**Unit I-INTRODUCTION TO SOFTWARE PROCESS**

**Software Engineering Tutorial**



Software Engineering Tutorial delivers basic and advanced concepts of Software Engineering. Software Engineering Tutorial is designed to help beginners and professionals both.

Software Engineering provides a standard procedure to design and develop a software.

Our Software Engineering Tutorial contains all the topics of Software Engineering like Software Engineering Models, Software Development Life Cycle, Requirement Engineering, Software Design tools, Software Design Strategies, Software Design levels, Software Project Management, Software Management activities, Software Management Tools, Software Testing levels, Software Testing approaches, Quality Assurance Vs. Quality control, Manual Testing, Software Maintenance, Software Re-engineering and Software Development Tool such as CASE Tool.

**What is Software Engineering?**

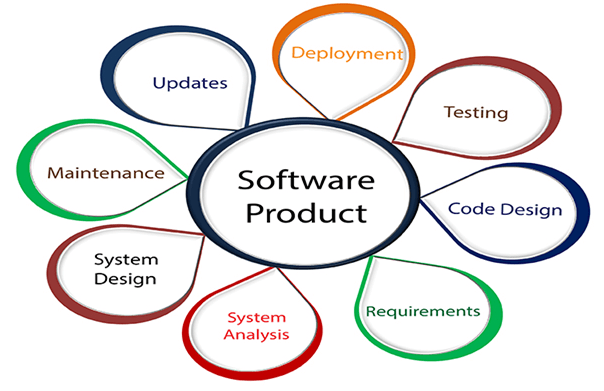
The term **software engineering** is the product of two words, **software**, and **engineering**.

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.

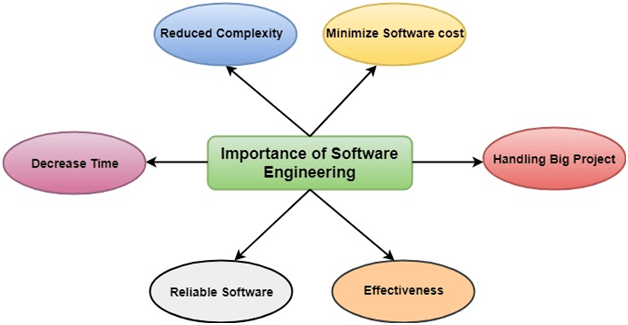
Sound knowledge of fundamentals of computer science.

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:

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.

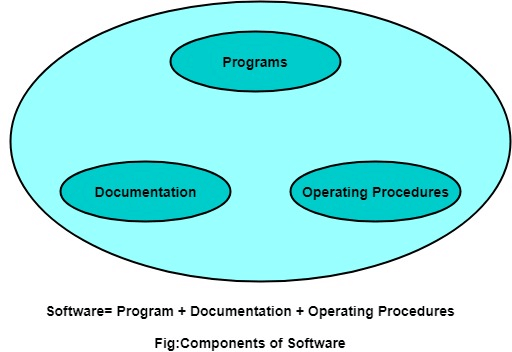
**Software Crisis**

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**

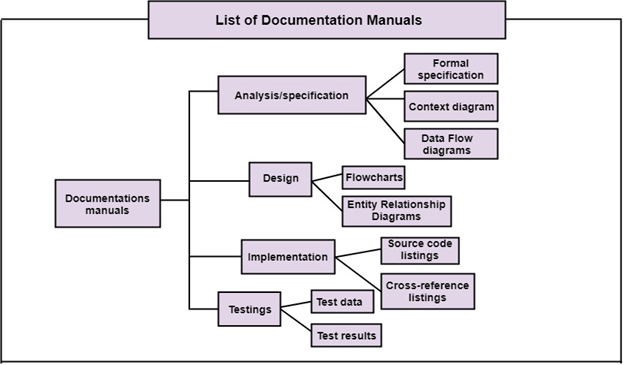
Software is more than programs. Any program is a subset of software, and it becomes software only if documentation & operating procedures manuals are prepared.

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

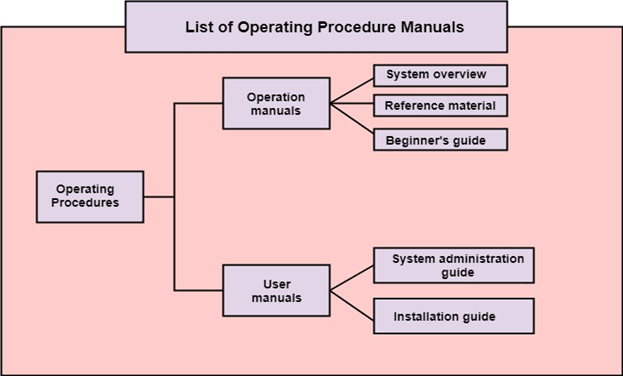


**1. Program:** Program is a combination of source code & object code.

**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.

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.

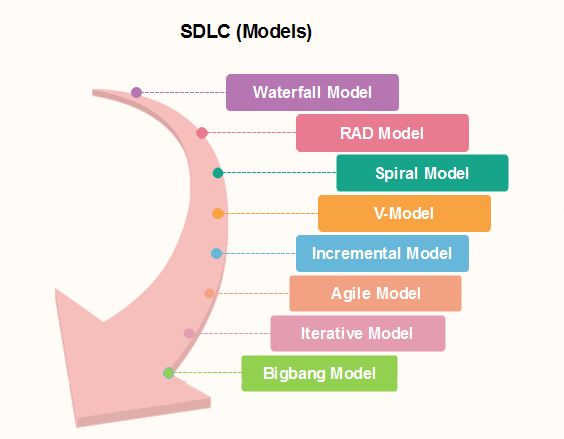
This procedure where the care is taken for the developed product is known as maintenance.

# 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.

# 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:**

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.

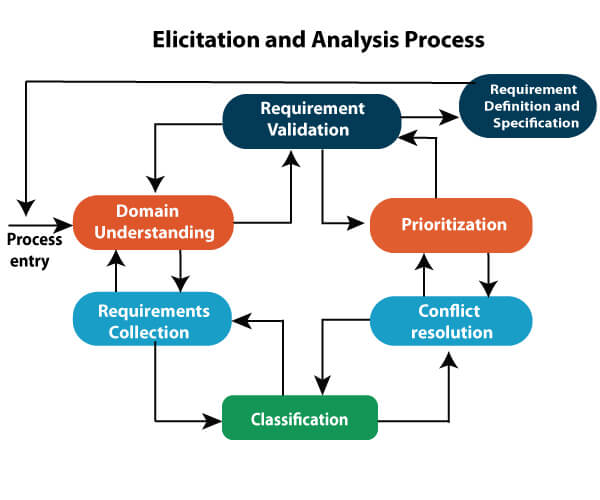
### 2. Requirement Elicitation and Analysis:

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

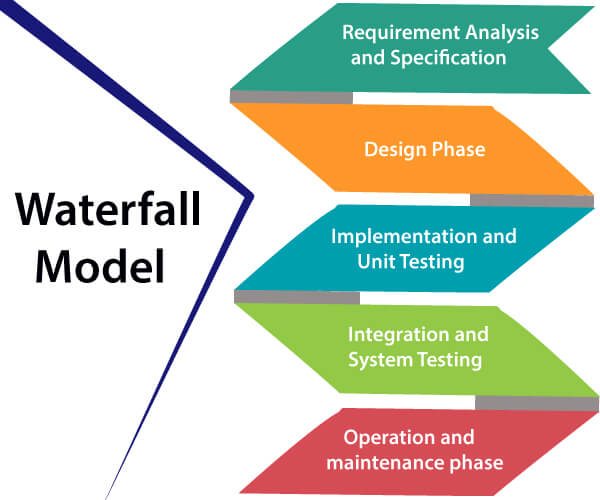
**Software Requirements:** Largely software requirements must be categorized into two categories:

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.

**Waterfall model**

Winston Royce introduced the Waterfall Model in 1970.This model has five phases: Requirements analysis and specification, design, implementation, and unit testing, integration and system testing, and operation and maintenance. The steps always follow in this order and do not overlap. The developer must complete every phase before the next phase begins. This model is named "**Waterfall Model**", because its diagrammatic representation resembles a cascade of waterfalls.

**1. Requirements analysis and specification phase:** The aim of this phase is to understand the exact requirements of the customer and to document them properly. Both the customer and the software developer work together so as to document all the functions, performance, and interfacing requirement of the software. It describes the "what" of the system to be produced and not "how."In this phase, a large document called **Software Requirement Specification (SRS)** document is created which contained a detailed description of what the system will do in the common language.



**2. Design Phase:** This phase aims to transform the requirements gathered in the SRS into a suitable form which permits further coding in a programming language. It defines the overall software architecture together with high level and detailed design. All this work is documented as a Software Design Document (SDD).

**3. Implementation and unit testing:** During this phase, design is implemented. If the SDD is complete, the implementation or coding phase proceeds smoothly, because all the information needed by software developers is contained in the SDD.

During testing, the code is thoroughly examined and modified. Small modules are tested in isolation initially. After that these modules are tested by writing some overhead code to check the interaction between these modules and the flow of intermediate output.

**4. Integration and System Testing:** This phase is highly crucial as the quality of the end product is determined by the effectiveness of the testing carried out. The better output will lead to satisfied customers, lower maintenance costs, and accurate results. Unit testing determines the efficiency of individual modules. However, in this phase, the modules are tested for their interactions with each other and with the system.

**5. Operation and maintenance phase:** Maintenance is the task performed by every user once the software has been delivered to the customer, installed, and operational.

