UNIT I

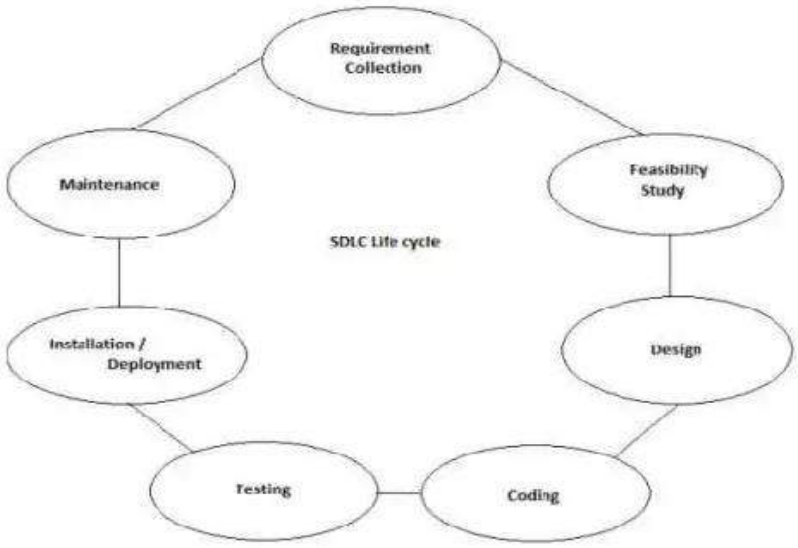
INTRODUCTION

Software is more than just a program code. A program is an executable code, which serves some computational purpose. Software is considered to be collection of executable programming code, associated libraries and documentations.

Software Engineering is an engineering branch associated with development of software product using well-defined scientific principles, methods and procedures.

→ Software Engineering process Paradigms (SDLC)

SDLC: SDLC is a step by step procedure or systematic approach to develop software and it is followed within a software organization. It consists of various phases which describe how to design, develop, enhance and maintain particular software.



Phase 1: Requirement collection and analysis:

In this phase mainly focus on gathering the business needs from the customer. Business Analyst collects the requirement from the customer and prepares the BRS (Business Requirement Specification) which has the requirement in the business form. Then a group of people sits together and determines the requirements like;

What should be input data to the system?

Who is going to use the system?

What should be output data by the system?

These questions are getting answered during this phase. After this, a Requirement Specification document is created which gives the guideline for the upcoming phase of the model.

Phase 2: Feasibility study:

Once the BRS document is completed, a set of people like Human Resource department, Finance department, Business analyst, Architect and Project manager are sit together and analyze if the project is do able or not. This decision is taken based on the cost, time, resources and etc.

Phase 3: Design:

In this phase system design specification is prepared from the requirement document. Design is a blue print of the application and it helps in specifying hardware and requirements of the system and helps in defining architecture of the system

Phase 4: Coding:

Once the system design document is ready, in this phase developer's starts writing the code using any programming language i.e., they start developing the software. Generally task is divided in units or modules and assigned to the developers and this coding phase is the longest phase of SDLC.

Phase 5: Testing:

Once the software is ready and is deployed in the testing environment, test engineers starts testing, if the functionality of an application is working according to requirement or not. During this phase test engineers may encounter some bugs/defects which need to be sent to developers, the developers fix the bug and sent back to test engineers for testing. This process continuous until the software is bug free/stable/working according to the requirement.

Phase 6: Installation/Deployment:

Once the product developed, tested and works according to the requirement it is installed / deployed at customer place for their use.

Phase 7: Maintenance:

When the customers starts using the software they may face some issues and needs to be solved, to fix those issue, tested and handed over back to the customer as soon as possible, which is done in the maintenance phase.

