in design and development activities.

Each cycle in the spiral begins with the identification of objectives for that cycle, the different alternatives that are possible for achieving the goals, and the constraints that exist. This is the first quadrant of the cycle (upper-left quadrant).

The next step in the cycle is to evaluate these different alternatives based on the objectives and constraints. The focus of evaluation in this step is based on the risk perception for the project.

The next step is to develop strategies that solve uncertainties and risks. This step may involve activities such as benchmarking, simulation, and prototyping.

### [**V-Model**](https://www.javatpoint.com/software-engineering-v-model)

In this type of SDLC model testing and the development, the step is planned in parallel. So, there are verification phases on the side and the validation phase on the other side. V-Model joins by Coding phase.

### [**Incremental Model**](https://www.javatpoint.com/software-engineering-incremental-model)

The incremental model is not a separate model. It is necessarily a series of waterfall cycles. The requirements are divided into groups at the start of the project. For each group, the SDLC model is followed to develop software. The SDLC process is repeated, with each release adding more functionality until all requirements are met. In this method, each cycle act as the maintenance phase for the previous software release. Modification to the incremental model allows development cycles to overlap. After that subsequent cycle may begin before the previous cycle is complete.

### [**Agile Model**](https://www.javatpoint.com/software-engineering-agile-model)

Agile methodology is a practice which promotes continues interaction of development and testing during the SDLC process of any project. In the Agile method, the entire project is divided into small incremental builds. All of these builds are provided in iterations, and each iteration lasts from one to three weeks.

Any agile software phase is characterized in a manner that addresses several key assumptions about the bulk of software projects:

1. It is difficult to think in advance which software requirements will persist and which will change. It is equally difficult to predict how user priorities will change as the project proceeds.
2. For many types of software, design and development are interleaved. That is, both activities should be performed in tandem so that design models are proven as they are created. It is difficult to think about how much design is necessary before construction is used to test the configuration.
3. Analysis, design, development, and testing are not as predictable (from a planning point of view) as we might like.

### [**Iterative Model**](https://www.javatpoint.com/software-engineering-iterative-model)

It is a particular implementation of a software development life cycle that focuses on an initial, simplified implementation, which then progressively gains more complexity and a broader feature set until the final system is complete. In short, iterative development is a way of breaking down the software development of a large application into smaller pieces.

### [Big bang model](https://www.javatpoint.com/software-engineering-big-bang-model)

Big bang model is focusing on all types of resources in software development and coding, with no or very little planning. The requirements are understood and implemented when they come.

This model works best for small projects with smaller size development team which are working together. It is also useful for academic software development projects. It is an ideal model where requirements are either unknown or final release date is not given.

### [**Prototype Model**](https://www.javatpoint.com/software-engineering-prototype-model)

The prototyping model starts with the requirements gathering. The developer and the user meet and define the purpose of the software, identify the needs, etc.

A '**quick design**' is then created. This design focuses on those aspects of the software that will be visible to the user. It then leads to the development of a prototype. The customer then checks the prototype, and any modifications or changes that are needed are made to the prototype.

Looping takes place in this step, and better versions of the prototype are created. These are continuously shown to the user so that any new changes can be updated in the prototype. This process continue until the customer is satisfied with the system. Once a user is satisfied, the prototype is converted to the actual system with all considerations for quality and security.

# What is a unified process model?

The **Unified Process (UP)** is a software development framework used for object-oriented modeling. The framework is also known as Rational Unified Process (RUP) and the Open Unified Process (Open UP). Some of the key features of this process include:

* It defines the order of phases.
* It is component-based, meaning a software system is built as a set of software components. There must be well-defined interfaces between the components for smooth communication.
* It follows an iterative, incremental, architecture-centric, and use-case driven approach

### The case-driven approach

Use a case-driven approach that follows a set of actions performed by one or more entities. A use case refers to the process of the team performing the development work from the functional requirements. The functional requirements are made from the list of requirements that were specified by the client. For example, an online learning management system can be specified in terms of use cases such as "add a course," "delete a course," "pay fees," and so on.

### The architecture-centric approach

The architecture-centric approach defines the form of the system and how it should be structured to provide a specific functionality whereas the use case defines the functionality.

### The iterative and incremental approach

An iterative and incremental approach means that the product will be developed in multiple phases. During these phases, the developers evaluate and test.

### Phases

We can represent a unified process model as a series of cycles. Each cycle ends with the release of a new system version for the customers. We have four phases in every cycle:

* Inception
* Elaboration
* Construction
* Transition

The phases of the unified process

### Inception

The main goal of this phase involves delimiting the project scope. This is where we define why we are making this product in the first place. It should have the following:

* What are the key features?
* How does this benefit the customers?
* Which methodology will we follow?
* What are the risks involved in executing the project?
* Schedule and cost estimates.

### Elaboration

We build the system given the requirements, cost, and time constraints and all the risks involved. It should include the following:

* Develop with the majority of the functional requirements implemented.
* Finalize the methodology to be used.
* Deal with the significant risks involved.

### Construction

This phase is where the development, integration, and testing take place. We build the complete architecture in this phase and hand the final documentation to the client.

### Transition

This phase involves the deployment, multiple iterations, beta releases, and improvements of the software. The users will test the software, which may raise potential issues. The development team will then fix those errors.

### Conclusion

This method allows us to deal with the changing requirements throughout the development period. The unified process model has various applications which also makes it complex in nature. Therefore, it's most suitable for smaller projects and should be implemented by a team of professionals.

**Phases of RUP:** There is total of five phases of the life cycle of RUP:

1. **Inception –**
   * Communication and planning are the main ones.
   * Identifies the scope of the project using a use-case model allowing managers to estimate costs and time required.
   * Customers’ requirements are identified and then it becomes easy to plan for the project.
   * The project plan, Project goal, risks, use-case model, and Project description, are made.
   * The project is checked against the milestone criteria and if it couldn’t pass these criteria then the project can be either cancelled or redesigned.
2. **Elaboration –**
   * Planning and modelling are the main ones.
   * A detailed evaluation and development plan are carried out and diminishes the risks.
   * Revise or redefine the use-case model (approx. 80%), business case, and risks.
   * Again, checked against milestone criteria and if it couldn’t pass these criteria then again project can be cancelled or redesigned.
   * Executable architecture baseline.
3. **Construction –**
   * The project is developed and completed.
   * System or source code is created and then testing is done.
   * Coding takes place.
4. **Transition –**
   * The final project is released to the public.
   * Transit the project from development into production.
   * Update project documentation.
   * Beta testing is conducted.
   * Defects are removed from the project based on feedback from the public.
5. **Production –**
   * The final phase of the model.
   * The project is maintained and updated accordingly.

**Advantages:**

1. It provides good documentation, it completes the process in itself.
2. It provides risk-management support.
3. It reuses the components, and hence total time duration is less.
4. Good online support is available in the form of tutorials and training.

**Disadvantages:**

1. Team of expert professional is required, as the process is complex.
2. Complex and not properly organized process.
3. More dependency on risk management.
4. Hard to integrate again and again.

**Unit-2**

**Role of System Analyst**

The **Analyst** starts requirements gathering and analysis activity by collection all info from the client that may be want to develop the necessities of the system. He then analyzes the collected info to get a transparent and thorough understanding of the merchandise to be developed, with a read to removing all ambiguities and inconsistencies from the initial client perception of the matter. The subsequent basic queries per the project ought to be clearly understood by the analyst so as to get a decent grasp of the problem:

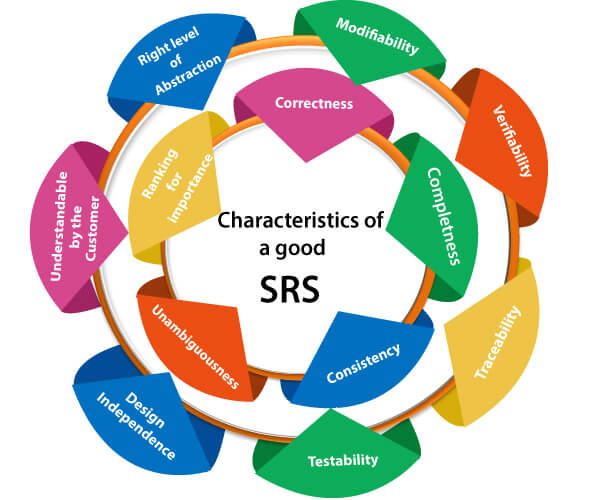
* What is the problem?
* Why is it important to solve the problem?
* What are the possible solutions to the problem?
* What precisely are the information input to the system and what exactly are the data output by the system?
* “What are the probable complexities which may arise whereas finding the problem?
* If there are external software package or hardware with that the developed software must interface, then what precisely would the information interchange formats with the external system be?

**Software Requirement Specifications (SRS)**

The production of the requirements stage of the software development process is **Software Requirements Specifications (SRS)** (also called a **requirements document**). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analysed. **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. Also, it comprises user requirements for a system as well as detailed specifications of the system 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. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

## Characteristics of good SRS



**Following are the features of a good SRS document:**

**1. Correctness:** User review is used to provide the accuracy of requirements stated in the SRS. SRS is said to be perfect if it covers all the needs that are truly expected from the system.

**2. Completeness:** The SRS is complete if, and only if, it includes the following elements:

**(1).** All essential requirements, whether relating to functionality, performance, design, constraints, attributes, or external interfaces.

**(2).** Definition of their responses of the software to all realizable classes of input data in all available categories of situations.

#### **Note: It is essential to specify the responses to both valid and invalid values.**

**(3).** Full labels and references to all figures, tables, and diagrams in the SRS and definitions of all terms and units of measure.

**3. Consistency:** The SRS is consistent if, and only if, no subset of individual requirements described in its conflict. There are three types of possible conflict in the SRS:

**(1).** The specified characteristics of real-world objects may conflict. For example,

(a) The format of an output report may be described in one requirement as tabular but in another as textual.

