SOFTWARE ENGINEERING

Unit - 1

INTRODUCTION TO SOFTWARE ENGINEERING

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

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be a collection of executable programming code, associated libraries and documentations. Software, when made for a specific requirement is called **software product.**

Engineering on the other hand, is all about developing products, using well-defined, scientific principles and methods.

Software Engineering:

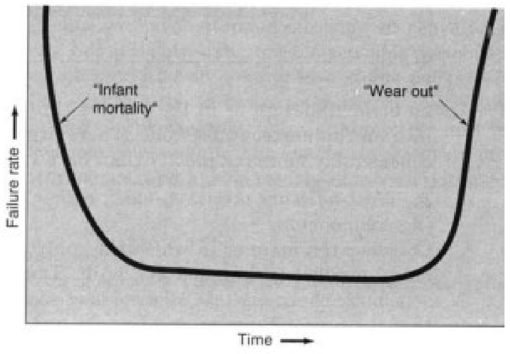
It is defined as systematic , disciplined and quantifiable approach in the development , operation and maintenance of software.

Software Characteristics

Software is developed or engineered; it is not manufactured in the classical sense.

Although some similarities exist between software development & hardware manufacturing both the activities are different. In both activities high quality is achieved through good design, but manufacturing phase will introduce quality problems that are non-existent in software. Both activities depend on people but relationship between people applied and work accomplished is different.

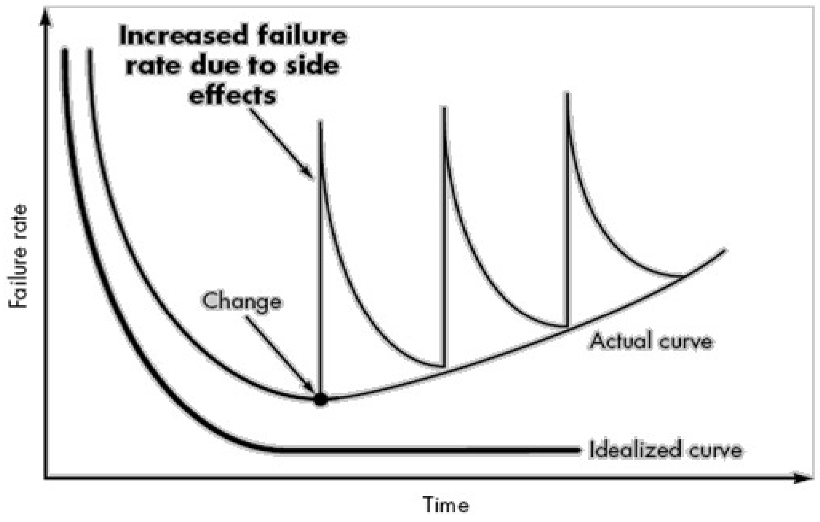
Software does not wear out. However it deteriorates due to change



Failure curve for H/W

It indicates that hardware exhibits high failure rates early in its life. Defects are corrected, and failure rate drops to a steady-state level for some period of time. As time passes, the failure rate rises again as hardware components suffer from cumulative effects of dust, temperature and other environmental maladies. Simply hardware begins to wear out.

Failure Curve or S/W



Software doesn't wear out so it should take form of idealized curve. Undiscovered defects will cause high failure rates early in its life of a program. However these are corrected and curve flattens the actual curve shows that during its life software will undergo change. As changes are made, it is likely that errors will be introduced, causing failure rate curve to spike again. Before curve can return to original state, another change is requested, causing curve to spike again. Slowly failure rate level begins to rise – **the software is deteriorating due to change.**

-Although the industry is moving towards component based construction, most software continues to be custom built

Custom-built: building according to needs of customer

The main of component-**based construction is reuse. In the hardware world, component reuse is a** natural part of engineering process. In software world it has only begun to achieve on a broad scale.

Abstraction versus Decomposition

Abstraction

It refers to the construction of a simpler version of a problem by ignoring the details. The principle of constructing an abstraction is popularly known as modelling.

Decomposition

Decomposition is a process of breaking down. It will be breaking down functions into smaller parts. It is another important principle of software engineering to handle problem complexity. The decomposition principle is popularly is says the divide and conquer principle.

Evolution of software:

1950's-60's (mid): Batch orientation, limited distribution

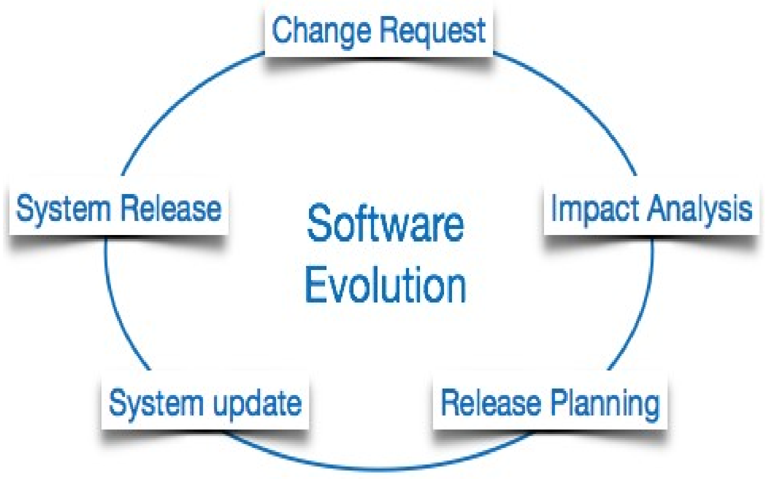
1960's (mid)-1970's (mid): multiuser, realtime, database, product s/w

1970's mid-1980's (late): distributed systems, low cost h/w, and consumer impact s/w

1980's (late)-2000: object-oriented techniques, artificial neural networks.

Evolution Of Software Engineering Techniques

- The process of developing a software product using software engineering principles and methods is referred to as **software evolution.**
- This includes the initial development of software and its maintenance and updates, till desired software product is developed, which satisfies the expected requirements.