**When to use SDLC Waterfall Model?**

Some Circumstances where the use of the Waterfall model is most suited are:

* When the requirements are constant and not changed regularly.
* A project is short
* The situation is calm
* Where the tools and technology used is consistent and is not changing
* When resources are well prepared and are available to use.

**Advantages of Waterfall model**

* This model is simple to implement also the number of resources that are required for it is minimal.
* The requirements are simple and explicitly declared; they remain unchanged during the entire project development.
* The start and end points for each phase is fixed, which makes it easy to cover progress.
* The release date for the complete product, as well as its final cost, can be determined before development.
* It gives easy to control and clarity for the customer due to a strict reporting system.

**Disadvantages of Waterfall model**

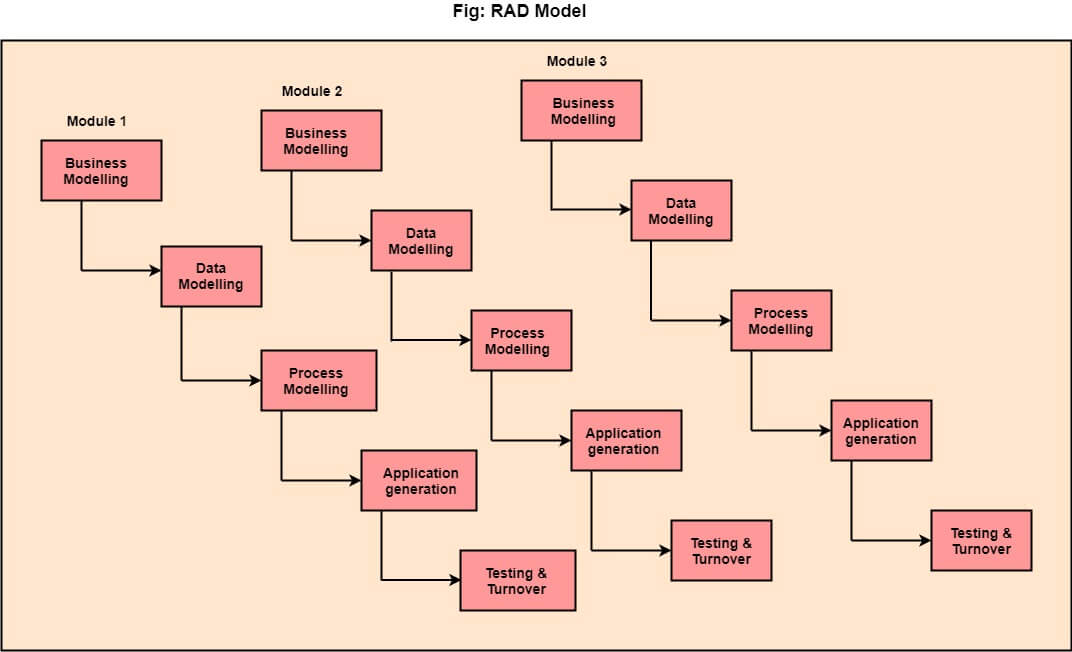
* In this model, the risk factor is higher, so this model is not suitable for more significant and complex projects.
* This model cannot accept the changes in requirements during development.
* It becomes tough to go back to the phase. For example, if the application has now shifted to the coding phase, and there is a change in requirement, It becomes tough to go back and change it.
* Since the testing done at a later stage, it does not allow identifying the challenges and risks in the earlier phase, so the risk reduction strategy is difficult to prepare.

**RAD (Rapid Application Development) Model**

RAD is a linear sequential software development process model that emphasizes a concise development cycle using an element based construction approach. If the requirements are well understood and described, and the project scope is a constraint, the RAD process enables a development team to create a fully functional system within a concise time period.

RAD (Rapid Application Development) is a concept that products can be developed faster and of higher quality through:

* Gathering requirements using workshops or focus groups
* Prototyping and early, reiterative user testing of designs
* The re-use of software components
* A rigidly paced schedule that refers design improvements to the next product version
* Less formality in reviews and other team communication



**The various phases of RAD are as follows:**

**1.Business Modelling:** The information flow among business functions is defined by answering questions like what data drives the business process, what data is generated, who generates it, where does the information go, who process it and so on.

**2. Data Modelling:** The data collected from business modeling is refined into a set of data objects (entities) that are needed to support the business. The attributes (character of each entity) are identified, and the relation between these data objects (entities) is defined.

**3. Process Modelling:** The information object defined in the data modeling phase are transformed to achieve the data flow necessary to implement a business function. Processing descriptions are created for adding, modifying, deleting, or retrieving a data object.

**4. Application Generation:** Automated tools are used to facilitate construction of the software; even they use the 4th GL techniques.

**5. Testing & Turnover:** Many of the programming components have already been tested since RAD emphasis reuse. This reduces the overall testing time. But the new part must be tested, and all interfaces must be fully exercised.

**When to use RAD Model?**

* When the system should need to create the project that modularizes in a short span time (2-3 months).
* When the requirements are well-known.
* When the technical risk is limited.
* When there's a necessity to make a system, which modularized in 2-3 months of period.
* It should be used only if the budget allows the use of automatic code generating tools.

**Advantage of RAD Model**

* This model is flexible for change.
* In this model, changes are adoptable.
* Each phase in RAD brings highest priority functionality to the customer.
* It reduced development time.
* It increases the reusability of features.

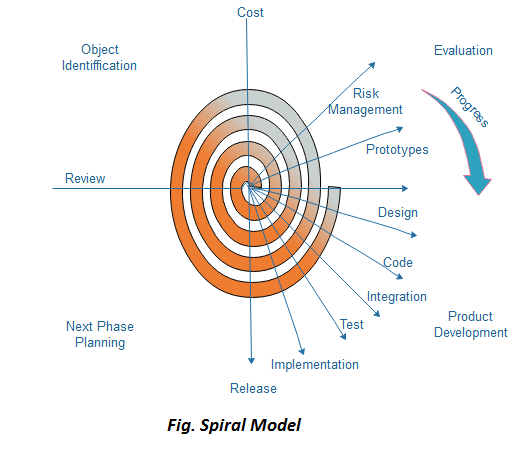
**Disadvantage of RAD Model**

* It required highly skilled designers.
* All application is not compatible with RAD.
* For smaller projects, we cannot use the RAD model.
* On the high technical risk, it's not suitable.
* Required user involvement.

**Spiral Model**

The spiral model, initially proposed by Boehm, is an evolutionary software process model that couples the iterative feature of prototyping with the controlled and systematic aspects of the linear sequential model. It implements the potential for rapid development of new versions of the software. Using the spiral model, the software is developed in a series of incremental releases. During the early iterations, the additional release may be a paper model or prototype. During later iterations, more and more complete versions of the engineered system are produced.

**The Spiral Model is shown in fig:**



**Each cycle in the spiral is divided into four parts:**

**Objective setting:** Each cycle in the spiral starts with the identification of purpose for that cycle, the various alternatives that are possible for achieving the targets, and the constraints that exists.

**Risk Assessment and reduction:** The next phase in the cycle is to calculate these various alternatives based on the goals and constraints. The focus of evaluation in this stage is located on the risk perception for the project.

**Development and validation:** The next phase is to develop strategies that resolve uncertainties and risks. This process may include activities such as benchmarking, simulation, and prototyping.

**Planning:** Finally, the next step is planned. The project is reviewed, and a choice made whether to continue with a further period of the spiral. If it is determined to keep, plans are drawn up for the next step of the project.

The development phase depends on the remaining risks. For example, if performance or user-interface risks are treated more essential than the program development risks, the next phase may be an evolutionary development that includes developing a more detailed prototype for solving the risks.

The **risk-driven** feature of the spiral model allows it to accommodate any mixture of a specification-oriented, prototype-oriented, simulation-oriented, or another type of approach. An essential element of the model is that each period of the spiral is completed by a review that includes all the products developed during that cycle, including plans for the next cycle. The spiral model works for development as well as enhancement projects.

**When to use Spiral Model?**

* When deliverance is required to be frequent.
* When the project is large
* When requirements are unclear and complex
* When changes may require at any time
* Large and high budget projects

**Advantages**

* High amount of risk analysis
* Useful for large and mission-critical projects.

**Disadvantages**

* Can be a costly model to use.
* Risk analysis needed highly particular expertise
* Doesn't work well for smaller projects.

**V-Model**

V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.



**Verification:** It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

**Validation:** It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.

So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation process is joined by coding phase in V-shape. Thus it is known as V-Model.

**There are the various phases of Verification Phase of V-model:**

1. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

**There are the various phases of Validation Phase of V-model:**

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client?s business team. System Test ensures that expectations from an application developer are met.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

**When to use V-Model?**

* When the requirement is well defined and not ambiguous.
* The V-shaped model should be used for small to medium-sized projects where requirements are clearly defined and fixed.
* The V-shaped model should be chosen when sample technical resources are available with essential technical expertise.

**Advantage (Pros) of V-Model:**

1. Easy to Understand.
2. Testing Methods like planning, test designing happens well before coding.
3. This saves a lot of time. Hence a higher chance of success over the waterfall model.
4. Avoids the downward flow of the defects.
5. Works well for small plans where requirements are easily understood.

**Disadvantage (Cons) of V-Model:**

1. Very rigid and least flexible.
2. Not a good for a complex project.
3. Software is developed during the implementation stage, so no early prototypes of the software are produced.
4. If any changes happen in the midway, then the test documents along with the required documents, has to be updated.

# Agile Model

The meaning of Agile is swift or versatile."**Agile process model**" refers to a software development approach based on iterative development. Agile methods break tasks into smaller iterations, or parts do not directly involve long term planning. The project scope and requirements are laid down at the beginning of the development process. Plans regarding the number of iterations, the duration and the scope of each iteration are clearly defined in advance.

