Software Engineering

Software Engineering

The term software engineering is the product of two words, software, and engineering.
The **software** is a collection of integrated programs.
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.

Definitions

Def-1: IEEE defines software engineering as:

a. The application of a systematic, disciplined, quantifiable approach to the development, operation and maintenance of software; that is, the application of engineering to software.
b. The study of approaches as in the above statement.

Def-2: Fritz Bauer, a German computer scientist, defines software engineering as:
Software engineering is the establishment and use of sound engineering principles in order to obtain economically software that is reliable and work efficiently on real machines.

Why is Software Engineering required? (Need/importance of Software Engineering)

- ✓ Without using software engineering principles it would be difficult to develop large programs. In industry it is usually needed to develop large programs to accommodate multiple functions.
- ✓ A problem with developing such large commercial programs is that the complexity and difficulty levels of the programs increase exponentially with their sizes.
- ✓ For example, a program of size 1,000 lines of code has some complexity. But a program with 10,000 LOC is not just 10 times more difficult to develop, but may as well turn out to be 100 times more difficult unless software engineering principles are used.
- ✓ In such situations software engineering techniques come to the rescue.

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

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**: The cost of programming remains high if the proper process is not adapted.

- **Dynamic Nature**: 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.

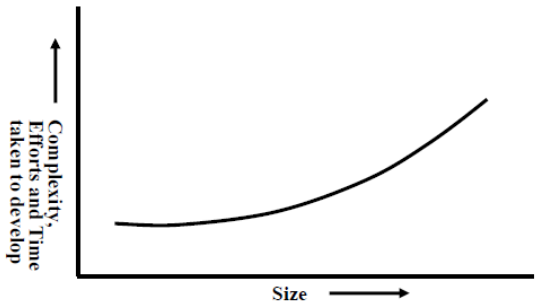


Fig. 33.3 Increase in development time and effort with problem size

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 and 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.
- Ability to work in a team and Discipline, etc.

Abstraction and Decomposition

Software engineering helps to reduce programming complexity. Software engineering principles use two important techniques to reduce problem complexity: **Abstraction and Decomposition**.

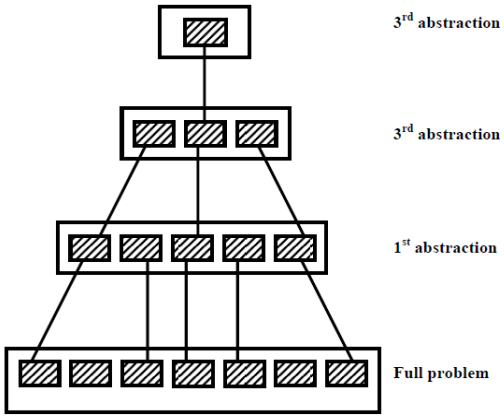


Fig. 33.4 A hierarchy of abstraction

Principle of abstraction

- Principle of abstraction (in fig) implies that a problem can be simplified by omitting irrelevant details.
- Once the simpler problem is solved then the omitted details can be taken into consideration to solve the next lower level abstraction, and so on.

Decomposition:

- In this technique, a complex problem is divided into several smaller problems and then the smaller problems are solved one by one.
- However, in this technique any random decomposition of a problem into smaller parts will not help.

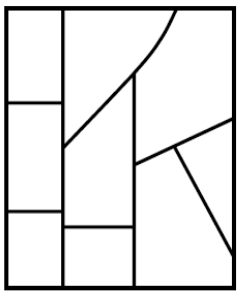


Fig. 33.5 Decomposition of a large problem into a set of smaller problems

- The problem has to be decomposed such that each component of the decomposed problem can be solved independently and then the solution of the different components can be combined to get the full solution.
- A good decomposition of a problem as shown in fig. 33.5 should minimize interactions among various components.
- If the different subcomponents are interrelated, then the different components cannot be solved separately and the desired reduction in complexity will not be realized.

Program vs. Software Product

- Programs are developed by individuals for their personal use. They are therefore, small in size and have limited functionality but software products are extremely large.
- In case of a program, the programmer himself is the sole user but on the other hand, in case of a software product, most users are not involved with the development.
- In case of a program, a single developer is involved but in case of a software product, a large number of developers are involved.
- For a program, the user interface may not be very important, because the programmer is the sole user.
- On the other hand, for a software product, user interface must be carefully designed and implemented because developers of that product and users of that product are totally different.
- In case of a program, very little documentation is expected, but a software product must be well documented.
- A program can be developed according to the programmer's individual style of development, but a software product must be developed using the accepted software engineering principles.

Evolution of Program Design (software Engineering) Techniques

a. During the 1950s, most programs were being written in **assembly language**. These programs were limited to about a few hundreds of lines of assembly code. Every programmer developed programs in his own individual style - based on his intuition. This type of programming was called **Exploratory Programming**.

b. The next significant development which occurred during early 1960s in the area computer programming was the high-level language programming. Use of **high-level language** programming reduced development efforts and development time significantly. Languages like FORTRAN, ALGOL, and COBOL were introduced at that time.

c. **Structured Programming**: As the size and complexity of programs kept on increasing, the exploratory programming style proved to be insufficient. To cope with this problem, experienced programmers advised other programmers to pay particular attention to the design of the program's **control flow structure.**

   - A structured program uses three types of program constructs i.e. *selection, sequence and iteration*.
   - Structured programs avoid unstructured control flows by restricting the use of GOTO statements.

- Structured programming uses single entry, single-exit program constructs such as if-then-else, do-while, etc.
- Structured programs are easier to maintain. They require less effort and time for development. They are amenable to easier debugging and usually fewer errors are made in the course of writing such programs.

d. **Data Structure-Oriented Design**: Pay more attention to the design of data structure, of the program rather than to the design of its control structure.

e. **Data Flow-Oriented Design**: Next significant development in the late 1970s was the development of data flow-oriented design technique. Every program reads data and then processes that data to produce some output.

f. **Object-Oriented Design**: Object-oriented design (1980s) is the latest and very widely used technique. It has an intuitively appealing design approach in which natural objects (such as employees, pay-roll register, etc.) occurring in a problem are first identified. Relationships among objects are determined.

g. **Modern practice:** The modern practice of software development is to develop the software through several well-defined stages such as requirements specification, design, coding, testing, etc., and attempts are made to detect and fix as many errors as possible in the same phase in which they occur.

Now, projects are first thoroughly planned. Project planning normally includes preparation of various types of estimates, resource scheduling, and development of project tracking plans. Several techniques and tools for tasks such as configuration management, cost estimation, scheduling, etc. are used for effective software project management.

Software development life cycle (SDLC) models

Software Life Cycle Model (also called Process Model)

1. A software life cycle model (also called **process model**) is a descriptive and diagrammatic representation of the software life cycle.
2. A life cycle model represents all the activities required to make a software product transit through its life cycle phases.
3. It also captures the order in which these activities are to be undertaken.
4. In other words, a life cycle model maps the different activities performed on a software product from its inception to its retirement.
5. Different life cycle models may map the basic development activities to phases in different ways.
6. Thus, no matter which life cycle model is followed, the basic activities are included in all life cycle models though the activities may be carried out in different orders in different life cycle models.
7. During any life cycle phase, more than one activity may also be carried out.

The Need for a Life Cycle Model

- The development team must identify a suitable life cycle model for the particular project and then adhere to it. Without using a particular life cycle model, the development of a software product would not be in a systematic and disciplined manner.

- When a software product is being developed by a team there must be a clear understanding among team members about when and what to do. Otherwise it would lead to chaos and project failure.

Example:

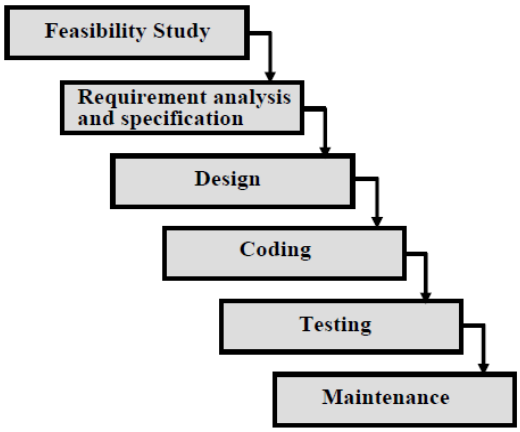
- Suppose a software development problem is divided into several parts and the parts are assigned to the team members. From then on, suppose the team members are allowed the freedom to develop the parts assigned to them in whatever way they like. It is possible that one member might start writing the code for his part, another might decide to prepare the test documents first, and some other engineer might begin with the design phase of the parts assigned to him. This would be one of the perfect recipes for project failure.
- A software life cycle model defines **entry and exit** criteria for every phase. So without a software life cycle model, the entry and exit criteria for a phase cannot be recognized.

