SoftWare Engineer:

What is Software Engineering?

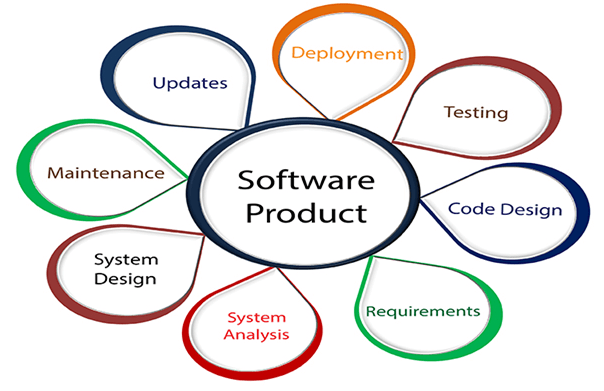
The term **software engineering** is the product of two words, **software**, and **engineering**.

The **software** is a collection of integrated programs.

Software subsists of carefully-organized instructions and code written by developers on any of various particular computer languages.

Computer programs and related documentation such as requirements, design models and user manuals.

**Engineering** is the application of **scientific** and **practical** knowledge to **invent, design, build, maintain**, and **improve frameworks, processes, etc**.



**Software Engineering** is an engineering branch related to the evolution of software product using well-defined scientific principles, techniques, and procedures. The result of software engineering is an effective and reliable software product.

## 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.

## Characteristics of a good software engineer

**The features that good software engineers should possess are as follows:**

Exposure to systematic methods, i.e., familiarity with software engineering principles.

Good technical knowledge of the project range (Domain knowledge).

Good programming abilities.

Good communication skills. These skills comprise of oral, written, and interpersonal skills.

High motivation.

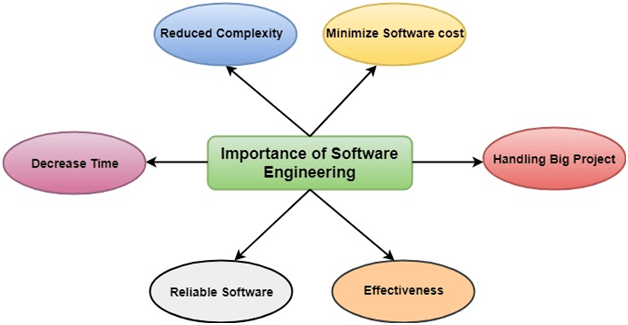
Sound knowledge of fundamentals of computer science.

Intelligence.

Ability to work in a team

Discipline, etc.

## 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 hardwork 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 its 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.

# Software Processes

The term **software** specifies to the set of computer programs, procedures and associated documents (Flowcharts, manuals, etc.) that describe the program and how they are to be used.

A software process is the set of activities and associated outcome that produce a software product. Software engineers mostly carry out these activities. These are four key process activities, which are common to all software processes. These activities are:

1. **Software specifications:** The functionality of the software and constraints on its operation must be defined.
2. **Software development:** The software to meet the requirement must be produced.
3. **Software validation:** The software must be validated to ensure that it does what the customer wants.
4. **Software evolution:** The software must evolve to meet changing client needs.

## 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.

There are several various general models or paradigms of software development:

OOPs Concepts in Java

1. **The waterfall approach:** This takes the above activities and produces them as separate process phases such as requirements specification, software design, implementation, testing, and so on. After each stage is defined, it is "signed off" and development goes onto the following stage.
2. **Evolutionary development:** This method interleaves the activities of specification, development, and validation. An initial system is rapidly developed from a very abstract specification.
3. **Formal transformation:** This method is based on producing a formal mathematical system specification and transforming this specification, using mathematical methods to a program. These transformations are 'correctness preserving.' This means that you can be sure that the developed programs meet its specification.
4. **System assembly from reusable components:** This method assumes the parts of the system already exist. The system development process target on integrating these parts rather than developing them from scratch.

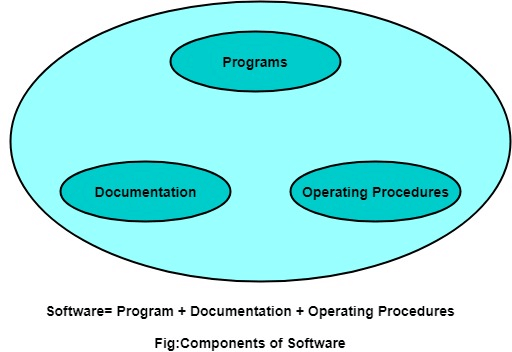
## Software Crisis

1. **Size:** Software is becoming more expensive and more complex with the growing complexity and expectation out of software. For example, the code in the consumer product is doubling every couple of years.
2. **Quality:** Many software products have poor quality, i.e., the software products defects after putting into use due to ineffective testing technique. For example, Software testing typically finds 25 errors per 1000 lines of code.
3. **Cost:** Software development is costly i.e. in terms of time taken to develop and the money involved. For example, Development of the FAA's Advanced Automation System cost over $700 per lines of code.
4. **Delayed Delivery:** Serious schedule overruns are common. Very often the software takes longer than the estimated time to develop, which in turn leads to cost shooting up. For example, one in four large-scale development projects is never completed.

## Program vs. Software

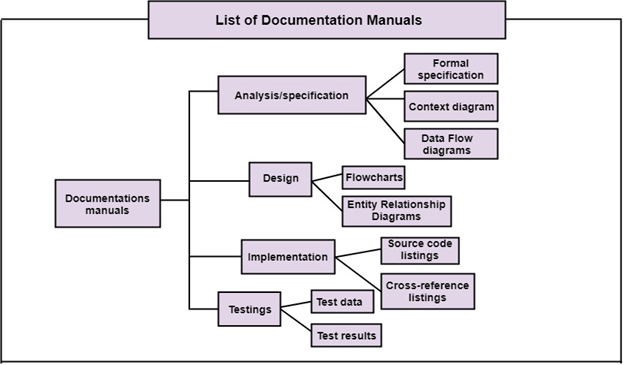
Software is more than programs. Any program is a subset of software, and it becomes software only if documentation & operating procedures manuals are prepared.