→ Waterfall Model

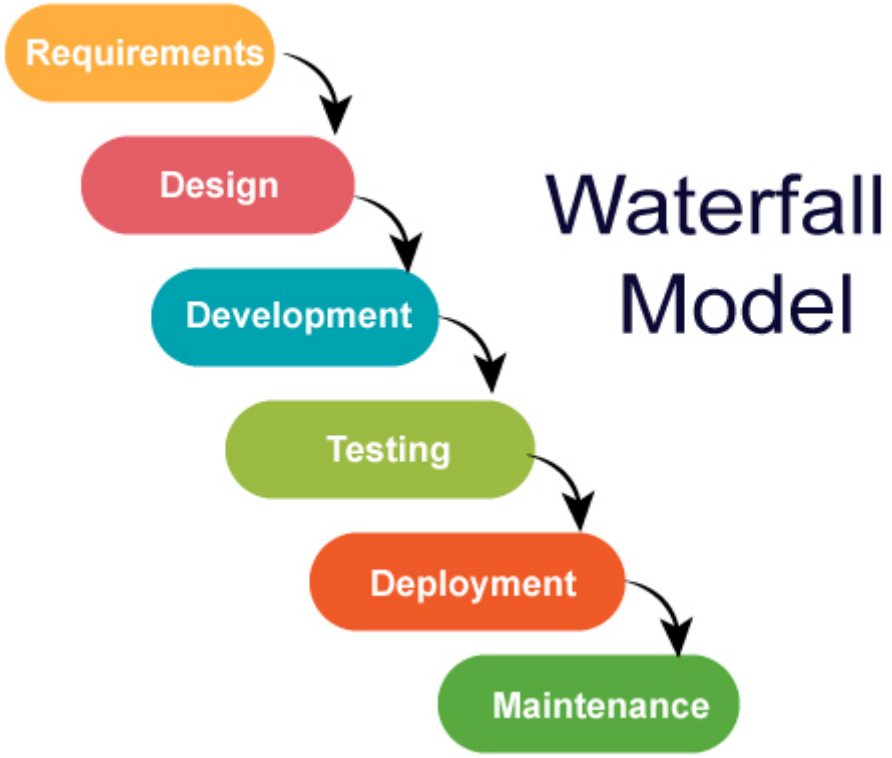
The Waterfall Model was the first Process Model to be introduced. It is also referred to as a **linear-sequential life cycle model**. It is very simple to understand and use. In a waterfall

model, each phase must be completed before the next phase can begin and there is no overlapping in the phases.

Waterfall Model-Design

In "The Waterfall" approach, the whole process of software development is divided into separate phases. In this Waterfall model, typically, the outcome of one phase acts as the input for the next phase sequentially.

The following illustration is a representation of the different phases of the Waterfall Model.



The sequential phases in Waterfall model are –

- **Requirement Gathering and analysis** − All possible requirements of the system to be developed are captured in this phase and documented in a requirement specification document.
- **System Design** − The requirement specifications from first phase are studied in this phase and the system design is prepared. This system design helps in specifying hardware and system requirements and helps in defining the overall system architecture.

- **Implementation** − With inputs from the system design, the system is first developed in small programs called units, which are integrated in the next phase. Each unit is developed and tested for its functionality, which is referred to as Unit Testing.
- **Integration and Testing** − All the units developed in the implementation phase are integrated into a system after testing of each unit. Post integration the entire system is tested for any faults and failures.
- **Deployment of system** − Once the functional and non-functional testing is done; the product is deployed in the customer environment or released into the market.
- **Maintenance** − There are some issues which come up in the client environment. To fix those issues, patches are released. Also to enhance the product some better versions are released. Maintenance is done to deliver these changes in the customer environment.

Waterfall Model – Advantages

- Simple and easy to understand and use
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Easy to arrange tasks.

Waterfall Model – Disadvantages

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

→ Iterative Model:

1. **Planning Phase:** This is the first stage of the iterative model, where proper planning is done by the team, which helps them in mapping out the specifications documents, establish software or hardware requirements and generally prepare for the upcoming stages of the cycle.

2. **Analysis and Design Phase:** Once the planning is complete for the cycle, an analysis is performed to point out the appropriate business logic, database models and to know any other requirements of this particular stage. Moreover, the design stage also occurs in this phase of iterative model, where the technical requirements are established that will be utilized in order to meet the need of analysis stage.

3. **Implementation Phase:** This is the third and the most important phase of the iterative model. Here, the actual implementation and coding process is executed. All planning, specification, and design documents up to this point are coded and implemented into this initial iteration of the project.

4. **Testing Phase:** After the current build iteration is coded and implemented, testing is initiated in the cycle to identify and locate any potential bugs or issues that may have been in the software.

5. **Evaluation Phase:** The final phase of the Iterative life cycle is the evaluation phase, where the entire team along with the client, examine the status of the project and validate whether it is as per the suggested requirements.

