PAGE No
DATE

Software Engineering

Software:- It is a program/ set of programs containing instructions which provide desired functionality.

Engineering: The process of designing and building something that serves a particular purpose.

Software engineering
- An engineering discipline which is concerned with all aspects of soft-ware production.

Software engineers should-
a) adopt a systematic and organised approach to their work
b) use appropriate tools and techniques
c) use the resources avaible
- At first conference of SE in 1968, Fritz Bauer defined SE as:- The establishment and use of sound engineering principles in order to obtain economically developed software that is reliable and works efficiently on real machines.
- Acc. to stephen schach:- A discipline whose aim is the production of quality software, software that is delivered on time, within budget, and that satisfies its requirements

Why Software Engineering is Important:-
1) Enables us to build complex system in timely manner.
2) Enable high quality of software.
3) Impart discipline to work.

Manuals in Documentation

Documentation Manual
- Analysis/specification
  - Formal specification
  - content - Diagram
  - Data flow diagrams
- Design
  - Flow charts
  - Entity - Relationship Diagram
- Implementation
  - Source code listings
  - cross - Reference listings
- Testing
  - Test - Data
  - Test Results

Need of software engineering -

1) Reduce complexities :- It divides big problems into various small issues and then start solving each small problem one by one.
2) Decrease time :- if you are making software according to the software engineering method, then it will decrease lot of time.

Characteristics -

1) Functionality
- It refers to the degree of performance of the software against its intended purpose. It basically means the required functions.

2) Reliability
- A set of attributes that bear on the capability of software to maintain its level of performances underlated conditions for a stated period of time.

3) Efficiency
- ability of software to use system resources in most effective and efficient manner.

4) Usability
- refers to the extent to which the software can be used with easi. Or the amount of effort or time required to learn how to use software should be less.

5) Maintainability
- refers to ease with which modifications can be made in software system to extend its functionality, improvement, performance or correct errors.

6) Portability
- a set of attributes that bears on the ability of the software to be transferred from one environment to another, without or minimum changes.

Software Development life cycle
- software development life cycle (SDLC) is a framework that defines the steps involve in the development of software.
- It covers the detailed plan for building, deploying and maintaining the software.
- Purpose of SDLC is to deliver a high quality product which is as per customer's requirement.

SDLC phases:-

(i) Requirement gathering and analysis
(ii) Design
(iii) Implementation or coding
(iv) Testing
(v) Deployment
(vi) Maintenance

→ Waterfall Model

Requirement analysis

System Design

Implementation

Testing

Deployment

Maintenance

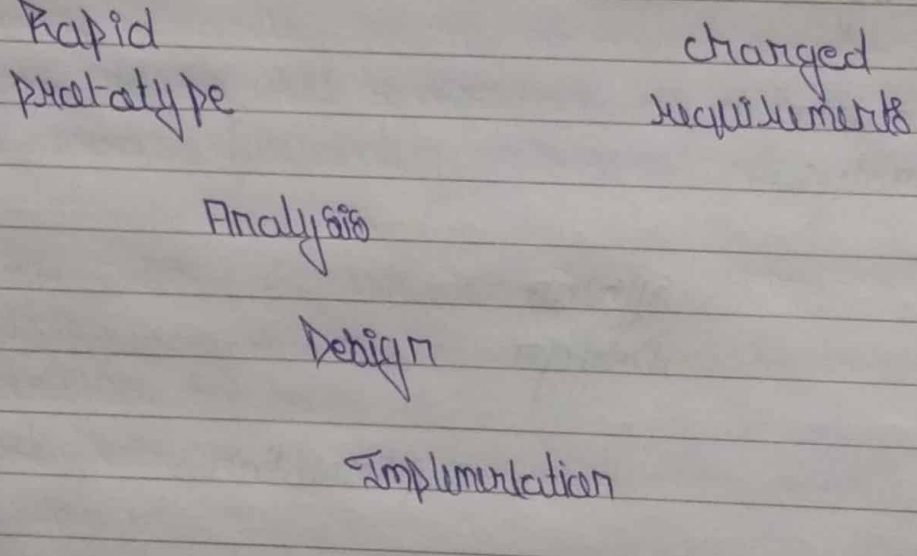
Drawbacks of waterfall Model:

(i) The model expects complete and accurate requirements early in the process which is unrealistic.
(ii) We can't go back in process of waterfall model.