There are three components of the software as shown in fig:

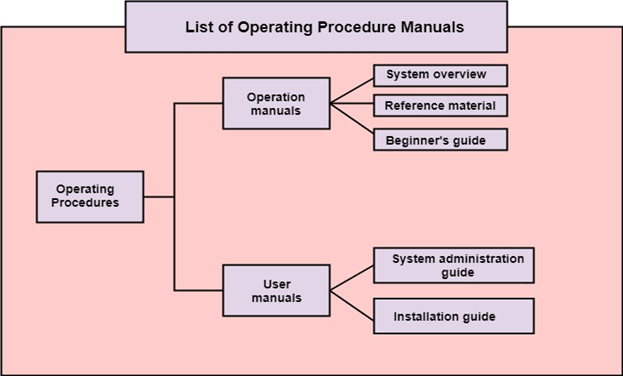


**1. Program:** Program is a combination of source code & object code.

**2. Documentation:** Documentation consists of different types of manuals. Examples of documentation manuals are: Data Flow Diagram, Flow Charts, ER diagrams, etc.



**3. Operating Procedures:** Operating Procedures consist of instructions to set up and use the software system and instructions on how react to the system failure. Example of operating system procedures manuals is: installation guide, Beginner's guide, reference guide, system administration guide, etc.



# 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.

Competitive questions on Structures in Hindi

Keep Watching

A software life cycle model describes entry and exit criteria for each phase. A phase can begin only if its stage-entry criteria have been fulfilled. So without a software life cycle model, the entry and exit criteria for a stage cannot be recognized. Without software life cycle models, it becomes tough for software project managers to monitor the progress of the project.

## SDLC Cycle

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 are 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**

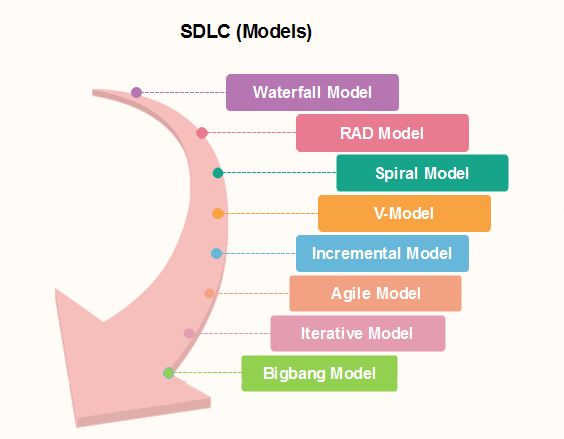
Once when the client starts using the developed systems, then the real issues come up and requirements to be solved from time to time.

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

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.

Features of Java - Javatpoint

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.

# Requirement Engineering

**Requirements engineering (RE)** refers to the process of defining, documenting, and maintaining requirements in the engineering design process. Requirement engineering provides the appropriate mechanism to understand what the customer desires, analyzing the need, and assessing feasibility, negotiating a reasonable solution, specifying the solution clearly, validating the specifications and managing the requirements as they are transformed into a working system. Thus, requirement engineering is the disciplined application of proven principles, methods, tools, and notation to describe a proposed system's intended behavior and its associated constraints.

## Requirement Engineering Process

It is a four-step process, which includes -

1. Feasibility Study
2. Requirement Elicitation and Analysis
3. Software Requirement Specification
4. Software Requirement Validation
5. Software Requirement Management



### 1. Feasibility Study:

The objective behind the feasibility study is to create the reasons for developing the software that is acceptable to users, flexible to change and conformable to established standards.

**Types of Feasibility:**

29.4M

683

OOPs Concepts in Java

1. **Technical Feasibility** - Technical feasibility evaluates the current technologies, which are needed to accomplish customer requirements within the time and budget.
2. **Operational Feasibility** - Operational feasibility assesses the range in which the required software performs a series of levels to solve business problems and customer requirements.
3. **Economic Feasibility** - Economic feasibility decides whether the necessary software can generate financial profits for an organization.

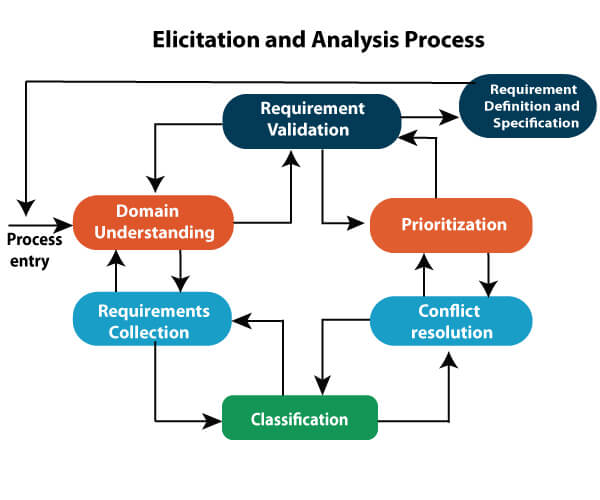
### 2. Requirement Elicitation and Analysis:

This is also known as the **gathering of requirements**. Here, requirements are identified with the help of customers and existing systems processes, if available.

Analysis of requirements starts with requirement elicitation. The requirements are analyzed to identify inconsistencies, defects, omission, etc. We describe requirements in terms of relationships and also resolve conflicts if any.

**Problems of Elicitation and Analysis**

* Getting all, and only, the right people involved.
* Stakeholders often don't know what they want
* Stakeholders express requirements in their terms.
* Stakeholders may have conflicting requirements.
* Requirement change during the analysis process.
* Organizational and political factors may influence system requirements.



### 3. Software Requirement Specification:

Software requirement specification is a kind of document which is created by a software analyst after the requirements collected from the various sources - the requirement received by the customer written in ordinary language. It is the job of the analyst to write the requirement in technical language so that they can be understood and beneficial by the development team.