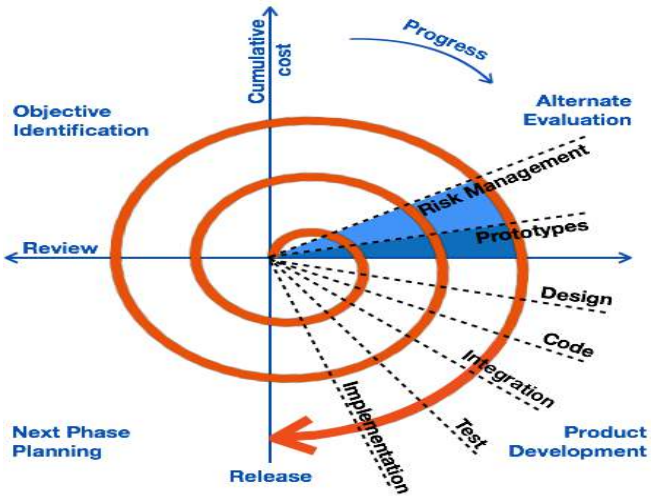
Advantages of Iterative Model:

- It is easily adaptable to the ever changing needs of the project as well as the client.
- It is more cost effective to change the scope or requirements in Iterative model.
- Parallel development can be planned.
- Testing and debugging during smaller iteration is easy.
- Risks are identified and resolved during iteration; and each iteration is an easily managed.
- In iterative model less time is spent on documenting and more time is given for designing.

Disadvantages of Iterative Model:

- More resources may be required.
- More management attention is required.
- It is not suitable for smaller projects.
- Highly skilled resources are required for skill analysis.
- Project progress is highly dependent upon the risk analysis phase.

→ Spiral Model



Spiral model is a combination of sequential and prototype model. This model is best used for large projects which involves continuous enhancements. There are specific activities which are done in one iteration (spiral) where the output is a small prototype of the large software. The same activities are then repeated for all the spirals till the entire software is build. **A spiral model has 4 phases described below:**

1. Planning phase
2. Risk analysis phase
3. Engineering phase
4. Evaluation phase.

Activities which are performed in the spiral model phases are shown below:

Phase Name	**Activities performed**	**Deliverables / Output**
Planning	-Requirements are studied and gathered. - Feasibility study - Reviews and walkthroughs to streamline the requirements	Requirements understanding document Finalized list of requirements.
Risk Analysis	Requirements are studied and brain storming sessions are done to identify the potential risks Once the risks are identified , risk mitigation strategy is planned and finalized	Document which highlights all the risk & its mitigation plans.
Engineering	Actual development and testing if the software takes place in this phase	Code Test cases and test results Test summary report and defect report.
Evaluation	Customers evaluate the software and provide their feedback and approval	Features implemented document

Advantages of using Spiral Model:

- Development is fast
- Larger projects / software are created and handled in a strategic way
- Risk evaluation is proper.
- More and more features are added in a systematic way.
- Software is produced early.

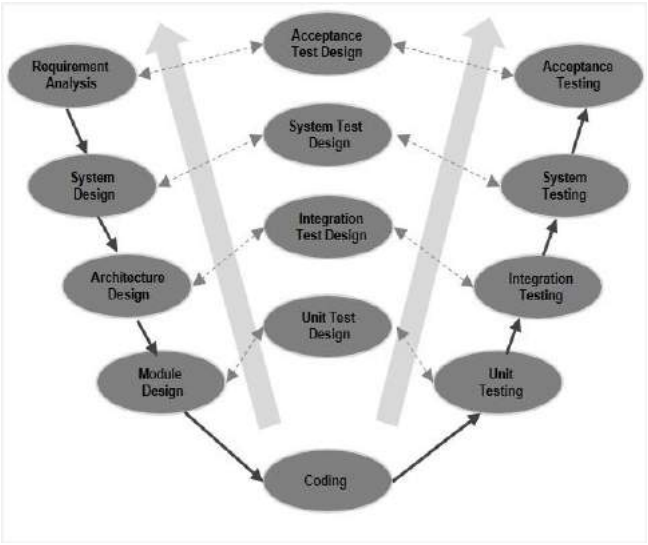
Disadvantages of using Spiral model:

- Risk analysis is important phase so requires expert people.
- Is not beneficial for smaller projects.
- Spiral may go infinitely.
- Documentation is more as it has intermediate phases.
- It is costly for smaller projects.