A few important and commonly used life cycle models are as follows:

Waterfall Model

The classical waterfall model is intuitively the most obvious way to develop software. Though the classical waterfall model is elegant and intuitively obvious, we will see that it is not a practical model in the sense that it cannot be used in actual software development projects.

Thus, we can consider this model to be a *theoretical way of developing software*. But all other life cycle models are essentially derived from the classical waterfall model. So, in order to be able to appreciate other life cycle models, we must first learn the classical waterfall model.



Classical waterfall model divides the life cycle into the following phases as shown in fig.

1. Feasibility study
2. Requirements analysis and specification
3. Design
4. Coding and unit testing
5. Integration and system testing
6. Maintenance

1. Feasibility Study

The main aim of feasibility study is to determine whether it would be financially, technically, timely, operationally etc feasible to develop the product

- At first project managers or team leaders try to have a rough understanding of what is required to be done by visiting the client side. They study different input data to the system , output data and they look at the various constraints on the behaviour of the system.
- After they have an overall understanding of the problem, they investigate the different solutions that are possible.
- They pick the best solution and determine whether the solution is feasible financially and technically. They check whether the customer budget would meet the cost of the product and whether they have sufficient technical expertise in the area of development.

2. Requirements Analysis and Specification

The aim of the requirements analysis and specification phase is to understand the exact requirements of the customer and to document them properly. This phase consists of two distinct activities, namely

- Requirements gathering and analysis, and
- Requirements specification

➢ **Requirements gathering and analysis** :
The goal of the requirements gathering activity is to collect all relevant information from the customer regarding the product to be developed with a view to clearly understand the customer requirements.

➢ **Requirements specification** :
After requirements gathering, the requirements specification activity can start. During this activity, the user requirements are systematically organized into a **Software Requirements Specification (SRS**) document.
The important components of SRS document are **functional requirements**, the **non-functional requirements**, and the goals of implementation.

3. Design

The goal of the design phase is to transform the requirements specified in the SRS document into a structure that is suitable for implementation in some programming language. In technical terms, during the design phase the **software architecture** is derived from the SRS document. Two distinctly different approaches are available:

- Traditional design approach and
- Object-oriented design approach.

➢ **Traditional design approach:** Traditional design consists of two different activities; first a *structured analysis* of the requirements specification is carried out where the detailed structure of the problem is examined. This is followed by a *structured design* activity. During structured design, the results of structured analysis are transformed into the software design.

➢ **Object-oriented design approach:** In this technique, various *objects* that occur in the problem domain and the solution domain are first identified, and the different *relationships* that exist among these objects are identified. The object structure is further refined to obtain the detailed design.

➢ At the start of design phase, context diagram and different levels of **DFDs** are produced according to the SRS document. At the end of this phase module structure (**structure chart**) is produced.

4. Coding and Unit Testing

The purpose of the coding and unit testing phase (sometimes called the implementation phase) of software development is to translate the software design into source code. Each component of the design is implemented as a program module. The end-product of this phase is a set of program modules that have been individually tested.

During this phase, each module is unit tested to determine the correct working of all the individual modules. It involves testing each module in isolation as this is the most efficient way to debug the errors identified at this stage.

5. Integration and System Testing

Integration of different modules is undertaken once they have been coded and unit tested. During the integration and **system testing phase**, the modules are integrated in a planned manner.

Integration is normally carried out incrementally over a number of steps. Finally, when all the modules have been successfully integrated and tested, system testing is carried out.
The goal of **system testing** is to ensure that the developed system conforms to the requirements laid out in the SRS document.
System testing usually consists of three different kinds of testing activities:

- ✓ **α – testing**: It is the system testing performed by the development team.
- ✓ **β – testing:** It is the system testing performed by a friendly set of customers.
- ✓ **Acceptance testing:** It is the system testing performed by the customer himself after product delivery to determine whether to accept or reject the delivered product.

6. Maintenance

Maintenance involves performing any one or more of the following three kinds of activities:

- ✓ Correcting errors that were not discovered during the product development phase. This is called **corrective** maintenance.
- ✓ Improving the implementation of the system, and enhancing the functionalities of the system according to the customer's requirements. This is called **perfective** maintenance.
- ✓ Porting the software to work in a new environment. For example, porting may be required to get the software to work on a new computer platform or with a new operating system. This is called **adaptive** maintenance.

Shortcomings of the Classical Waterfall Model

- ➢ The classical waterfall model is an idealistic one since it assumes that no development error is ever committed by the engineers during any of the life cycle phases.
- ➢ However, in practical development environments, the engineers do commit a large number of errors in almost every phase of the life cycle.

Disadvantages:

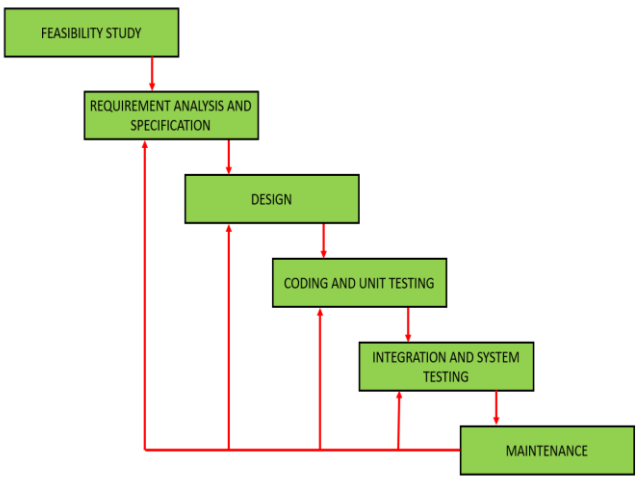
- It is difficult to measure progress within stages.
- Poor model for long and ongoing projects.
- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for long and object oriented projects.
- Cannot accommodate changing requirements.

When Should You Use It ?

1. Requirements are clear and fixed that may not change.
2. There are no ambiguous requirements (no confusion).
3. It is good to use this model when the technology is well understood.
4. The project is short and cast is low.
5. Risk is zero or minimum.

Advantages:

- ✓ It is simple and easy to understand and use.
- ✓ It is easy to manage.
- ✓ It works well for smaller and low budget projects where requirements are very well understood.
- ✓ Clearly defined stages and well understood.
- ✓ It is easy to arrange tasks.
- ✓ Process and results are well documented.



Iterative Waterfall model

In a practical software development, the classical waterfall model is hard to

use. So, Iterative waterfall model can be thought of as incorporating the necessary changes to the classical waterfall model to make it usable in practical software development.

It is almost same as the classical waterfall model except some changes are made to increase the efficiency of the software development.

The iterative waterfall model provides feedback paths from every phase to its preceding phases, which is the main difference from the classical waterfall model.

Feedback paths introduced by the iterative waterfall model are shown in the figure below.

When errors are detected at some later phase, these feedback paths allow correcting errors committed by programmers during some phase.

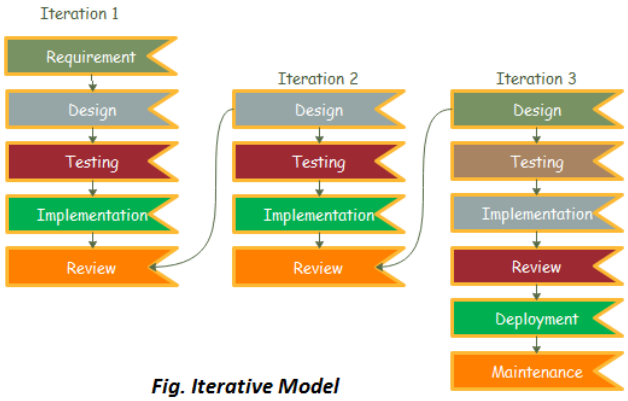


Fig. Iterative Model

Incremental Model

- Incremental Model is a process of software development where requirements divided into multiple standalone modules of the software development cycle.
- In this model, each module goes through the requirements, design, implementation and testing phases. Every subsequent release of the module adds function to the previous release.
- The process continues until the complete system achieved.