The models used at this stage include ER diagrams, data flow diagrams (DFDs), function decomposition diagrams (FDDs), data dictionaries, etc.

* **Data Flow Diagrams:** Data Flow Diagrams (DFDs) are used widely for modeling the requirements. DFD shows the flow of data through a system. The system may be a company, an organization, a set of procedures, a computer hardware system, a software system, or any combination of the preceding. The DFD is also known as a data flow graph or bubble chart.
* **Data Dictionaries:** Data Dictionaries are simply repositories to store information about all data items defined in DFDs. At the requirements stage, the data dictionary should at least define customer data items, to ensure that the customer and developers use the same definition and terminologies.
* **Entity-Relationship Diagrams:** Another tool for requirement specification is the entity-relationship diagram, often called an "**E-R diagram**." It is a detailed logical representation of the data for the organization and uses three main constructs i.e. data entities, relationships, and their associated attributes.

### 4. Software Requirement Validation:

After requirement specifications developed, the requirements discussed in this document are validated. The user might demand illegal, impossible solution or experts may misinterpret the needs. Requirements can be the check against the following conditions -

* If they can practically implement
* If they are correct and as per the functionality and specially of software
* If there are any ambiguities
* If they are full
* If they can describe

**Requirements Validation Techniques**

* **Requirements reviews/inspections:** systematic manual analysis of the requirements.
* **Prototyping:** Using an executable model of the system to check requirements.
* **Test-case generation:** Developing tests for requirements to check testability.
* **Automated consistency analysis:** checking for the consistency of structured requirements descriptions.

### Software Requirement Management:

Requirement management is the process of managing changing requirements during the requirements engineering process and system development.

New requirements emerge during the process as business needs a change, and a better understanding of the system is developed.

The priority of requirements from different viewpoints changes during development process.

The business and technical environment of the system changes during the development.

## Prerequisite of Software requirements

Collection of software requirements is the basis of the entire software development project. Hence they should be clear, correct, and well-defined.

A complete Software Requirement Specifications should be:

* Clear
* Correct
* Consistent
* Coherent
* Comprehensible
* Modifiable
* Verifiable
* Prioritized
* Unambiguous
* Traceable
* Credible source

**Software Requirements:** Largely software requirements must be categorized into two categories:

1. **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 are describing the behavior of the system as it correlates to the system's functionality.
2. **Non-functional Requirements:** This can be the necessities that specify the criteria that can be used to decide the operation instead of specific behaviors of the system.  
   Non-functional requirements 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.

Understand the Software Design Fundamentals:

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

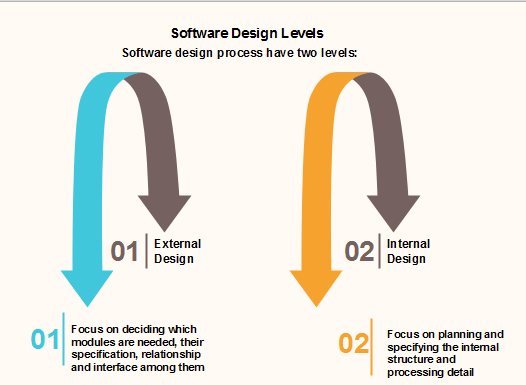
For assessing user requirements, an SRS (Software Requirement Specification) document is created whereas for coding and implementation, there is a need of more specific and detailed requirements in software terms. The output of this process can directly be used into implementation in programming languages.

Software design is the first step in SDLC (Software Design Life Cycle), which moves the concentration from problem domain to solution domain. It tries to specify how to fulfill the requirements mentioned in SRS.

## Software Design Levels

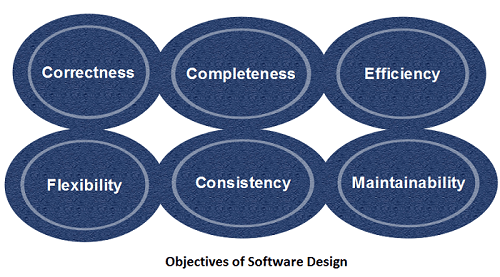
Software design yields three levels of results:

* **Architectural Design -**The architectural design is the highest abstract version of the system. It identifies the software as a system with many components interacting with each other. At this level, the designers get the idea of proposed solution domain.
* **High-level Design-**The high-level design breaks the ‘single entity-multiple component’ concept of architectural design into less-abstracted view of sub-systems and modules and depicts their interaction with each other. High-level design focuses on how the system along with all of its components can be implemented in forms of modules. It recognizes modular structure of each sub-system and their relation and interaction among each other.
* **Detailed Design-**Detailed design deals with the implementation part of what is seen as a system and its sub-systems in the previous two designs. It is more detailed towards modules and their implementations. It defines logical structure of each module and their interfaces to communicate with other modules.



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.

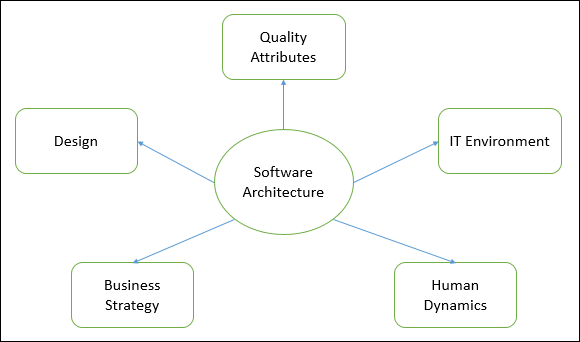
Know about the Key Issues in Software Design:

* Increasing customer demands. ...
* Time limitations. ...
* Limited infrastructure/resources. ...
* Conflicts with software testing teams.

Some of the major challenges include requirements volatility, design process, quality issues (**e.g., performance, usability, security, etc.**), distributed software development, efficient allocation of human resources to development tasks, limited budgets, unreasonable expectations and schedules, fast-changing technology,

Learn about the Software Structure and Architecture:

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment.



We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

## Software Architecture

Architecture serves as a **blueprint for a system**. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components.