(b) One condition may state that all lights shall be green while another states that all lights shall be blue.

**(2).** There may be a reasonable or temporal conflict between the two specified actions. For example,

(a) One requirement may determine that the program will add two inputs, and another may determine that the program will multiply them.

(b) One condition may state that "A" must always follow "B," while other requires that "A and B" co-occurs.

**(3).** Two or more requirements may define the same real-world object but use different terms for that object. For example, a program's request for user input may be called a "prompt" in one requirement's and a "cue" in another. The use of standard terminology and descriptions promotes consistency.

**4. Unambiguousness:** SRS is unambiguous when every fixed requirement has only one interpretation. This suggests that each element is uniquely interpreted. In case there is a method used with multiple definitions, the requirements report should determine the implications in the SRS so that it is clear and simple to understand.

**5. Ranking for importance and stability:** The SRS is ranked for importance and stability if each requirement in it has an identifier to indicate either the significance or stability of that particular requirement.

Typically, all requirements are not equally important. Some prerequisites may be essential, especially for life-critical applications, while others may be desirable. Each element should be identified to make these differences clear and explicit. Another way to rank requirements is to distinguish classes of items as essential, conditional, and optional.

**6. Modifiability:** SRS should be made as modifiable as likely and should be capable of quickly obtain changes to the system to some extent. Modifications should be perfectly indexed and cross-referenced.

**7. Verifiability:** SRS is correct when the specified requirements can be verified with a cost-effective system to check whether the final software meets those requirements. The requirements are verified with the help of reviews.

**8. Traceability:** The SRS is traceable if the origin of each of the requirements is clear and if it facilitates the referencing of each condition in future development or enhancement documentation.

**There are two types of Traceability:**

**1. Backward Traceability:** This depends upon each requirement explicitly referencing its source in earlier documents.

**2. Forward Traceability:** This depends upon each element in the SRS having a unique name or reference number.

The forward traceability of the SRS is especially crucial when the software product enters the operation and maintenance phase. As code and design document is modified, it is necessary to be able to ascertain the complete set of requirements that may be concerned by those modifications.

**9. Design Independence:** There should be an option to select from multiple design alternatives for the final system. More specifically, the SRS should not contain any implementation details.

**10. Testability:** An SRS should be written in such a method that it is simple to generate test cases and test plans from the report.

**11. Understandable by the customer:** An end user may be an expert in his/her explicit domain but might not be trained in computer science. Hence, the purpose of formal notations and symbols should be avoided too as much extent as possible. The language should be kept simple and clear.

**12. The right level of abstraction:** If the SRS is written for the requirements stage, the details should be explained explicitly. Whereas, for a feasibility study, fewer analysis can be used. Hence, the level of abstraction modifies according to the objective of the SRS.

## Properties of a good SRS document

**The essential properties of a good SRS document are the following:**

**Concise:** The SRS report should be concise and at the same time, unambiguous, consistent, and complete. Verbose and irrelevant descriptions decrease readability and also increase error possibilities.

**Structured:** It should be well-structured. A well-structured document is simple to understand and modify. In practice, the SRS document undergoes several revisions to cope up with the user requirements. Often, user requirements evolve over a period of time. Therefore, to make the modifications to the SRS document easy, it is vital to make the report well-structured.

**Black-box view:** It should only define what the system should do and refrain from stating how to do these. This means that the SRS document should define the external behaviour of the system and not discuss the implementation issues. The SRS report should view the system to be developed as a black box and should define the externally visible behaviour of the system. For this reason, the SRS report is also known as the black-box specification of a system.

**Conceptual integrity:** It should show conceptual integrity so that the reader can merely understand it. Response to undesired events: It should characterize acceptable responses to unwanted events. These are called system response to exceptional conditions.

**Verifiable:** All requirements of the system, as documented in the SRS document, should be correct. This means that it should be possible to decide whether or not requirements have been met in an implementation.

**Functional and Non-Functional Requirements**

## Functional Requirements

Functional requirements define a function that a system or system element must be qualified to perform and must be documented in different forms. The functional requirements describe the behaviour of the system as it correlates to the system's functionality.

Functional requirements should be written in a simple language, so that it is easily understandable. The examples of functional requirements are authentication, business rules, audit tracking, certification requirements, transaction corrections, etc.

These requirements allow us to verify whether the application provides all functionalities mentioned in the application's functional requirements. They support tasks, activities, user goals for easier project management.

There are a number of ways to prepare functional requirements. The most common way is that they are documented in the text form. Other formats of preparing the functional requirements are use cases, models, prototypes, user stories, and diagrams.

## Non-functional requirements

Non-functional requirements are not related to the software's functional aspect. They can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviours of the system. Basic non-functional requirements are - usability, reliability, security, storage, cost, flexibility, configuration, performance, legal or regulatory requirements, etc.

They are divided into two main categories:

**Execution qualities** like security and usability, which are observable at run time.

**Evolution qualities** like testability, maintainability, extensibility, and scalability that embodied in the static structure of the software system.

## Functional requirements v/s Non-functional requirements

|  |  |
| --- | --- |
| **Functional Requirements** | **Non-functional requirements** |
| Functional requirements help to understand the functions of the system. | They help to understand the system's performance. |
| Functional requirements are mandatory. | While non-functional requirements are not mandatory. |
| They are easy to define. | They are hard to define. |
| They describe what the product does. | They describe the working of product. |
| It concentrates on the user's requirement. | It concentrates on the expectation and experience of the user. |
| It helps us to verify the software's functionality. | It helps us to verify the software's performance. |
| These requirements are specified by the user. | These requirements are specified by the software developers, architects, and technical persons. |
| There is functional testing such as API testing, system, integration, etc. | There is non-functional testing such as usability, performance, stress, security, etc. |
| Examples of the functional requirements are - Authentication of a user on trying to log in to the system. | Examples of the non-functional requirements are - The background color of the screens should be light blue. |
| These requirements are important to system operation. | These are not always the important requirements, they may be desirable. |
| Completion of Functional requirements allows the system to perform, irrespective of meeting the non-functional requirements. | While system will not work only with non-functional requirements. |

# **Difference between Decision Table and Decision Tree**

**1.**[**Decision Table:**](https://www.geeksforgeeks.org/software-engineering-decision-table/) Decision Table is just a tabular representation of all conditions and actions. Decision Table are always used whenever the processing logic is very complicated and involves multiple conditions. The main components used for the formation of the Data Table are Conditions Stubs, Action Stubs, and rules.

**Types of decision tables:**

* Extended entry table
* Limited entry table

**Benefits:**

* Visualization of Cause and effect relationships in the table.
* Easy to understand
* In the case of a complex table, it can be readily broken down into simpler tables.
* Tables are formatted consistently.
* Suggestions of possible actions need to be taken from the summarized outcomes of a situation.
* In these tables, semi-standardized languages might be used.
* Table users are not necessarily known how to use a computer.

**Drawbacks:**

* Decision tables are not well suited to large-scale applications. There is a requirement of splitting huge tables into smaller ones to eliminate redundancy.
* The complete sequence of actions is not reflected in the decision tables.
* A partial solution is presented.

**2.**[**Decision Tree:**](https://www.geeksforgeeks.org/decision-tree-in-software-engineering/) A decision tree is a graph that always uses a branching method in order to demonstrate all the possible outcomes of any decision. Decision Trees are graphical and show a better representation of decision outcomes. It consists of three nodes namely Decision Nodes, Chance Nodes, and Terminal Nodes.

**Types of the decision tree:**

* Categorical variable decision tree
* Continuous variable decision tree

**Benefits:**

* A decision tree is simple to comprehend and use.
* New scenarios are simple to add.
* Can be combined with other decision-making methods.
* Handling of both numerical and categorial variables
* The classification does not require many computations.
* Useful in analyzing and solving various business problems.

**Drawbacks:**

* They are inherently unstable, which means that a slight change in the data can have a result in a change in the structure of the optimal decision tree, and they are frequently wrong.
* These are less suitable for estimation tasks where the outcome required is the value of a continuous variable.
* The alternative options perform better with the same data. A random forest of decision trees can be used as a replacement but it is not as straightforward to comprehend as a single decision tree.
* Calculations can become quite complicated, especially when several values are uncertain and/or multiple outcomes are related.

### Difference between Decision Table and Decision Tree:

| **S. No.** | **Decision Table** | **Decision Tree** |
| --- | --- | --- |
| 1. | Decision Tables are a tabular representation of conditions and actions. | Decision Trees are a graphical representation of every possible outcome of a decision. |
| 2. | We can derive a decision table from the decision tree. | We cannot derive a decision tree from the decision table. |
| 3. | It helps to clarify the criteria. | It helps to consider the possible relevant outcomes of the decision. |
| 4. | In Decision Tables, we can include more than one ‘or’ condition. | In Decision Trees, we cannot include more than one ‘or’ condition. |
| 5. | It is used when there are small number of properties. | It is used when there are a greater number of properties. |
| 6. | It is used for simple logic only. | It can be used for complex logic as well. |
| 7. | It is constructed of rows and tables. | It is constructed of branches and nodes. |
| 8. | The goal of using a decision table is the generation of rules for structuring logic on the basis of data entered in the table. | A decision tree’s objective is to provide an effective means to visualize and understand a decision’s available possibilities and range of possible outcomes. |

The **formal methods model** is concerned with the application of a mathematical technique to design and implement the software. This model lays the foundation for developing a complex system and supporting the program development. The formal methods used during the development process provide a mechanism for eliminating problems, which are difficult to overcome using other software process models. The software engineer creates formal specifications for this model. These methods minimize specification errors and this result in fewer errors when the user begins using the system.