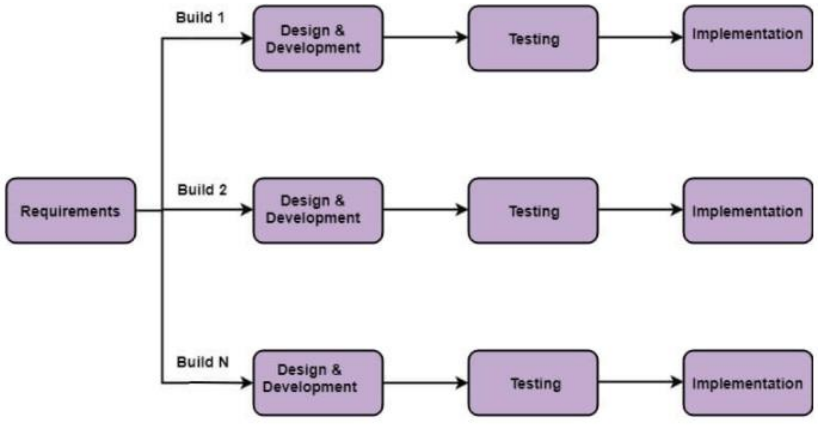


Fig: Incremental Model

When we use the Incremental Model?

- When the requirements are superior.
- A project has a lengthy development schedule.
- When Software team are not very well skilled or trained.
- When the customer demands a quick release of the product.
- You can develop prioritized requirements first.

Advantage of Incremental Model

- Errors are easy to be recognized.
- Easier to test and debug
- More flexible.
- Simple to manage risk because it handled during its iteration.
- The Client gets important functionality early.

Disadvantage of Incremental Model

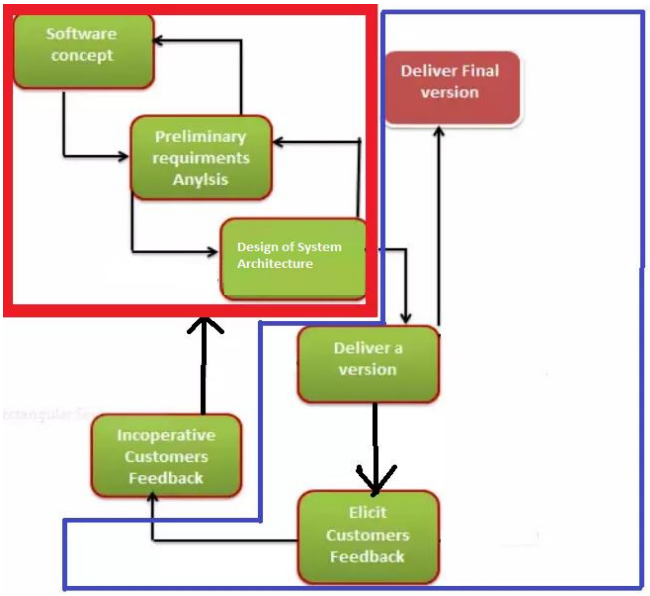
- Need for good planning
- Total Cost is high.
- Well defined module interfaces are needed.

Evolutionary model

This model is a combination of **incremental** and **iterative** models. In the evolutionary model, all work divided into smaller chunks. These chunks present to the customer one by one. The confidence of the customer increased. This model also allows for changing requirements as well as all development done into different pieces and maintains all the work as a chunk.

Where the evolutionary model is useful

- It is very useful in a large project where you can easily find a module for step by step implementation.
- The evolutionary model is used when the users need to start using the many features instead of waiting for the complete software.
- The evolutionary model is also very useful in object-oriented software development because all the development is divided into different units.



Disadvantages of Evolutionary Model

- It is difficult to divide the problem into several parts, that would be acceptable to the customer which can be incrementally implemented and delivered.

The following are the evolutionary models.

1. *The prototyping model*
2. *the spiral model*
3. *the concurrent development model*

Prototype Model

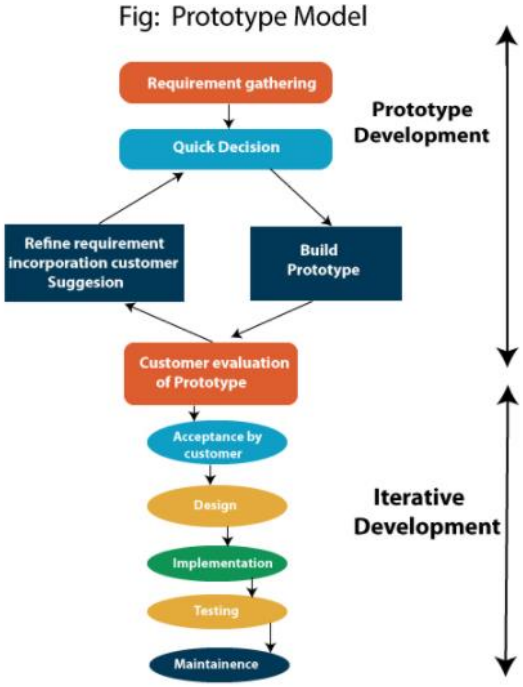
The prototype model requires that before carrying out the development of actual software, a working prototype of the system should be built. A prototype is a toy implementation of the system.

The Need for a Prototype

There are several uses of a prototype. An important purpose is to illustrate the input data formats, messages, reports, and the interactive dialogues to the customer. This is a valuable mechanism for gaining better understanding of the customer's needs.

- how screens might look like
- how the user interface would behave
- how the system would produce outputs, etc.

Fig: Prototype Model



Steps of Prototype Model

1. Requirement Gathering and Analyze
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. Require extensive customer collaboration
    - Costs customer money
    - Needs committed customer
    - Difficult to finish if customer withdraw
    - May be too customer specific, no broad market
2. Difficult to know how long the project will last.
3. It is a time-consuming process.

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.

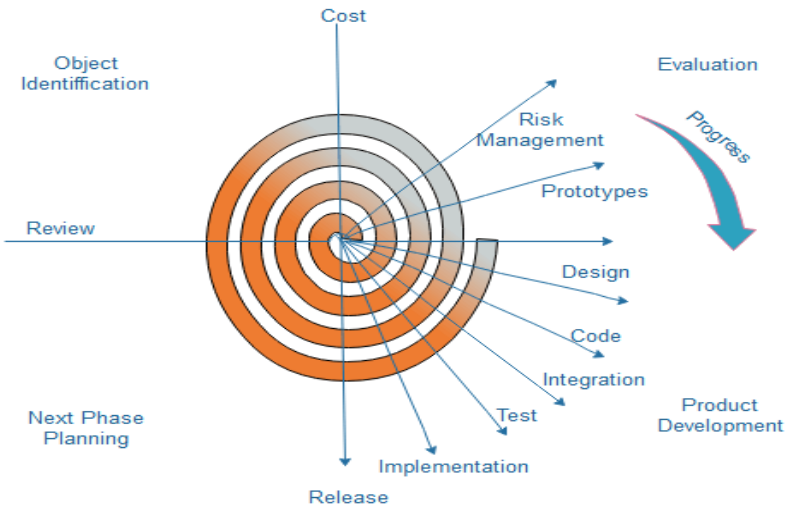


Fig. Spiral Model

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.

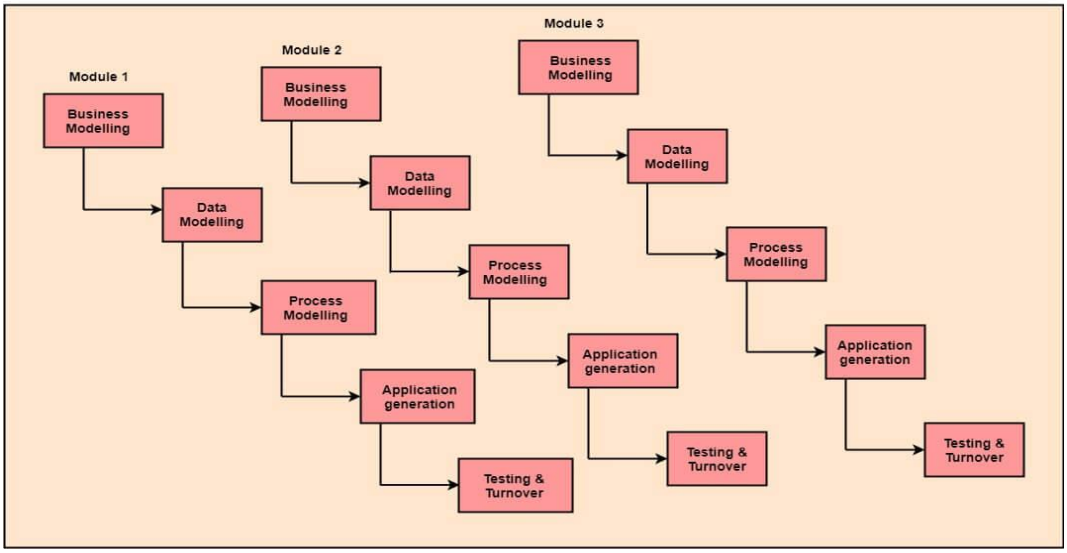
RAD (Rapid Application Development) Model

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

Fig: RAD Model



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.