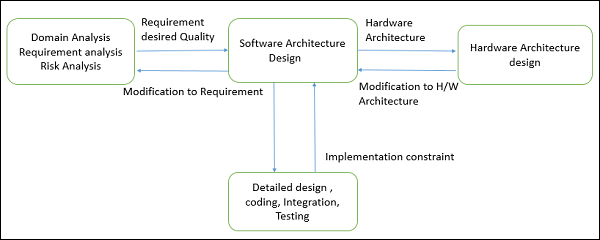
* It defines a **structured solution** to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security.
* Further, it involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of −
  + Selection of structural elements and their interfaces by which the system is composed.
  + Behavior as specified in collaborations among those elements.
  + Composition of these structural and behavioral elements into large subsystem.
  + Architectural decisions align with business objectives.
  + Architectural styles guide the organization.

## Software Design

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows −

* To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.
* Act as a blueprint during the development process.
* Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.



What Are the Metrics for Measuring Software Quality?

The metrics for measuring software quality can be extremely technical, but can be boiled down into the following four essential categories:

* Code quality
* Performance
* Security
* Usability

### Code Quality

Bug-free and semantically correct code is very important for premium software. Code quality standards can be divided into quantitative and qualitative metrics. Quantitative quality metrics measure how big or complex the software program is, the number of lines and functions it contains, how many bugs there are per 1,000 lines of code, and more. Qualitative code quality metrics measure features like maintainability, readability, clarity, efficiency, and documentation. These metrics measure how easy the code is to read, understand, and if it is written according to coding standards.

### Performance

Every software program is built for a purpose. Performance metrics measure if the product fulfils its purpose and if it performs the way it is meant to. It also refers to how the application uses resources, its scalability, customer satisfaction, and response times.

### Security

Software security metrics measure the inherent safety of a software program, and ensure there are no unauthorised changes in the product when it is handed over to the client.

Software quality product is defined in term of its fitness of purpose. That is, a quality product does precisely what the users want it to do. For software products, the fitness of use is generally explained in terms of satisfaction of the requirements laid down in the SRS document. Although "fitness of purpose" is a satisfactory interpretation of quality for many devices such as a car, a table fan, a grinding machine, etc.for software products, "fitness of purpose" is not a wholly satisfactory definition of quality.

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

**The modern view of a quality associated with a software product several quality methods such as the following:**

**Portability:** A software device is said to be portable, if it can be freely made to work in various operating system environments, in multiple machines, with other software products, etc.

25.9M

525

Java Try Catch

**Usability:** A software product has better usability if various categories of users can easily invoke the functions of the product.

**Reusability:** A software product has excellent reusability if different modules of the product can quickly be reused to develop new products.

**Correctness:** A software product is correct if various requirements as specified in the SRS document have been correctly implemented.

**Maintainability:** A software product is maintainable if bugs can be easily corrected as and when they show up, new tasks can be easily added to the product, and the functionalities of the product can be easily modified, etc.

## Software Quality Management System

A quality management system is the principal methods used by organizations to provide that the products they develop have the desired quality.

**A quality system subsists of the following:**

**Managerial Structure and Individual Responsibilities:** A quality system is the responsibility of the organization as a whole. However, every organization has a sever quality department to perform various quality system activities. The quality system of an arrangement should have the support of the top management. Without help for the quality system at a high level in a company, some members of staff will take the quality system seriously.

**Quality System Activities:** The quality system activities encompass the following:

Auditing of projects

Review of the quality system

Development of standards, methods, and guidelines, etc.

Production of documents for the top management summarizing the effectiveness of the quality system in the organization.

## Evolution of Quality Management System

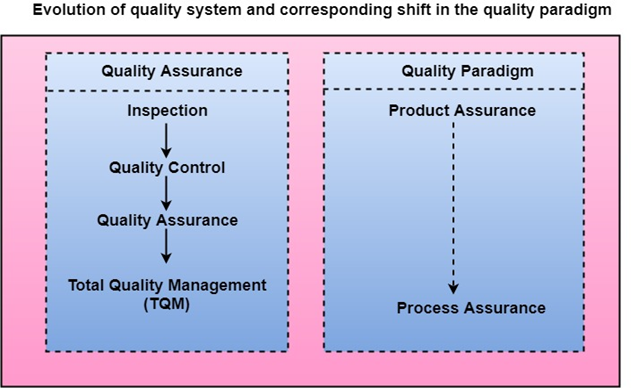
Quality systems have increasingly evolved over the last five decades. Before World War II, the usual function to produce quality products was to inspect the finished products to remove defective devices. Since that time, quality systems of organizations have undergone through four steps of evolution, as shown in the fig. The first product inspection task gave method to quality control (QC).

Quality control target not only on detecting the defective devices and removes them but also on determining the causes behind the defects. Thus, quality control aims at correcting the reasons for bugs and not just rejecting the products. The next breakthrough in quality methods was the development of quality assurance methods.

The primary premise of modern quality assurance is that if an organization's processes are proper and are followed rigorously, then the products are obligated to be of good quality. The new quality functions include guidance for recognizing, defining, analyzing, and improving the production process.

Total quality management (TQM) advocates that the procedure followed by an organization must be continuously improved through process measurements. TQM goes stages further than quality assurance and aims at frequently process improvement. TQM goes beyond documenting steps to optimizing them through a redesign. A term linked to TQM is Business Process Reengineering (BPR).

BPR aims at reengineering the method business is carried out in an organization. From the above conversation, it can be stated that over the years, the quality paradigm has changed from product assurance to process assurance, as shown in fig.



SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software.

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 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

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 are 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.

Software Designs:

Software analysis and design includes all activities, which help the transformation of requirement specification into implementation. Requirement specifications specify all functional and non-functional expectations from the software. These requirement specifications come in the shape of human readable and understandable documents, to which a computer has nothing to do.

Software analysis and design is the intermediate stage, which helps human-readable requirements to be transformed into actual code.