Each iteration is considered as a short time "frame" in the Agile process model, which typically lasts from one to four weeks. The division of the entire project into smaller parts helps to minimize the project risk and to reduce the overall project delivery time requirements. Each iteration involves a team working through a full software development life cycle including planning, requirements analysis, design, coding, and testing before a working product is demonstrated to the client.



## Phases of Agile Model:

Following are the phases in the Agile model are as follows:

1. Requirements gathering
2. Design the requirements
3. Construction/ iteration
4. Testing/ Quality assurance
5. Deployment
6. Feedback

**1. Requirements gathering:** In this phase, you must define the requirements. You should explain business opportunities and plan the time and effort needed to build the project. Based on this information, you can evaluate technical and economic feasibility.

**2. Design the requirements:** When you have identified the project, work with stakeholders to define requirements. You can use the user flow diagram or the high-level UML diagram to show the work of new features and show how it will apply to your existing system.

**3. Construction/ iteration:** When the team defines the requirements, the work begins. Designers and developers start working on their project, which aims to deploy a working product. The product will undergo various stages of improvement, so it includes simple, minimal functionality.

**4. Testing:** In this phase, the Quality Assurance team examines the product's performance and looks for the bug.

**5. Deployment:** In this phase, the team issues a product for the user's work environment.

**6. Feedback:** After releasing the product, the last step is feedback. In this, the team receives feedback about the product and works through the feedback.

## Agile Testing Methods:

* Scrum
* Crystal
* Dynamic Software Development Method(DSDM)
* Feature Driven Development(FDD)
* Lean Software Development
* eXtreme Programming(XP)

### Scrum

SCRUM is an agile development process focused primarily on ways to manage tasks in team-based development conditions.

There are three roles in it, and their responsibilities are:

* **Scrum Master:** The scrum can set up the master team, arrange the meeting and remove obstacles for the process
* **Product owner:** The product owner makes the product backlog, prioritizes the delay and is responsible for the distribution of functionality on each repetition.
* **Scrum Team:** The team manages its work and organizes the work to complete the sprint or cycle.

### eXtreme Programming(XP)

This type of methodology is used when customers are constantly changing demands or requirements, or when they are not sure about the system's performance.

### Crystal:

There are three concepts of this method-

1. Chartering: Multi activities are involved in this phase such as making a development team, performing feasibility analysis, developing plans, etc.
2. Cyclic delivery: under this, two more cycles consist, these are:
   * Team updates the release plan.
   * Integrated product delivers to the users.
3. Wrap up: According to the user environment, this phase performs deployment, post-deployment.

### Dynamic Software Development Method(DSDM):

DSDM is a rapid application development strategy for software development and gives an agile project distribution structure. The essential features of DSDM are that users must be actively connected, and teams have been given the right to make decisions. The techniques used in DSDM are:

1. Time Boxing
2. MoSCoW Rules
3. Prototyping

**The DSDM project contains seven stages:**

1. Pre-project
2. Feasibility Study
3. Business Study
4. Functional Model Iteration
5. Design and build Iteration
6. Implementation
7. Post-project

### Feature Driven Development(FDD):

This method focuses on "Designing and Building" features. In contrast to other smart methods, FDD describes the small steps of the work that should be obtained separately per function.

### Lean Software Development:

Lean software development methodology follows the principle "just in time production." The lean method indicates the increasing speed of software development and reducing costs. Lean development can be summarized in seven phases.

1. Eliminating Waste
2. Amplifying learning
3. Defer commitment (deciding as late as possible)
4. Early delivery
5. Empowering the team
6. Building Integrity
7. Optimize the whole

## When to use the Agile Model?

* When frequent changes are required.
* When a highly qualified and experienced team is available.
* When a customer is ready to have a meeting with a software team all the time.
* When project size is small.

## Advantage(Pros) of Agile Method:

1. Frequent Delivery
2. Face-to-Face Communication with clients.
3. Efficient design and fulfils the business requirement.
4. Anytime changes are acceptable.
5. It reduces total development time.

## Disadvantages(Cons) of Agile Model:

1. Due to the shortage of formal documents, it creates confusion and crucial decisions taken throughout various phases can be misinterpreted at any time by different team members.
2. Due to the lack of proper documentation, once the project completes and the developers allotted to another project, maintenance of the finished project can become a difficulty.

# Iterative Model

In this Model, you can start with some of the software specifications and develop the first version of the software. After the first version if there is a need to change the software, then a new version of the software is created with a new iteration. Every release of the Iterative Model finishes in an exact and fixed period that is called iteration.

The Iterative Model allows the accessing earlier phases, in which the variations made respectively. The final output of the project renewed at the end of the Software Development Life Cycle (SDLC) process.



### The various phases of Iterative model are as follows:

**1. Requirement gathering & analysis:** In this phase, requirements are gathered from customers and check by an analyst whether requirements will fulfil or not. Analyst checks that need will achieve within budget or not. After all of this, the software team skips to the next phase.

**2. Design:** In the design phase, team design the software by the different diagrams like Data Flow diagram, activity diagram, class diagram, state transition diagram, etc.

**3. Implementation:** In the implementation, requirements are written in the coding language and transformed into computer programmes which are called Software.

**4. Testing:** After completing the coding phase, software testing starts using different test methods. There are many test methods, but the most common are white box, black box, and grey box test methods.

**5. Deployment:** After completing all the phases, software is deployed to its work environment.

**6. Review:** In this phase, after the product deployment, review phase is performed to check the behaviour and validity of the developed product. And if there are any error found then the process starts again from the requirement gathering.

**7. Maintenance:** In the maintenance phase, after deployment of the software in the working environment there may be some bugs, some errors or new updates are required. Maintenance involves debugging and new addition options.

## When to use the Iterative Model?

1. When requirements are defined clearly and easy to understand.
2. When the software application is large.
3. When there is a requirement of changes in future.

## Advantage(Pros) of Iterative Model:

1. Testing and debugging during smaller iteration is easy.
2. A Parallel development can plan.
3. It is easily acceptable to ever-changing needs of the project.
4. Risks are identified and resolved during iteration.
5. Limited time spent on documentation and extra time on designing.

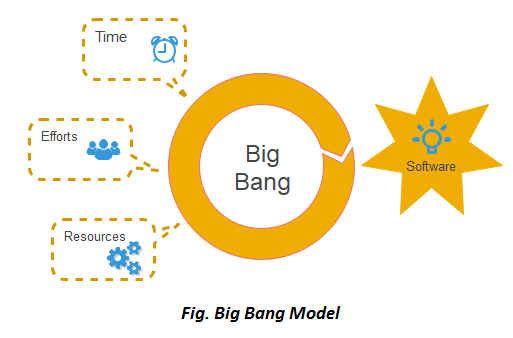
## Disadvantage(Cons) of Iterative Model:

1. It is not suitable for smaller projects.
2. More Resources may be required.
3. Design can be changed again and again because of imperfect requirements.
4. Requirement changes can cause over budget.
5. Project completion date not confirmed because of changing requirements.

**Big Bang Model**

In this model, developers do not follow any specific process. Development begins with the necessary funds and efforts in the form of inputs. And the result may or may not be as per the customer's requirement, because in this model, even the customer requirements are not defined.

This model is ideal for small projects like academic projects or practical projects. One or two developers can work together on this model.



**When to use Big Bang Model?**

As we discussed above, this model is required when this project is small like an academic project or a practical project. This method is also used when the size of the developer team is small and when requirements are not defined, and the release date is not confirmed or given by the customer.

**Advantage(Pros) of Big Bang Model:**

1. There is no planning required.
2. Simple Model.
3. Few resources required.
4. Easy to manage.
5. Flexible for developers.

**Disadvantage(Cons) of Big Bang Model:**

1. There are high risk and uncertainty.
2. Not acceptable for a large project.
3. If requirements are not clear that can cause very expensive.

**Prototype Model**

The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system. A prototype usually turns out to be a very crude version of the actual system, possible exhibiting limited functional capabilities, low reliability, and inefficient performance as compared to actual software. In many instances, the client only has a general view of what is expected from the software product. In such a scenario where there is an absence of detailed information regarding the input to the system, the processing needs, and the output requirement, the prototyping model may be employed.



**Steps of Prototype Model**

1. Requirement Gathering and Analyst
2. Quick Decision
3. Build a Prototype
4. Assessment or User Evaluation
5. Prototype Refinement
6. Engineer Product

**Advantage of Prototype Model**

1. Reduce the risk of incorrect user requirement
2. Good where requirement are changing/uncommitted
3. Regular visible process aids management
4. Support early product marketing
5. Reduce Maintenance cost.
6. Errors can be detected much earlier as the system is made side by side.

**Disadvantage of Prototype Model**

1. An unstable/badly implemented prototype often becomes the final product.
2. Require extensive customer collaboration
   * Costs customer money
   * Needs committed customer
   * Difficult to finish if customer withdraw
   * May be too customer specific, no broad market
3. Difficult to know how long the project will last.
4. Easy to fall back into the code and fix without proper requirement analysis, design, customer evaluation, and feedback.
5. Prototyping tools are expensive.
6. Special tools & techniques are required to build a prototype.
7. It is a time-consuming process.

**Evolutionary Process Model**

Evolutionary process model resembles the iterative enhancement model. The same phases are defined for the waterfall model occurs here in a cyclical fashion. This model differs from the iterative enhancement model in the sense that this does not require a useful product at the end of each cycle. In evolutionary development, requirements are implemented by category rather than by priority.