→V-Model

The V-model is an SDLC model where execution of processes happens in a sequential manner in a V-shape. It is also known as **Verification and Validation model**.

The following illustration depicts the different phases in a V-Model of the SDLC.



V-Model - Verification Phases

There are several Verification phases in the V-Model, each of these are explained in detail below.

Business Requirement Analysis

This phase involves detailed communication with the customer to understand expectations and exact requirement. The **acceptance test design planning** is done at this stage as business requirements can be used as an input for acceptance testing.

System Design

The system design will have the understanding and detailing the complete hardware and communication setup for the product under development. The system test plan is developed based on the system design.

Architectural Design: Architectural specifications are understood and designed in this phase. Usually more than one technical approach is proposed and based on the technical and financial feasibility the final decision is taken. The system design is broken down further into modules taking up different functionality. This is also referred to as **High Level Design (HLD)**.

Module Design

In this phase, the detailed internal design for all the system modules is specified, referred to as **Low Level Design (LLD)**. It is important that the design is compatible with the other modules in the system architecture and the other external systems. The unit tests are an essential part of any development process and helps eliminate the maximum faults and errors at a very early stage. These unit tests can be designed at this stage based on the internal module designs.

Coding Phase

The coding is performed based on the coding guidelines and standards. The code goes through numerous code reviews and is optimized for best performance before the final build is checked into the repository.

Validation Phases

The different Validation Phases in a V-Model are explained in detail below.

Unit Testing

Unit testing is the testing at code level and helps eliminate bugs at an early stage, though all defects cannot be uncovered by unit testing.

Integration Testing

Integration testing is associated with the architectural design phase.

System Testing

System tests check the entire system functionality and the communication of the system under development with external systems.

Acceptance Testing

Acceptance tests uncover the compatibility issues with the other systems available in the user environment.

The advantages of the V-Model method are as follows −

- This is a highly-disciplined model and Phases are completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Simple and easy to understand and use.

The disadvantages of the V-Model method are as follows −

- High risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Poor model for long and ongoing projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing.

****

→Project Management Concepts (Or) Spectrum

- The People
- The Product (or) problem
- The Process
- The Project

The People: The Stakeholders

Four categories of stakeholders

- Senior managers – define business issues that often have significant influence on the project.
- Project (technical) managers – plan, motivate, organize, and control the practitioners who do the work.
- Customers – specify the requirements for the software engineer.
- Users – interact with the software once it is released for production use

The People: Team Leaders

- Team leaders should use a problem-solving management style.
  - Concentrate on understanding the problem to be solved
  - Manage the flow of ideas.

The People: The Software Team

The People: Coordination and Communication Issues

Documents, Milestones, Memos, Review Meetings, Inspections , Information Meetings, Problem Solving ,E-Mail, Bulletin Boards, Video Conferencing ,Discussion With People Outside Project Team

The Product

- The scope of the software development must be established and bounded
  - **Context** – How does the software to be built fit into a larger system? And what constraints are imposed as a result of the context?
  - **Information objectives** – What customer-visible data objects are produced as output from the software? What data objects are required for input?
  - **Function and performance** – What functions does the software perform to transform input data into output? Are there any special performance characteristics to be addressed?

The Process

The project manager must decide which process model is most appropriate based on framework activities.

- Customer communication
- Planning
- Risk analysis

- Engineering
- Customer evaluation

Once a process model is selected, a preliminary project plan is established based on the process framework activities.