Formal methods comprise *formal specification*using mathematics to specify the desired properties of the system. Formal specification is expressed in a language whose syntax and semantics are formally defined. This language comprises a syntax that defines specific notation used for specification representation; semantic*,*which uses objects to describe the system; and a set of relations*,*which uses rules to indicate the objects for satisfying the specification.

Generally, the formal method comprises two approaches, namely, property basedand model-based*.*The **property-based specification**describes the operationsperformed on the system. In addition, it describes the relationship that exists amongthese operations. A property-based specification consists of two parts: signatures*,*which determine the syntax of operations and an equation*,*which defines thesemantics of the operations through a set of equations known as **axioms.**The**model-based specification**utilizes the tools of set theory, function theory, andlogic to develop an abstract model of the system. In addition, it specifies theoperations performed on the abstract model. The model thus developed is of a highlevel and idealized. A model-based specification comprises a definition of the set ofstates of the system and definitions of the legal operations performed on the systemto indicate how these legal operations change the current state.

# **Software Cost Estimation**

For any new software project, it is necessary to know how much it will cost to develop and how much development time will it take. These estimates are needed before development is initiated, but how is this done? Several estimation procedures have been developed and are having the following attributes in common.

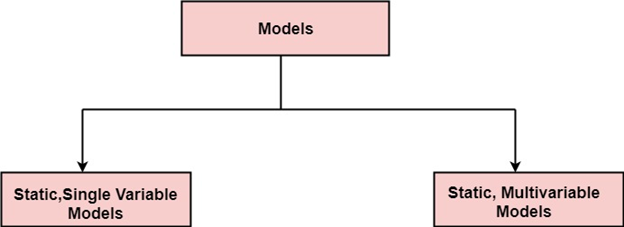
1. Project scope must be established in advanced.
2. Software metrics are used as a support from which evaluation is made.
3. The project is broken into small PCs which are estimated individually.  
   To achieve true cost & schedule estimate, several options arise.
4. Delay estimation
5. Used symbol decomposition techniques to generate project cost and schedule estimates.
6. Acquire one or more automated estimation tools.

## Uses of Cost Estimation

1. During the planning stage, one needs to choose how many engineers are required for the project and to develop a schedule.
2. In monitoring the project's progress, one needs to access whether the project is progressing according to the procedure and takes corrective action, if necessary.

## Cost Estimation Models

A model may be static or dynamic. In a static model, a single variable is taken as a key element for calculating cost and time. In a dynamic model, all variables are interdependent, and there is no basic variable.



**Static, Single Variable Models:** When a model makes use of single variables to calculate desired values such as cost, time, efforts, etc. is said to be a single variable model. The most common equation is:

**C=aLb**

**Where**   C=cost L=size  
                a and b are constants

**Static, Multivariable Models:** These models are based on method (1), they depend on several variables describing various aspects of the software development environment. In some model, several variables are needed to describe the software development process, and selected equation combined these variables to give the estimate of time & cost. These models are called multivariable models.

# **COCOMO Model**

Boehm proposed COCOMO (Constructive Cost Estimation Model) in 1981.COCOMO is one of the most generally used software estimation models in the world. COCOMO predicts the efforts and schedule of a software product based on the size of the software.

**The necessary steps in this model are:**

1. Get an initial estimate of the development effort from evaluation of thousands of delivered lines of source code (KDLOC).
2. Determine a set of 15 multiplying factors from various attributes of the project.
3. Calculate the effort estimate by multiplying the initial estimate with all the multiplying factors i.e., multiply the values in step1 and step2.

The initial estimate (also called nominal estimate) is determined by an equation of the form used in the static single variable models, using KDLOC as the measure of the size. To determine the initial effort Ei in person-months the equation used is of the type is shown below

**Ei=a\*(KDLOC)b**

The value of the constant a and b are depends on the project type.

**In COCOMO, projects are categorized into three types:**

1. Organic
2. Semidetached
3. Embedded

**1.Organic:** A development project can be treated of the organic type, if the project deals with developing a well-understood application program, the size of the development team is reasonably small, and the team members are experienced in developing similar methods of projects. **Examples of this type of projects are simple business systems, simple inventory management systems, and data processing systems.**

**2. Semidetached:** A development project can be treated with semidetached type if the development consists of a mixture of experienced and inexperienced staff. Team members may have finite experience in related systems but may be unfamiliar with some aspects of the order being developed. **Example of Semidetached system includes developing a new operating system (OS), a Database Management System (DBMS), and complex inventory management system.**

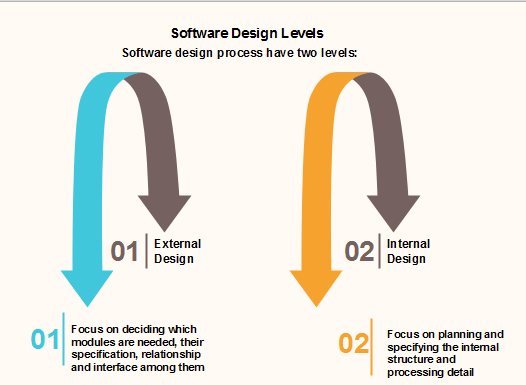
**3. Embedded:** A development project is treated to be of an embedded type, if the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist. **For Example:** ATM, Air Traffic control.

**Unit-3**

# **Software Design**

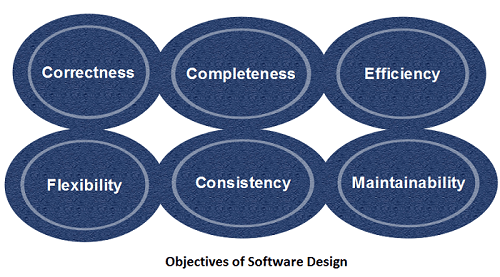
Software design is a mechanism to transform user requirements into some suitable form, which helps the programmer in software coding and implementation. It deals with representing the client's requirement, as described in SRS (Software Requirement Specification) document, into a form, i.e., easily implementable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

The software design phase is the first step in **SDLC (Software Design Life Cycle)**, which moves the concentration from the problem domain to the solution domain. In software design, we consider the system to be a set of components or modules with clearly defined behaviours & boundaries.



## Objectives of Software Design

Following are the purposes of Software design:



1. **Correctness:** Software design should be correct as per requirement.
2. **Completeness:** The design should have all components like data structures, modules, and external interfaces, etc.
3. **Efficiency:** Resources should be used efficiently by the program.
4. **Flexibility:** Able to modify on changing needs.
5. **Consistency:** There should not be any inconsistency in the design.
6. **Maintainability:** The design should be so simple so that it can be easily maintainable by other designers.

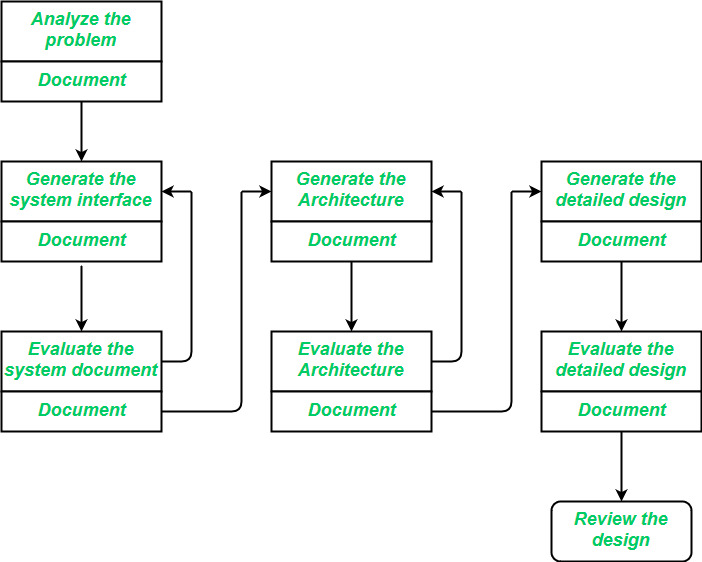
# **Software Engineering | Software Design Process**

The design phase of software development deals with transforming the customer requirements as described in the SRS documents into a form implementable using a programming language. The software design process can be divided into the following three levels of phases of design:

1. Interface Design
2. Architectural Design
3. Detailed Design

**Elements of a System:**

1. **Architecture –** This is the conceptual model that defines the structure, behaviour, and views of a system. We can use flowcharts to represent and illustrate the architecture.
2. **Modules –** These are components that handle one specific task in a system. A combination of the modules makes up the system.
3. **Components –** This provides a particular function or group of related functions. They are made up of modules.
4. **Interfaces –**This is the shared boundary across which the components of a system exchange information and relate.
5. **Data –**This is the management of the information and data flow.

  
   
**Interface Design:** *Interface design* is the specification of the interaction between a system and its environment. this phase proceeds at a high level of abstraction with respect to the inner workings of the system i.e., during interface design, the internal of the systems are completely ignored and the system is treated as a black box. Attention is focused on the dialogue between the target system and the users, devices, and other systems with which it interacts. The design problem statement produced during the problem analysis step should identify the people, other systems, and devices which are collectively called *agents*. Interface design should include the following details:

* Precise description of events in the environment, or messages from agents to which the system must respond.
* Precise description of the events or messages that the system must produce.
* Specification of the data, and the formats of the data coming into and going out of the system.
* Specification of the ordering and timing relationships between incoming events or messages, and outgoing events or outputs.

**Architectural Design:** *Architectural design* is the specification of the major components of a system, their responsibilities, properties, interfaces, and the relationships and interactions between them. In architectural design, the overall structure of the system is chosen, but the internal details of major components are ignored. Issues in architectural design includes:

* Gross decomposition of the systems into major components.
* Allocation of functional responsibilities to components.
* Component Interfaces
* Component scaling and performance properties, resource consumption properties, reliability properties, and so forth.
* Communication and interaction between components.