For example, in a simple database application, one cycle might implement the graphical user Interface (GUI), another file manipulation, another queries and another updates. All four cycles must complete before there is a working product available. GUI allows the users to interact with the system, file manipulation allow the data to be saved and retrieved, queries allow user to get out of the system, and updates allows users to put data into the system.

**Benefits of Evolutionary Process Model**

Use of EVO brings a significant reduction in risk for software projects.

EVO can reduce costs by providing a structured, disciplined avenue for experimentation.

EVO allows the marketing department access to early deliveries, facilitating the development of documentation and demonstration.

Better fit the product to user needs and market requirements.

Manage project risk with the definition of early cycle content.

Uncover key issues early and focus attention appropriately.

Increase the opportunity to hit market windows.

Accelerate sales cycles with early customer exposure.

Increase management visibility of project progress.

Increase product team productivity and motivations.

Software Project Management

# What is Project?

A project is a group of tasks that need to complete to reach a clear result. A project also defines as a set of inputs and outputs which are required to achieve a goal. Projects can vary from simple to difficult and can be operated by one person or a hundred.

Projects usually described and approved by a project manager or team executive. They go beyond their expectations and objects, and it's up to the team to handle logistics and complete the project on time. For good project development, some teams split the project into specific tasks so they can manage responsibility and utilize team strengths.

## What is software project management?

Software project management is an art and discipline of planning and supervising software projects. It is a sub-discipline of software project management in which software projects planned, implemented, monitored and controlled.

It is a procedure of managing, allocating and timing resources to develop computer software that fulfills requirements.

In software Project Management, the client and the developers need to know the length, period and cost of the project.

### Prerequisite of software project management?

There are three needs for software project management. These are:

1. Time
2. Cost
3. Quality

It is an essential part of the software organization to deliver a quality product, keeping the cost within the client?s budget and deliver the project as per schedule. There are various factors, both external and internal, which may impact this triple factor. Any of three-factor can severely affect the other two.

## Project Manager

A project manager is a character who has the overall responsibility for the planning, design, execution, monitoring, controlling and closure of a project. A project manager represents an essential role in the achievement of the projects.

A project manager is a character who is responsible for giving decisions, both large and small projects. The project manager is used to manage the risk and minimize uncertainty. Every decision the project manager makes must directly profit their project.

### Role of a Project Manager:

**1. Leader**

A project manager must lead his team and should provide them direction to make them understand what is expected from all of them.

**2. Medium:**

The Project manager is a medium between his clients and his team. He must coordinate and transfer all the appropriate information from the clients to his team and report to the senior management.

**3. Mentor:**

He should be there to guide his team at each step and make sure that the team has an attachment. He provides a recommendation to his team and points them in the right direction.

### Responsibilities of a Project Manager:

1. Managing risks and issues.
2. Create the project team and assigns tasks to several team members.
3. Activity planning and sequencing.
4. Monitoring and reporting progress.
5. Modifies the project plan to deal with the situation.

**Activities**

Software Project Management consists of many activities, that includes planning of the project, deciding the scope of product, estimation of cost in different terms, scheduling of tasks, etc.

**The list of activities are as follows:**

1. Project planning and Tracking
2. Project Resource Management
3. Scope Management
4. Estimation Management
5. Project Risk Management
6. Scheduling Management
7. Project Communication Management
8. Configuration Management

Now we will discuss all these activities -

**1. Project Planning:** It is a set of multiple processes, or we can say that it a task that performed before the construction of the product starts.

**2. Scope Management:** It describes the scope of the project. Scope management is important because it clearly defines what would do and what would not. Scope Management create the project to contain restricted and quantitative tasks, which may merely be documented and successively avoids price and time overrun.

**3. Estimation management:** This is not only about cost estimation because whenever we start to develop software, but we also figure out their size(line of code), efforts, time as well as cost.

If we talk about the size, then Line of code depends upon user or software requirement.

If we talk about effort, we should know about the size of the software, because based on the size we can quickly estimate how big team required to produce the software.

If we talk about time, when size and efforts are estimated, the time required to develop the software can easily determine.

And if we talk about cost, it includes all the elements such as:

* Size of software
* Quality
* Hardware
* Communication
* Training
* Additional Software and tools
* Skilled manpower

**4. Scheduling Management:** Scheduling Management in software refers to all the activities to complete in the specified order and within time slotted to each activity. Project managers define multiple tasks and arrange them keeping various factors in mind.

**For scheduling, it is compulsory -**

* Find out multiple tasks and correlate them.
* Divide time into units.
* Assign the respective number of work-units for every job.
* Calculate the total time from start to finish.
* Break down the project into modules.

**5. Project Resource Management:** In software Development, all the elements are referred to as resources for the project. It can be a human resource, productive tools, and libraries.

Resource management includes:

* Create a project team and assign responsibilities to every team member
* Developing a resource plan is derived from the project plan.
* Adjustment of resources.

**6. Project Risk Management:** Risk management consists of all the activities like identification, analyzing and preparing the plan for predictable and unpredictable risk in the project.

Several points show the risks in the project:

* The Experienced team leaves the project, and the new team joins it.
* Changes in requirement.
* Change in technologies and the environment.
* Market competition.

**7. Project Communication Management:** Communication is an essential factor in the success of the project. It is a bridge between client, organization, team members and as well as other stakeholders of the project such as hardware suppliers.

From the planning to closure, communication plays a vital role. In all the phases, communication must be clear and understood. Miscommunication can create a big blunder in the project.

**8. Project Configuration Management:** Configuration management is about to control the changes in software like requirements, design, and development of the product.

The Primary goal is to increase productivity with fewer errors.

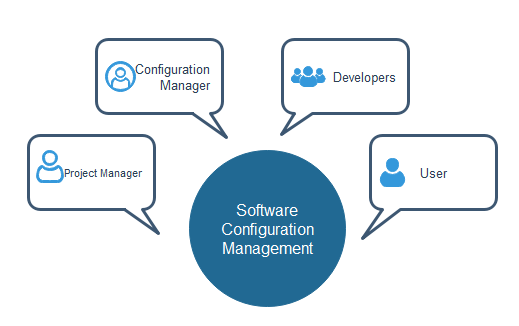
**Some reasons show the need for configuration management:**

* Several people work on software that is continually update.
* Help to build coordination among suppliers.
* Changes in requirement, budget, schedule need to accommodate.
* Software should run on multiple systems.

**Tasks perform in Configuration management:**

* Identification
* Baseline
* Change Control
* Configuration Status Accounting
* Configuration Audits and Reviews

**People involved in Configuration Management:**



# Project Management Tools

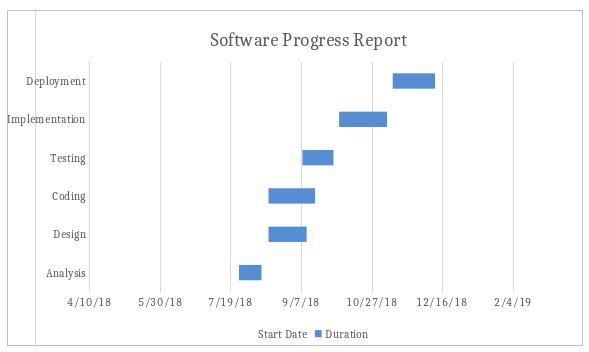
To manage the Project management system adequately and efficiently, we use Project management tools.

**Here are some standard tools:**

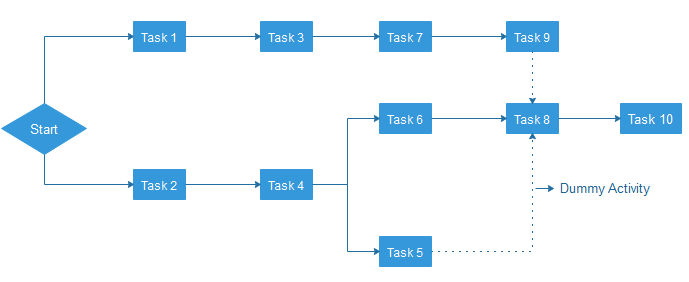
## Gantt chart

Gantt Chart first developed by Henry Gantt in 1917. Gantt chart usually utilized in project management, and it is one of the most popular and helpful ways of showing activities displayed against time. Each activity represented by a bar.

Gantt chart is a useful tool when you want to see the entire landscape of either one or multiple projects. It helps you to view which tasks are dependent on one another and which event is coming up.



## PERT chart



PERT is an acronym of Programme Evaluation Review Technique. In the 1950s, it is developed by the U.S. Navy to handle the Polaris submarine missile programme.

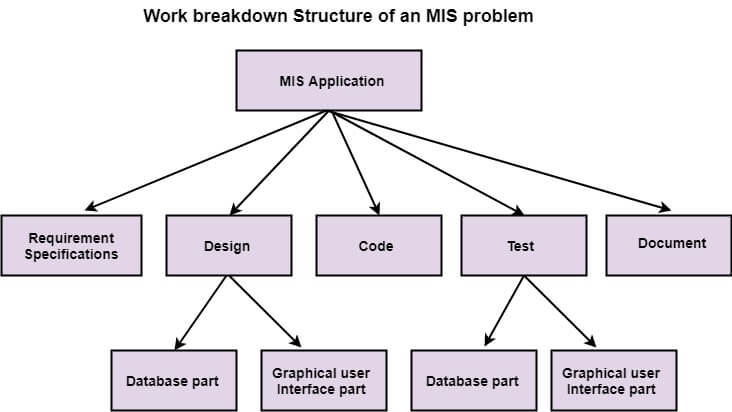
In Project Management, PERT chart represented as a network diagram concerning the number of nodes, which represents events.