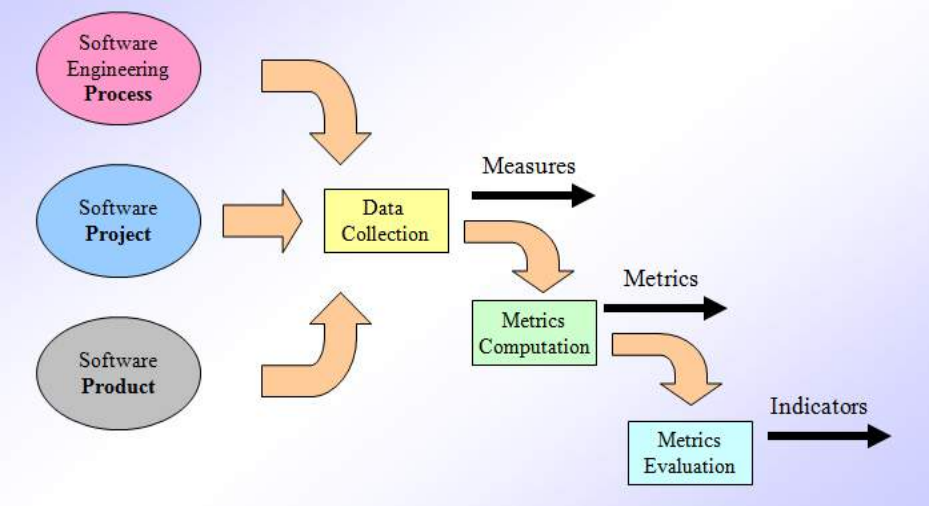
The Project

- Defining and setting up project scope
- Managing project management activities
- Monitoring progress and performance
- Risk analysis at every phase
- Take necessary step to avoid or come out of problems

****

Process and Project metrics

Measure, Metrics, Indicators



- **Measure.**
  - Provides a quantitative indication of the amount, dimension, capacity, or size of some attributes of a product or process.
- **Metrics** Relates the individual measures in some way.(quantitative measure of the degree to which a system i.e, The goal of software metrics is to identify and control essential parameters that affect software development.)

- **Indicator.**
  - These indicators provide a detailed insight into the software process, software project, or intermediate product. Indicators also enable software engineers or project managers to adjust software processes and improve software products, if required.

Process and project Indicator

Process Indicator

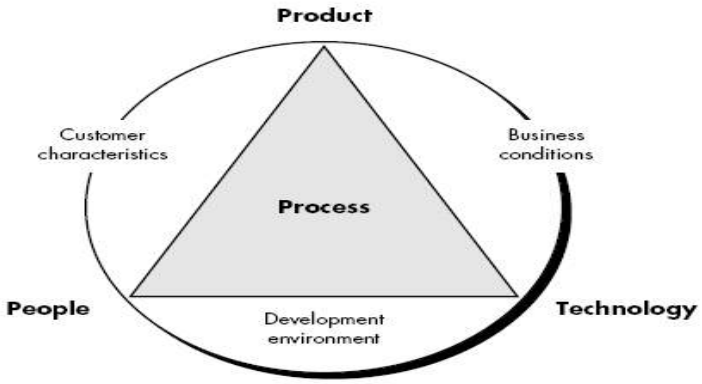
Process Indicators are collected across all projects and over long periods of time. Their intent is to provide indicators that lead to long term software process improvement.

Project Indicator

Enables a software project manager to

1) Assess the status of an ongoing project
2) Track potential risks.
3) Uncover problem areas before they go "Critical"
4) Adjust work flow or tasks
5) Evaluate the project team's ability to control quality of software work products.

Process Metrics and Software Process Improvement



Factors that influence quality:

- people - skills and experience of SW people
- technology - used in development (e.g. CASE)
- product complexity

Types of process metrics:

Private &. public metrics

SW process improvement should begin at the individual level

Private metrics:

- defect rates by individual
- defect rates by module
- errors found during development

Public metrics:

Use information from individual and team metrics

Some public metrics:

- project-level defect rates
- effort
- Calendar times

Software Measurement

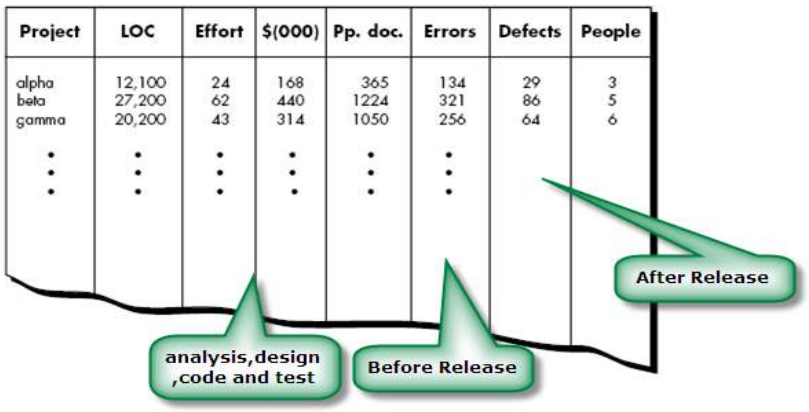
Categories in 2 ways:

- ***Direct measure*** of the software process & Product
  - E.g. Lines of code (LOC), execution speed, and defect)
- ***Indirect measures*** of the product that includes functionality, complexity, efficiency, reliability, maintainability etc.