The architectural design adds important details ignored during the interface design. Design of the internals of the major components is ignored until the last phase of the design.

**Detailed Design:** *Design* is the specification of the internal elements of all major system components, their properties, relationships, processing, and often their algorithms and the data structures. The detailed design may include:

* Decomposition of major system components into program units.
* Allocation of functional responsibilities to units.
* User interfaces
* Unit states and state changes
* Data and control interaction between units
* Data packaging and implementation, including issues of scope and visibility of program elements
* Algorithms and data structures

### **Importance of Design Documentation:**

**1. Requirements are well understood:**With proper documentation, we can remove inconsistencies and conflicts about the requirements. Requirements are well understood by every team member.

**2. Architecture/Design of product:**Architecture/Design documents give us a complete overview of how the product look like and better insight to the customer/user about their product.

**3. New Person can also work on the project:**New person to the project can very easily understand the project through documentations and start working on it. So, developers need to maintain the documentation and keep upgrading it according to the changes made in the product/software.

**4. Everything is well Stated:**This documentation is helpful to understand each and every working of the product. It explains each and every feature of the product/software.

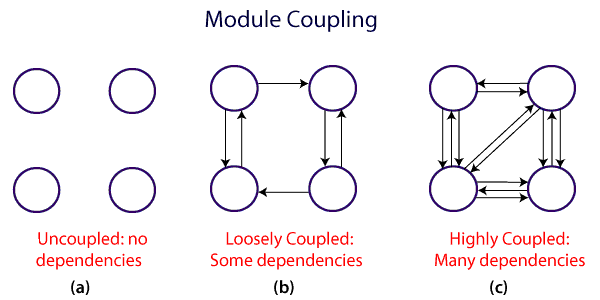
**5. Proper Communication:**Through documentation, we have good communication with every member who is part of the project/software. Helpful in understanding role and contribution of each and every member.

# **Coupling and Cohesion**

## Module Coupling

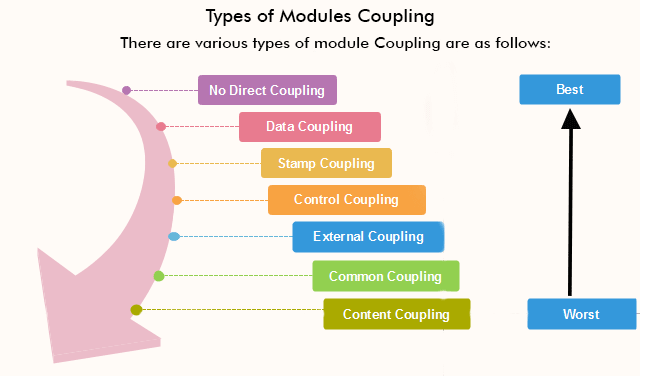
In software engineering, the coupling is the degree of interdependence between software modules. Two modules that are tightly coupled are strongly dependent on each other. However, two modules that are loosely coupled are not dependent on each other. **Uncoupled modules** have no interdependence at all within them.

**The various types of coupling techniques are shown in fig:**

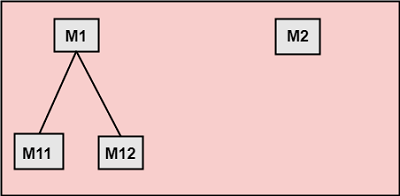


A good design is the one that has low coupling. Coupling is measured by the number of relations between the modules. That is, the coupling increases as the number of calls between modules increase or the amount of shared data is large. Thus, it can be said that a design with high coupling will have more errors.

### **Types of Module Coupling**

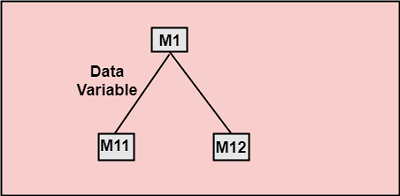


**1. No Direct Coupling:** There is no direct coupling between M1 and M2.



In this case, modules are subordinates to different modules. Therefore, no direct coupling.

**2. Data Coupling:** When data of one module is passed to another module, this is called data coupling.



**3. Stamp Coupling:** Two modules are stamp coupled if they communicate using composite data items such as structure, objects, etc. When the module passes non-global data structure or entire structure to another module, they are said to be stamp coupled. For example, passing structure variable in C or object in C++ language to a module.

**4. Control Coupling:** Control Coupling exists among two modules if data from one module is used to direct the structure of instruction execution in another.

**5. External Coupling:** External Coupling arises when two modules share an externally imposed data format, communication protocols, or device interface. This is related to communication to external tools and devices.

**6. Common Coupling:** Two modules are common coupled if they share information through some global data items.

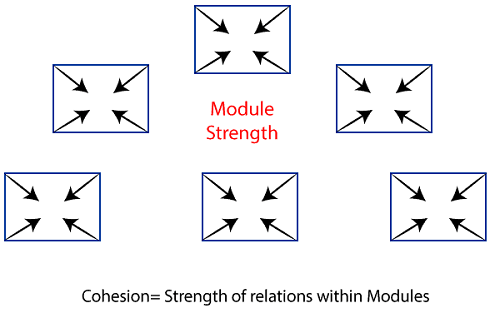


**7. Content Coupling:** Content Coupling exists among two modules if they share code, e.g., a branch from one module into another module.

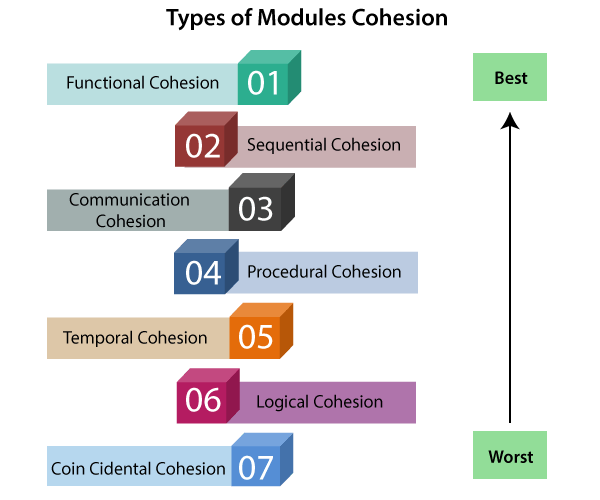
## Module Cohesion

In computer programming, cohesion defines to the degree to which the elements of a module belong together. Thus, cohesion measures the strength of relationships between pieces of functionality within a given module. For example, in highly cohesive systems, functionality is strongly related.

Cohesion is an **ordinal** type of measurement and is generally described as "high cohesion" or "low cohesion."



### **Types of Modules Cohesion**



1. **Functional Cohesion:** Functional Cohesion is said to exist if the different elements of a module, cooperate to achieve a single function.
2. **Sequential Cohesion:** A module is said to possess sequential cohesion if the element of a module forms the components of the sequence, where the output from one component of the sequence is input to the next.
3. **Communicational Cohesion:** A module is said to have communicational cohesion, if all tasks of the module refer to or update the same data structure, e.g., the set of functions defined on an array or a stack.
4. **Procedural Cohesion:** A module is said to be procedural cohesion if the set of purpose of the module are all parts of a procedure in which particular sequence of steps has to be carried out for achieving a goal, e.g., the algorithm for decoding a message.
5. **Temporal Cohesion:** When a module includes functions that are associated by the fact that all the methods must be executed in the same time, the module is said to exhibit temporal cohesion.
6. **Logical Cohesion:** A module is said to be logically cohesive if all the elements of the module perform a similar operation. For example Error handling, data input and data output, etc.
7. **Coincidental Cohesion:** A module is said to have coincidental cohesion if it performs a set of tasks that are associated with each other very loosely, if at all.

## Differentiate 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. |

## What is Structured Analysis?

Structured Analysis is a development method that allows the analyst to understand the system and its activities in a logical way.

It is a systematic approach, which uses graphical tools that analyze and refine the objectives of an existing system and develop a new system specification which can be easily understandable by user.

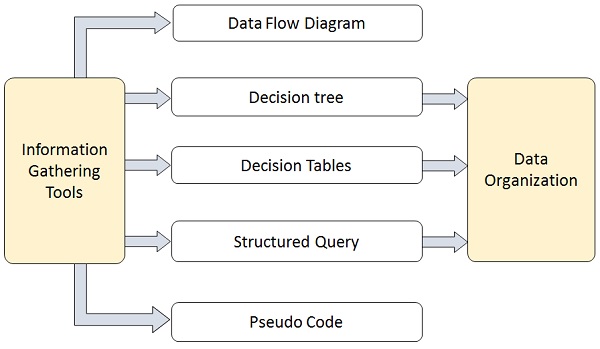
It has following attributes −

* It is graphic which specifies the presentation of application.
* It divides the processes so that it gives a clear picture of system flow.
* It is logical rather than physical i.e., the elements of system do not depend on vendor or hardware.
* It is an approach that works from high-level overviews to lower-level details.

## Structured Analysis Tools

During Structured Analysis, various tools and techniques are used for system development. They are −

* Data Flow Diagrams
* Data Dictionary
* Decision Trees
* Decision Tables
* Structured English
* Pseudocode



## Data Flow Diagrams (DFD) or Bubble Chart

It is a technique developed by Larry Constantine to express the requirements of system in a graphical form.

* It shows the flow of data between various functions of system and specifies how the current system is implemented.
* It is an initial stage of design phase that functionally divides the requirement specifications down to the lowest level of detail.
* Its graphical nature makes it a good communication tool between user and analyst or analyst and system designer.
* It gives an overview of what data a system processes, what transformations are performed, what data are stored, what results are produced and where they flow.

### **Basic Elements of DFD**

DFD is easy to understand and quite effective when the required design is not clear and the user wants a notational language for communication. However, it requires a large number of iterations for obtaining the most accurate and complete solution.