The direction of the lines indicates the sequence of the task. In the above example, tasks between "Task 1 to Task 9" must complete, and these are known as a dependent or serial task. Between Task 4 and 5, and Task 4 and 6, nodes are not depended and can undertake simultaneously. These are known as Parallel or concurrent tasks. Without resource or completion time, the task must complete in the sequence which is considered as event dependency, and these are known as Dummy activity and represented by dotted lines.

## Logic Network

The Logic Network shows the order of activities over time. It shows the sequence in which activities are to do. Distinguishing events and pinning down the project are the two primary uses. Moreover, it will help with understanding task dependencies, a timescale, and overall project workflow.

## Product Breakdown Structure



Product Breakdown Structure (BBS) is a management tool and necessary a part of the project designing. It's a task-oriented system for subdividing a project into product parts. The product breakdown structure describes subtasks or work packages and represents the connection between work packages. Within the product breakdown Structure, the project work has diagrammatically pictured with various types of lists. The product breakdown structure is just like the work breakdown structure (WBS).

## Work Breakdown Structure

It is an important project deliverable that classifies the team's work into flexible segments. "Project Management Body of Knowledge (PMBOK)" is a group of terminology that describes the work breakdown structure as a "deliverable-oriented hierarchical breakdown of the work which is performed by the project team."

There are two ways to generate a Work Breakdown Structure ? The top-down and

The bottom-up approach.

In the **top-down approach**, the WBS derived by crumbling the overall project into subprojects or lower-level tasks.

The **bottom-up approach** is more alike to a brainstorming exercise where team members are asked to make a list of low-level tasks which is required to complete the project.

## Resource Histogram

The resource histogram is precisely a bar chart that used for displaying the amounts of time that a resource is scheduled to be worked on over a prearranged and specific period. Resource histograms can also contain the related feature of resource availability, used for comparison on purposes of contrast.

## Critical Path Analysis

Critical path analysis is a technique that is used to categorize the activities which are required to complete a task, as well as classifying the time which is needed to finish each activity and the relationships between the activities. It is also called a critical path method. CPA helps in predicting whether a project will expire on time.

**Functional Point (FP) Analysis**

Allan J. Albrecht initially developed function Point Analysis in 1979 at IBM and it has been further modified by the International Function Point Users Group (IFPUG). FPA is used to make estimate of the software project, including its testing in terms of functionality or function size of the software product. However, functional point analysis may be used for the test estimation of the product. The functional size of the product is measured in terms of the function point, which is a standard of measurement to measure the software application.

**Objectives of FPA**

The basic and primary purpose of the functional point analysis is to measure and provide the software application functional size to the client, customer, and the stakeholder on their request. Further, it is used to measure the software project development along with its maintenance, consistently throughout the project irrespective of the tools and the technologies.

**Following are the points regarding FPs**

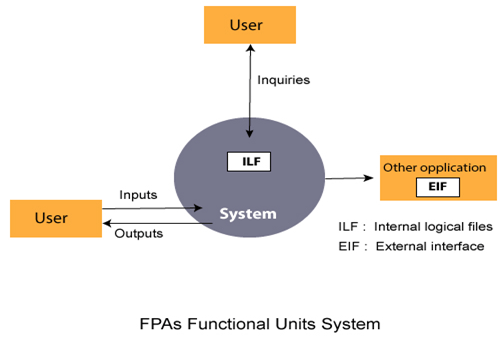
1. FPs of an application is found out by counting the number and types of functions used in the applications. Various functions used in an application can be put under five types, as shown in Table:

**Types of FP Attributes**

|  |  |
| --- | --- |
| **Measurements Parameters** | **Examples** |
| 1.Number of External Inputs(EI) | Input screen and tables |
| 2. Number of External Output (EO) | Output screens and reports |
| 3. Number of external inquiries (EQ) | Prompts and interrupts. |
| 4. Number of internal files (ILF) | Databases and directories |
| 5. Number of external interfaces (EIF) | Shared databases and shared routines. |

All these parameters are then individually assessed for complexity.

**The FPA functional units are shown in Fig:**



2. FP characterizes the complexity of the software system and hence can be used to depict the project time and the manpower requirement.

3. The effort required to develop the project depends on what the software does.

4. FP is programming language independent.

5. FP method is used for data processing systems, business systems like information systems.

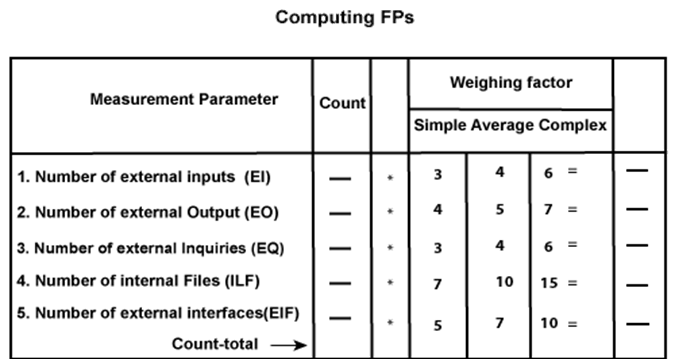
6. The five parameters mentioned above are also known as information domain characteristics.

7. All the parameters mentioned above are assigned some weights that have been experimentally determined and are shown in Table

**Weights of 5-FP Attributes**

|  |  |  |  |
| --- | --- | --- | --- |
| **Measurement Parameter** | **Low** | **Average** | **High** |
| 1. Number of external inputs (EI) | 7 | 10 | 15 |
| 2. Number of external outputs (EO) | 5 | 7 | 10 |
| 3. Number of external inquiries (EQ) | 3 | 4 | 6 |
| 4. Number of internal files (ILF) | 4 | 5 | 7 |
| 5. Number of external interfaces (EIF) | 3 | 4 | 6 |

The functional complexities are multiplied with the corresponding weights against each function, and the values are added up to determine the UFP (Unadjusted Function Point) of the subsystem.



Here that weighing factor will be simple, average, or complex for a measurement parameter type.

The Function Point (FP) is thus calculated with the following formula.

**FP = Count-total \* [0.65 + 0.01 \* ∑(fi)]**  
              **= Count-total \* CAF**

where Count-total is obtained from the above Table.

**CAF = [0.65 + 0.01 \*∑(fi)]**

and **∑(fi)** is the sum of all 14 questionnaires and show the complexity adjustment value/ factor-CAF (where i ranges from 1 to 14). Usually, a student is provided with the value of ∑(fi)

Also note that **∑(fi)** ranges from 0 to 70, i.e.,

              0 <= ∑(fi) <=70

and CAF ranges from 0.65 to 1.35 because

1. When **∑(fi)** = 0 then CAF = 0.65
2. When **∑(fi)** = 70 then CAF = 0.65 + (0.01 \* 70) = 0.65 + 0.7 = 1.35

Based on the FP measure of software many other metrics can be computed:

1. Errors/FP
2. $/FP.
3. Defects/FP
4. Pages of documentation/FP
5. Errors/PM.
6. Productivity = FP/PM (effort is measured in person-months).
7. $/Page of Documentation.

8. LOCs of an application can be estimated from FPs. That is, they are interconvertible. **This process is known as backfiring**. For example, 1 FP is equal to about 100 lines of COBOL code.

9. FP metrics is used mostly for measuring the size of Management Information System (MIS) software.

10. But the function points obtained above are unadjusted function points (UFPs). These (UFPs) of a subsystem are further adjusted by considering some more General System Characteristics (GSCs). It is a set of 14 GSCs that need to be considered. The procedure for adjusting UFPs is as follows:

1. Degree of Influence (DI) for each of these 14 GSCs is assessed on a scale of 0 to 5. (b) If a particular GSC has no influence, then its weight is taken as 0 and if it has a strong influence then its weight is 5.
2. The score of all 14 GSCs is totaled to determine Total Degree of Influence (TDI).
3. Then Value Adjustment Factor (VAF) is computed from TDI by using the formula: **VAF = (TDI \* 0.01) + 0.65**

Remember that the value of VAF lies within 0.65 to 1.35 because

1. When TDI = 0, VAF = 0.65
2. When TDI = 70, VAF = 1.35
3. VAF is then multiplied with the UFP to get the final FP count: **FP = VAF \* UFP**

**Example:** Compute the function point, productivity, documentation, cost per function for the following data:

1. Number of user inputs = 24
2. Number of user outputs = 46
3. Number of inquiries = 8
4. Number of files = 4
5. Number of external interfaces = 2
6. Effort = 36.9 p-m
7. Technical documents = 265 pages
8. User documents = 122 pages
9. Cost = $7744/ month

Various processing complexity factors are: 4, 1, 0, 3, 3, 5, 4, 4, 3, 3, 2, 2, 4, 5.

**Solution:**

|  |  |  |  |
| --- | --- | --- | --- |
| **Measurement Parameter** | **Count** |  | **Weighing factor** |
| 1. Number of external inputs (EI) | 24 | \* | 4 = 96 |
| 2. Number of external outputs (EO) | 46 | \* | 4 = 184 |
| 3. Number of external inquiries (EQ) | 8 | \* | 6 = 48 |
| 4. Number of internal files (ILF) | 4 | \* | 10 = 40 |
| 5. Number of external interfaces (EIF) Count-total → | 2 | \* | 5 = 10  378 |

So sum of all fi (i ← 1 to 14) = 4 + 1 + 0 + 3 + 5 + 4 + 4 + 3 + 3 + 2 + 2 + 4 + 5 = 43

                FP = Count-total \* [0.65 + 0.01 \*∑(fi)]  
                = 378 \* [0.65 + 0.01 \* 43]  
                = 378 \* [0.65 + 0.43]  
                = 378 \* 1.08 = 408

Functional Point (FP) Analysis

Total pages of documentation = technical document + user document  
                = 265 + 122 = 387pages

Documentation = Pages of documentation/FP  
                = 387/408 = 0.94



**Differentiate between FP and LOC**

|  |  |
| --- | --- |
| **FP** | **LOC** |
| 1. FP is specification based. | 1. LOC is an analogy based. |
| 2. FP is language independent. | 2. LOC is language dependent. |
| 3. FP is user-oriented. | 3. LOC is design-oriented. |
| 4. It is extendible to LOC. | 4. It is convertible to FP (backfiring) |

**Software Project Planning**

A Software Project is the complete methodology of programming advancement from requirement gathering to testing and support, completed by the execution procedures, in a specified period to achieve intended software product.