PAGE No
DATE

Prototype Model

- used when the customers do not know the exact project requirements beforehand.
- In this, a prototype of end model is first developed, tested and refined as per customer feedback repeatedly till a final ~~project~~ acceptable prototype is achieved which forms the basis for developing the final project.



Advantages -

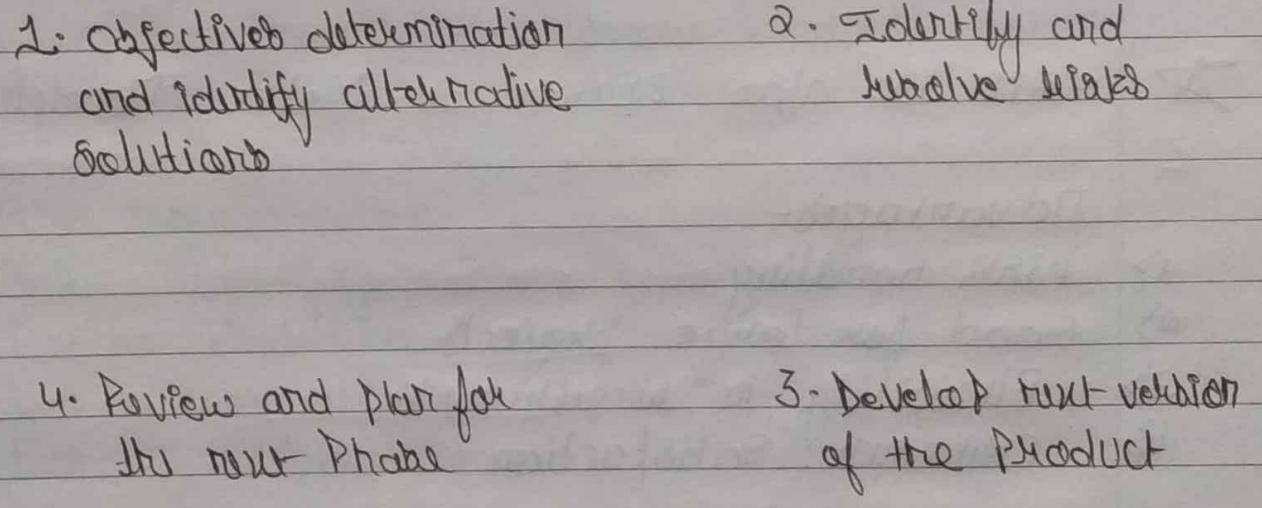
- New requirements can easily be accomodated as there is scope for refinement.
- Missing functionalities can easily be figured out.
- ☆ The developed prototype can be reused by the developer for more complicated projects in future.

PAGE No
DATE

Disadvantages -
- costly w.r.t time as well as money.
- Poor documentation due to continuously changing customer requirements

Spiral Model
- provides support for risk handling.
- Each loop of the spiral is called Phase of the software Development process.

* The radius of the spiral at any point represents the expenses of the project so far, and the angular dimension represents the progress made so far in the current phase.



1- objectives determination and identify alternative solutions:
- Requirements are gathered from the customers and objectives are identified, elaborated, and analyzed at the start of every phase.

2) Identify and resolve risks
- all possible solutions are evaluated to select the best possible solution.
- then the risks associated with the solution are identified and the risks are resolved using best possible strategy.

- At the end of this quadrant, the prototype is built for the best possible cost.

3- Develop next version of software/product-
- in this, identified features are developed and verified through testing.
- At the end, the next version of software is available.