Software Evolution Laws

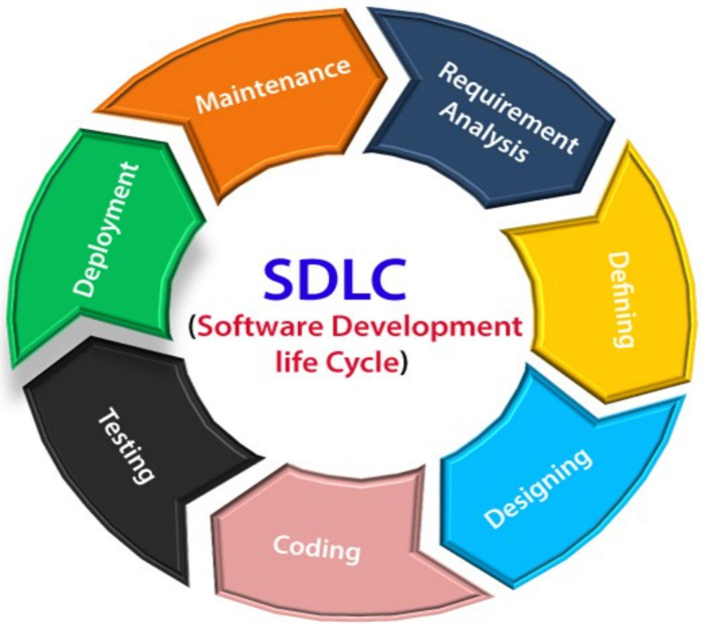
Lehman has given laws for software evolution. He divided the software into **three different categories:**

S-type (static-type) - This is a software, which works strictly according to defined **specifications and solutions.** The solution and the method to achieve it, both are immediately understood before coding. The s-type software is least subjected to changes hence this is the simplest of all. For example, calculator program for mathematical computation.

P-type (practical-type) - This is a software with a collection of **procedures.** This is defined by exactly what procedures can do. In this software, the specifications can be described but the solution is not obvious instantly. For example, gaming software.

E-type (embedded-type) - This software works closely as the requirement of real-world **environment.** This software has a high degree of evolution as there are various changes in laws, taxes etc. in the real world situations. For example, Online trading software.

Software development life cycle (SDLC) models



SDLC is a process followed for a software project, within a software organization. It consists of a detailed plan describing how to develop, maintain, replace and alter or enhance specific software. The life cycle defines a methodology for improving the quality of software and the overall development process.

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 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 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.

SRS (software requirement specification)

Definition- SRS is a complete reading base documentation focus on the particular desired software to the specific client or customer. After collecting the necessary data from SDLC we have to summarize the useful and appropriate data for making desired software. SRS has some objectives which is help to the software developer as well as the customer for making a successfully software.

Characteristics of SRS 1) complete 2) traceable 3) appropriate for the developer 4) modifiable 5) simple language 6) software requirement view

Good SRS: - SRS is a very important for fact gathering technique which include the consider delimiting with the customer gather previous information related to particular software and returned on investment the good characteristics of SRS includes the following activities:

1) It focuses on summarized from for a particular software specification.

2) The completeness deserves the related phase for making a required specification.

3) The traceable factor focus on the modification part when ever it necessary for the development of SRS.

SDLC Models

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.

Advantages of software process model

- Enables effective communication
- Facilitates process reuse
- Effective
- Facilitates process management

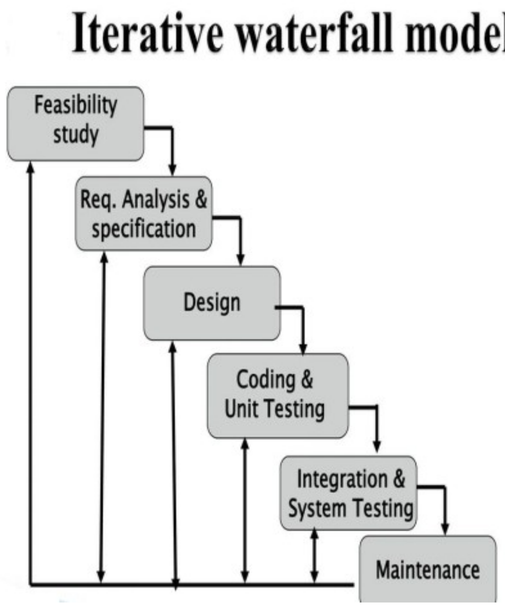
- Various kinds of process models
  - Waterfall model
  - Prototyping model
  - Spiral model
  - Incremental model
  - RAD model
  - V model
  - Build and fix model

Classical Waterfall Model – Limitations

- But, in this model, complete requirements should be available at the time of commencement of the project
- But in actual practice, the requirements keep on originating during different phases.
- Moreover, it does not incorporate any kind of risk assessment. In waterfall model, a working model of software is not available.
- Thus, there is no methods to judge the problems of software in between different phases. So it is not suitable for the large projects

Iterative Waterfall Model

- This model was developed to remove the shortcomings of waterfall model.
- This is also referred as Modified Waterfall Life-Cycle Model
- The classical waterfall model is an idealistic model because it assumes that development error is ever committed by the engineers during any phase of life cycle of software development.
- But in practical environment of development, engineers do commit large number of errors in almost every phase of life cycle of software development
- Once defect is detected, the engineers need to go back to the phase where the defects had occurred and redo some work done during that phase and the subsequent phases to correct the defects and its effect on the latter phases.
- Therefore a feedback path is needed in the classical waterfall model from every phase to its preceding phase.
- The principle of detecting errors as close to their points of introduction as possible is called phase containment error.


Iterative waterfall model
Feasibility study
Req. Analysis & specification
Design
Coding & Unit Testing
Integration & System Testing
Maintenance

Requirements Phase: In the requirements phase of software development, the system related information is gathered and analyzed. The collected requirements are then planned accordingly for developing the system.

Design Phase: In the Design phase, the software solution is prepared to meet the necessities for the design. The system design may be a new one or the extension of a previous build one.

Implementation / coding and unit testing: During this phase, design is implemented. If the Software Design Document (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.

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.

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

Phase	**Input**	**Output**
Requirement analysis	Requirement defined through communication	● SRS document ● Feasibility report
Design	SRS document	● Design specification document ● Test class design
Coding	Design specification document	● Executable software modules ● Test plan ● Software documentation
Testing	Executable software modules	● Integrated and tested system ● Software documentation

Implementation	Integrated product	● Delivered software
Maintenance	Delivered software	● Changed requirements

Advantages and Disadvantages

Advantages

- It is most widely used software development model
- Simple to understand and easy to implement.