Agile Model

- The meaning of Agile is versatile.
- Agile method proposes incremental and iterative approach to software design.
- AGILE methodology is a practice that promotes continuous **iteration of development and testing** throughout the software development lifecycle of the project. In the Agile model, both development and testing activities are concurrent.
- **Agile software development refers** to a **group of software development methodologies** based on iterative development, where requirements and solutions evolve through collaboration between self-organizing cross-functional teams.
- Agility is achieved by fitting the process to the project, removing activities that may not be essential for a specific project. Also, anything that is wastage of time and effort is avoided.

Agile principles

1. The highest priority of this process is to satisfy the customer.
2. Acceptance of changing requirement even late in development.
3. Frequently deliver working software in small time span.
4. Throughout the project business people and developers work together on daily basis.
5. Primary measure of progress is working software.

Fig. Agile Model

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

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.

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.
- SCRUM concentrates specifically on how to manage tasks within a team-based development environment.
- Basically, Scrum is derived from activity that occurs during a rugby match.
- Scrum believes in empowering the development team and advocates working in small teams (say- 7 to 9 members).

It consists of three roles, and their responsibilities are explained as follows:

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

Process flow of Scrum Methodologies:

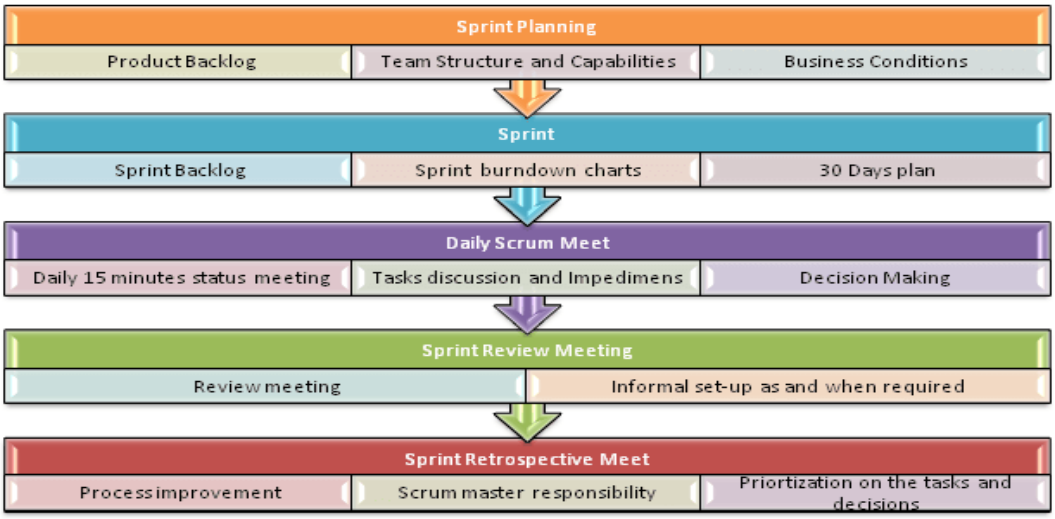
Process flow of scrum testing is as follows:

- Each iteration of a scrum is known as **Sprint**
- Product **backlog** is a list where all details are entered to get the end-product
- During each Sprint, top user stories of Product backlog are selected and turned into Sprint backlog
- Team works on the defined sprint backlog
- Team checks for the daily work

- At the end of the sprint, team delivers product functionality

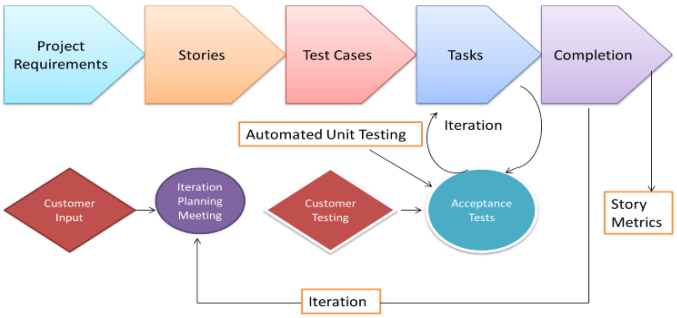
Scrum Practices

Practices are described in detailed



eXtreme Programming(XP)

Extreme Programming technique is very helpful when there is **constantly changing demands** or requirements from the customers or when they are not sure about the functionality of the system. It advocates frequent "releases" of the product in short development cycles, which inherently improves the productivity of the system and also introduces a checkpoint where any customer requirements can be easily implemented.



Extreme Programming (XP)

Business requirements are gathered in terms of **stories**. All those stories are stored in a place called the **parking lot**.

In this type of methodology, **releases** are based on the shorter cycles called **Iterations** with span of **14** days time period. Each iteration includes phases like coding, unit testing and system testing where at each phase some minor or major functionality will be built in the application.

SCRUM vs xP

There are however some differences, some of them very subtle, and particularly in the following 4 aspects:

1. **Iteration length**

Scrum

- Typically from two weeks to one month long.

XP

- Typically one or two weeks long.

2. Whether requirements are allowed to be modified in iteration

Scrum

- Do not allow changes into their sprints.
- Once the sprint planning meeting is completed and a commitment made to deliver a set of product backlog items, that set of items remains unchanged through the end of the sprint.

XP

- Much more amenable to change within their iterations.
- As long as the team hasn't started work on a particular feature, a new feature of equivalent size can be swapped into the XP team's iteration in exchange for the un-started feature.

3. Whether User Story is implemented strictly according to priority in iterations.

Scrum

- Scrum product owner prioritizes the product backlog but the team determines the sequence in which they will develop the backlog items.
- A Scrum team will very likely choose to work on the second most important.

XP

- Work in a strict priority order.
- Features to be developed are prioritized by the customer (Scrum's Product Owner) and the team is required to work on them in that order.

Crystal

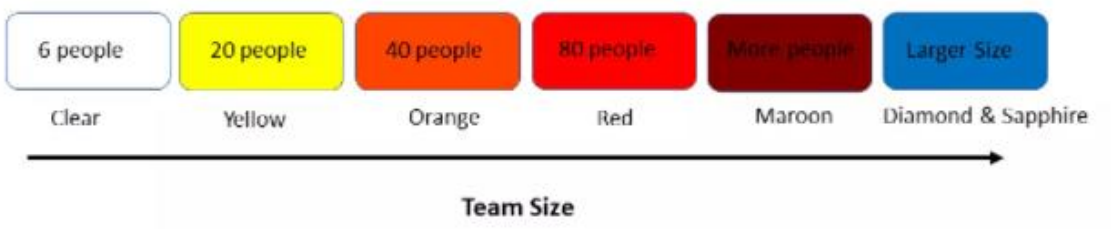
- Introduced by **Alistair Cockburn**, Crystal Methods, which is a **collection** of Agile software development approaches, focuses primarily on people and the interaction among them while they work on a software development project.
- Unlike more fixed frameworks like scrum, crystal recognizes that **different teams** will perform differently depending on **team size**, criticality, and priority of the project and encourages users to adapt the framework for their individual situation.
- For example, a small team can keep itself aligned with regular communication, so it **doesn't** need much status reporting and documentation, whereas a large team is likely to get out-of-sync and would benefit from a more structured approach.

These are categorized by color, according to the number of people in the project;

- ✓ Crystal clear - Teams with less than 8 people
- ✓ Crystal yellow - Teams with between 10 and 20 people
- ✓ Crystal orange - Teams with between 20-50 people
- ✓ Crystal red - Teams with between 50-100 people

Main practices recommended by Crystal

- An iterative and incremental development approach
- Active user involvement
- Delivering on commitments



SOFTWARE ENGINEERING (Unit-1)

Dynamic Software Development Method (DSDM)

DSDM is a Rapid Application Development (RAD) approach to software development and provides an agile project delivery framework. The important aspect of DSDM is that the users are required to be involved actively, and the teams are given the power to make decisions. Frequent delivery of product becomes the active focus with DSDM. The techniques used in DSDM are

1. Time Boxing
2. MoSCoW Rules (Must Have, Should Have, Could Have, Won't Have this time)
3. Prototyping

Feature Driven Development (FDD):

This method focuses on "Designing and Building" **features**. Feature-Driven Development (FDD) is customer-centric, iterative, and incremental, with the goal of delivering tangible software results often and efficiently.