4) Review and plan for the next phase.
- the customer evaluate the so far developed version of soft-ware. In the end, planning for next-phase started

* This is also known as Meta Model

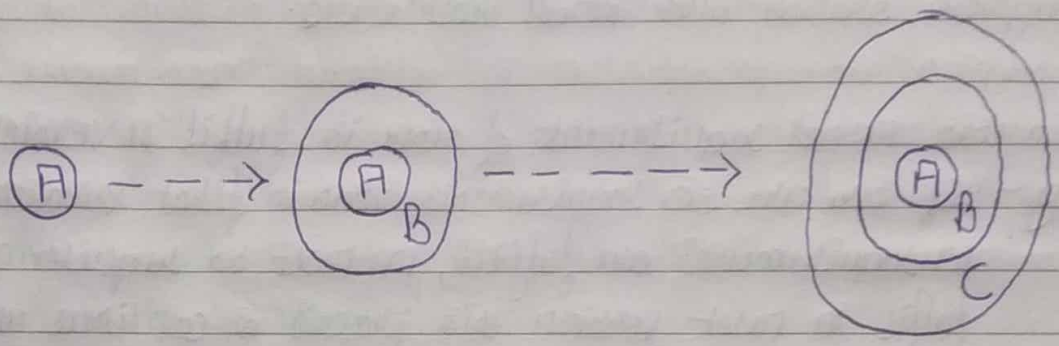
Advantages-
1) Risk handling
2) Good for large projects
3) Flexibility in requirements
4) Customer satisfaction

Disadvantages -
1) complex
2) Expensive
3) Too much dependability on Risk analysis
4) Difficulty in time management

PAGE No
DATE

Incremental process model

- also known as successive version model.



- First, a simple working system implementing only a few basic features is built and then that is delivered to customer. Then thereafter many successive iterations/versions are implemented and delivered to the customer until desired system is released.
- A, B, C are modules of software product that are incrementally developed and delivered.

A → Communication
B → Planning
C → Modelling
D → Construction
E → Deployment

Advantages -

- Work with small team
- Initial product delivery is faster.
- Can accomodate changes.
- Customer response / feedback is considered.

PAGE No
DATE

Disadvantage
- Actual cost may exceed the estimated cost.
- System broken into small instructions.

In this model requirement of user is fulfil in versions.
eg - If there are 50 requirements then in first version of software 10 requirements are fulfill, in next 10 requirement are fulfil in next version this process is continue until all requirement are not fulfil.

- Sequence of requirement in version is divided by user than that what requirements he wants to fulfill in first version.

AGILE Software Development

- primarily designed to help a project to adapt to change requests quickly.
- main aim of Agile model is to facilitate quick project completion.
- agility is achieved by fitting the process to the project, removing activities that may not be essential for a specific project.
- Agile model refers to a group of development processes. A few Agile SDLC models are given below:

→ crystal
→ atern
→ Feature-driven development
→ scrum
→ unified process

- In this, the requirements are decomposed into many small parts that can be incrementally developed. Agile model iterative development.
- Each incremental part is developed over an iteration.
- Each iteration is intended to be small and easily manageable and that can be completed within a couple of weeks only.

Agile model is the combination of iterative and incremental process models. Steps involve in agile SDLC models are:

→ Requirement gathering
→ Requirement Analysis
→ Design
→ coding
→ Unit testing
→ Acceptance testing

The time to complete an iteration is known as a Time Box.
The critical principle of the Agile model is the delivery of an increment to the customer after each time-box.

Principles of Agile model

- Agile model relies on working software development rather than comprehensive document.
- Frequent delivery of incremental versions of software to the customer representative in intervals of few weeks.

Advantages-

- It reduces total development time of the whole project.
- Customer representatives get the idea of updated software products after each iteration. So, it is easy for him to change any requirement if needed.