Disadvantages

- Going back a phase or two can be a costly affair.

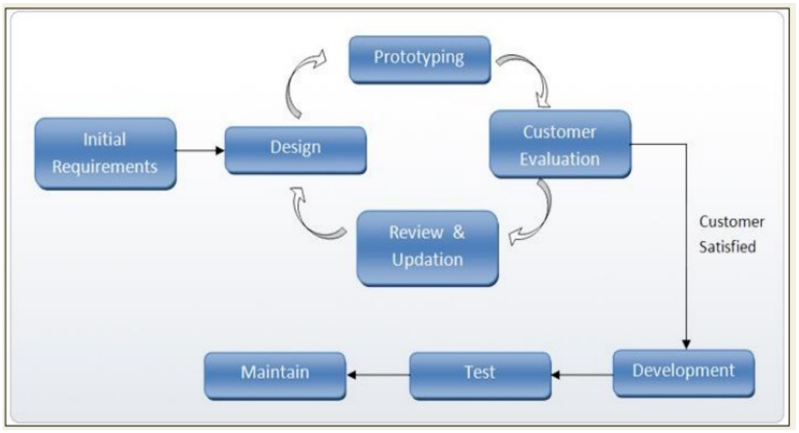
Even a small change in any previous stage can cause big problem for subsequent phases as all phases are dependent on each-other.

When to use Iterative waterfall model?

- Suitable only for well-understood problems
- Not suitable for very large and risk prone projects
- This methodology is preferred in projects where quality is more important as compared to schedule or cost.
- In development of database-related software and commercial projects.
- In development of E-commerce website or portal.
- In development of network protocol software.

Prototype Model

- Applied when there is an absence of detailed information regarding input and output requirements in the software
- Increases flexibility of the development process by allowing the user to interact and experiment with a working representation of the product known as **prototype**
- At any stage, if the user is not satisfied with the prototype, it can be thrown away and an entirely new system is developed



Step 1: Requirements gathering and analysis

A prototyping model starts with requirement analysis. In this phase, the requirements of the system are defined in detail. During the process, the users of the system are interviewed to know what is their expectation from the system.

Step 2: Quick design

The second phase is a preliminary design or a quick design. In this stage, a simple design of the system is created. However, it is not a complete design. It gives a brief idea of the system to the user. The quick design helps in developing the prototype.

Step 3: Build a Prototype

In this phase, an actual prototype is designed based on the information gathered from quick design. It is a small working model of the required system.

Step 4: Initial user evaluation

In this stage, the proposed system is presented to the client for an initial evaluation. It helps to find out the strength and weakness of the working model. Comment and suggestion are collected from the customer and provided to the developer.

Step 5: Refining prototype

If the user is not happy with the current prototype, you need to refine the prototype according to the user's feedback and suggestions.

This phase will not over until all the requirements specified by the user are met. Once the user is satisfied with the developed prototype, a final system is developed based on the approved final prototype.

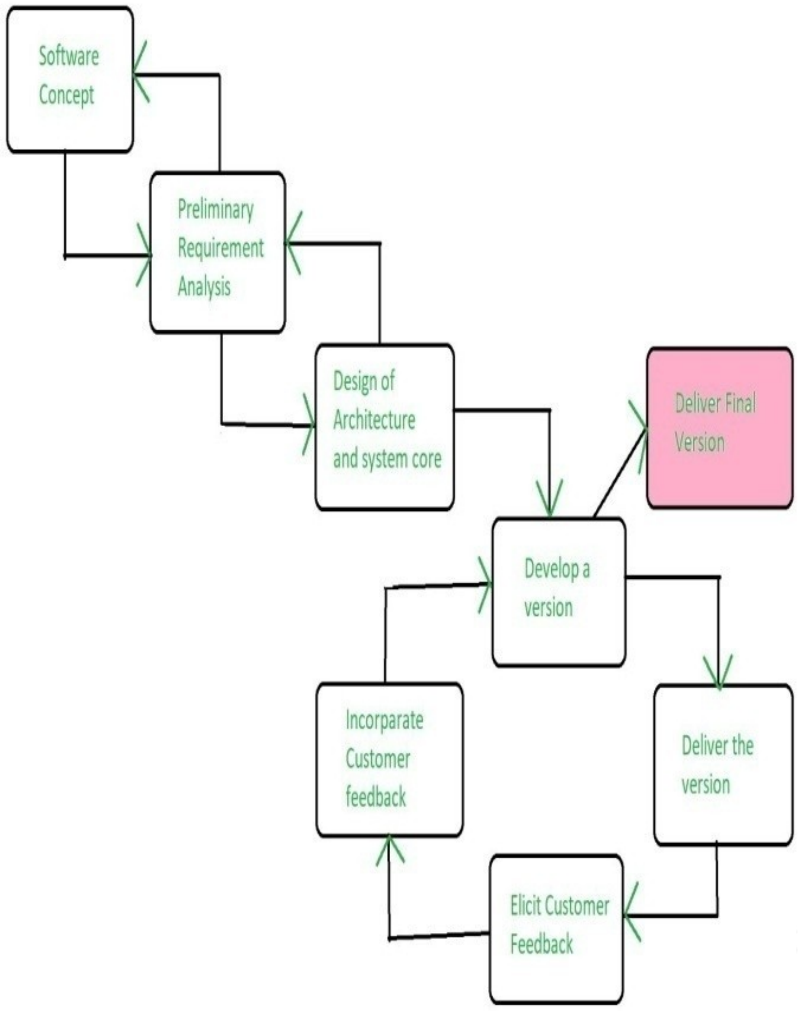
Step 6: Implement Product and Maintain

Once the final system is developed based on the final prototype, it is thoroughly tested and deployed to production. The system undergoes routine maintenance for minimizing downtime and prevent large-scale failures.

Evolutionary Model

Evolutionary model is a combination of Iterative and Incremental model of software development life cycle.

- Construct a partial implementation of a total system
- Then slowly add increased functionality
- The incremental model prioritizes requirements of the system and then implements them in groups.
- Each subsequent release of the system adds function to the previous release, until all designed functionality has been implemented.