Size Oriented Metrics

Size-oriented software metrics are derived by normalizing quality and/or productivity measures by considering the **size** of the **software** that has been produced. A set of simple **size-oriented metrics** can be developed for each project: Errors per KLOC (thousand lines of code). Defects4 per KLOC.

Project	LOC	Effort	$(000)	Pp. doc.	Errors	Defects	People
alpha	12,100	24	168	365	134	29	3
beta	27,200	62	440	1224	321	86	5
gamma	20,200	43	314	1050	256	64	6
.	.	.	.	.	.		
.	.	.	.	.	.		
.	.	.	.	.	.		



Size-oriented metrics measures on LOC as normalization value.

- Errors per KLOC (thousand lines of code)
- Defects per KLOC
- $ per LOC
- Pages of documentation per KLOC

Function-Oriented Metrics

- It uses a measure of functionality delivered by the application as a normalization value.
- Function Point (FP) is widely used as function oriented metrics.
- FP derived using an empirical relationship based on countable (direct) measures of software's information domain and assessments of software complexity.
- FP is based on characteristic of Software information domain and complexity.

measurement parameter	count	weighting factor simple	avg.	complex	
number of user inputs		X 3	4	6	=
number of user outputs		X 4	5	7	=
number of user inquiries		X 3	4	6	=
number of files		X 7	10	15	=
number of ext.interfaces		X 5	7	10	=
count-total →					
complexity multiplier					
function points →					

Number of user inputs

Each unique user input that provides application-oriented data to the SW.

Number of user outputs

Each user output that provides application-oriented information to user (reports, screens, error messages, etc.).

Number of inquiries

Inquiry is an on-line input that results in generation of an immediate SW response in form of an on-line output.

Number of internal files

Include each logical file or if using a DB, logical grouping of data, that is generated, used and maintained by the application.

Number of external interfaces

Files passed or shared between applications should be counted.

Compute: the function points calculation

$$\textbf{FP= count-total * [0.65+.01* } \Sigma \textbf{ Fi]}$$

Reconciling LOC and FP metric:

- Relationship between lines of code and function points depends upon the programming language that is used to implement the software and the quality of the design.
- Following table provides rough estimates of the average number of LOC required to build one FP in various programming languages:

Programming Language	LOC/FP (average)
Assembly language	320
C	128
COBOL	106
FORTRAN	106
Pascal	90
C++	64
Ada95	53
Visual Basic	32
Smalltalk	22
Powerbuilder (code generator)	16
SQL	12

Metrics for SW Quality

Focus on the process, the project and the product (as do productivity metrics)

Factors that affect quality

- product operation - using it
- product revision - changing it
- product transition - portability

Measuring quality:

- **correctness**
- **degree to which SW performs its required function**
    - Defects per KLOC - most common measure for correctness.
    - Maintainability.
    - Ease with which a program can be corrected, adapted, or enhanced.
        - **MTTC - mean time to change -**

Simple metric - time it takes to analyze, implement change, test it, and distribute it to users.

- **Integrity**
    - Measures system's ability to withstand attacks on its security.
- **Usability**
    - Quantify user friendliness.

Defect Removal Efficiency(DRE)

- Defect removal efficiency provides benefits at both the project and process level.
- It is a measure of the filtering ability of QA activities as they are applied throughout all process framework activities.
    - It indicates the percentage of software errors found before software release.
- It is defined as DRE = E / (E + D).
    - E is the number of errors found before delivery of the software to the end user.
    - D is the number of defects found after delivery.

****

→SOFTWARE PROJECT ESTIMATION

Estimation is the process of finding an estimate, or approximation, which is a value that can be used for some purpose even if input data may be incomplete, uncertain, or unstable.