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.

Software Project Management (SPM)

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.

What is software project management?

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

Goal of SPM:

The main goal of software project management is to enable a group of developers to work effectively towards the successful completion of a project.

Prerequisite of software project management?

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

- Time
- Cost
- Quality

Software Project Management Complexities

1. Invisibility: Software remains invisible, until its development is complete and it is operational. Anything that is invisible is difficult to manage and control.
2. Changeability: Frequent changes to the requirements and the invisibility of software are possibly the two major factors making software project management a complex task.
3. Complexity: Even a moderate sized software has millions of functions that interact with each other in many ways—data coupling, serial and concurrent runs, state transitions, control dependency, file sharing, etc.
4. Uniqueness: Every software project is usually associated with many unique features or situations.
5. Exactness of the solution: Exactness of the solution introduces additional risks and contributes to the complexity of managing software projects

Responsibilities of a software project manager (or SPM Activities)

A software project manager takes the overall responsibility of steering a project to success. We can broadly classify a project manager's varied responsibilities into the following two major categories:

1. Project planning, and
2. Project monitoring and control.

1. **Project planning**:
   - ☑ Project planning is undertaken immediately after the feasibility study phase and before the starting of the requirements analysis and specification phase.
   - ☑ Project planning involves estimating several characteristics of a project and then planning the project activities based on these estimates made.
2. **Project monitoring and control**:
   - ☑ Project monitoring and control activities are undertaken once the development activities start.
   - ☑ The focus of project monitoring and control activities is to ensure that the software development proceeds as per plan.

SOFTWARE ENGINEERING (Unit-1)

Software Project Planning

Definition: Project planning is a discipline for stating how to complete a project within a certain timeframe, usually with defined stages, and with designated resources. During project planning, the project manager performs the following activities.

1. Estimation: The following project attributes are estimated.

- *Cost*: How much is it going to cost to develop the software product?
- *Duration*: How long is it going to take to develop the product?
- Effort: How much effort would be necessary to develop the product?

2. Scheduling: After all the necessary project parameters have been estimated, the schedules for manpower and other resources are developed.

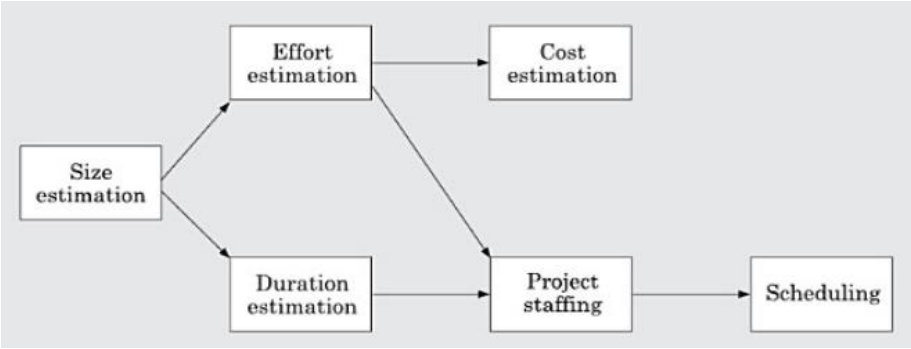
3. Staffing: Staff organization and staffing plans are made.

4. Risk management: This includes risk identification, analysis, and abatement planning.

5. Miscellaneous plans: This includes making several other plans such as *quality assurance plan*, and *configuration management* plan, etc

Precedence ordering among planning activities

Size is the most fundamental parameter based on which all other estimations and project plans are made.



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

1. **Sliding Window Planning**: In the sliding window planning technique, starting with an initial plan, the project is planned more accurately over a **number of stages**.
2. **The SPMP Document of Project Planning**: Once project planning is complete, project managers document their plans in a software project management plan (SPMP) document.
   Organization of the software project management plan (SPMP) document

1. Introduction	
	(a) Objectives (b) Major Functions (c) Performance Issues (d) Management and Technical Constraints
2. Project estimates	
	(a) Historical Data Used (b) Estimation Techniques Used (c) Effort, Resource, Cost, and Project Duration Estimates
3. Schedule	
	(a) Work Breakdown Structure (b) Gantt Chart Representation (d) PERT Chart Representation

4. Project resources	
	(a) People (b) Hardware and Software (c) Special Resources
5. Staff organization	
	(a) Team Structure (b) Management Reporting
6. Risk management plan	
	(a) Risk Analysis (b) Risk Identification (c) Risk Estimation (d) Risk Abatement (reduction) Procedures
7. Project tracking and control plan	
	(a) Metrics to be tracked (b) Tracking plan (c) Control plan
8. Miscellaneous plans	
	(a) Process Tailoring (b) Quality Assurance Plan (c) Configuration Management Plan (d) Validation and Verification (e) System Testing Plan (f) Delivery, Installation, and Maintenance Plan

Metrics for Project Size Estimation

Currently, two metrics are popularly being used to measure size:

—lines of code (LOC)

— function point (FP).

-Lines of Code (LOC): This metric measures the size of a project by counting the number of source instructions in the developed program. Obviously, while counting the number of source instructions, comment lines, and header lines are ignored.

Estimating LoC

- Accurate estimation of LOC count at the beginning of a project is a very difficult task.
- One can possibly estimate the LOC count at the starting of a project, only by using some form of systematic guess typically involves the following.
  - ☑ The project manager divides the problem into modules, and each module into sub-modules and so on, until the LOC of the leaf-level modules are small enough to be predicted.
  - ☑ To be able to predict the LOC count for the various leaf-level modules sufficiently

—function point (FP): This metric measures the size of a project by considering that "a software product is directly dependent on the number of different high-level functions or features it supports".

Function point (FP) metric computation

FP is computed using the following three steps:

Step 1: using a **heuristic** expression

The unadjusted function points (UFP) is computed as the weighted sum of five characteristics

UFP = (I)*4 + (O)*5 + (Q)*4 + (F)*10 + (N)*10

- ✓ **(I):Number of Inputs**: Each data item input by the user is counted.
- ✓ **(O): Number of Outputs**: include reports printed, screen outputs, error messages produced, etc.

- ✓ **(Q) :Number inquiries**: An inquiry is a user command (without any data input) inquiries are print account balance, print all student grades, display rank holders' names, etc.
- ✓ **(F): Number of Files**: The files referred to here are logical files
- ✓ **(N): Number of Interfaces**: different mechanisms that are used to exchange information like data files on tapes, disks, communication links with other systems, etc.

Step 2: Refine parameters

Each parameter (input, output, etc.) refined as Simple Average Complex for computing.

Table 3.1: Refinement of Function Point Entities

Type	Simple	Average	Complex
Input(I)	3	4	6
Output (O)	4	5	7
Inquiry (E)	3	4	6
Number of files (F)	7	10	15
Number of interfaces	5	7	10

Step 3: Refine UFP based on complexity of the overall project

In the final step, several factors (14 parameters) that can impact the overall project size are considered to refine the UFP computed in step 2.

A technical complexity factor (TCF) is computed as (0.65+0.01*DI).

Project Estimation

A large number of estimation techniques have been proposed by researchers. These can broadly be classified into three main categories:

1. Empirical estimation techniques
2. Heuristic techniques
3. Analytical estimation techniques

1. Empirical Estimation Techniques

While using this technique, **prior experience** with development of similar products is helpful.

Empirical estimation techniques are based on common sense and subjective decisions, over the years,

2. Heuristic Techniques

Heuristic techniques assume that the relationships that exist among the different project parameters can be satisfactorily modeled using suitable mathematical expressions**.**

Different heuristic estimation models can be divided into the following two broad Categories

[1]. single variable and

[2]. multivariable models.

[1].*Single variable* estimation models assume that various project characteristic can be predicted based on a single previously estimatedcharacteristic of the software such as its **size**.

$$\text{Estimated Parameter} = c_1 \square e^{d_1}$$

Example: Basic COCOMO Model.

[2].*A multivariable* cost estimation model assumes that a parameter can be predicted based on the values of more than one independent parameter.

$$\text{Estimated Resource} = c_1 \square p_1^{d_1} + c_2 \square p_2^{d_2} + \ldots$$

Example: Intermediate COCOMO Model.

3. Analytical Estimation Techniques:

Unlike empirical and heuristic techniques, analytical techniques do have certain scientific basis. An example of an analytical technique is Halstead's software science.

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.