Software Concept
Preliminary Requirement Analysis
Design of Architecture and system core
Deliver Final Version
Develop a version
Incorparate Customer feedback
Deliver the version
Elicit Customer Feedback

Applications:

- For **small or medium-size** interactive systems
- For **parts of large systems** (e.g. the user interface)
- This model is **useful for projects using new technology** that is not well understood.
- This is also **used for complex projects where all functionality must be delivered at one time, but the requirements are unstable or not well understood at the beginning**.

Advantages:

1. User gets a chance to experiment partially developed system
2. Reduce the error because the core modules get tested thoroughly

Disadvantages:

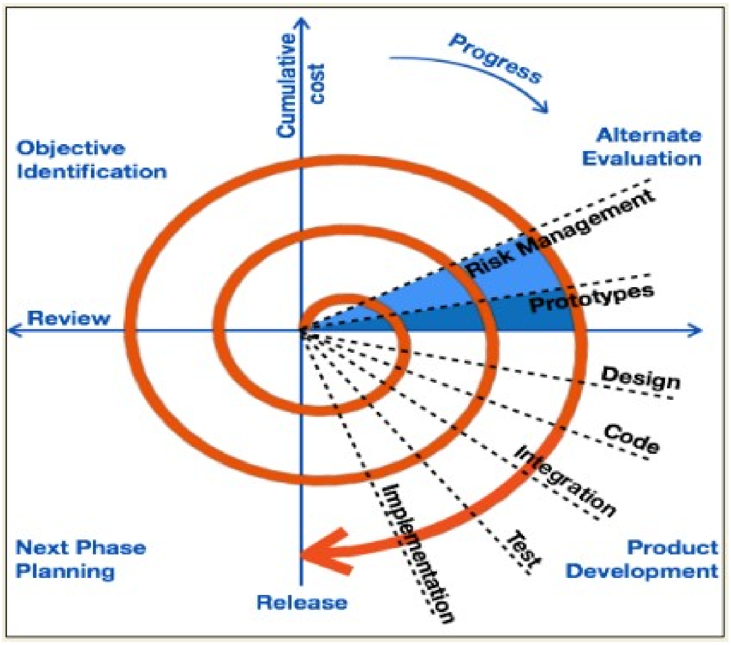
It is difficult to divide the problem into several versions that would be acceptable to the customer which can be incrementally implemented & delivered

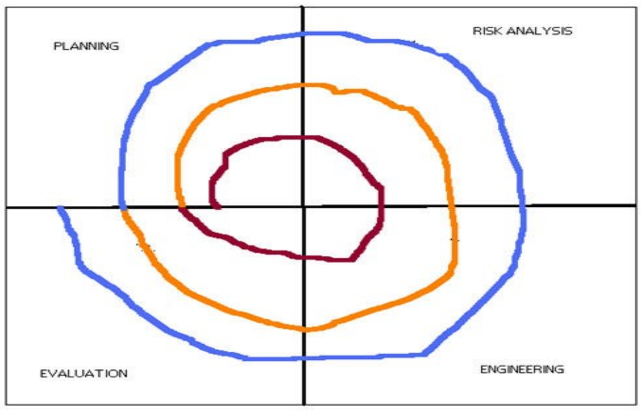
Spiral Model

1.Spiral model is an evolutionary software model.

2.Spiral model may be viewed as a Meta model, because it can accommodate any process model.

3.Spiral model focuses on identifying and eliminating high risk problems.





First quadrant (Objective Setting)

During the first quadrant, it is needed to identify the objectives of the phase. Examine the risks associated with these objectives

Second Quadrant (Risk Assessment and Reduction)

- A detailed analysis is carried out for each identified project risk.
- Steps are taken to reduce the risks. For example, if there is a risk that the requirements are inappropriate, a prototype system may be developed.

Third Quadrant (Development and Validation)

- Develop and validate the next level of the product after resolving the identified risks.

Fourth Quadrant (Review and Planning)

- Review the results achieved so far with the customer and plan the next iteration around the spiral.
- Progressively more complete version of the software gets built with each iteration around the spiral.

Spiral Model helps to adopt software development elements of multiple process models for the software project based on unique risk patterns ensuring efficient development process. Each phase of spiral model in software engineering begins with a design goal and ends with the client reviewing the progress

Advantages of Spiral Model

Below are some advantages of the Spiral Model.

1. **Risk Handling:** The projects with many unknown risks that occur as the development proceeds, in that case, Spiral Model is the best development model to follow due to the risk analysis and risk handling at every phase.
2. **Good for large projects:** It is recommended to use the Spiral Model in large and complex projects.
3. **Flexibility in Requirements:** Change requests in the Requirements at later phase can be incorporated accurately by using this model.
4. **Customer Satisfaction:** Customer can see the development of the product at the early phase of the software development and thus, they habituated with the system by using it before completion of the total product.

Disadvantages of Spiral Model

Below are some main disadvantages of the spiral model.

1. **Complex:** The Spiral Model is much more complex than other SDLC models.
2. **Expensive:** Spiral Model is not suitable for small projects as it is expensive.
3. Too much dependability on Risk Analysis: The successful completion of the project is very much dependent on Risk Analysis. Without very highly experienced experts, it is going to be a failure to develop a project using this model.
4. **Difficulty in time management:** As the number of phases is unknown at the start of the project, so time estimation is very difficult.