## Data Flow Diagram

Data flow diagram is graphical representation of flow of data in an information system. It is capable of depicting incoming data flow, outgoing data flow and stored data. The DFD does not mention anything about how data flows through the system.

There is a prominent difference between DFD and Flowchart. The flowchart depicts flow of control in program modules. DFDs depict flow of data in the system at various levels. DFD does not contain any control or branch elements.

### Types of DFD

Data Flow Diagrams are either Logical or Physical.

* **Logical DFD** - This type of DFD concentrates on the system process, and flow of data in the system.For example in a Banking software system, how data is moved between different entities.
* **Physical DFD** - This type of DFD shows how the data flow is actually implemented in the system. It is more specific and close to the implementation.

### DFD Components

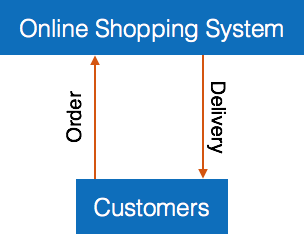
DFD can represent Source, destination, storage and flow of data using the following set of components -



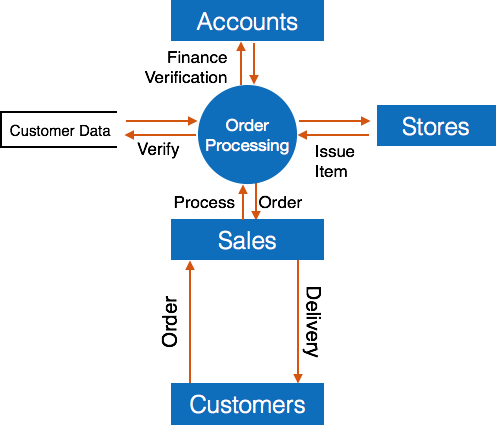
* **Entities** - Entities are source and destination of information data. Entities are represented by a rectangles with their respective names.
* **Process** - Activities and action taken on the data are represented by Circle or Round-edged rectangles.
* **Data Storage** - There are two variants of data storage - it can either be represented as a rectangle with absence of both smaller sides or as an open-sided rectangle with only one side missing.
* **Data Flow** - Movement of data is shown by pointed arrows. Data movement is shown from the base of arrow as its source towards head of the arrow as destination.

### Levels of DFD

* **Level 0** - Highest abstraction level DFD is known as Level 0 DFD, which depicts the entire information system as one diagram concealing all the underlying details. Level 0 DFDs are also known as context level DFDs.



* **Level 1** - The Level 0 DFD is broken down into more specific, Level 1 DFD. Level 1 DFD depicts basic modules in the system and flow of data among various modules. Level 1 DFD also mentions basic processes and sources of information.



* **Level 2** - At this level, DFD shows how data flows inside the modules mentioned in Level 1.

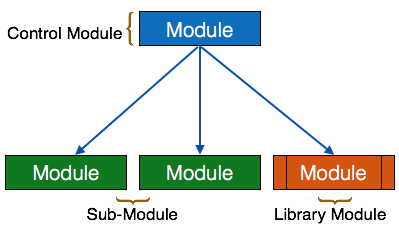
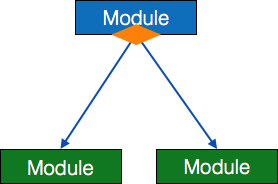
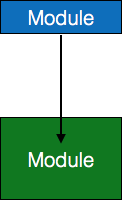
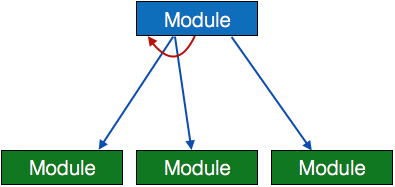
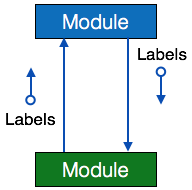
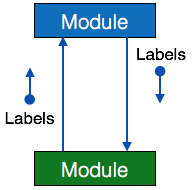
Higher level DFDs can be transformed into more specific lower level DFDs with deeper level of understanding unless the desired level of specification is achieved.

## Structure Charts

Structure chart is a chart derived from Data Flow Diagram. It represents the system in more detail than DFD. It breaks down the entire system into lowest functional modules, describes functions and sub-functions of each module of the system to a greater detail than DFD.

Structure chart represents hierarchical structure of modules. At each layer a specific task is performed.

Here are the symbols used in construction of structure charts -

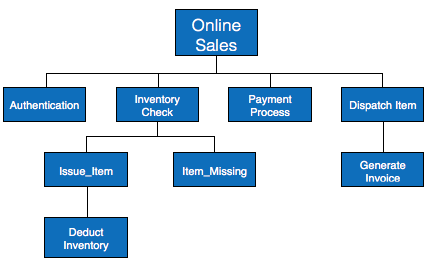
* **Module** - It represents process or subroutine or task. A control module branches to more than one sub-module. Library Modules are re-usable and invokable from any module.
* **Condition** - It is represented by small diamond at the base of module. It depicts that control module can select any of sub-routine based on some condition.
* **Jump** - An arrow is shown pointing inside the module to depict that the control will jump in the middle of the sub-module.
* **Loop** - A curved arrow represents loop in the module. All sub-modules covered by loop repeat execution of module.
* **Data flow** - A directed arrow with empty circle at the end represents data flow.
* **Control flow** - A directed arrow with filled circle at the end represents control flow.

## HIPO Diagram

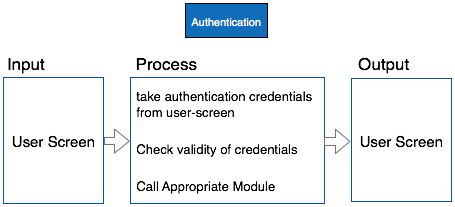
HIPO (Hierarchical Input Process Output) diagram is a combination of two organized method to analyze the system and provide the means of documentation. HIPO model was developed by IBM in year 1970.

HIPO diagram represents the hierarchy of modules in the software system. Analyst uses HIPO diagram in order to obtain high-level view of system functions. It decomposes functions into sub-functions in a hierarchical manner. It depicts the functions performed by system.