To determine the initial effort E_i in person-months the equation used is of the type is shown below

$$E_i=a*(KDLOC)^b$$

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: 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: 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: new operating system (OS), a Database Management System (DBMS

3. Embedded: If the software being developed is strongly coupled to complex hardware, or if the stringent regulations on the operational method exist.

Example: ATM, Air Traffic control.

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 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 following expressions give the basic COCOMO estimation model:

   $$\textbf{Effort}=a_1*(KLOC)^{a2}\ \textbf{PM}$$

   $$\textbf{Tdev}=b_1*(efforts)^{b2}\ \textbf{Months}$$

Where

- **KLOC** is the estimated size of the software product indicate in Kilo Lines of Code
- a_1,a_2,b_1,b_2 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**.

What is a person-month?

- Person-month (PM) is a popular unit for effort measurement.
- Person-month (PM) is considered to be an appropriate unit for measuring effort,
- because developers are typically assigned to a project for a certain number of months.

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

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=a_1*(KLOC) ^{a2} PM
Tdev=b_1*(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

2. **Intermediate Model:** The intermediate COCOMO model 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:

Product attributes -

1. Required software **reliability** extent
2. **Size** of the application database
3. The **complexity** of the product

Hardware attributes -

4. **Run-time** performance constraints
5. **Memory** constraints
6. The **volatility** of the virtual machine environment
7. Required **turnabout** time

Personnel attributes -

8. Analyst capability
9. Software engineering capability
10. Applications experience
11. Virtual machine experience
12. Programming language experience

Project attributes -

13. Use of software tools
14. Application of software engineering methods
15. Required development schedule

Intermediate COCOMO equation:
E=ai $(KLOC)^{bi}$*EAF
D=ci $(E)^{di}$

Coefficients for intermediate COCOMO

Project	a_i	b_i	c_i	d_i
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 drivers effect on each method of the software engineering process.

Halstead's Software Science

Halstead's software science is an **analytical technique** to measure size development effort, and development cost of software products.

According to Halstead's "A computer program is an implementation of an algorithm considered to be a collection of tokens which can be classified as either operators or operand."

Token Count

In these metrics, a computer program is considered to be a collection of tokens, which may be classified as either operators or operands. All software science metrics can be defined in terms of these basic symbols. These symbols are called as a token.

The basic measures are

- **n1** = count of unique operators.
- **n2** = count of unique operands.
- **N1** = count of total occurrences of operators.
- **N2** = count of total occurrence of operands.
- V* = Min volume of the most briefed program in which a problem can be coded.

Vocabulary (n)	$n = n_1 + n_2$
Length (N)	$N = N_1 + N_2$ $= n_1 \log_2(n_1) + n_2 \log_2(n_2)$
Volume (V)	$V = N \log_2(n)$ $= N \log_2(n_1 + n_2)$
Level (L)	$L = V^* / V$
Difficulty (D) (inverse of level)	$D = V / V^*$
Effort (E)	$E = V / L$
Faults (B)	$B = V / S^*$

Halstead metrics are:

1. **Vocabulary**(n) :(total tokens) , that is the size of the program can be expressed as N = N1 + N2.
2. **Estimated Program Length(N):** is a the number of unique operators and operands
3. **Program Volume (V)** :It is the actual size of a program in "bits" .
4. **Program Level (L):** representing a program written at the highest possible level.
5. **Program Difficulty(D):** is proportional to the number of the unique operator in the program.
6. **Programming Effort (E):** The amount of mental activity needed to translate the existing algorithm into implementation in the specified program language.
7. **Faults (B):** The number of faults in a program is a function of its volume.

Example 3.6 Let us consider the following C program:

```
main()
{
  int a,b,c,avg;
  scanf("%d %d %d",&a,&b,&c);
  avg=(a+b+c)/3;
  printf("avg= %d",avg);
}
```

The unique operators are: `main, (), {}, int, scanf, &, ",", ";", =, +, /, printf`

The unique operands are: `a,b,c,&a,&b,&c,a+b+c,avg,3,"%d %d %d", "avg=%d"`

Therefore,

$$\eta_1 = 12, \eta_2 = 11$$

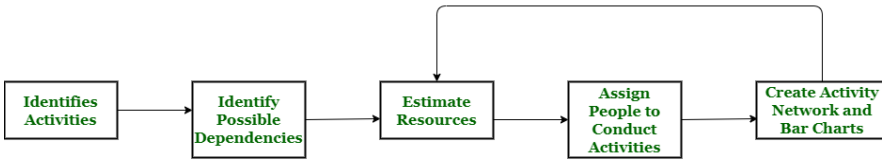
$$\text{Estimated Length} = (12 * \log 12 + 11 * \log 11)$$

$$= (12 * 3.58 + 11 * 3.45) = (43 + 38) = 81$$

$$\text{Volume} = \text{Length} * \log(23) = 81 * 4.52 = 366$$

Project Scheduling

- A schedule in your project's time table actually consists of **sequenced activities** and **milestones** that are needed to be delivered under a **given period of time**.
- The most common and important form of project schedule is **PERT** ,**CPM** and **Gantt chart**.



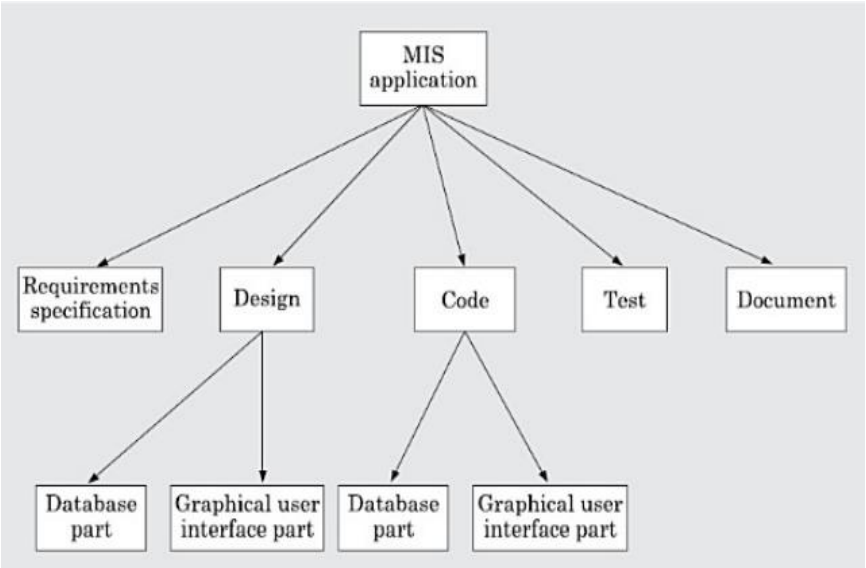
Project Scheduling Process

Scheduling Process:

1. Identify all the major activities that need to be carried out to complete the project.
2. Break down each activity into tasks.
3. Determine the dependency among different tasks.
4. Establish the estimates for the time durations necessary to complete the tasks.
5. Represent the information in the form of an activity network.
6. Determine task starting and ending dates from the information represented in the activity network.
7. Determine the critical path. A critical path is a chain of tasks that determines the duration of the project.
8. Allocate resources to tasks.

Work Breakdown Structure

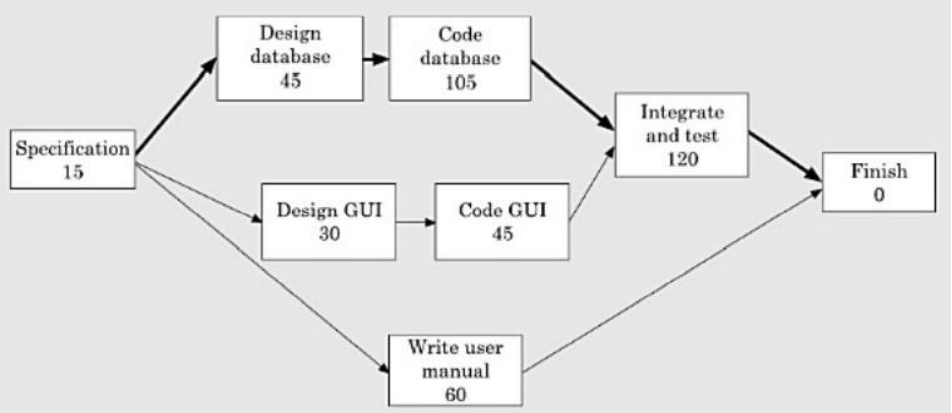
1. Work breakdown structure (WBS) is used to recursively decompose a given set of activities into smaller activities.
2. WBS provides a notation for representing the activities, sub-activities, and tasks needed to be carried out in order to solve a problem. Each of these is represented using a rectangle (see Figure).
3. The root of the tree is labeled by the project name. Each node of the tree is broken down into smaller activities that are made the children of the node.
4. Figure 3.7 represents the WBS of management information system (MIS) software.