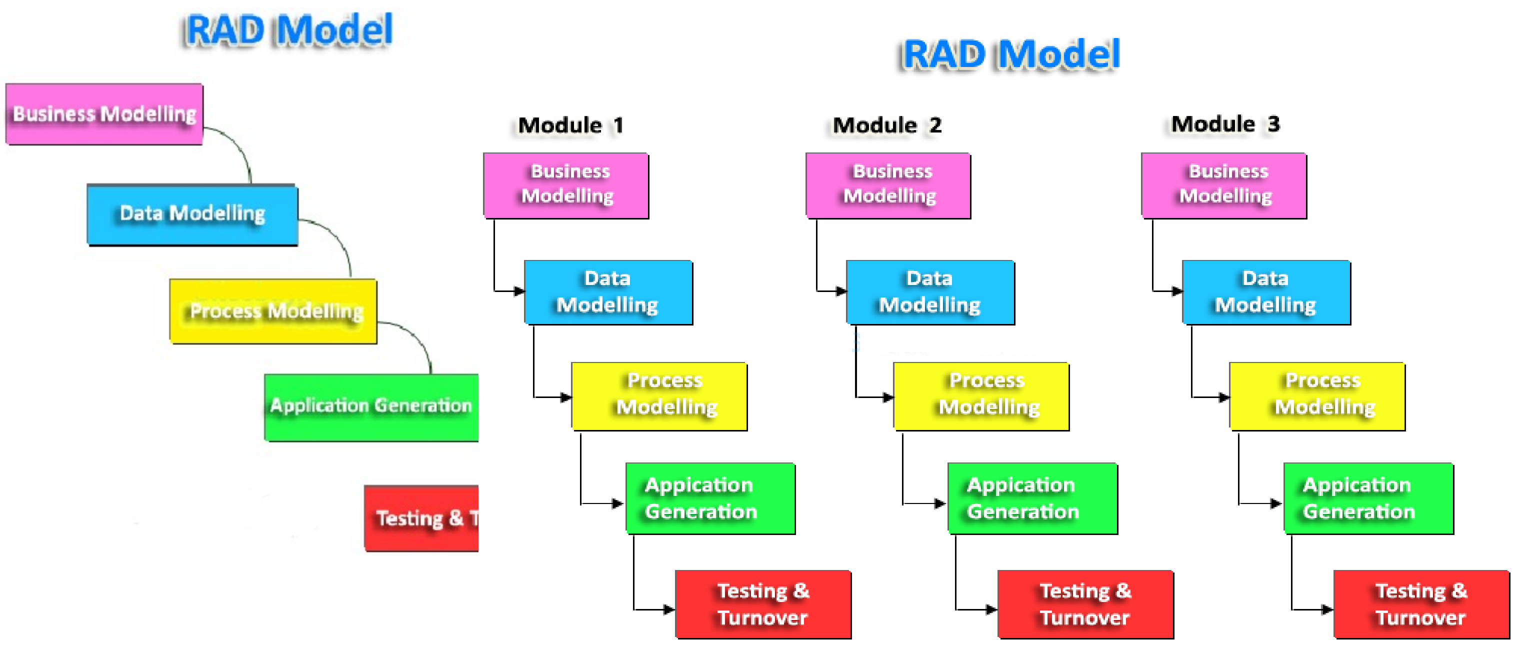
When to use Spiral Model?

- ✓ When the project is large
- ✓ Large and high budget projects
- ✓ When releases are required to be frequent, spiral methodology is used
- ✓ When creation of a prototype is applicable
- ✓ When risk and costs evaluation is important
- ✓ Spiral methodology is useful for medium to high-risk projects
- ✓ When requirements are unclear and complex, Spiral model in SDLC is useful
- ✓ When changes may require at any time
- ✓ When long term project commitment is not feasible due to changes in economic priorities

RAD (Rapid Application Development) Model

RAD stand for Rapid Application Development. Rapid Application Development model is a software development process based on prototyping without any specific planning.

It focuses on input-output source and destination of the information. It emphasizes on delivering projects in small pieces; the larger projects are divided into a series of smaller projects. The main features of RAD modeling are that it focuses on the reuse of templates, tools, processes, and code.



In RAD model, there is less attention paid to the planning and more priority is given to the development tasks. It targets at developing software in a short span of time. SDLC RAD modeling has following phases

SDLC RAD Modeling Phases
1. Business Modeling
2. Data Modeling
3. Process Modeling
4. Application Generation
5. Testing and Turnover

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 Methodology?

- When there is need to develop system within short period of time (2-**3 months).**
- When there is high availability of developers.
- When budget is high enough to afford the development cost.

Advantages of RAD Model:

- ✓ Increases re-**usability of components.**
- ✓ Less development time.
- ✓ Quick initial responses.
- ✓ Encourages customer feedback's.
- ✓ Involvement of end users from very beginning solves lot of development issues.

Disadvantages of RAD Model:

- ✓ Projects which can be developed into mini projects/components can use RAD Model.
- ✓ Requires strong and skilled developers team.
- ✓ Cost is very high..

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 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.

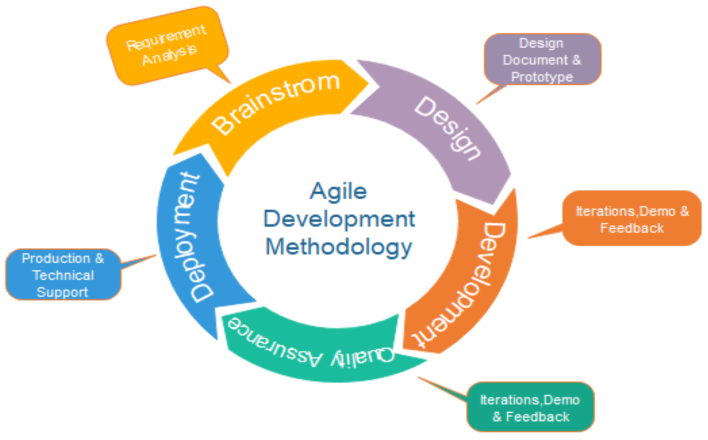

Requirement Analysis
Brainstrom
Design
Design Document & Prototype
Agile Development Methodology
Iterations,Demo & Feedback
Development
Production & Technical Support
Deployment
Quality Assurance
Iterations,Demo & Feedback

Fig. Agile Model

Phases of Agile Model:

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

- ✓ Requirements gathering
- ✓ Design the requirements
- ✓ Construction/ iteration
- ✓ Testing/ Quality assurance

- ✓ Deployment
- ✓ 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 is an umbrella, and lots of different types fall under this umbrella
- Different type like Scrum, XP and many more.
- Some people think that Agile itself is a development approach.
- But No, agile is not development approach agile is a bag which contains the lots development approaches that follow the same core theme
- The core theme is iterative development , Faster delivery, increased quality

Agile Methodologies

- **Agile Methodologies means the different approaches of agile**
    - ➢ Scrum
    - ➢ Crystal
    - ➢ Dynamic Software Development Method(DSDM)

- Feature Driven Development(FDD)
- Lean Software Development
- eXtreme Programming(XP)

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 of Agile Method:

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

Disadvantages of Agile Model:

- 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.
- 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.