The following table shows the symbols used in designing a DFD and their significance −

|  |  |  |
| --- | --- | --- |
| **Symbol Name** | **Symbol** | **Meaning** |
| Square | Square | Source or Destination of Data |
| Arrow | Arrow | Data flow |
| Circle | Circle | Process transforming data flow |
| Open Rectangle | Rectangle | Data Store |

### **Types of DFD**

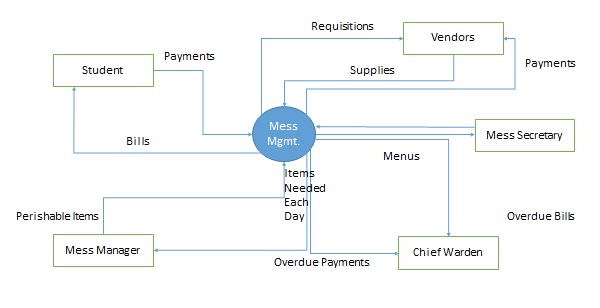
DFDs are of two types: Physical DFD and Logical DFD. The following table lists the points that differentiate a physical DFD from a logical DFD.

|  |  |
| --- | --- |
| **Physical DFD** | **Logical DFD** |
| It is implementation dependent. It shows which functions are performed. | It is implementation independent. It focuses only on the flow of data between processes. |
| It provides low level details of hardware, software, files, and people. | It explains events of systems and data required by each event. |
| It depicts how the current system operates and how a system will be implemented. | It shows how business operates; not how the system can be implemented. |

### **Context Diagram**

A context diagram helps in understanding the entire system by one DFD which gives the overview of a system. It starts with mentioning major processes with little details and then goes onto giving more details of the processes with the top-down approach.

The context diagram of mess management is shown below.



## Data Dictionary

A data dictionary is a structured repository of data elements in the system. It stores the descriptions of all DFD data elements that is, details and definitions of data flows, data stores, data stored in data stores, and the processes.

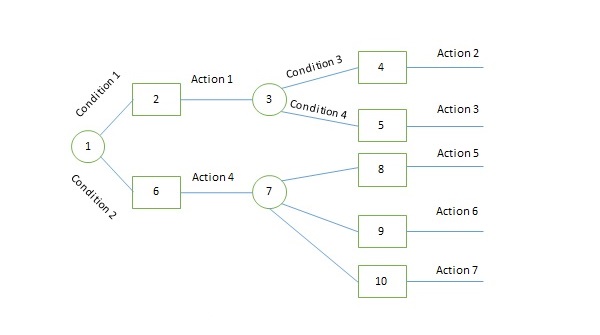
A data dictionary improves the communication between the analyst and the user. It plays an important role in building a database. Most DBMSs have a data dictionary as a standard feature. For example, refer the following table −

|  |  |  |  |
| --- | --- | --- | --- |
| **Sr.No.** | **Data Name** | **Description** | **No. of Characters** |
| 1 | ISBN | ISBN Number | 10 |
| 2 | TITLE | title | 60 |
| 3 | SUB | Book Subjects | 80 |
| 4 | ANAME | Author Name | 15 |

## Decision Trees

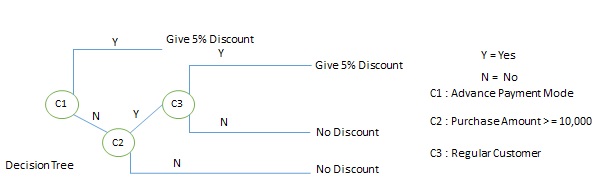
Decision trees are a method for defining complex relationships by describing decisions and avoiding the problems in communication. A decision tree is a diagram that shows alternative actions and conditions within horizontal tree framework. Thus, it depicts which conditions to consider first, second, and so on.

Decision trees depict the relationship of each condition and their permissible actions. A square node indicates an action and a circle indicates a condition. It forces analysts to consider the sequence of decisions and identifies the actual decision that must be made.



The major limitation of a decision tree is that it lacks information in its format to describe what other combinations of conditions you can take for testing. It is a single representation of the relationships between conditions and actions.

For example, refer the following decision tree −



## Decision Tables

Decision tables are a method of describing the complex logical relationship in a precise manner which is easily understandable.

* It is useful in situations where the resulting actions depend on the occurrence of one or several combinations of independent conditions.
* It is a matrix containing row or columns for defining a problem and the actions.

### **Components of a Decision Table**

* **Condition Stub** − It is in the upper left quadrant which lists all the condition to be checked.
* **Action Stub** − It is in the lower left quadrant which outlines all the action to be carried out to meet such condition.
* **Condition Entry** − It is in upper right quadrant which provides answers to questions asked in condition stub quadrant.
* **Action Entry** − It is in lower right quadrant which indicates the appropriate action resulting from the answers to the conditions in the condition entry quadrant.

The entries in decision table are given by Decision Rules which define the relationships between combinations of conditions and courses of action. In rules section,

* Y shows the existence of a condition.
* N represents the condition, which is not satisfied.
* A blank - against action states it is to be ignored.
* X (or a check mark will do) against action states it is to be carried out.

For example, refer the following table −

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **CONDITIONS** | **Rule 1** | **Rule 2** | **Rule 3** | **Rule 4** |
| Advance payment made | Y | N | N | N |
| Purchase amount = Rs 10,000/- | - | Y | Y | N |
| Regular Customer | - | Y | N | - |
| **ACTIONS** |  |  |  |  |
| Give 5% discount | X | X | - | - |
| Give no discount | - | - | X | X |

## Structured English

Structure English is derived from structured programming language which gives more understandable and precise description of process. It is based on procedural logic that uses construction and imperative sentences designed to perform operation for action.

* It is best used when sequences and loops in a program must be considered and the problem needs sequences of actions with decisions.
* It does not have strict syntax rule. It expresses all logic in terms of sequential decision structures and iterations.

For example, see the following sequence of actions −

if customer pays advance

then

Give 5% Discount

else

if purchase amount >=10,000

then

if the customer is a regular customer

then Give 5% Discount

else No Discount

end if

else No Discount

end if

end if

## Pseudocode

A pseudocode does not conform to any programming language and expresses logic in plain English.

* It may specify the physical programming logic without actual coding during and after the physical design.
* It is used in conjunction with structured programming.
* It replaces the flowcharts of a program.

## Guidelines for Selecting Appropriate Tools

Use the following guidelines for selecting the most appropriate tool that would suit your requirements −

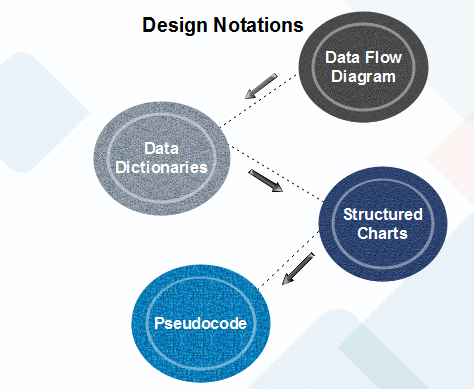
* Use DFD at high- or low-level analysis for providing good system documentations.
* Use data dictionary to simplify the structure for meeting the data requirement of the system.
* Use structured English if there are many loops and actions are complex.
* Use decision tables when there are a large number of conditions to check and logic is complex.
* Use decision trees when sequencing of conditions is important and if there are few conditions to be tested.

# **Function Oriented Design**

Function Oriented design is a method to software design where the model is decomposed into a set of interacting units or modules where each unit or module has a clearly defined function. Thus, the system is designed from a functional viewpoint.

## Design Notations

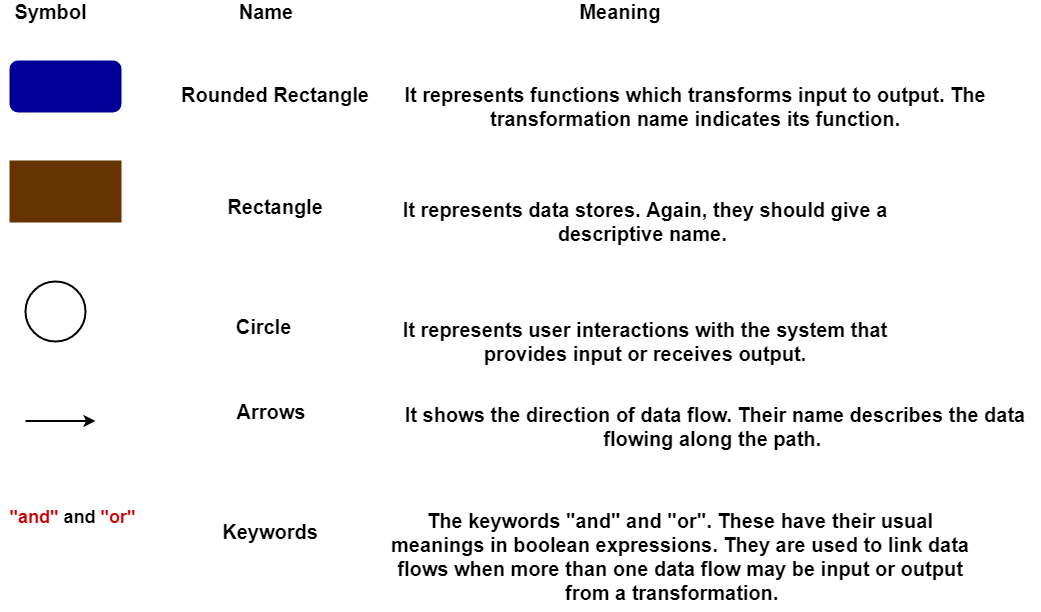
Design Notations are primarily meant to be used during the process of design and are used to represent design or design decisions. For a function-oriented design, the design can be represented graphically or mathematically by the following:

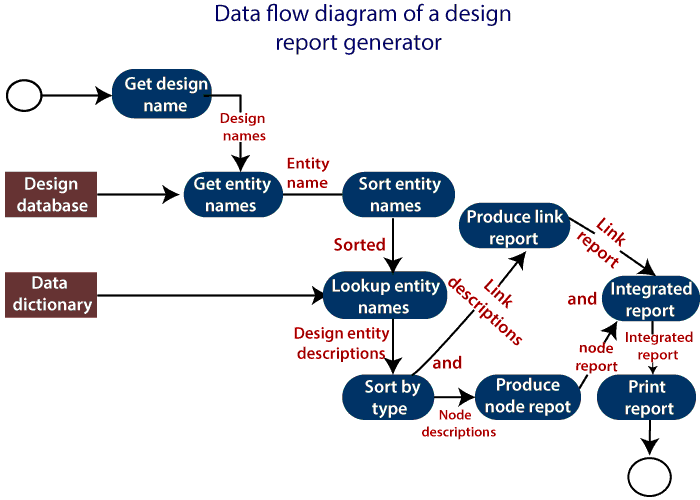


### **Data Flow Diagram**

Data-flow design is concerned with designing a series of functional transformations that convert system inputs into the required outputs. The design is described as data-flow diagrams. These diagrams show how data flows through a system and how the output is derived from the input through a series of functional transformations.