**Need of Software Project Management**

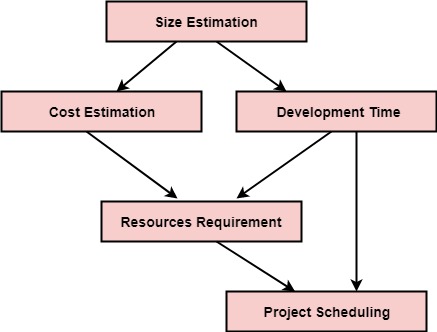
Software development is a sort of all new streams in world business, and there's next to no involvement in structure programming items. Most programming items are customized to accommodate customer's necessities. The most significant is that the underlying technology changes and advances so generally and rapidly that experience of one element may not be connected to the other one. All such business and ecological imperatives bring risk in software development; hence, it is fundamental to manage software projects efficiently.

**Software Project Manager**

Software manager is responsible for planning and scheduling project development. They manage the work to ensure that it is completed to the required standard. They monitor the progress to check that the event is on time and within budget. The project planning must incorporate the major issues like size & cost estimation scheduling, project monitoring, personnel selection evaluation & risk management. To plan a successful software project, we must understand:

* Scope of work to be completed
* Risk analysis
* The resources mandatory
* The project to be accomplished
* Record of being followed

Software Project planning starts before technical work start. The various steps of planning activities are:



The size is the crucial parameter for the estimation of other activities. Resources requirement are required based on cost and development time. Project schedule may prove to be very useful for controlling and monitoring the progress of the project. This is dependent on resources & development time.

**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.

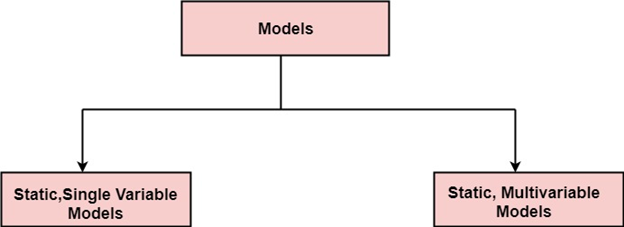
1. Project scope must be established in advanced.
2. Software metrics are used as a support from which evaluation is made.
3. The project is broken into small PCs which are estimated individually.   
   To achieve true cost & schedule estimate, several option arise.
4. Delay estimation
5. Used symbol 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 variable 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 = Costs  
                L= size  
                a and b are constants

The Software Engineering Laboratory established a model called SEL model, for estimating its software production. This model is an example of the static, single variable model.

                E=1.4L0.93  
                DOC=30.4L0.90  
                D=4.6L0.26

**Where**    E= Efforts (Person Per Month)  
                DOC=Documentation (Number of Pages)  
                D = Duration (D, in months)  
                L = Number of Lines per code

**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.

WALSTON and FELIX develop the models at IBM provide the following equation gives a relationship between lines of source code and effort:

**E=5.2L0.91**

In the same manner duration of development is given by

**D=4.1L0.36**

The productivity index uses 29 variables which are found to be highly correlated productivity as follows:

Software Cost Estimation

Where **Wi** is the weight factor for the **ith**variable and **Xi={-1,0,+1}** the estimator gives **Xi**one of the values **-1, 0 or +1** depending on the variable decreases, has no effect or increases the productivity.

**Example:** Compare the Walston-Felix Model with the SEL model on a software development expected to involve 8 person-years of effort.

1. Calculate the number of lines of source code that can be produced.
2. Calculate the duration of the development.
3. Calculate the productivity in LOC/PY
4. Calculate the average manning

**Solution:**

The amount of manpower involved = 8PY=96persons-months

(a)Number of lines of source code can be obtained by reversing equation to give:

Software Cost Estimation

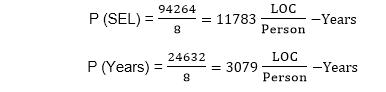
Then

                L (SEL) = (96/1.4)1⁄0.93=94264 LOC  
                L (SEL) = (96/5.2)1⁄0.91=24632 LOC

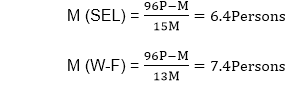
(b)Duration in months can be calculated by means of equation

                D (SEL) = 4.6 (L) 0.26  
                               = 4.6 (94.264)0.26 = 15 months  
                D (W-F) = 4.1 L0.36  
                               = 4.1 (24.632)0.36 = 13 months

(c) Productivity is the lines of code produced per persons/month (year)



(d)Average manning is the average number of persons required per month in the project



**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.

For three product categories, Bohem provides a different set of expression to predict effort (in a unit of person month)and development time from the size of estimation in KLOC(Kilo Line of code) efforts estimation takes into account the productivity loss due to holidays, weekly off, coffee breaks, etc.

According to Boehm, software cost estimation should be done through three stages:

1. Basic Model
2. Intermediate Model
3. Detailed Model

**1. Basic COCOMO Model:** The basic COCOMO model provide an accurate size of the project parameters. The following expressions give the basic COCOMO estimation model:

**Effort=a1\*(KLOC) a2 PM**  
                **Tdev=b1\*(efforts)b2 Months**

Where

**KLOC** is the estimated size of the software product indicate in Kilo Lines of Code,

a1,a2,b1,b2 are constants for each group of software products,

**Tdev** is the estimated time to develop the software, expressed in months,

**Effort** is the total effort required to develop the software product, expressed in **person months (PMs)**.

**Estimation of development effort**

For the three classes of software products, the formulas for estimating the effort based on the code size are shown below:

**Organic:** Effort = 2.4(KLOC) 1.05 PM

**Semi-detached:** Effort = 3.0(KLOC) 1.12 PM

**Embedded:** Effort = 3.6(KLOC) 1.20 PM

**Estimation of development time**

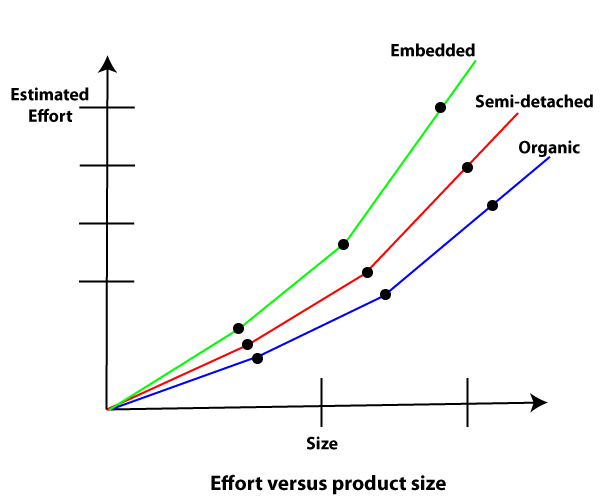
For the three classes of software products, the formulas for estimating the development time based on the effort are given below:

**Organic:** Tdev = 2.5(Effort) 0.38 Months

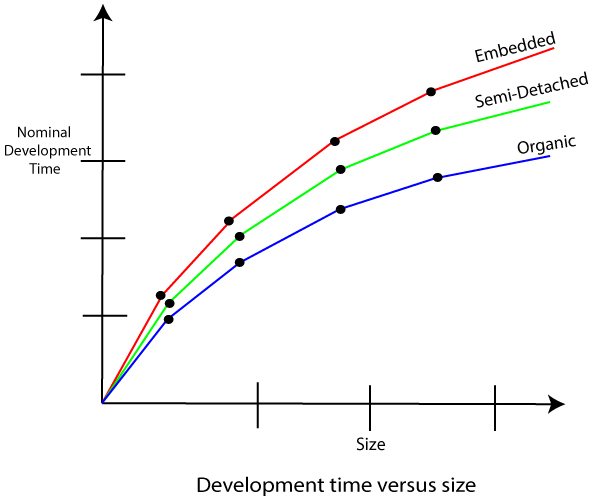
**Semi-detached:** Tdev = 2.5(Effort) 0.35 Months

**Embedded:** Tdev = 2.5(Effort) 0.32 Months

Some insight into the basic COCOMO model can be obtained by plotting the estimated characteristics for different software sizes. Fig shows a plot of estimated effort versus product size. From fig, we can observe that the effort is somewhat superliner in the size of the software product. Thus, the effort required to develop a product increases very rapidly with project size.



The development time versus the product size in KLOC is plotted in fig. From fig it can be observed that the development time is a sub linear function of the size of the product, i.e. when the size of the product increases by two times, the time to develop the product does not double but rises moderately. This can be explained by the fact that for larger products, a larger number of activities which can be carried out concurrently can be identified. The parallel activities can be carried out simultaneously by the engineers. This reduces the time to complete the project. Further, from fig, it can be observed that the development time is roughly the same for all three categories of products. For example, a 60 KLOC program can be developed in approximately 18 months, regardless of whether it is of organic, semidetached, or embedded type.