Software Project Management

Software project management is an art and science of planning and leading software projects.

In which software projects are planned, implemented, monitored and controlled

What is software project planning?

Software project planning is an aspect of project management, which comprises of various processes.

The aim of these processes is to ensure that various project tasks are well coordinated and they meet the various project objectives including timely completion of the project.

Where does project planning help?

Project planning helps in

- ➢ facilitating communication
- ➢ monitoring/measuring the project progress.
- ➢ provides overall documentation of assumptions/planning decisions.

Software Project Manager

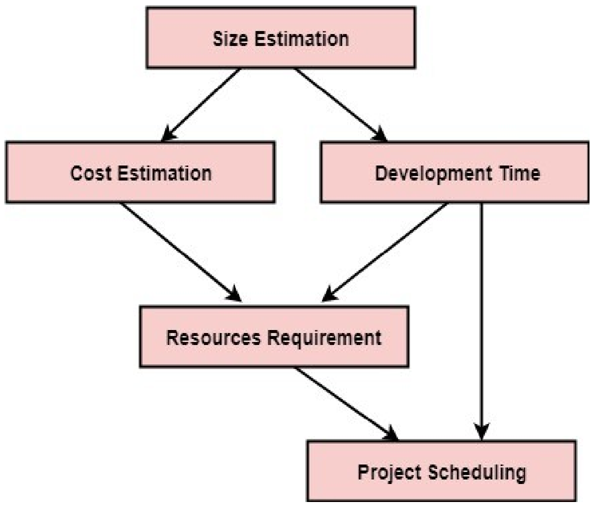
Software manager is responsible for planning and scheduling project development. They manage the work to ensure that it is completed to the required standard. They monitor the progress to check that the event is on time and within budget. The project planning must incorporate the major issues like size & 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:



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.

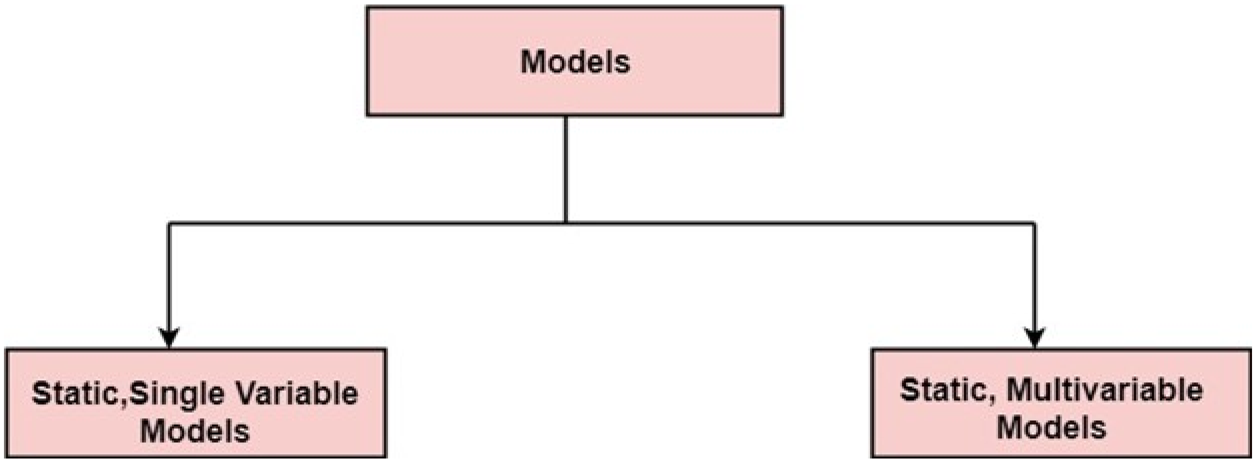
Uses of Cost Estimation

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

JA 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: **$C=aL^b$**

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

$E=1.4L^{0.93}$
$DOC=30.4L^{0.90}$
$D=4.6L^{0.26}$

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 also known as **multivariable models.** This model is often based on the first equation (eq., **$C=aL^b$**)and actually depends on several variables representing different aspects of the software development environment.

For Ex: user participator , customer oriented, etc.,

Equations are:

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

$$\mathbf{E = 5.2L^{0.91}}$$

In the same manner duration of development is given by

$$\mathbf{D = 4.1L^{0.36}}$$

Where E is in Person-months, D is duration which is months.

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

1. Number of lines of source code can be obtained by reversing equation to give:

Then

L (SEL) = (96/1.4)⅟.93=94264 LOC
L (SEL) = (96/5.2)⅟.91=24632 LOC

2. 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) = $4.1\ L^{0.36}$
= 4.1 (24.632)0.36 = 13 months

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

$$P\ (SEL) = \frac{94264}{8} = 11783\ \frac{LOC}{Person}\ -Years$$

$$P\ (Years) = \frac{24632}{8} = 3079\ \frac{LOC}{Person}\ -Years$$

4. Average manning is the average number of persons required per month in the project

$$M\ (SEL) = \frac{96P-M}{15M} = 6.4Persons$$

$$M\ (W\text{-}F) = \frac{96P-M}{13M} = 7.4Persons$$

COCOMO Model

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.(used for calculate the effort, development time, average staff size & productive)

COCOMO has three different models that reflect complexity:

1.Basic Model

2.Intermediate Model

3.Detailed Model

1.Basic Model: It computes the effort & related cost applied on the software development process as a function of program size expressed in terms of estimated lines of code (LOC or KLOC).

2. The Intermediate COCMO model: - It computes the software development effort as a function of – a) Program size, and b)A set of cost drivers that includes subjective assessments of product, hardware, personnel, and project attributes

3. The Advanced/detailed COCOMO model: - It incorporates all the characteristics of the intermediate version along with an assessment of cost driver's impact on each step of software engineering process.