HIPO diagrams are good for documentation purpose. Their graphical representation makes it easier for designers and managers to get the pictorial idea of the system structure.



In contrast to IPO (Input Process Output) diagram, which depicts the flow of control and data in a module, HIPO does not provide any information about data flow or control flow.



### Example

Both parts of HIPO diagram, Hierarchical presentation and IPO Chart are used for structure design of software program as well as documentation of the same.

## Structured English

Most programmers are unaware of the large picture of software so they only rely on what their managers tell them to do. It is the responsibility of higher software management to provide accurate information to the programmers to develop accurate yet fast code.

Other forms of methods, which use graphs or diagrams, may are sometimes interpreted differently by different people.

Hence, analysts and designers of the software come up with tools such as Structured English. It is nothing but the description of what is required to code and how to code it. Structured English helps the programmer to write error-free code.

Other form of methods, which use graphs or diagrams, may are sometimes interpreted differently by different people. Here, both Structured English and Pseudo-Code tries to mitigate that understanding gap.

Structured English is the It uses plain English words in structured programming paradigm. It is not the ultimate code but a kind of description what is required to code and how to code it. The following are some tokens of structured programming.

IF-THEN-ELSE,

DO-WHILE-UNTIL

Analyst uses the same variable and data name, which are stored in Data Dictionary, making it much simpler to write and understand the code.

### Example

We take the same example of Customer Authentication in the online shopping environment. This procedure to authenticate customer can be written in Structured English as:

Enter Customer\_Name

SEEK Customer\_Name in Customer\_Name\_DB file

IF Customer\_Name found THEN

Call procedure USER\_PASSWORD\_AUTHENTICATE()

ELSE

PRINT error message

Call procedure NEW\_CUSTOMER\_REQUEST()

ENDIF

The code written in Structured English is more like day-to-day spoken English. It can not be implemented directly as a code of software. Structured English is independent of programming language.

## Pseudo-Code

Pseudo code is written more close to programming language. It may be considered as augmented programming language, full of comments and descriptions.

Pseudo code avoids variable declaration but they are written using some actual programming language’s constructs, like C, Fortran, Pascal etc.

Pseudo code contains more programming details than Structured English. It provides a method to perform the task, as if a computer is executing the code.

### Example

Program to print Fibonacci up to n numbers.

void function Fibonacci

Get value of n;

Set value of a to 1;

Set value of b to 1;

Initialize I to 0

for (i=0; i< n; i++)

{

if a greater than b

{

Increase b by a;

Print b;

}

else if b greater than a

{

increase a by b;

print a;

}

}

## Decision Tables

A Decision table represents conditions and the respective actions to be taken to address them, in a structured tabular format.

It is a powerful tool to debug and prevent errors. It helps group similar information into a single table and then by combining tables it delivers easy and convenient decision-making.

### Creating Decision Table

To create the decision table, the developer must follow basic four steps:

* Identify all possible conditions to be addressed
* Determine actions for all identified conditions
* Create Maximum possible rules
* Define action for each rule

Decision Tables should be verified by end-users and can lately be simplified by eliminating duplicate rules and actions.

### Example

Let us take a simple example of day-to-day problem with our Internet connectivity. We begin by identifying all problems that can arise while starting the internet and their respective possible solutions.

We list all possible problems under column conditions and the prospective actions under column Actions.

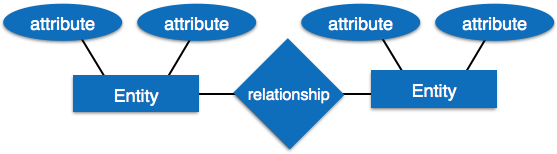
|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Conditions/Actions** | **Rules** | | | | | | | |
| **Conditions** | Shows Connected | N | N | N | N | Y | Y | Y | Y |
| Ping is Working | N | N | Y | Y | N | N | Y | Y |
| Opens Website | Y | N | Y | N | Y | N | Y | N |
| **Actions** | Check network cable | X |  |  |  |  |  |  |  |
| Check internet router | X |  |  |  | X | X | X |  |
| Restart Web Browser |  |  |  |  |  |  | X |  |
| Contact Service provider |  | X | X | X | X | X | X |  |
| Do no action |  |  |  |  |  |  |  |  |

Table : Decision Table – In-house Internet Troubleshooting

## Entity-Relationship Model

Entity-Relationship model is a type of database model based on the notion of real world entities and relationship among them. We can map real world scenario onto ER database model. ER Model creates a set of entities with their attributes, a set of constraints and relation among them.

ER Model is best used for the conceptual design of database. ER Model can be represented as follows :



* **Entity** - An entity in ER Model is a real world being, which has some properties called ***attributes***. Every attribute is defined by its corresponding set of values, called ***domain***.

For example, Consider a school database. Here, a student is an entity. Student has various attributes like name, id, age and class etc.

* **Relationship** - The logical association among entities is called ***relationship***. Relationships are mapped with entities in various ways. Mapping cardinalities define the number of associations between two entities.

Mapping cardinalities:

* + one to one
  + one to many
  + many to one
  + many to many

## Data Dictionary

Data dictionary is the centralized collection of information about data. It stores meaning and origin of data, its relationship with other data, data format for usage etc. Data dictionary has rigorous definitions of all names in order to facilitate user and software designers.

Data dictionary is often referenced as meta-data (data about data) repository. It is created along with DFD (Data Flow Diagram) model of software program and is expected to be updated whenever DFD is changed or updated.

### Requirement of Data Dictionary

The data is referenced via data dictionary while designing and implementing software. Data dictionary removes any chances of ambiguity. It helps keeping work of programmers and designers synchronized while using same object reference everywhere in the program.

Data dictionary provides a way of documentation for the complete database system in one place. Validation of DFD is carried out using data dictionary.