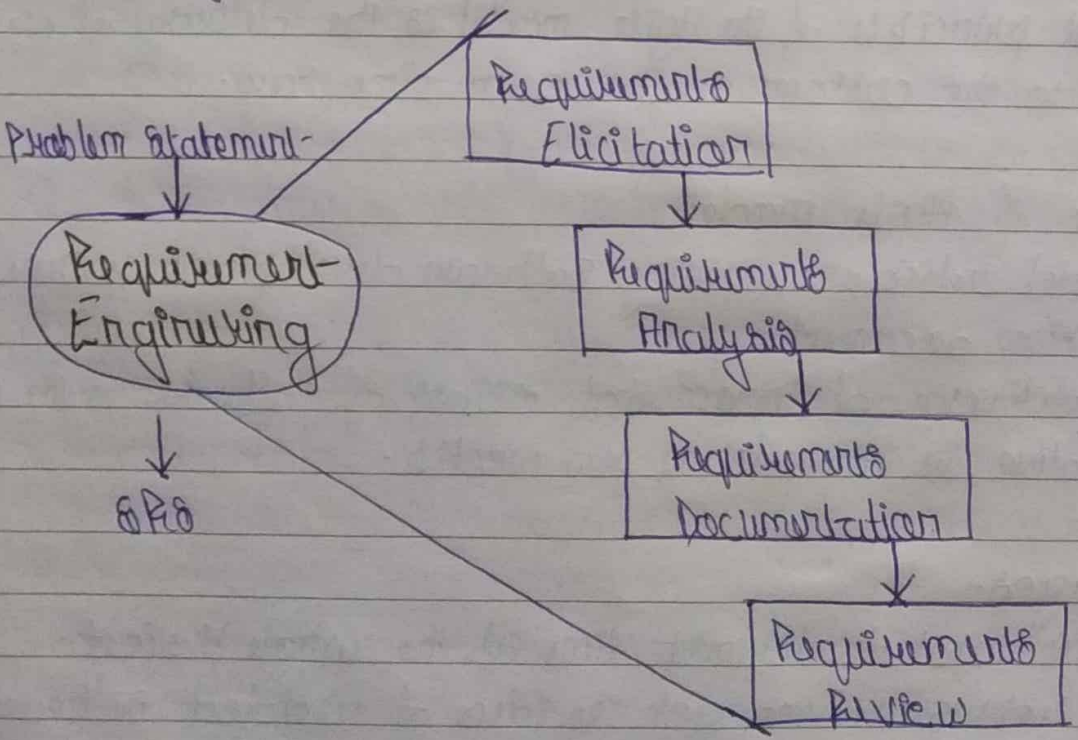
Disadvantages-

- Due to lack of formal documents, it creates confusion and important decisions taken during different phases can be misinterpreted at any time by different team members.
- Due to the absence of proper documentation, when the project completes and the developers are assigned to another project, maintenance of the developed project can become a problem.

Requirement Engineering

Requirement Engineering is the disciplined application of proven principles, methods, tools, and notations to describe a proposed system's intended behaviour and its associated constraints.

- SRS may act as a contract between developer and customer.



SRS → Software Requirement Specification Document
A contractual document where requirements are written in the natural language.

Types of System supported -
- Modelling - To know the system's physical behaviour
- Simulation - where we unable to check what is going on physically or behaviour of the system.

Types of Requirements -
1) Functional requirements
- Describe what the software has to do. They are often called product features.
2) Non-Functional requirements
- mostly quality requirements. That stipulate how well the software does, what it has to do.

Requirements Elicitation
- it is the most difficult, most error-prone and most communication intensive software development.
- It can be successfully through an effective customer-developer partnership.

Requirement Elicitation Activities -
- Detailed investigation of user needs.
- Define the constraints for system development
- The details of the precise customer problem where the system are going to be applied must be understood.

PAGE No
DATE

Requirements Elicitation methods-

1- Interviews
- conduct to understand the customer's expectations from the soft-ware.
- Interviews may be open-ended or structured
- In open-ended interviews there is no pre-set agenda. Context free questions may be asked to understand the problem.
- In structured interview, agenda of fairly open questions is prepared. Sometimes a proper questionnaire is designed for interview.

2- Brain-Storming Sessions
- It is intended to generate lots of new ideas hence providing a platform to share views.
- Every idea is documented so that everyone can see it.
- Finally, a document is prepared which consists of the list of requirements and their priority if possible.