Data-flow diagrams are a useful and intuitive way of describing a system. They are generally understandable without specialized training, notably if control information is excluded. They show end-to-end processing. That is the flow of processing from when data enters the system to where it leaves the system can be traced. Data-flow design is an integral part of several design methods, and most CASE tools support data-flow diagram creation. Different ways may use different icons to represent data-flow diagram entities, but their meanings are similar.





The report generator produces a report which describes all of the named entities in a data-flow diagram. The user inputs the name of the design represented by the diagram. The report generator then finds all the names used in the data-flow diagram. It looks up a data dictionary and retrieves information about each name. This is then collated into a report which is output by the system.

### **Data Dictionaries**

A data dictionary lists all data elements appearing in the DFD model of a system. The data items listed contain all data flows and the contents of all data stores looking on the DFDs in the DFD model of a system.

A data dictionary lists the objective of all data items and the definition of all composite data elements in terms of their component data items. For example, a data dictionary entry may contain that the data grossPay consists of the parts regularPay and overtimePay.

**grossPay = regularPay + overtimePay**

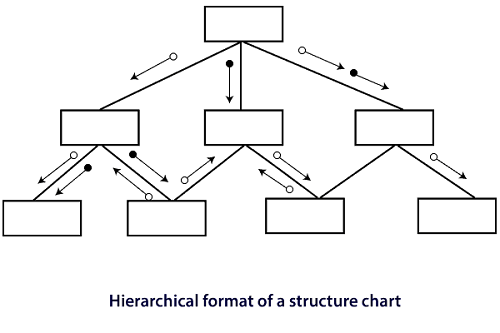
For the smallest units of data elements, the data dictionary lists their name and their type.

A data dictionary plays a significant role in any software development process because of the following reasons:

* A Data dictionary provides a standard language for all relevant information for use by engineers working in a project. A consistent vocabulary for data items is essential since, in large projects, different engineers of the project tend to use different terms to refer to the same data, which unnecessarily causes confusion.
* The data dictionary provides the analyst with a means to determine the definition of various data structures in terms of their component elements.

### **Structured Charts**

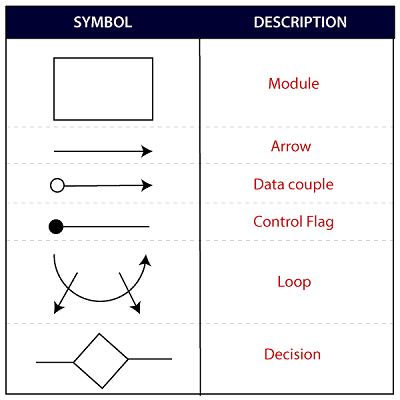
It partitions a system into block boxes. A Black box system that functionality is known to the user without the knowledge of internal design.



Structured Chart is a graphical representation which shows:

* System partitions into modules
* Hierarchy of component modules
* The relation between processing modules
* Interaction between modules
* Information passed between modules

**The following notations are used in structured chart:**



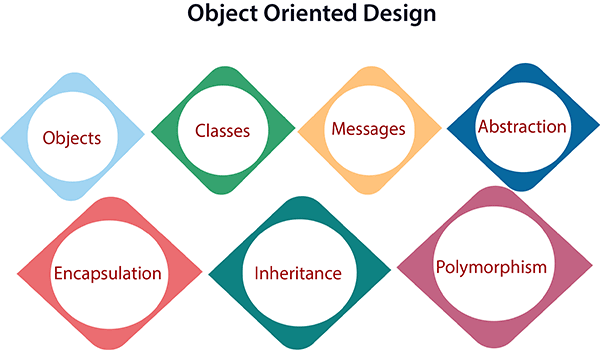
### **Pseudo-code**

Pseudo-code notations can be used in both the preliminary and detailed design phases. Using pseudo-code, the designer describes system characteristics using short, concise, English Language phases that are structured by keywords such as If-Then-Else, While-Do, and End.

# **Object-Oriented Design**

In the object-oriented design method, the system is viewed as a collection of objects (i.e., entities). The state is distributed among the objects, and each object handles its state data. For example, in a Library Automation Software, each library representative may be a separate object with its data and functions to operate on these data. The tasks defined for one purpose cannot refer or change data of other objects. Objects have their internal data which represent their state. Similar objects create a class. In other words, each object is a member of some class. Classes may inherit features from the superclass.

**The different terms related to object design are:**



1. **Objects:** All entities involved in the solution design are known as objects. For example, person, banks, company, and users are considered as objects. Every entity has some attributes associated with it and has some methods to perform on the attributes.
2. **Classes:** A class is a generalized description of an object. An object is an instance of a class. A class defines all the attributes, which an object can have and methods, which represents the functionality of the object.
3. **Messages:** Objects communicate by message passing. Messages consist of the integrity of the target object, the name of the requested operation, and any other action needed to perform the function. Messages are often implemented as procedure or function calls.
4. **Abstraction** In object-oriented design, complexity is handled using abstraction. Abstraction is the removal of the irrelevant and the amplification of the essentials.
5. **Encapsulation:** Encapsulation is also called an information hiding concept. The data and operations are linked to a single unit. Encapsulation not only bundles essential information of an object together but also restricts access to the data and methods from the outside world.
6. **Inheritance:** OOD allows similar classes to stack up in a hierarchical manner where the lower or sub-classes can import, implement, and re-use allowed variables and functions from their immediate superclasses. This property of OOD is called an inheritance. This makes it easier to define a specific class and to create generalized classes from specific ones.
7. **Polymorphism:** OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned the same name. This is known as polymorphism, which allows a single interface is performing functions for different types. Depending upon how the service is invoked, the respective portion of the code gets executed.

**Unit-4**

# **Software Testing Principles**

Software testing is a procedure of implementing software or the application to identify the defects or bugs. For testing an application or software, we need to follow some principles to make our product defects free, and that also helps the test engineers to test the software with their effort and time. Here, in this section, we are going to learn about the seven essential principles of software testing.

Let us see the seven different testing principles, one by one:

* Testing shows the presence of defects
* Exhaustive Testing is not possible
* Early Testing
* Defect Clustering
* Pesticide Paradox
* Testing is context-dependent
* Absence of errors fallacy

**Types of Testing**

**Manual Testing**

Manual testing is a software testing process in which test cases are executed manually without using any automated tool. All test cases executed by the tester manually according to the end user's perspective. It ensures whether the application is working, as mentioned in the requirement document or not. Test cases are planned and implemented to complete almost 100 percent of the software application. Test case reports are also generated manually.

Manual Testing is one of the most fundamental testing processes as it can find both visible and hidden defects of the software. The difference between expected output and output, given by the software, is defined as a defect. The developer fixed the defects and handed it to the tester for retesting.

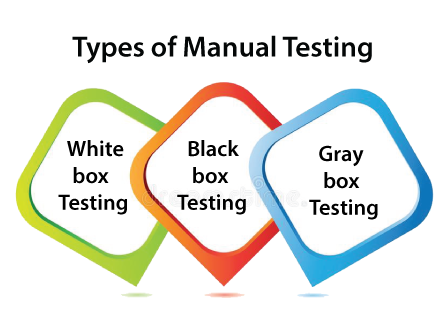
Manual testing is mandatory for every newly developed software before automated testing. This testing requires great efforts and time, but it gives the surety of bug-free software. Manual Testing requires knowledge of manual testing techniques but not of any automated testing tool.

Manual testing is essential because one of the [software testing](https://www.javatpoint.com/software-testing-tutorial) fundamentals is "100% automation is not possible."

## Types of Manual Testing

There are various methods used for manual testing. Each technique is used according to its testing criteria. Types of manual testing are given below:

* White Box Testing
* Black Box Testing
* Gray Box Testing



### **White-box testing**

The white box testing is done by Developer, where they check every line of a code before giving it to the Test Engineer. Since the code is visible for the Developer during the testing, that's why it is also known as White box testing.

For more information about white box testing, refers to the below link:

<https://www.javatpoint.com/white-box-testing>

### **Black box testing**

The black box testing is done by the Test Engineer, where they can check the functionality of an application or the software according to the customer /client's needs. In this, the code is not visible while performing the testing; that's why it is known as black-box testing.

For more information about black-box testing, refers to the below link:

<https://www.javatpoint.com/black-box-testing>

### **Gray Box testing**

Gray box testing is a combination of white box and Black box testing. It can be performed by a person who knew both coding and testing. And if the single person performs white box, as well as black-box testing for the application, is known as Gray box testing.