### Contents

Data dictionary should contain information about the following

* Data Flow
* Data Structure
* Data Elements
* Data Stores
* Data Processing

Data Flow is described by means of DFDs as studied earlier and represented in algebraic form as described.

|  |  |
| --- | --- |
| **=** | **Composed of** |
| {} | Repetition |
| () | Optional |
| + | And |
| [ / ] | Or |

### Example

Address = House No + (Street / Area) + City + State

Course ID = Course Number + Course Name + Course Level + Course Grades

### Data Elements

Data elements consist of Name and descriptions of Data and Control Items, Internal or External data stores etc. with the following details:

* Primary Name
* Secondary Name (Alias)
* Use-case (How and where to use)
* Content Description (Notation etc. )
* Supplementary Information (preset values, constraints etc.)

### Data Store

It stores the information from where the data enters into the system and exists out of the system. The Data Store may include -

* **Files**
  + Internal to software.
  + External to software but on the same machine.
  + External to software and system, located on different machine.
* **Tables**
  + Naming convention
  + Indexing property

### Data Processing

There are two types of Data Processing:

* **Logical:** As user sees it
* **Physical:** As software sees it

There are multiple variants of software design. Let us study them briefly:

## Structured Design

Structured design is a conceptualization of problem into several well-organized elements of solution. It is basically concerned with the solution design. Benefit of structured design is, it gives better understanding of how the problem is being solved. Structured design also makes it simpler for designer to concentrate on the problem more accurately.

Structured design is mostly based on ‘divide and conquer’ strategy where a problem is broken into several small problems and each small problem is individually solved until the whole problem is solved.

The small pieces of problem are solved by means of solution modules. Structured design emphasis that these modules be well organized in order to achieve precise solution.

These modules are arranged in hierarchy. They communicate with each other. A good structured design always follows some rules for communication among multiple modules, namely -

**Cohesion** - grouping of all functionally related elements.

**Coupling** - communication between different modules.

A good structured design has high cohesion and low coupling arrangements.

## Function Oriented Design

In function-oriented design, the system is comprised of many smaller sub-systems known as functions. These functions are capable of performing significant task in the system. The system is considered as top view of all functions.

Function oriented design inherits some properties of structured design where divide and conquer methodology is used.

This design mechanism divides the whole system into smaller functions, which provides means of abstraction by concealing the information and their operation.. These functional modules can share information among themselves by means of information passing and using information available globally.

Another characteristic of functions is that when a program calls a function, the function changes the state of the program, which sometimes is not acceptable by other modules. Function oriented design works well where the system state does not matter and program/functions work on input rather than on a state.

### Design Process

* The whole system is seen as how data flows in the system by means of data flow diagram.
* DFD depicts how functions changes data and state of entire system.
* The entire system is logically broken down into smaller units known as functions on the basis of their operation in the system.
* Each function is then described at large.

## Object Oriented Design

Object oriented design works around the entities and their characteristics instead of functions involved in the software system. This design strategies focuses on entities and its characteristics. The whole concept of software solution revolves around the engaged entities.

Let us see the important concepts of Object Oriented Design:

* **Objects -**All entities involved in the solution design are known as objects. For example, person, banks, company and customers are treated as objects. Every entity has some attributes associated to it and has some methods to perform on the attributes.
* **Classes -**A class is a generalized description of an object. An object is an instance of a class. Class defines all the attributes, which an object can have and methods, which defines the functionality of the object.

In the solution design, attributes are stored as variables and functionalities are defined by means of methods or procedures.

* **Encapsulation -**In OOD, the attributes (data variables) and methods (operation on the data) are bundled together is called encapsulation. Encapsulation not only bundles important information of an object together, but also restricts access of the data and methods from the outside world. This is called information hiding.
* **Inheritance -**OOD allows similar classes to stack up in hierarchical manner where the lower or sub-classes can import, implement and re-use allowed variables and methods from their immediate super classes. This property of OOD is known as inheritance. This makes it easier to define specific class and to create generalized classes from specific ones.
* **Polymorphism -**OOD languages provide a mechanism where methods performing similar tasks but vary in arguments, can be assigned same name. This is called polymorphism, which allows a single interface performing tasks for different types. Depending upon how the function is invoked, respective portion of the code gets executed.

User interface is the front-end application view to which user interacts in order to use the software. User can manipulate and control the software as well as hardware by means of user interface. Today, user interface is found at almost every place where digital technology exists, right from computers, mobile phones, cars, music players, airplanes, ships etc.

User interface is part of software and is designed such a way that it is expected to provide the user insight of the software. UI provides fundamental platform for human-computer interaction.

UI can be graphical, text-based, audio-video based, depending upon the underlying hardware and software combination. UI can be hardware or software or a combination of both.

The software becomes more popular if its user interface is:

* Attractive
* Simple to use
* Responsive in short time
* Clear to understand
* Consistent on all interfacing screens

UI is broadly divided into two categories:

* Command Line Interface
* Graphical User Interface

## Command Line Interface (CLI)

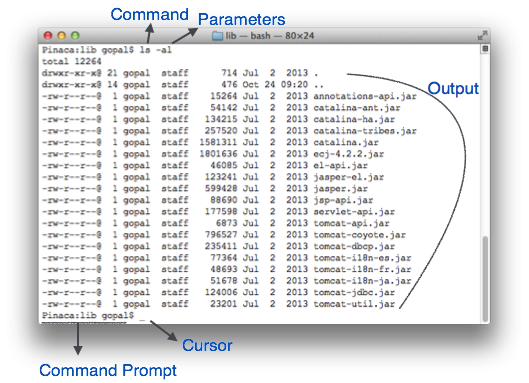
CLI has been a great tool of interaction with computers until the video display monitors came into existence. CLI is first choice of many technical users and programmers. CLI is minimum interface a software can provide to its users.

CLI provides a command prompt, the place where the user types the command and feeds to the system. The user needs to remember the syntax of command and its use. Earlier CLI were not programmed to handle the user errors effectively.

A command is a text-based reference to set of instructions, which are expected to be executed by the system. There are methods like macros, scripts that make it easy for the user to operate.

CLI uses less amount of computer resource as compared to GUI.