3) Facilitated Application specification Technique
- it's objective is to bridge the expectation gap.
- A team oriented approach is developed for requirements gathering. Each attendee is asked to make a list of objects that are-

a) Part of the environment that surrounds the system.
b) Produced by the system.
c) Used by the system

PAGE No
DATE

* Each participant prepares his/her list, different lists are then combined, redundant entries are eliminated, team is divided into smaller sub-teams to develop mini-specifications and finally a draft of specifications is written down using all the inputs from the meeting.

4- Quality Function Deployment-

- In this technique, customer satisfaction is of prime concern, hence it emphasizes on the requirements which are valuable to customer.
- 3 types of requirements are identified -

a) Normal requirements
b) Expected requirements
c) Exciting requirements

The major steps involved in this procedure are -

- Identify all stakeholders
- List-out all requirements from customer
- A value indicating degree of importance is assigned to each requirement.

5 Points: V. important
4 Points: Important
3 points: Not imp. but nice to have
2 points: Not imp.
1 point: Unrealistic, required further exploration

PAGE No
DATE

5- Use case approach
- This technique combines text and pictures to provide a better understanding of the requirements.
- The components of the use-case design includes three major things - Actor, Use cases and use case diagram.

a) Actor
- It is external agent that lies outside the system but interacts with it in some way.
- Actors may be primary actors or secondary actors.
- Primary actors - It requires assistance from the system to achieve a goal.
- Secondary actor - It is an actor from which the system need assistance.

b) Use cases
- they describe the sequence of interactions between actors and the system.
- a complete set of use cases specifies all possible ways to use the system.

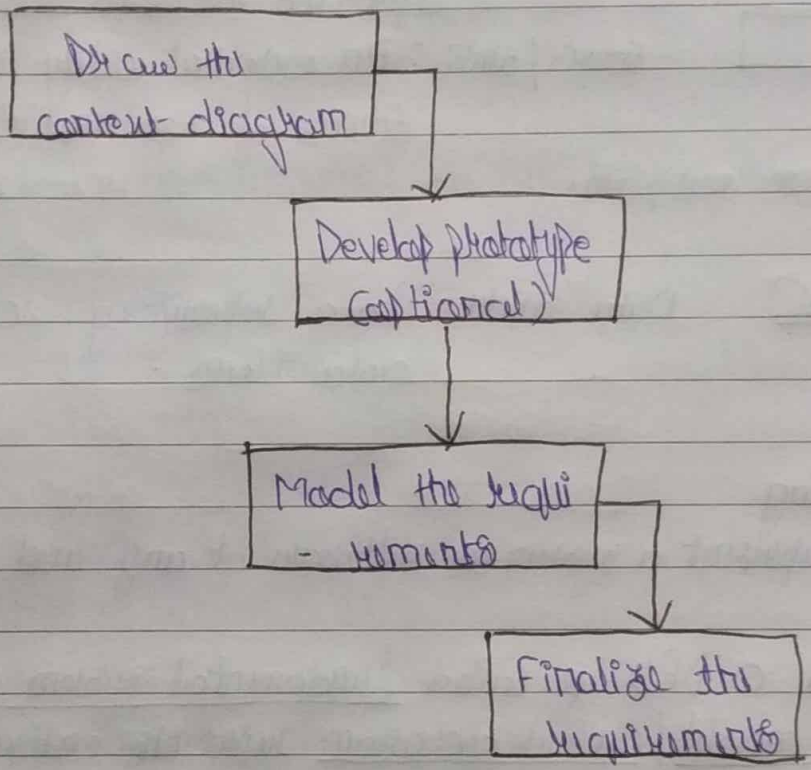
c) Use case diagram
- It graphically represents what happens when an actor interacts with system.
- It captures the functional aspect of system.
- A stick fig. is used to represent actor.
- An oval is used to represent a use case.
- A line is used to represent a relationship b/w an actor and a use case.

PAGE No
DATE

Requirement Analysis

- We analyse, refine and examine requirements to make consistent and unambiguous requirements.