Estimation determines how much money, effort, resources, and time it will take to build a specific system or product. Estimation is based on −

- Past Data/Past Experience
- Available Documents/Knowledge
- Assumptions
- Identified Risks

Estimation need not be a one-time task in a project. It can take place during −

  - Acquiring a Project.
  - Planning the Project.
  - Execution of the Project as the need arises.

Project Estimation Approach

The Project Estimation Approach that is widely used is **Decomposition Technique**. Decomposition techniques take a divide and conquer approach. Size, Effort and Cost estimation are performed in a stepwise manner by breaking down a Project into major Functions or related Software Engineering Activities.

Step 1 − Understand the scope of the software to be built.

Step 2 − Generate an estimate of the software size.

Step 3 − Generate an estimate of the effort and cost. You can arrive at the effort and cost estimates by breaking down a project into related software engineering activities.

Step 4 − Reconcile estimates: Compare the resulting values from Step 3 to those obtained from Step 2. If both sets of estimates agree, then your numbers are highly reliable.

Step 5 − Determine the cause of divergence and then reconcile the estimates.

****

→EMPERICAL ESTIMATION MODELS

The structure of empirical estimation models is a formula, derived from data collected from past software projects, that uses software size to estimate effort. Size, itself, is an estimate, described as either lines of code (LOC) or function points (FP). No estimation model is appropriate for all development environments, development processes, or application types. Models must be customised (values in the formula must be altered) so that results from the model agree with the data from the particular environment.

The typical formula of estimation models is: $E = a + b(S)c$

where;

E represents effort, in person months,

S is the size of the software development, in LOC or FP, and,

a, b, and c are values derived from data

COCOMO: When Barry Boehm wrote 'Software Engineering Economics', published in 1981, he introduced an empirical effort estimation model (COCOMO - COnstructive COst MOdel) that is still referenced by the software engineering community. The model has been reviewed since 1981 and details of the revised and updated COCOMO 2 model.

The original COCOMO model was a set of models; 3 development modes (organic, semi-detached, and embedded) and 3 levels (basic, intermediate, and advanced). COCOMO model levels:

Basic - predicted software size (lines of code) was used to estimate development effort.

Intermediate - predicted software size (lines of code), plus a set of 15 subjectively assessed 'cost drivers' was used to estimate development effort

Advanced - on top of the intermediate model, the advanced model allows phase-based cost driver adjustments and some adjustments at the module, component, and system level.

COCOMO development models:

Organic - small relatively small, simple software projects in which small teams with good application experience work to a set of flexible requirements.

Embedded - the software project has tight software, hardware and operational constraints.

Semi-detached – an intermediate (in size and complexity) software project in which teams with mixed experience levels must meet a mix of rigid and less than rigid requirements.

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:

$$\text{Effort}=a_1*(\text{KLOC})\ a_2\ \text{PM}$$

$$\text{Tdev}=b_1*(\text{efforts})b_2\ \text{Months}$$

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

****

→Project Planning

The project plan sets out the resources available about to the project, the work breakdown and a schedule for carrying out the work.

Most plans should include the following sections:

1. **Introduction:** This briefly describes the objectives of the project and sets out the constraints (Eg: budget, time etc) which affect the project management.
2. **Project Organisation:** This describes the way in which the development team is organized, the people involved and their roles in the team.
3. **Risk analysis:** This describes possible project risks, the likelihood of these risks arising and the risk reduction strategies, which are proposed.
4. **Hardware and Software resource Requirements :** This describes the hardware and the support software required to carry out the development .
5. **Work Breakdown :** This describes the breakdown of the project into activities and identifies the milestones ans deliverables associated with each activity.
6. **Project Schedule:** This describes the dependencies between activities , the estimated time required to reach each milestone and the allocation of people to the activities.
7. **Monitoring and reporting mechanism :** This describes the management reports which should be produced , when these should be produced and the project monitoring mechanisms used.

****

→Risk Management

Risk Management can be defined as follows:

- **Project risks**: are risk s which are affect the project schedule or resources.
- **Product risks:** are risks which are affect the quality or performance of the software being developed.
- **Business Risks :** are risks which affect the organization developing or procuring the software.