### CLI Elements



A text-based command line interface can have the following elements:

* **Command Prompt** - It is text-based notifier that is mostly shows the context in which the user is working. It is generated by the software system.
* **Cursor** - It is a small horizontal line or a vertical bar of the height of line, to represent position of character while typing. Cursor is mostly found in blinking state. It moves as the user writes or deletes something.
* **Command** - A command is an executable instruction. It may have one or more parameters. Output on command execution is shown inline on the screen. When output is produced, command prompt is displayed on the next line.

## Graphical User Interface

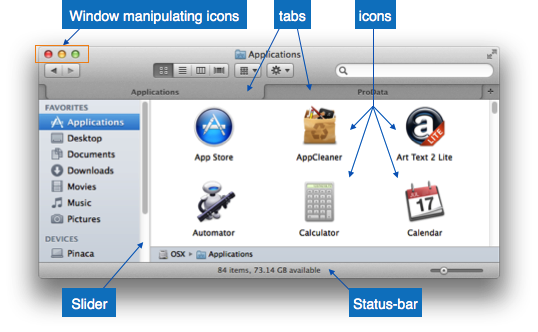
Graphical User Interface provides the user graphical means to interact with the system. GUI can be combination of both hardware and software. Using GUI, user interprets the software.

Typically, GUI is more resource consuming than that of CLI. With advancing technology, the programmers and designers create complex GUI designs that work with more efficiency, accuracy and speed.

### GUI Elements

GUI provides a set of components to interact with software or hardware.

Every graphical component provides a way to work with the system. A GUI system has following elements such as:

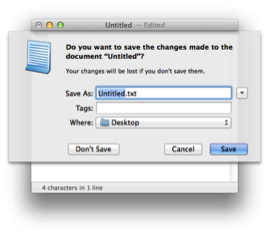


* **Window** - An area where contents of application are displayed. Contents in a window can be displayed in the form of icons or lists, if the window represents file structure. It is easier for a user to navigate in the file system in an exploring window. Windows can be minimized, resized or maximized to the size of screen. They can be moved anywhere on the screen. A window may contain another window of the same application, called child window.
* **Tabs** - If an application allows executing multiple instances of itself, they appear on the screen as separate windows.**Tabbed Document Interface** has come up to open multiple documents in the same window. This interface also helps in viewing preference panel in application. All modern web-browsers use this feature.
* **Menu** - Menu is an array of standard commands, grouped together and placed at a visible place (usually top) inside the application window. The menu can be programmed to appear or hide on mouse clicks.
* **Icon** - An icon is small picture representing an associated application. When these icons are clicked or double clicked, the application window is opened. Icon displays application and programs installed on a system in the form of small pictures.
* **Cursor** - Interacting devices such as mouse, touch pad, digital pen are represented in GUI as cursors. On screen cursor follows the instructions from hardware in almost real-time. Cursors are also named pointers in GUI systems. They are used to select menus, windows and other application features.

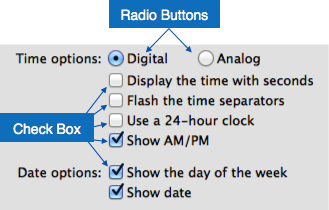
### Application specific GUI components

A GUI of an application contains one or more of the listed GUI elements:

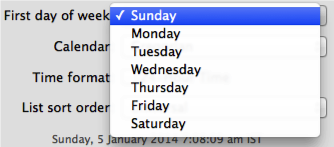
* **Application Window** - Most application windows uses the constructs supplied by operating systems but many use their own customer created windows to contain the contents of application.
* **Dialogue Box**- It is a child window that contains message for the user and request for some action to be taken. For Example: Application generate a dialogue to get confirmation from user to delete a file.



* **Text-Box** - Provides an area for user to type and enter text-based data.
* **Buttons** - They imitate real life buttons and are used to submit inputs to the software.



* **Radio-button** - Displays available options for selection. Only one can be selected among all offered.
* **Check-box** - Functions similar to list-box. When an option is selected, the box is marked as checked. Multiple options represented by check boxes can be selected.
* **List-box**- Provides list of available items for selection. More than one item can be selected.



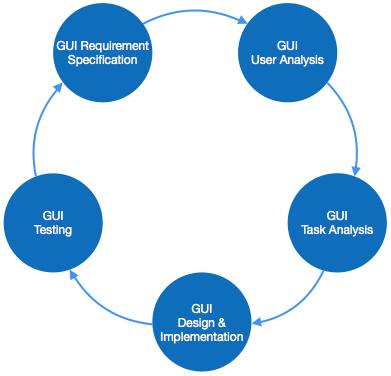
Other impressive GUI components are:

* Sliders
* Combo-box
* Data-grid
* Drop-down list

## User Interface Design Activities

There are a number of activities performed for designing user interface. The process of GUI design and implementation is alike SDLC. Any model can be used for GUI implementation among Waterfall, Iterative or Spiral Model.

A model used for GUI design and development should fulfill these GUI specific steps.



* **GUI Requirement Gathering** - The designers may like to have list of all functional and non-functional requirements of GUI. This can be taken from user and their existing software solution.
* **User Analysis** - The designer studies who is going to use the software GUI. The target audience matters as the design details change according to the knowledge and competency level of the user. If user is technical savvy, advanced and complex GUI can be incorporated. For a novice user, more information is included on how-to of software.
* **Task Analysis** - Designers have to analyze what task is to be done by the software solution. Here in GUI, it does not matter how it will be done. Tasks can be represented in hierarchical manner taking one major task and dividing it further into smaller sub-tasks. Tasks provide goals for GUI presentation. Flow of information among sub-tasks determines the flow of GUI contents in the software.
* **GUI Design & implementation** - Designers after having information about requirements, tasks and user environment, design the GUI and implements into code and embed the GUI with working or dummy software in the background. It is then self-tested by the developers.
* **Testing** - GUI testing can be done in various ways. Organization can have in-house inspection, direct involvement of users and release of beta version are few of them. Testing may include usability, compatibility, user acceptance etc.