Steps →



Data Flow Diagrams

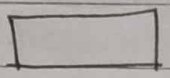
- Data Flow Diagrams (DFD) show the flow of data through the system.
- The primary purpose of DFD is to understand the requirement
- DFD is also known as Bubble Chart
- Some major points:-
  a) All names should be unique
  b) It is not a flow chart
  c) Suppress logical decisions
  d) Defer error conditions and handling until the end of the analysis.
- One bubble → Context diagram / 0-Level DFD

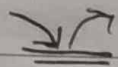
PAGE No
DATE

symbols -

1-  Process: Defines a process that transforms data inputs into data outputs

2-  Dataflow: shows flow of data into or out of a process or data store.

3-  Source/sink: An external entity that acts as a source of system inputs or sink of system outputs.

4-  Data store: Data repository, a collection of data items.

Leveling

DFD represent a system or software at any level of abstraction.

* A level 0 DFD is called fundamental system model or context model / context diagram represents entire software element as a single bubble with input and output data indicating by incoming and out-going arrows.

→ Module is an independent part of system.

Requirements Documentation

- This is the way of representing requirements in a consistent format.
- SRS [Software Requirement specification] document serves many purposes depending upon who is writing it.
  → It is a type of contractual document between developer & customer

PAGE No
DATE

Nature of SRS

Basic issues
- Functionality
- External Interfaces
- Performance
- Attributes
- Design constraints imposed on an implementation

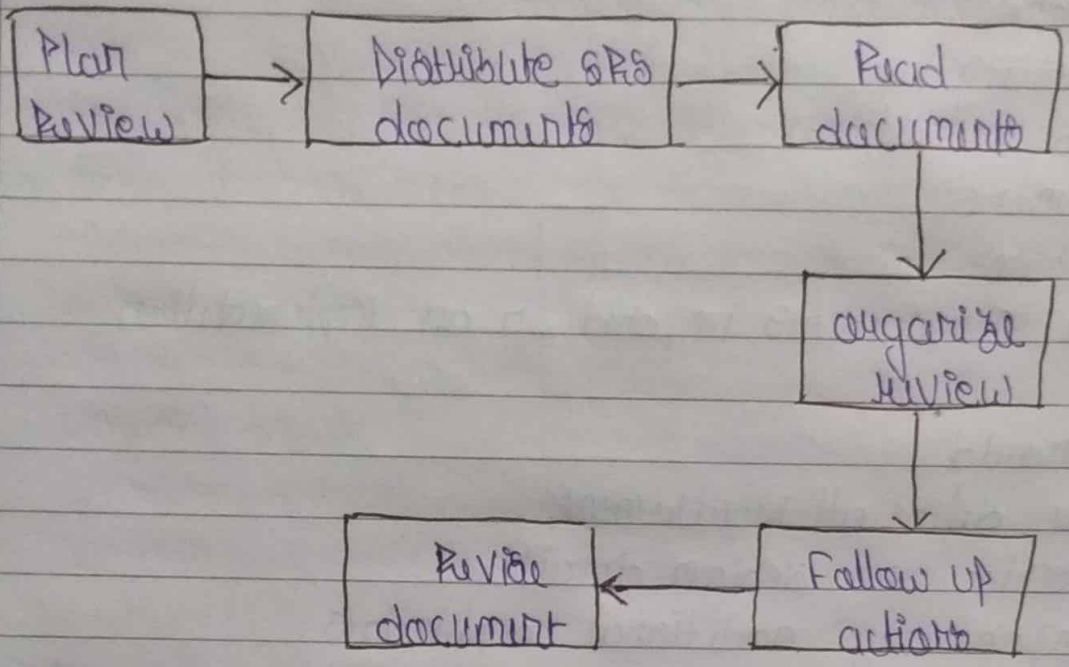
SRS should
- correctly define all requirements
- not describe any design details
- not impose any additional constraints

characteristics of a good SRS
- Correct
- ~~Una~~ Unambiguous
- Complete
- Consistent
- Ranked for importance / stability
- Verifiable
- Modifiable
- Traceable

* IEEE has published guidelines and standards to organize an SRS.