Activity Networks:

An activity network shows the different activities making up a project, their estimated durations, and their interdependencies. Two equivalent representations for activity networks are possible and are in use:

Activity on Node (AoN): In this representation, each activity is represented by a rectangular (some use circular) node and the duration of the activity is shown alongside each task in the node. The inter-task dependencies are shown using directional edges (see Figure).



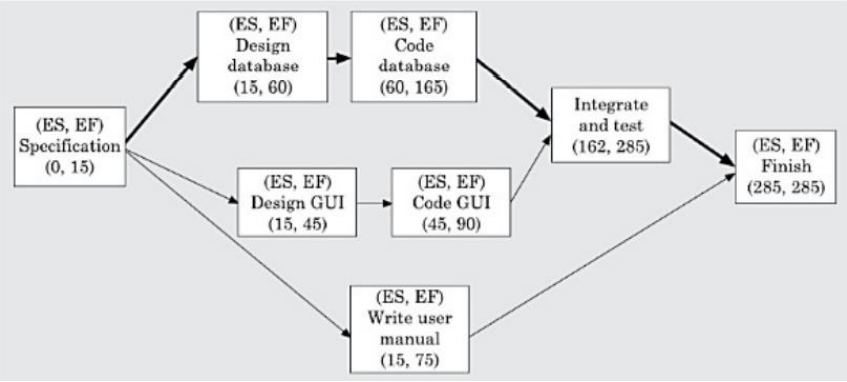
Activity on Edge (AoE): In this representation tasks are associated with the edges. The edges are also annotated with the task duration. The nodes in the graph represent project milestones.

Critical Path Method (CPM)

CPM is an algorithmic approach to determine the critical paths and slack times for tasks not on the critical paths involves calculating the following quantities:

1. **Minimum time (MT):** It is the minimum time required to complete the project.
2. **Earliest start (ES):** It is the time of a task is the maximum of all paths from the start to this task.
3. **Latest start time (LST):** It is the difference between MT and the maximum of all paths from this task to the finish.
4. **Earliest finish time (EF**): The EF for a task is the sum of the earliest start time of the task and the duration of the task.
5. **Latest finish (LF):** LF indicates the latest time by which a task can finish without affecting the final completion time of the project.
6. **Slack time (ST):** The slack time (or float time) is the total time that a task may be delayed before it will affect the end time of the project.

Example: : The activity network with computed ES and EF values has been shown in Figure



The activity network with computed LS and LF values has been shown in Figure

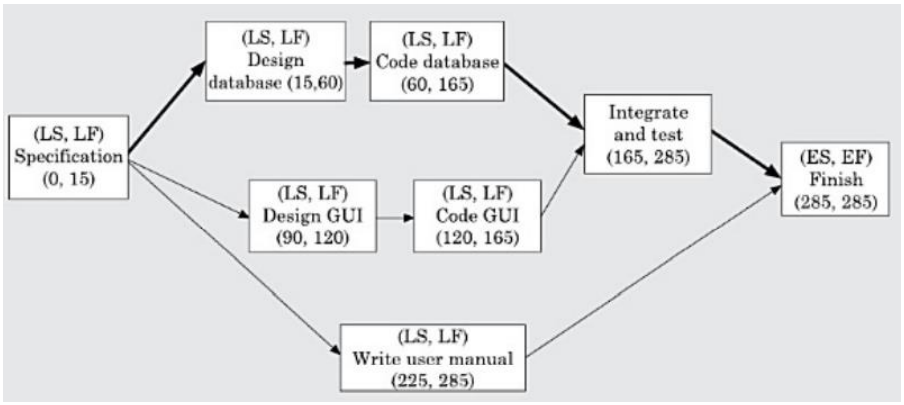


Table 3.8: Project Parameters Computed From Activity Network

Task	ES	EF	LS	LF	ST
Specification	0	15	0	15	0
Design data base	15	60	15	60	0
Design GUI part	15	45	90	120	75
Code data base	60	165	60	165	0
Code GUI part	45	90	120	165	75
Integrate and test	165	285	165	285	0
Write user manual	15	75	225	285	210

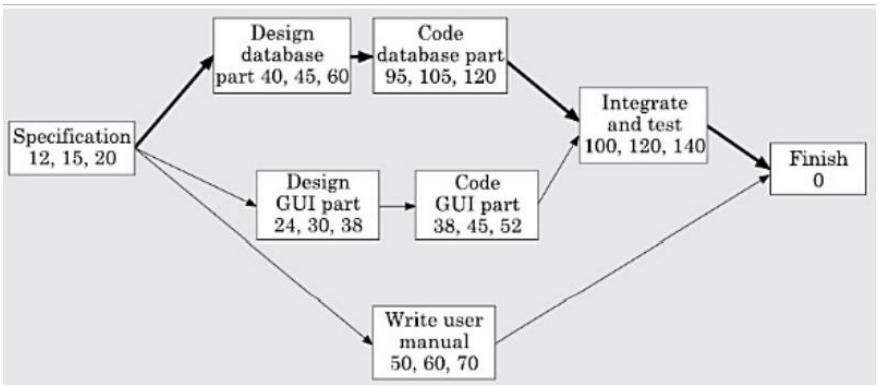
- ✓ The CPM can be used to determine the duration of a project, but does not provide any indication of the probability of meeting that schedule.

PERT Charts

Project evaluation and review technique (PERT) charts are a more sophisticated form of activity chart. Each task is annotated with three estimates:

- Optimistic (O): The best possible case task completion time.
- Most likely estimate (M): Most likely task completion time.
- Worst case (W): The worst possible case task completion time.

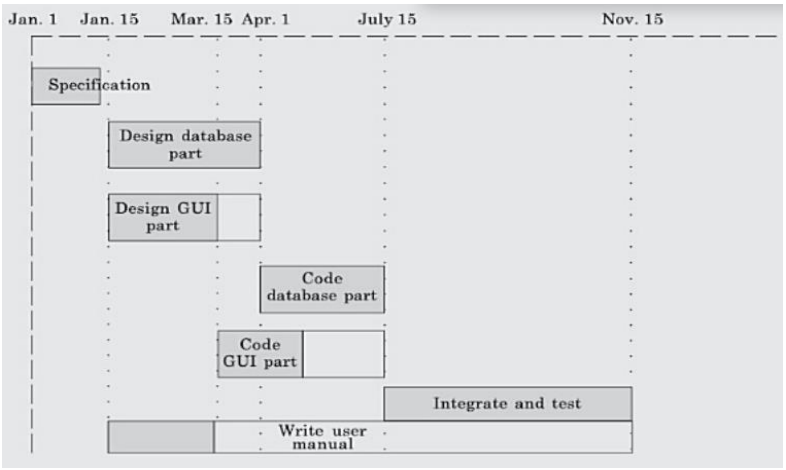
The PERT chart representation of the MIS problem of Figure 3



Gantt Charts

Gantt chart has been named after its developer Henry Gantt. Gantt chart is a special type of bar chart where each bar represents an activity. The bars are drawn along a time line. The length of each bar is proportional to the duration of time planned for the corresponding activity.

A Gantt chart representation for the MIS problem of Figure 3



We can summarize the differences between the two as listed in the table below:

Gantt chart	PERT chart
Gantt chart is defined as the bar chart.	PERT chart is similar to a network diagram
Gantt chart is often used for Small Projects	PERT chart can be used for large and complex Projects
Gantt chart focuses on the time required to complete a task	PERT chart focuses on the dependency of relationships.
Gantt chart is simpler and more straightforward	PERT chart could be sometimes confusing and complex but can be used for visualizing critical path

Personnel Planning(Staffing)

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.
- ➢ 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.

SOFTWARE ENGINEERING (Unit-1)

Organization and Team Structures

Usually every software development organization handles several projects at any time. Software organizations assign different teams of developers to handle different software projects.