From the effort estimation, the project cost can be obtained by multiplying the required effort by the manpower cost per month. But, implicit in this project cost computation is the assumption that the entire project cost is incurred on account of the manpower cost alone. In addition to manpower cost, a project would incur costs due to hardware and software required for the project and the company overheads for administration, office space, etc.

It is important to note that the effort and the duration estimations obtained using the COCOMO model are called a nominal effort estimate and nominal duration estimate. The term nominal implies that if anyone tries to complete the project in a time shorter than the estimated duration, then the cost will increase drastically. But, if anyone completes the project over a longer period of time than the estimated, then there is almost no decrease in the estimated cost value.

**Example1:** Suppose a project was estimated to be 400 KLOC. Calculate the effort and development time for each of the three model i.e., organic, semi-detached & embedded.

**Solution:** The basic COCOMO equation takes the form:

                Effort=a1\*(KLOC) a2 PM  
                Tdev=b1\*(efforts)b2 Months  
                Estimated Size of project= 400 KLOC

**(i)Organic Mode**

                E = 2.4 \* (400)1.05 = 1295.31 PM  
                D = 2.5 \* (1295.31)0.38=38.07 PM

**(ii)Semidetached Mode**

                E = 3.0 \* (400)1.12=2462.79 PM  
                D = 2.5 \* (2462.79)0.35=38.45 PM

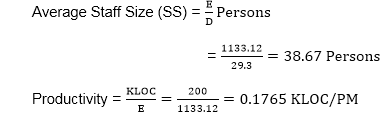
**(iii) Embedded Mode**

                E = 3.6 \* (400)1.20 = 4772.81 PM  
                D = 2.5 \* (4772.8)0.32 = 38 PM

**Example2:** A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of projects. The project schedule is not very tight. Calculate the Effort, development time, average staff size, and productivity of the project.

**Solution:** The semidetached mode is the most appropriate mode, keeping in view the size, schedule and experience of development time.

Hence       E=3.0(200)1.12=1133.12PM  
                D=2.5(1133.12)0.35=29.3PM



            P = 176 LOC/PM

**2. Intermediate Model:** The basic Cocomo model considers that the effort is only a function of the number of lines of code and some constants calculated according to the various software systems. The intermediate COCOMO model recognizes these facts and refines the initial estimates obtained through the basic COCOMO model by using a set of 15 cost drivers based on various attributes of software engineering.

**Classification of Cost Drivers and their attributes:**

**(i) Product attributes -**

* Required software reliability extent
* Size of the application database
* The complexity of the product

**Hardware attributes -**

* Run-time performance constraints
* Memory constraints
* The volatility of the virtual machine environment
* Required turnabout time

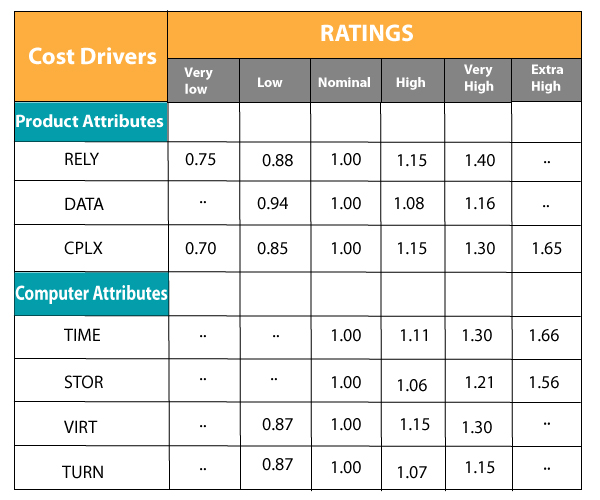
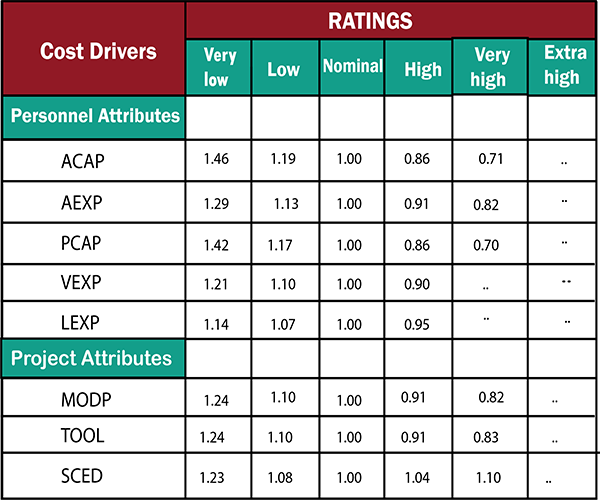
**Personnel attributes -**

* Analyst capability
* Software engineering capability
* Applications experience
* Virtual machine experience
* Programming language experience

**Project attributes -**

* Use of software tools
* Application of software engineering methods
* Required development schedule

**The cost drivers are divided into four categories:**

**Intermediate COCOMO equation:**

**E=ai (KLOC) bi\*EAF**  
                **D=ci (E)di**

Coefficients for intermediate COCOMO

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Project** | **ai** | **bi** | **ci** | **di** |
| Organic | 2.4 | 1.05 | 2.5 | 0.38 |
| Semidetached | 3.0 | 1.12 | 2.5 | 0.35 |
| Embedded | 3.6 | 1.20 | 2.5 | 0.32 |

**3. Detailed COCOMO Model:**Detailed COCOMO incorporates all qualities of the standard version with an assessment of the cost driver?s effect on each method of the software engineering process. The detailed model uses various effort multipliers for each cost driver property. In detailed cocomo, the whole software is differentiated into multiple modules, and then we apply COCOMO in various modules to estimate effort and then sum the effort.

The Six phases of detailed COCOMO are:

1. Planning and requirements
2. System structure
3. Complete structure
4. Module code and test
5. Integration and test
6. Cost Constructive model

The effort is determined as a function of program estimate, and a set of cost drivers are given according to every phase of the software lifecycle.

**What is Risk?**

"Tomorrow problems are today's risk." Hence, a clear definition of a "risk" is a problem that could cause some loss or threaten the progress of the project, but which has not happened yet.

These potential issues might harm cost, schedule or technical success of the project and the quality of our software device, or project team morale.

Risk Management is the system of identifying addressing and eliminating these problems before they can damage the project.

We need to differentiate risks, as potential issues, from the current problems of the project.

Different methods are required to address these two kinds of issues.

For example, staff storage, because we have not been able to select people with the right technical skills is a current problem, but the threat of our technical persons being hired away by the competition is a risk.

**Risk Management**

A software project can be concerned with a large variety of risks. In order to be adept to systematically identify the significant risks which might affect a software project, it is essential to classify risks into different classes. The project manager can then check which risks from each class are relevant to the project.

There are three main classifications of risks which can affect a software project:

1. Project risks
2. Technical risks
3. Business risks

**1. Project risks:** Project risks concern differ forms of budgetary, schedule, personnel, resource, and customer-related problems. A vital project risk is schedule slippage. Since the software is intangible, it is very tough to monitor and control a software project. It is very tough to control something which cannot be identified. For any manufacturing program, such as the manufacturing of cars, the plan executive can recognize the product taking shape.

**2. Technical risks:** Technical risks concern potential method, implementation, interfacing, testing, and maintenance issue. It also consists of an ambiguous specification, incomplete specification, changing specification, technical uncertainty, and technical obsolescence. Most technical risks appear due to the development team's insufficient knowledge about the project.

**3. Business risks:** This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

**Other risk categories**

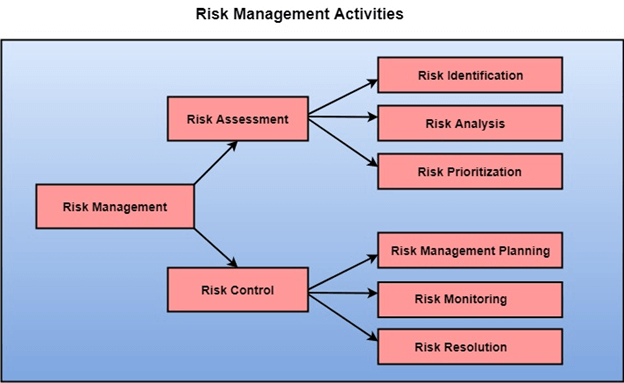
1. **1. Known risks:** Those risks that can be uncovered after careful assessment of the project program, the business and technical environment in which the plan is being developed, and more reliable data sources (e.g., unrealistic delivery date)
2. **2. Predictable risks:** Those risks that are hypothesized from previous project experience (e.g., past turnover)
3. **3. Unpredictable risks:** Those risks that can and do occur, but are extremely tough to identify in advance.

**Principle of Risk Management**

1. **Global Perspective:** In this, we review the bigger system description, design, and implementation. We look at the chance and the impact the risk is going to have.
2. **Take a forward-looking view:** Consider the threat which may appear in the future and create future plans for directing the next events.
3. **Open Communication:** This is to allow the free flow of communications between the client and the team members so that they have certainty about the risks.
4. **Integrated management:** In this method risk management is made an integral part of project management.
5. **Continuous process:** In this phase, the risks are tracked continuously throughout the risk management paradigm.

**Risk Management Activities**

**Risk management consists of three main activities, as shown in fig:**



**Risk Assessment**

The objective of risk assessment is to division the risks in the condition of their loss, causing potential. For risk assessment, first, every risk should be rated in two methods:

* The possibility of a risk coming true (denoted as r).
* The consequence of the issues relates to that risk (denoted as s).