Check the document for:-
- completeness and consistency
- conformance to standards
- Requirements conflicts
- Technical errors
- Ambiguous requirements

Requirements Review Process



Software Project Planning

After the finalization of SRS, we would like to estimate size, cost and development time of the project. Also, in many cases, customer may like to know the cost and development time even prior to finalization of the SRS.

In order to conduct a successful software project, we must understand:

- Scope of work to be done
- The risk to be incurred
- The resources required
- The task to be accomplished
- The cost to be expended
- The schedule to be followed

- Software planning begins before technical work starts, continues as the software evolves from concept to reality, and culminates only when the software is retired.

Estimation-

- Size estimation
- Development-time
- Cost estimation
- Resources requirements
- Project scheduling

Size Estimation

Lines of code

- A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragements of statements on the line. This specially includes all lines containing program header, declaration and executable and non-executable statements.

Function count

- Alan Albrecht developed a technique [which is called Function Point Analysis]
- The principle of Albrecht's function point analysis (FPA) is that a system is decomposed into functional units.

a) Inputs: information entering the system
b) outputs: information leaving the system
c) Enquiries: requests for instant access to information
d) Internal logical files: information held within the system

- External interface files: information held by other system that is used by the system being analyzed.

The five functional units are divided into two categories:

(i) Data function types
- Internal Logical Files
- External Interface Files

(ii) Transactional function types
- External input
- External output
- External Enquiry

Special features
- Function point approach is independent of language
- Function points can be estimated from requirement specification or design specification, thus making it possible to estimate development efforts in early phases of development.
- Function points are directly linked to statement of requirements.
- Function Points are based on system user's external view of system, non-technical users of the software system have a better understanding of what function points are measuring.

Counting Function Points

Functional units	weighting factors		
	Low	Average	High
External Input (EI)	3	4	6
External output (EO)	4	5	7
External enquiries (EQ)	3	4	6
External Logical Files (ILF)	7	10	15
External Interface Files (EIF)	5	7	10

UFP (Unadjusted Function Point) = $\sum_{i=1}^{5} \sum_{j=1}^{3} Z_{ij} W_{ij}$

[Rough estimation of FP]

CAF (Complexity Adjustment Factor) = $0.65 + 0.01 \times \sum_{i=1}^{14}$

→ There are 14 complexity factors.

Factor's Points :-

0	1	2	3	4	5
No influence	Incidental	Moderate	Average	Significant	Essential

$$FP = UFP * CAF$$

Q) Consider a project with the following functional units :-

Input = 50

output = 40

Enquiries = 35

User files = 06

External interfaces = 04

- All complexity factors & weighting factors are avg.

Unit of size of software [KLOC]

PAGE No
DATE

UFP = 50X4 + 40X5 + 35X4 + 6X10 + 4X7
= 628

CAF = (0.65 + 0.01 (14X3)) = 1.07

FP = 672

- suppose 'C' takes 128 LOC [Lines of code] per FP
size = 128 × 672 = 86016 ≈ 86 KLOC

Q) Input = 50 - low
output = 30 - average
Enquiry = 10 - high
File = 5 - low
Interface = 5 - high
complexity factors = moderate, weighting factors = low

UFP = 50X3 + 30X5 + 10X6 + 5X7 + 5X10
= 445

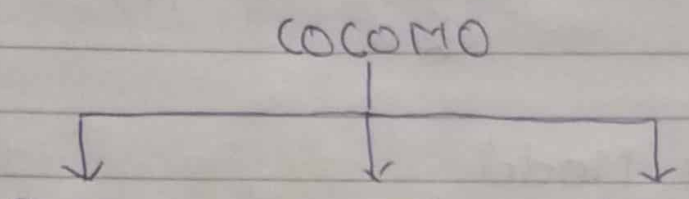
CAF = .65 + 0.01 × $\sum_{i=1}^{14}$

⇒ 0.93

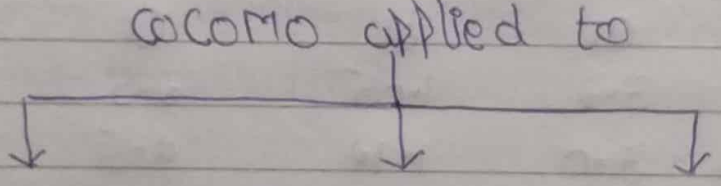
FP = 445 × 0.93 = 413.85
= 413.85 × 128
= 52972.8 LOC
≈ 53 KLOC