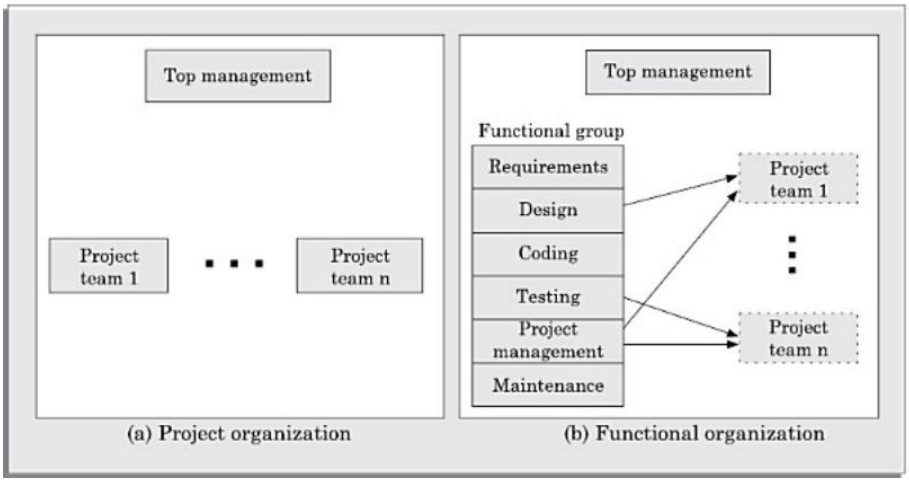
(A). Organization Structure

Essentially there are three broad ways in which a software development organization can be structured

- Functional format,
- project format, and
- Matrix format.

1. Functional format: In the functional format, the development staff are divided based on the specific functional group to which they belong to. This format has schematically been shown in Figure

2. Project format: In the project format, the development staff are divided based on the project for which they work (See Figure).



The main advantages of a **functional organization** are:

- Ease of staffing
- Production of good quality documents
- Job specialization
- Efficient handling of the problems associated with manpower turnover

3. Matrix format: A matrix organization is intended to provide the advantages of both functional and project structures. In a matrix organization, the pool of functional specialists is assigned to different projects as needed.

	Project			
Functional group	#1	#2	#3	
#1	2	0	3	Functional manager 1
#2	0	5	3	Functional manager 2
#3	0	4	2	Functional manager 3
#4	1	4	0	Functional manager 4
#5	0	4	6	Functional manager 5
	Project manager 1	Project manager 2	Project manager 3	

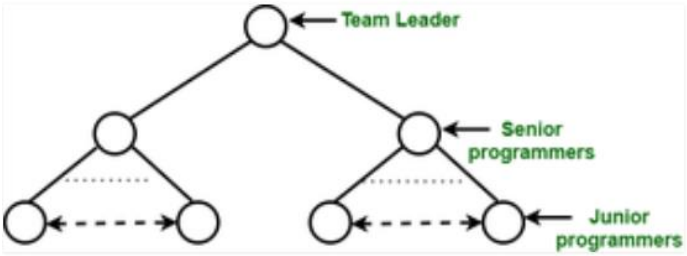
(B). Team Structure

There are many ways to organize the project team. Some important ways are as follows :

1. Hierarchical team organization
2. Chief-programmer team organization
3. Egoless team organization

1. Hierarchical team organization :

In this, the people of organization at different levels following a tree structure. People at bottom level generally possess most detailed knowledge about the system. People at higher levels have broader appreciation of the whole project.



Benefits of hierarchical team organization :

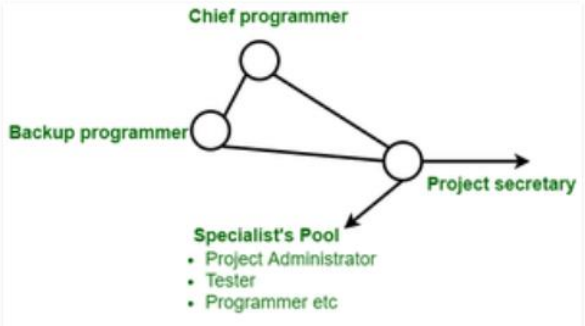
- It limits the number of communication paths and stills allows for the needed communication.
- It is well suited for the development of the hierarchical software products.
- Large software projects may have several levels.

Limitations of hierarchical team organization:

- As information has to be travel up the levels, it may get distorted.

2.Chief-programmer team organization:

This team organization is composed of a small team consisting the following team members :



- ☑ The Chief programmer : It is the person who is actively involved in the planning, specification and design process and ideally in the implementation process as well.
- ☑ The project assistant : It is the closest technical co-worker of the chief programmer.
- ☑ The project secretary : It relieves the chief programmer and all other programmers of administration tools.
- ☑ Specialists : These people select the implementation language, implement individual system components and employ software tools and carry out tasks.

Advantages of Chief-programmer team organization :

- Centralized decision-making
- Reduced communication paths
- Small teams are more productive than large teams
- The chief programmer is directly involved in system development and can exercise the better control function.

Disadvantages of Chief-programmer team organization :

- Project survival depends on one person only.
- Can cause the psychological problems as the “chief programmer” is like the “king” who takes all the credit and other members are resentful.

3. Egoless Team Organization:

Egoless programming is a state of mind in which programmer are supposed to separate themselves from their product. Here group, 'leadership' rotates based on tasks to be performed and differing abilities of members.

Risk management

Risk

- **Definition**: "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

Risk Management is the system of identifying addressing and eliminating **potential issues** (might harm cost, schedule or technical success of the project) before they can damage the project

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

Types of Risks

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.

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.

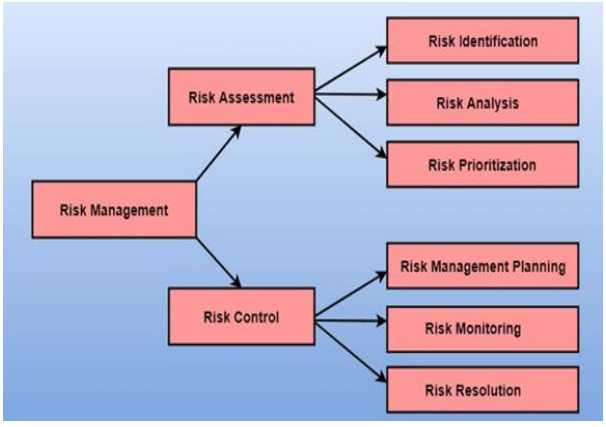
3. **Business risks**: This type of risks contain risks of building an excellent product that no one need, losing budgetary or personnel commitments, etc.

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 two main activities, as shown in fig:



(1).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:

1. The possibility of a risk coming true
2. The consequence of the issues relates to that risk

(2).Risk Control(Mitigation)

It is the process of managing risks to achieve desired outcomes.
There are three main methods:
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.
Transfer the risk: This method involves getting the risky element developed by a third party, buying insurance cover, etc.
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.

SOFTWARE ENGINEERING (Unit-1)

Software Configuration Management

Definition: Software Configuration Management (SCM) is a technique of identifying, organizing, and controlling modification to software being built by a programming team.

Why do we need Configuration Management?

- When we develop software, the product (software) undergoes many changes in their **maintenance** phase; we need to handle these changes effectively.
- Multiple people are working on software which is consistently updating. It may be a method where multiple version, branches, authors are involved in a software project, and the team is geographically distributed and works concurrently. It changes in user requirements, and policy, budget, schedules need to be accommodated

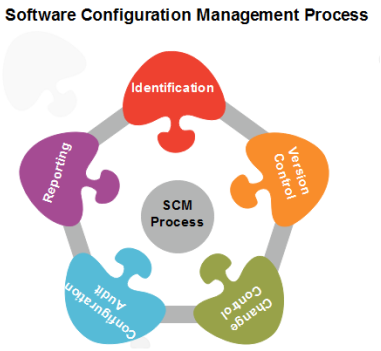
Importance of SCM

- It is practical in controlling and managing the access to various SCIs e.g., by preventing the two members of a team for **checking out** the same component for modification at the same time.
- It provides the tool to ensure that changes are being properly implemented.
- It has the capability of describing and storing the various constituent of software.
- SCM is used in keeping a system in a consistent state by automatically producing derived version upon modification of the same component.

SCM Process

It uses the tools which keep that the necessary change has been implemented adequately to the appropriate component. The SCM process defines a number of tasks:

1. Identification of objects in the software configuration
2. Version Control
3. Change Control
4. Configuration Audit
5. Status Reporting



1. **Identification**: Unit of Text created by a software engineer during analysis, design, code, or test.
2. **Version Control:** Version Control combines procedures and tools to handle different version of configuration objects that are generated during the software process.
3. **Change Control**: The "check-in" and "check-out" process implements two necessary elements of change control-access.
4. **Configuration Audit**: SCM audits to verify that the software product satisfies the baselines requirements and ensures that what is built and what is delivered.
5. **Status Reporting**: Configuration Status reporting providing accurate status and current configuration data to developers, testers, end users, customers and stakeholders