The process of risk management is involves in several stages:

1. **Risk identification:** Possible project, product and business risks are identified.
2. **Risk Analysis:** The likelihood and consequences of these risks are assessed.
3. **Risk Planning :** Plans to address the risk either by avoiding it or minimizing its effects on the project are drawn up.
4. **Risk Monitoring :** The risk is constantly assessed and plans for risks mitigation are revised as more information about risk becomes available.

1 . Risk Identification: This is the first stage of risk management. These types include

- ➔ **Technology risks:** Risks which are defines from the software or Hardware technologies
- ➔ **People risks:** Risks which are associated with the people in the development team.
- ➔ **Organisational risks :** Risks which are derive from the organizational environment where software is being developed.

- ➔ **Tool risks:** Risks which derive from the CASE tools and other support software used to develop the system.
- ➔ **Requirement risks:** Risks which are derive from changes to the customer's requirements and the process of managing the requirements change.
- ➔ **Estimation risks:** Risks which are derive from the management estimates of the system characteristics and the resources required to build the system.

2. Risk Analysis: During this risk analysis process, each identified risk is considered in turn and a judgment made about the probability and the seriousness of the risk..

Once the risks have been analyzed and ranked , a judgment must then be made about which are the most important risks which are the most important risks which must be considered during the project.

3. Risk Planning: These strategies fall into three categories:

- ➔ **Avoidance strategies :** The probability that the risk will arise will be reduced.
- ➔ **Minimization strategies :** The impact of the risk will reduced.
- ➔ **Contingency plans :** If the worst happens , prepared for it and have a strategy in place to deal with it.

4. Risk Monitoring : It involves regularly assessing each of the identified risks to decide whether or not that risk is becoming more or less probable and whether the effects of the risk have changed.

Risk monitoring should be a continuous process and, at every management progress review , each of the key risk should be considered separately and discussed by the meeting.

****

➔Project Scheduling:

Project scheduling involves separating the total work involved in a project into separate and judging the time required to complete these activities. Usually, some of these activities are carried out in parallel.

In estimating schedules , managers should not assume that every stage of the project will be problem free.

Two project scheduling methods:

- **Program Evaluation and Review Technique (PERT)** is a project management tool used to schedule, organize, and coordinate tasks within a project. It is basically a method to analyze the tasks involved in completing a given project, especially the time needed to complete each task, and to identify the minimum time needed to complete the total project.

- **Critical Path Method (CPM)** is an algorithm for scheduling a set of project activities. It is commonly used in conjunction with the program evaluation and review technique (PERT).

Both methods are driven by information developed in earlier project planning activities:

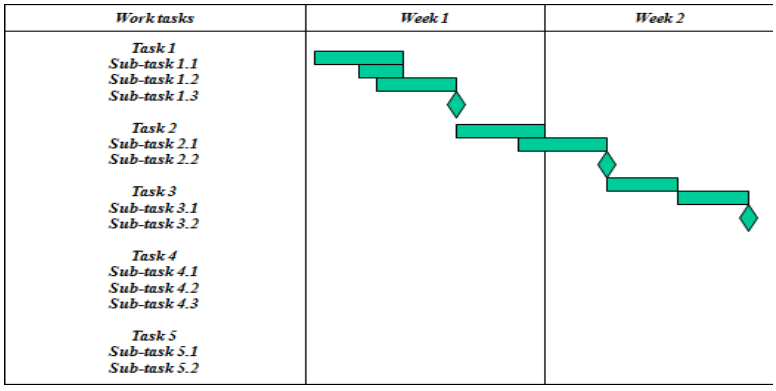
- Estimates of effort.
- A decomposition of product function.
- The selection of the appropriate process model.
- The selection of project type and task set.

Timeline Charts (Gantt charts)

Timeline charts, or also called *Gantt charts*, are developed for the entire project, for tracking and control of all activities that need to be performed for project development.

The timeline chart is a kind of a table with the following fields:

- The left hand column contains the project tasks
- The horizontal bars indicate the duration of each task
- The diamonds indicate milestones



Tracking the Schedule

The project schedule provides a road map for a software project manager. It defines the tasks and milestones.

Several ways to track a project schedule:

- conducting periodic project status meeting.
- evaluating the review results in the software process.
- determine if formal project milestones have been.

- compare actual start date to planned start date for each task.

- Informal meeting with practitioners.

Project manager takes the control of the schedule in the aspects of:

- project staffing - Project problems

- Project resources - Reviews - Project budget

****