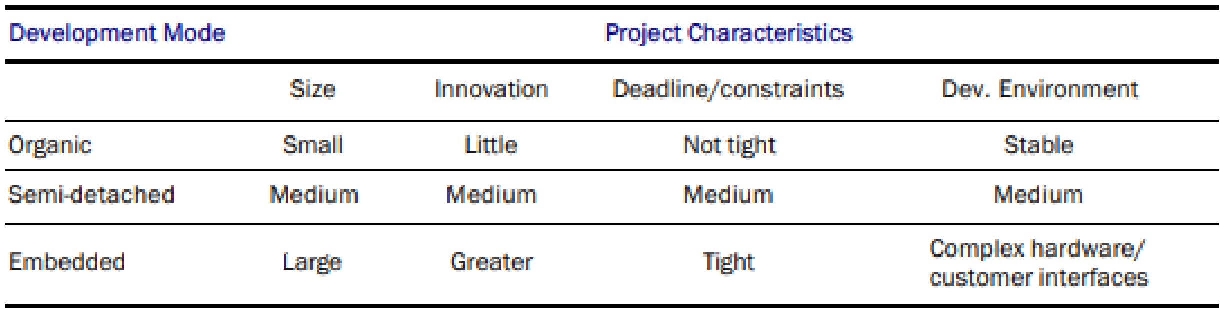
The COCOMO models are defined for three classes of software projects, stated as follows:

1.Organic projects: - These are relatively small and simple software projects which require small team structure having good application experience. The project requirements are not rigid.

2. Semi-detached projects: - These are medium size projects with a mix of rigid and less than rigid requirements level.

3. Embedded projects: - These are large projects that have a very rigid requirements level. Here the hardware, software and operational constraints are of prime importance.

There are three modes of development:



Development Mode	Project Characteristics			
	Size	Innovation	Deadline/constraints	Dev. Environment
Organic	Small	Little	Not tight	Stable
Semi-detached	Medium	Medium	Medium	Medium
Embedded	Large	Greater	Tight	Complex hardware/ customer interfaces

COCOMO'81 models depends on the two main equations

1.Development effort : **$MM = a * KLOC^{b}$** based on MM - man-month / person month / staff-month is one month of effort by one person.

2. Effort and development time (TDEV) : **$TDEV = 2.5 * MM^{c}$** The coefficents a, b and c depend on the mode of the development.

Basic COCOMO Model :

The effort equation is as follows: - $MM = a * (KLOC)^{b}$

$$D = c * (E)^{d}$$

Where E -> effort applied by per person per month,

D -> development time in consecutive months,

KLOC -> estimated thousands of lines of code delivered for the project. The coefficients a, b, and the coefficients c, d are given in the Table below

Software Project	a	b	c	D
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

Intermediate Model

This is extension of COCOMO model. This estimation model makes use of set of “Cost Driver Attributes” to compute the cost of software.

I. Product attributes

a. required software reliability

b. size of application data base

c. complexity of the product

II. Hardware attributes

a. run-time performance constraints

b. memory constraints

c. volatility of the virtual machine environment

d. required turnaround time

III. Personnel attributes

a. analyst capability

b. software engineer capability

c. applications experience

d. virtual machine experience

e. programming language experience

IV. Project attributes

a. use of software tools

b. application of software engineering methods

c. required development schedule

Project	**a**	**b**	**c**	**d**
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

Advanced/Detailed COCOMO Model

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

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) a_2 PM
Tdev=b_1*(efforts)b_2 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

$$\text{Average Staff Size (SS)} = \frac{E}{D} \text{ Persons}$$

$$= \frac{1133.12}{29.3} = 38.67 \text{ Persons}$$

$$\text{Productivity} = \frac{\text{KLOC}}{E} = \frac{200}{1133.12} = 0.1765 \text{ KLOC/PM}$$

P = 176 LOC/PM

Halstead's Software Metrics

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.

In terms of the total tokens used, the size of the program can be expressed as N = N1 + N2.

Halstead metrics are:

Program Volume (V)

The unit of measurement of volume is the standard unit for size "bits." It is the actual size of a program if a uniform binary encoding for the vocabulary is used.

$$V=N*\log_2 n$$

Program Level (L)

The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size).

$$L=V^*/ V$$

Program Difficulty

The difficulty level or error-proneness (D) of the program is proportional to the number of the unique operator in the program.

$$D= (n1/2) * (N2/n2)$$

Programming Effort (E)

The unit of measurement of E is elementary mental discriminations. $E=V/L=D*V$

Estimated Program Length

According to Halstead, The first Hypothesis of software science is that the length of a well-structured program is a function only of the number of unique operators and operands.

$N=N1+N2$

And estimated program length is denoted by N^

$N\hat{} = n1\log_2 n1 + n2\log_2 n2$

The following alternate expressions have been published to estimate program length:

- ✓ **$N_J = \log_2 (n1!) + \log_2 (n2!)$**
- ✓ **$N_B = n1 * \log_2 n2 + n2 * \log 2n1$**
- ✓ **$N_C = n1 * \mathrm{sqrt}(n1) + n2 * \mathrm{sqrt}(n2)$**
- ✓ **$N_S = (n * \log_2 n) / 2$**

Size of Vocabulary (n)

The size of the vocabulary of a program, which consists of the number of unique tokens used to build a program, is defined as:

$n=n1+n2$

where

n=vocabulary of a program
n1=number of unique operators
n2=number of unique operands

Potential Minimum Volume

The potential minimum volume V* is defined as the volume of the most short program in which a problem can be coded.

$V^* = (2 + n2^*) * \log_2 (2 + n2^*)$

Here, n2* is the count of unique input and output parameters

Program Level – To rank the programming languages, the level of abstraction provided by the programming language, Program Level (L) is considered. The higher the level of a language, the less effort it takes to develop a program using that language. **L = V* / V**

The value of L ranges between zero and one, with L=1 representing a program written at the highest possible level (i.e., with minimum size). And estimated program level is **$L^{\wedge}$ =2 * (n2) / (n1)(N2)**

Programming Effort – Measures the amount of mental activity needed to translate the existing algorithm into implementation in the specified program language. **E = V / L = D * V = Difficulty * Volume**

Language Level – Shows the algorithm implementation program language level. The same algorithm demands additional effort if it is written in a low-level program language. For example, it is easier to program in Pascal than in Assembler.

L' = V / D / D

lambda = L * V* = L^2 * V

Project Scheduling

Project scheduling in a project refers to roadmap of all activities to be done. 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.

Resources required for Development of Project :

- Human effort
- Sufficient disk space on server
- Specialized hardware

- ➢ Software technology
- ➢ Travel allowance required by project staff, etc.