To get more details about gray box testing, refers to the below link:

<https://www.javatpoint.com/grey-box-testing>

## How to perform Manual Testing

* First, tester observes all documents related to software, to select testing areas.
* Tester analyses requirement documents to cover all requirements stated by the customer.
* Tester develops the test cases according to the requirement document.
* All test cases are executed manually by using Black box testing and white box testing.
* If bugs occurred then the testing team informs the development team.
* The Development team fixes bugs and handed software to the testing team for a retest.

Advantages of Manual Testing

* It does not require programming knowledge while using the Black box method.
* It is used to test dynamically changing GUI designs.
* Tester interacts with software as a real user so that they are able to discover usability and user interface issues.
* It ensures that the software is a hundred percent bug-free.
* It is cost-effective.
* Easy to learn for new testers.

Disadvantages of Manual Testing

* It requires a large number of human resources.
* It is very time-consuming.
* Tester develops test cases based on their skills and experience. There is no evidence that they have covered all functions or not.
* Test cases cannot be used again. Need to develop separate test cases for each new software.
* It does not provide testing on all aspects of testing.
* Since two teams work together, sometimes it is difficult to understand each other's motives, it can mislead the process.

Manual testing tools

In manual testing, different types of testing like unit, integration, security, performance, and bug tracking, we have various tools such as [Jira](https://www.javatpoint.com/jira-tutorial), [Bugzilla](https://www.javatpoint.com/bugzilla), Mantis, Zap, NUnit, Tessy, LoadRunner, Citrus, SonarQube, etc. available in the market. Some of the tools are open-source, and some are commercial.

For more information about testing tools, refers to the below link:

<https://www.javatpoint.com/software-testing-tools>

# **Automation Testing**

## What is Automation Testing?

**"Automation testing refers to the automatic testing of the software in which developer or tester write the test script once with the help of testing tools and framework and run it on the software. The test script automatically test the software without human intervention and shows the result (either error, bugs are present or software is free from them)."**

Automation testing needs manual effort when creating initial scripts, and further process is performed automatically to compare the actual testing result with expected results.

In **automation testing**, the test automation engineer will write the test script or use the automation testing tools to execute the application. On the other hand, in manual testing, the test engineer will write the test cases and implement the software on the basis of written test cases.

In test automation, the **test engineer** can execute repetitive tasks and other related tasks. In manual testing, it is a tedious process to implement the repetitive take again and again.

In other words, we can say that the main concentration of **test Automation** is to change the manual human activity with systems or devices.

The **automation testing** process is a time-saving process as it spends less time in exploratory testing and more time in keeping the test scripts whereas enhancing the complete test coverage.

#### **Note: We perform exploratory testing whenever the requirement does not exist.**

## Why do we perform Automation Testing?

* In software testing, automation testing is required to test the application because it offers us a better application with less effort and time.
* Some organizations still perform only manual testing to test the application as those companies are not fully aware of the automation testing process.
* But now, they are aware of automated testing and executing the test automation procedure in their application development process.
* To implement the automation testing, we required pretty a considerable investment of resources and money.

The execution of automation testing provides us various advantages, which are as discussed below:

* **Reusability**
* **Consistency**
* **Running tests anytime (24/7)**
* **Early Bug detection**
* **Less Human Resources**

|  |  |
| --- | --- |
| **Manual testing** | **Automation testing** |
| Testing in which a human tester executes test cases | In automation testing, automation tools are used to execute the test cases |
| In this testing, human resources are involved, that's why it is time-consuming | It is much faster than the manual testing |
| It is repetitive and error-prone | Here automated tools are used that make it interesting and accurate |
| BVT (build verification testing) is time-consuming and tough in manual testing | It's easy to build verification testing |
| Instead of frameworks, this testing use checklist, guidelines, and stringent process for drafting test cases. | Frameworks like keyword, hybrid, and data drive to accelerate the automation process. |
| The process turnaround time is higher than the automation testing process (one testing cycle takes lots of time) | It completes a single round of testing within record time; therefore, a process turnaround time is much lower than a manual testing process. |
| The main goal of manual testing is user-friendliness or improved customer experience. | Automation testing can only guarantee a positive customer experience and user-friendliness. |
| It is best for usability and exploratory testing | It is widely used for performing testing, load testing and regression testing. |
| Low return on investment | The high return on investment |

You can read about functional and non-functional testing by clicking the following link: - <https://www.javatpoint.com/functional-testing>

<https://www.javatpoint.com/non-functional-testing>

UNIT TESTING: - <https://www.javatpoint.com/unit-testing>

INTEGRATION TESTING: - <https://www.javatpoint.com/integration-testing>

SYSTEM TESTING: - <https://www.javatpoint.com/system-testing>

# **Software Maintenance**

Software maintenance is a part of the Software Development Life Cycle. Its primary 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.

Software Maintenance is an inclusive activity that includes error corrections, enhancement of capabilities, deletion of obsolete capabilities, and optimization.

## Need for Maintenance

Software Maintenance is needed for: -

* 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.

Thus, the maintenance is required to ensure that the system continues to satisfy user requirements.

## Types of Software Maintenance

### **1. Corrective Maintenance**

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

### **2. Adaptive Maintenance**

It contains modifying the software to match changes in the ever-changing environment.

### **3. Preventive Maintenance**

It is the process by which we prevent our system from being obsolete. It involves the concept of reengineering & reverse engineering in which an old system with old technology is re-engineered using new technology. This maintenance prevents the system from dying out.

### **4. Perfective Maintenance**

It defines improving processing efficiency or performance or restricting the software to enhance changeability. This may contain enhancement of existing system functionality, improvement in computational efficiency, etc.

# Software Engineering | Re-engineering

**Software Re-engineering** is a process of software development which is done to improve the maintainability of a software system. Re-engineering is the examination and alteration of a system to reconstitute it in a new form.

Re-engineering, also known as reverse engineering or software re-engineering, is the process of analyzing, designing, and modifying existing software systems to improve their quality, performance, and maintainability. This can include updating the software to work with new hardware or software platforms, adding new features, or improving the software’s overall design and architecture.

#### Re-engineering can be done for a variety of reasons, such as:

1. **To add new features:** Re-engineering can be used to add new features or functionality to existing software.
2. **To support new platforms**: Re-engineering can be used to update existing software to work with new hardware or software platforms.
3. **To improve maintainability:** Re-engineering can be used to improve the software’s overall design and architecture, making it easier to maintain and update over time.
4. **To meet new regulations and compliance**: Re-engineering can be done to ensure that the software is compliant with new regulations and standards.
5. Re-engineering can be a complex and time-consuming process, and it requires a thorough understanding of the existing software system. It also requires a structured and disciplined approach to software development, similar to software engineering.
6. Re-engineering can be beneficial in many cases, it can help to improve the quality, performance, and maintainability of existing software systems, but it also has its own drawbacks, such as:
7. **High costs:** Re-engineering can be a resource-intensive process and can require a significant investment in tools and training.
8. Risk of introducing new bugs: Changing existing code can introduce new bugs and compatibility issues.
9. **High learning curve:** Re-engineering can be complex, and it requires a lot of learning and training, which can be challenging for new developers.

It’s important to weigh the pros and cons of re-engineering and determine if it is the right approach for a particular software system.

*Re-engineering is the reorganizing and modifying existing software systems to make them more maintainable.*