Cost Estimation

→ The Constructive cost model [COCOMO]



- Proposed by B. W. Boehm's



Basic Model

- Effort (E) = a_b (KLOC)^{b_b}
- Development time (D) = c_b (E)^{d_b}

Software Project	a_b	b_b	c_b	d_b
organic	2.4	1.05	2.5	0.38
~~embedded~~				
Semi-detached	3.0	1.12	2.5	0.35
embedded	3.6	1.20	2.5	0.32

- Average staff size (SS) = $\frac{E}{D}$ persons

- Productivity (P) = $\frac{KLOC}{E}$ KLOC/PM

Intermediate Model

- $E = a_i (KLOC)^{b_i} * EAF$ (Effort adjustment factor)
  $D = c_i (E)^{d_i}$

Project	a_i	b_i	c_i	d_i
organic	3.2	1.05	2.5	0.38
semi-detached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0.32

$$EAF = \prod_{i=1}^{15} f_i$$

- There are 15 effort factors.

Detailed COCOMO Model

→ Phase-sensitive effort multipliers
→ Three level product hierarchy

Principle of the effort estimate
size equivalent

- As the software might be partly developed from software already existing, a full development is not always required. In such cases, the parts of design document (DD), code (CC%) and integration (I%) to be modified are

estimated. Then, an adjustment factor, A, is calculated by means of the following eqn

$$A = 0.4\,DD + 0.3C + 0.3I$$

size $(S) = (S \times A)/100$

$E_P = \mu_P E$, $\mu_P \Rightarrow$ Effort multiplier

$D_P = \tau_P D$, $\tau_P =$ schedule multiplier

Distribution of software life cycle-

1- Requirement and product design
a- Plans and Requirements
b- system design
c- Detailed Design
d- Code & unit test
e) Module code & test
4- Integrati and Test

Software Designing

- Design is the blueprint of Idea
- Architecture is defined from design
- Software design is the process to transform the user requirements into some suitable form, which helps programmer in software coding and implementation.

The following items are designed and documented during design phase:

1- Different modules required
2- Control relationships among modules
3- Interface among different modules
4- Data structure among different modules
5- Algorithms required to implement among the individual modes.

Objectives of Software Designs:-

1- Correctness
2- Efficiency
3- Understandability
4- Completeness
5- Maintainability

Modularity

- A modular system consist of well defined manageable units with well defined interfaces among the units.
- Modularity is the single attribute of software that allows a program to be intellectually manageable.
- It enhances design clarity, which in turn eases implementation, debugging, testing, documenting and maintenance of software product.

Module coupling

- Coupling is the measure of the degree of interdependence between modules.

Types :-

1- Uncoupled: no dependencies
2- Loosely coupled: some dependencies
3- Highly coupled: many dependencies

This can be achieved as-

- controlling the no. of parameters passed amongst modules.
- avoid passing undesired data to calling module.
- Maintain parent/child relationship b/w called & calling modules.
- Pass data, not the control information.

NOTE

Poor design: Tight coupling
Good design: Loose coupling

Data coupling → Best

:

:

:

content coupling → worst

Data coupling

- if the dependency of module A & B is based on the fact they communicate by only passing of data. other than communicating through data, the two modules are independent.

stamp coupling

- occurs b/w module A & B when complete data structure is passed from one module to another.

Control coupling

- if modules communicate by passing of control information. This is usually accomplished by means of flags that are set by one module and acted upon by dependent module.