Based on these two methods, the priority of each risk can be estimated:

                    p = r \* s

Where p is the priority with which the risk must be controlled, r is the probability of the risk becoming true, and s is the severity of loss caused due to the risk becoming true. If all identified risks are set up, then the most likely and damaging risks can be controlled first, and more comprehensive risk abatement methods can be designed for these risks.

**1. Risk Identification:** The project organizer needs to anticipate the risk in the project as early as possible so that the impact of risk can be reduced by making effective risk management planning.

A project can be of use by a large variety of risk. To identify the significant risk, this might affect a project. It is necessary to categories into the different risk of classes.

There are different types of risks which can affect a software project:

1. **Technology risks:** Risks that assume from the software or hardware technologies that are used to develop the system.
2. **People risks:** Risks that are connected with the person in the development team.
3. **Organizational risks:** Risks that assume from the organizational environment where the software is being developed.
4. **Tools risks:** Risks that assume from the software tools and other support software used to create the system.
5. **Requirement risks:** Risks that assume from the changes to the customer requirement and the process of managing the requirements change.
6. **Estimation risks:** Risks that assume from the management estimates of the resources required to build the system

**2. Risk Analysis:** During the risk analysis process, you have to consider every identified risk and make a perception of the probability and seriousness of that risk.

There is no simple way to do this. You have to rely on your perception and experience of previous projects and the problems that arise in them.

It is not possible to make an exact, the numerical estimate of the probability and seriousness of each risk. Instead, you should authorize the risk to one of several bands:

1. The probability of the risk might be determined as very low (0-10%), low (10-25%), moderate (25-50%), high (50-75%) or very high (+75%).
2. The effect of the risk might be determined as catastrophic (threaten the survival of the plan), serious (would cause significant delays), tolerable (delays are within allowed contingency), or insignificant.

**Risk Control**

It is the process of managing risks to achieve desired outcomes. After all, the identified risks of a plan are determined; the project must be made to include the most harmful and the most likely risks. Different risks need different containment methods. In fact, most risks need ingenuity on the part of the project manager in tackling the risk.

**There are three main methods to plan for risk management:**

1. **Avoid the risk:** This may take several ways such as discussing with the client to change the requirements to decrease the scope of the work, giving incentives to the engineers to avoid the risk of human resources turnover, etc.
2. **Transfer the risk:** This method involves getting the risky element developed by a third party, buying insurance cover, etc.
3. **Risk reduction:** This means planning method to include the loss due to risk. For instance, if there is a risk that some key personnel might leave, new recruitment can be planned.

**Risk Leverage:** To choose between the various methods of handling risk, the project plan must consider the amount of controlling the risk and the corresponding reduction of risk. For this, the risk leverage of the various risks can be estimated.

Risk leverage is the variation in risk exposure divided by the amount of reducing the risk.

**Risk leverage = (risk exposure before reduction - risk exposure after reduction) / (cost of reduction)**

**1. Risk planning:** The risk planning method considers each of the key risks that have been identified and develop ways to maintain these risks.

For each of the risks, you have to think of the behavior that you may take to minimize the disruption to the plan if the issue identified in the risk occurs.

You also should think about data that you might need to collect while monitoring the plan so that issues can be anticipated.

Again, there is no easy process that can be followed for contingency planning. It rely on the judgment and experience of the project manager.

**2. Risk Monitoring:** Risk monitoring is the method king that your assumption about the product, process, and business risks has not changed.

**Project Scheduling**

Project-task scheduling is a significant project planning activity. It comprises deciding which functions would be taken up when. To schedule the project plan, a software project manager wants to do the following:

1. Identify all the functions required to complete the project.
2. Break down large functions into small activities.
3. Determine the dependency among various activities.
4. Establish the most likely size for the time duration required to complete the activities.
5. Allocate resources to activities.
6. Plan the beginning and ending dates for different activities.
7. Determine the critical path. A critical way is the group of activities that decide the duration of the project.

The first method in scheduling a software plan involves identifying all the functions required to complete the project. A good judgment of the intricacies of the project and the development process helps the supervisor to identify the critical role of the project effectively. Next, the large functions are broken down into a valid set of small activities which would be assigned to various engineers. The work breakdown structure formalism supports the manager to breakdown the function systematically after the project manager has broken down the purpose and constructs the work breakdown structure; he has to find the dependency among the activities. Dependency among the various activities determines the order in which the various events would be carried out. If an activity A necessary the results of another activity B, then activity A must be scheduled after activity B. In general, the function dependencies describe a partial ordering among functions, i.e., each service may precede a subset of other functions, but some functions might not have any precedence ordering describe between them (called concurrent function). The dependency among the activities is defined in the pattern of an activity network.

Once the activity network representation has been processed out, resources are allocated to every activity. Resource allocation is usually done using a Gantt chart. After resource allocation is completed, a PERT chart representation is developed. The PERT chart representation is useful for program monitoring and control. For task scheduling, the project plan needs to decompose the project functions into a set of activities. The time frame when every activity is to be performed is to be determined. The end of every action is called a milestone. The project manager tracks the function of a project by audit the timely completion of the milestones. If he examines that the milestones start getting delayed, then he has to handle the activities carefully so that the complete deadline can still be met.

# Personnel Planning

Personnel Planning deals with staffing. Staffing deals with the appoint personnel for the position that is identified by the organizational structure.

It involves:

* Defining requirement for personnel
* Recruiting (identifying, interviewing, and selecting candidates)
* Compensating
* Developing and promoting agent

For personnel planning and scheduling, it is helpful to have efforts and schedule size for the subsystems and necessary component in the system.

At planning time, when the system method has not been completed, the planner can only think to know about the large subsystems in the system and possibly the major modules in these subsystems.

Once the project plan is estimated, and the effort and schedule of various phases and functions are known, staff requirements can be achieved.

From the cost and overall duration of the projects, the average staff size for the projects can be determined by dividing the total efforts (in person-months) by the whole project duration (in months).

Typically the staff required for the project is small during requirement and design, the maximum during implementation and testing, and drops again during the last stage of integration and testing.

Using the COCOMO model, average staff requirement for various phases can be calculated as the effort and schedule for each method are known.

When the schedule and average staff level for every action are well-known, the overall personnel allocation for the project can be planned.

This plan will indicate how many people will be required for different activities at different times for the duration of the project.

The total effort for each month and the total effort for each step can easily be calculated from this plan.

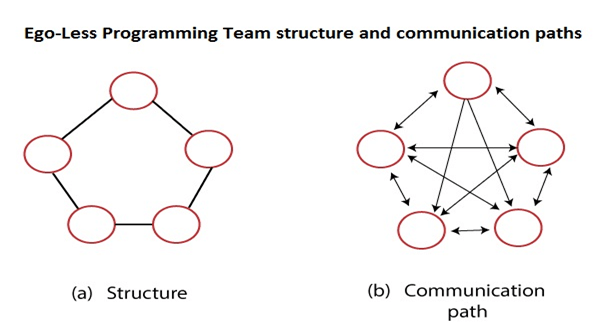
## Team Structure

Team structure addresses the issue of arrangement of the individual project teams. There are some possible methods in which the different project teams can be organized. There are primarily three formal team structures: **chief programmer, Ego-less or democratic, and the mixed team organizations** even several other variations to these structures are possible. Problems of various complexities and sizes often need different team structures for the chief solution.

## Ego-Less or Democratic Teams

Ego-Less teams subsist of a team of fewer programmers. The objective of the group is set by consensus, and input from each member is taken for significant decisions. Group leadership revolves among the group members. Due to its nature, egoless teams are consistently known as democratic teams.

The structure allows input from all representatives, which can lead to better decisions in various problems. This suggests that this method is well suited for long-term research-type projects that do not have time constraints.



## Chief Programmer Team

A chief-programmer team, in contrast to the ego-less team, has a hierarchy. It consists of a chief-programmer, who has a backup programmer, a program librarian, and some programmers.

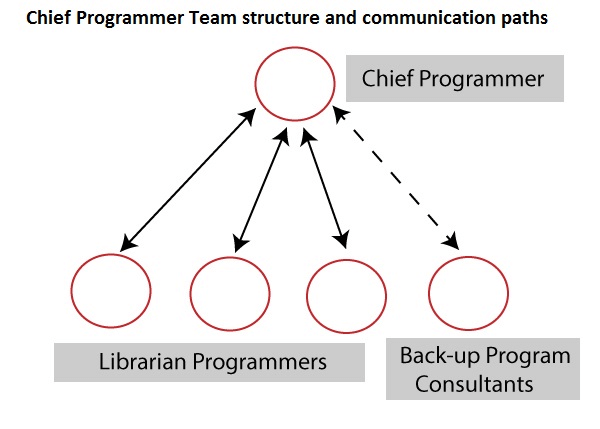
The chief programmer is essential for all major technical decisions of the project.

He does most of the designs, and he assigns coding of the different part of the design to the programmers.

The backup programmer uses the chief programmer makes technical decisions, and takes over the chief programmer if the chief programmer drops sick or leaves.

The program librarian is vital for maintaining the documentation and other communication-related work.

This structure considerably reduces interpersonal communication. The communication paths, as shown in fig:



## Controlled Decentralized Team

### (Hierarchical Team Structure)

A third team structure known as the controlled decentralized team tries to combine the strength of the democratic and chief programmer teams.

It consists of project leaders who have a class of senior programmers under him, while under every senior programmer is a group of a junior programmer.

The group of a senior programmer and his junior programmers behave like an ego-less team, but communication among different groups occurs only through the senior programmers of the group.

The senior programmer also communicates with the project leader.

Such a team has fewer communication paths than a democratic team but more paths compared to a chief programmer team.

This structure works best for large projects that are reasonably straightforward. It is not well suited for simple projects or research-type projects.