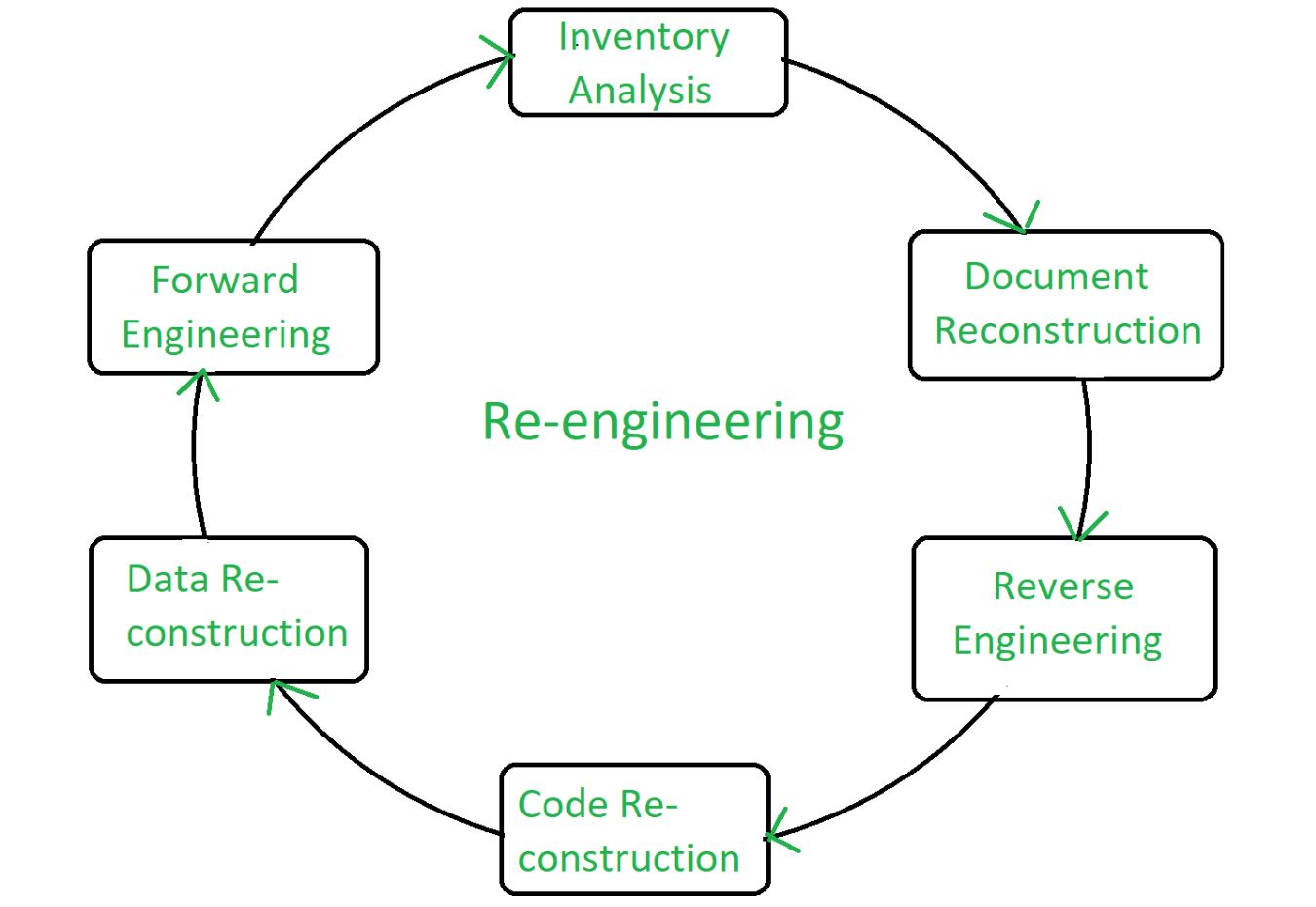
**Objectives of Re-engineering:** 

* To describe a cost-effective option for system evolution.
* To describe the activities involved in the software maintenance process.
* To distinguish between software and data re-engineering and to explain the problems of data re-engineering.

**Steps involved in Re-engineering:** 

1. Inventory Analysis
2. Document Reconstruction
3. Reverse Engineering
4. Code Reconstruction
5. Data Reconstruction
6. Forward Engineering

**Diagrammatic Representation:**



**Re-engineering Cost Factors:** 

* The quality of the software to be re-engineered
* The tool support available for re-engineering
* The extent of the required data conversion
* The availability of expert staff for re-engineering

#### Re-engineering can be a costly process, and there are several factors that can affect the cost of re-engineering a software system:

1. Size and complexity of the software: The larger and more complex the software system, the more time and resources will be required to analyze, design, and modify it.
2. Number of features to be added or modified: The more features that need to be added or modified, the more time and resources will be required.
3. Tools and technologies used: The cost of re-engineering can be affected by the tools and technologies used, such as the cost of software development tools and the cost of hardware and infrastructure.
4. Availability of documentation: If the documentation of the existing system is not available or is not accurate, then it will take more time and resources to understand the system.
5. Team size and skill level: The size and skill level of the development team can also affect the cost of re-engineering. A larger and more experienced team may be able to complete the project faster and with fewer resources.
6. Location and rate of the team: The location and rate of the development team can also affect the cost of re-engineering. Hiring a team in a lower-cost location or with lower rates can help to reduce the cost of re-engineering.
7. Testing and quality assurance: Testing and quality assurance are important aspects of re-engineering, and they can add significant costs to the project.
8. Post-deployment maintenance: The cost of post-deployment maintenance such as bug fixing, security updates, and feature additions can also play a role in the cost of re-engineering.

In summary, the cost of re-engineering a software system can vary depending on a variety of factors, including the size and complexity of the software, the number of features to be added or modified, the tools and technologies used, and the availability of documentation and the skill level of the development team. It’s important to carefully consider these factors when estimating the cost of re-engineering a software system.

**Advantages of Re-engineering:**

* **Reduced Risk:** As the software is already existing, the risk is less as compared to new software development. Development problems, staffing problems and specification problems are the lots of problems which may arise in new software development.
* **Reduced Cost:**  The cost of re-engineering is less than the costs of developing new software.
* **Revelation of Business Rules:** As a system is re-engineered, business rules that are embedded in the system are rediscovered.
* **Better use of Existing Staff:**Existing staff expertise can be maintained and extended accommodate new skills during re-engineering.
* Improved efficiency: By analyzing and redesigning processes, re-engineering can lead to significant improvements in productivity, speed, and cost-effectiveness.
* Increased flexibility: Re-engineering can make systems more adaptable to changing business needs and market conditions.
* Better customer service: By redesigning processes to focus on customer needs, re-engineering can lead to improved customer satisfaction and loyalty.
* Increased competitiveness: Re-engineering can help organizations become more competitive by improving efficiency, flexibility, and customer service.
* Improved quality: Re-engineering can lead to better quality products and services by identifying and eliminating defects and inefficiencies in processes.
* Increased innovation: Re-engineering can lead to new and innovative ways of doing things, helping organizations to stay ahead of their competitors.
* Improved compliance: Re-engineering can help organizations to comply with industry standards and regulations by identifying and addressing areas of non-compliance.
* **Disadvantages of Re-engineering:**
* Practical limits to the extent of re-engineering.
* Major architectural changes or radical reorganizing of the systems data management has to be done manually.
* Re-engineered system is not likely to be as maintainable as a new system developed using modern software Re-engineering methods.
* High costs: Re-engineering can be a costly process, requiring significant investments in time, resources, and technology.
* Disruption to business operations: Re-engineering can disrupt normal business operations and cause inconvenience to customers, employees and other stakeholders.
* Resistance to change: Re-engineering can encounter resistance from employees who may be resistant to change and uncomfortable with new processes and technologies.
* Risk of failure: Re-engineering projects can fail if they are not planned and executed properly, resulting in wasted resources and lost opportunities.
* Lack of employee involvement: Re-engineering projects that are not properly communicated and involve employees, may lead to lack of employee engagement and ownership resulting in failure of the project.
* Difficulty in measuring success: Re-engineering can be difficult to measure in terms of success, making it difficult to justify the cost and effort involved.
* Difficulty in maintaining continuity: Re-engineering can lead to significant changes in processes and systems, making it difficult to maintain continuity and consistency in the organization.

**Measures, Metrices and Indicators**

## Metrics

**Metrics** represent the different methods we employ to understand change over time across a number of dimensions or criteria. It is often used as a catch-all term to describe the method used to measure something, the resulting values obtained from measuring, as well as a calculated or combined set of measures.

We use the term **measures** when we mean the value measured by whatever mechanism we employ and the term **indicator** for values we combine and use to hint to specific outcomes and trends.

## Measures

**A measure is a number or a quantity that records a directly observable value or performance**. All measures are composed of a value (a number) and a unit of measure. The number provides magnitude for the measure (how much), while the unit gives number meaning (what is measured).

* 1,234,567 Pageviews
* 8,901,234 Sessions
* 567,890 Facebook Likes

## Indicators

**An indicator is a qualitative or quantitative factor or variable that provides a simple and reliable mean to express achievement, the attainment of a goal, or the results stemming from a specific change**. It often aggregates or combines multiple measures in an explicit formula.

* 1M weekly active users
* 1:3 users complete the story
* 23% homepage bounce rate

# **Software Measurement and Metrics**

**Software Measurement:** A measurement is a manifestation of the size, quantity, amount, or dimension of a particular attribute of a product or process. Software measurement is a titrate impute of a characteristic of a software product or the software process. It is an authority within software engineering. The software measurement process is defined and governed by ISO Standard.

### Software Measurement Principles:

The software measurement process can be characterized by five activities-

1. **Formulation:**The derivation of software measures and metrics appropriate for the representation of the software that is being considered.
2. **Collection:** The mechanism used to accumulate data required to derive the formulated metrics.
3. **Analysis:** The computation of metrics and the application of mathematical tools.
4. **Interpretation:**The evaluation of metrics resulting in insight into the quality of the representation.
5. **Feedback:**Recommendation derived from the interpretation of product metrics transmitted to the software team.

### Need for Software Measurement:

Software is measured to:

* Create the quality of the current product or process.
* Anticipate future qualities of the product or process.
* Enhance the quality of a product or process.
* Regulate the state of the project in relation to budget and schedule.

### Classification of Software Measurement:

There are 2 types of software measurement:

1. **Direct Measurement:** In direct measurement, the product, process, or thing is measured directly using a standard scale.
2. **Indirect Measurement:** In indirect measurement, the quantity or quality to be measured is measured using related parameters i.e. by use of reference.

### Metrics:

A metric is a measurement of the level at which any impute belongs to a system product or process.

Software metrics will be useful only if they are characterized effectively and validated so that their worth is proven. There are 4 functions related to software metrics:

1. Planning
2. Organizing
3. Controlling
4. Improving

### Characteristics of software Metrics:

1. **Quantitative:** Metrics must possess quantitative nature. It means metrics can be expressed in values.
2. **Understandable:** Metric computation should be easily understood, and the method of computing metrics should be clearly defined.
3. **Applicability:** Metrics should be applicable in the initial phases of the development of the software.
4. **Repeatable:** The metric values should be the same when measured repeatedly and consistent in nature.
5. **Economical:** The computation of metrics should be economical.
6. **Language Independent:** Metrics should not depend on any programming language.

### Classification of Software Metrics:

There are 3 types of software metrics:

1. **Product Metrics:** Product metrics are used to evaluate the state of the product, tracing risks and undercover prospective problem areas. The ability of the team to control quality is evaluated.
2. **Process Metrics:** Process metrics pay particular attention to enhancing the long-term process of the team or organization.
3. **Project Metrics:** The project matrix describes the project characteristic and execution process.
   * Number of software developer
   * Staffing patterns over the life cycle of software
   * Cost and schedule
   * Productivity

### Advantages of Software Metrics:

1. Reduction in cost or budget.
2. It helps to identify the particular area for improvising.
3. It helps to increase the product quality.
4. Managing the workloads and teams.
5. Reduction in overall time to produce the product.
6. It helps to determine the complexity of the code and to test the code with resources.
7. It helps in providing effective planning, controlling and managing of the entire product.