Advantages:-

- ✓ Create the project team and allocating responsibility
- ✓ Determine the recourses required at a particular stage and their availability
- ✓ Manage resources by de-allocating and reallocating

Staffing / 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

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).

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.

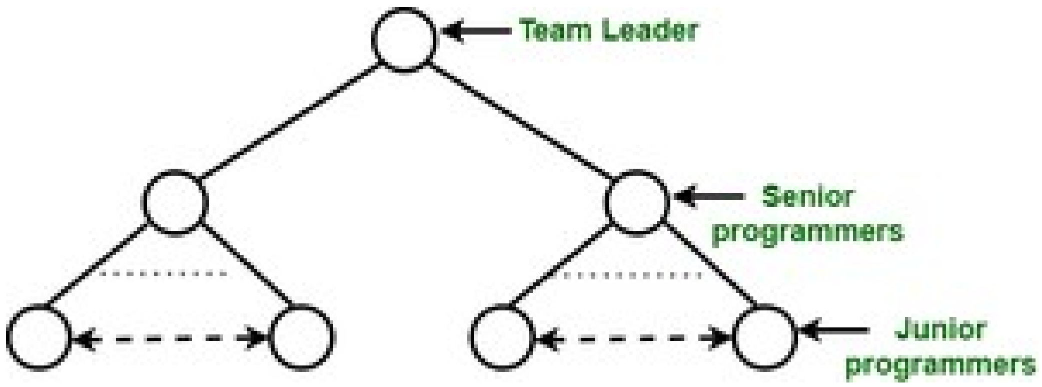
The total effort for each month and the total effort for each step can easily be calculated from this plan.

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.

1. Hierarchical team organization
2. Chief-programmer team organization
3. Matrix team, **organization**
4. Egoless team organization
5. Democratic 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 can be expanded over multiple levels.

- ✓ It is well suited for the development of the hierarchical software products.
- ✓ Large software projects may have several levels.

2.Chief-programmer team organization :

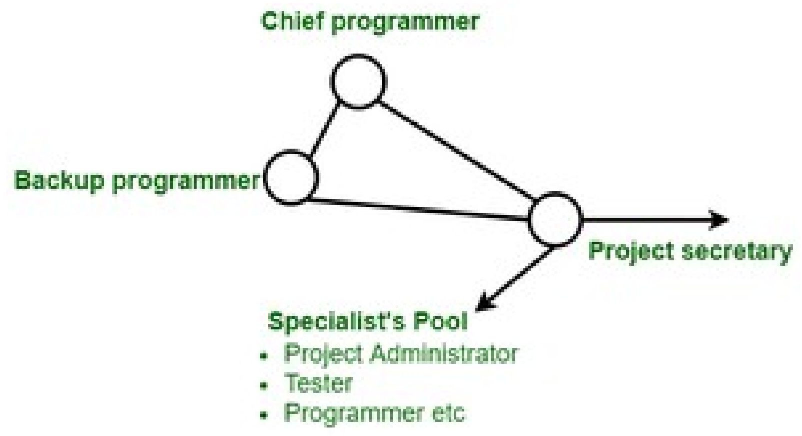
This team organization is composed of a small team consisting the following **team members :**

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 chi**ef 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.
- ✓ Team organization is limited to only small team and small team cannot handle every project.

3. **Matrix Team Organization :**
   In matrix team organization, people are divided into specialist groups. Each group has a manager. Example of Metric team organization is as follows

4. **Egoless Team Organization :**
   Egoless programming is a state of mind in which programmer are supposed to separate themselves from their product. In this team organization goals are set and decisions are made by group consensus. Here group, ‘leadership’ rotates based on tasks to be performed and differing abilities of members.

   In this organization work products are discussed openly and all freely examined all team members. There is a major risk which such organization, if teams are composed of inexperienced or incompetent members.

5. **Democratic Team Organization :**

   It is quite similar to the egoless team organization, but one member is the team leader with some responsibilities :
   - o Coordination
   - o Final decisions, when consensus cannot be reached

Advantages of Democratic Team Organization :

- ✓ Each member can contribute to decisions.
- ✓ Members can learn from each other.
- ✓ Improved job satisfaction.

Disadvantages of Democratic Team Organization :

- ✓ Communication overhead increased.
- ✓ Need for compatibility of members.
- ✓ Less individual responsibility and authority.

<u>Risk Management</u>

What is Risk Management?

Risk management is the process of identifying, assessing, and prioritizing the risks to minimize, monitor, and control the probability of unfortunate events.

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:**

   - ❑ Threaten the project plan. That is, if project risks become real, it is likely that the project schedule will slip and that costs will increase.
   - ❑ Project risks identify potential budgetary, schedule, personnel (staffing and organization), resource, stakeholder, and requirements problems and their impact on a software project.

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:

Threaten the viability of the software to be built and often jeopardize the project or the product.

Candidates for the top five business risks are

1. building an excellent product or system that no one really wants (market risk)
2. building a product that no longer fits into the overall business strategy for the company (strategic risk)
3. building a product that the sales force doesn‘t understand how to sell (sales risk)
4. losing the support of senior management due to a change in focus or a change in people (management risk)

5. losing budgetary or personnel commitment (budget risks).

Other risk categories

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. Predictable risks: Those risks that are hypothesized from previous project experience (e.g., past turnover,poor communication with customer etc.,)

3. Unpredictable risks: Those risks that can and do occur, but are extremely tough to identify in advance.

Configuration Management

What is Software Configuration Management?

In Software Engineering, **Software Configuration Management(SCM)** is a process to systematically manage, organize, and control the changes in the documents, codes, and other entities during the Software Development Life Cycle. The primary goal is to increase productivity with minimal mistakes. SCM is part of cross-disciplinary field of configuration management and it can accurately determine who made which revision.

A configuration of the product refers not only to the product's constituent but also to a particular version of the component.

Therefore, SCM is the discipline which

- ❖ Identify change
- ❖ Monitor and control change
- ❖ Ensure the proper implementation of change made to the item.
- ❖ Auditing and reporting on the change made.

Configuration Management (CM) is a technic of identifying, organizing, and controlling modification to software being built by a programming team.

The objective is to maximize productivity by minimizing mistakes (errors).

CM is used to essential due to the inventory management, library management, and updation management of the items essential for the project.

****