Unit - 1
Introduction

Software Engineering
↓
Systematic Approach
↓
Well mannered discipline.

SDLC (Software Development Life Cycle).

1. Feasibility Study
2. Requirement Gathering
3. Analysis
4. Design
5. Coding
6. Testing
7. Implementation
8. Maintainance.

Software Engineer : The establishment & use of sound engineering principle in order to obtain economically developed software i.e reliable & works efficiently on real machines in other words.

A discipline whose aim is the production of quality, software that is deliver on time, within budget & that satisfied its requirement.

What is Measure, Matrix, Measurement.

Measure : A measure provide a quantitative indication of the extent, dimension, size, capacity, efficiency, productivity or reliability of some attribute of a product or process.

Measurement : It is the act of evaluating a measure. When a single data point has been collected for eg the no. of error uncovered in the review of a single module, a measure has been established.

Measurement occurs as the result of the collection of data points. A Software matrix relates the individual measure in some way.

eg:- The average number of error founds per review.

Metrics A metrics is a quantitative measure of a degree to which a system, component or process posses a given attribute

1945 - 1965 → Build and fix model
1965 - 1980 → S/w crises
1990 - 2000 → internet

Software Characteristics

1. Functionality
2. Reliability
3. Efficient
4. Usability
5. Portability

IP/OT
↳ recoverable
↳ fault tolerance

Software Engineer Principle : (W⁵HH)

Why What When Who Where How

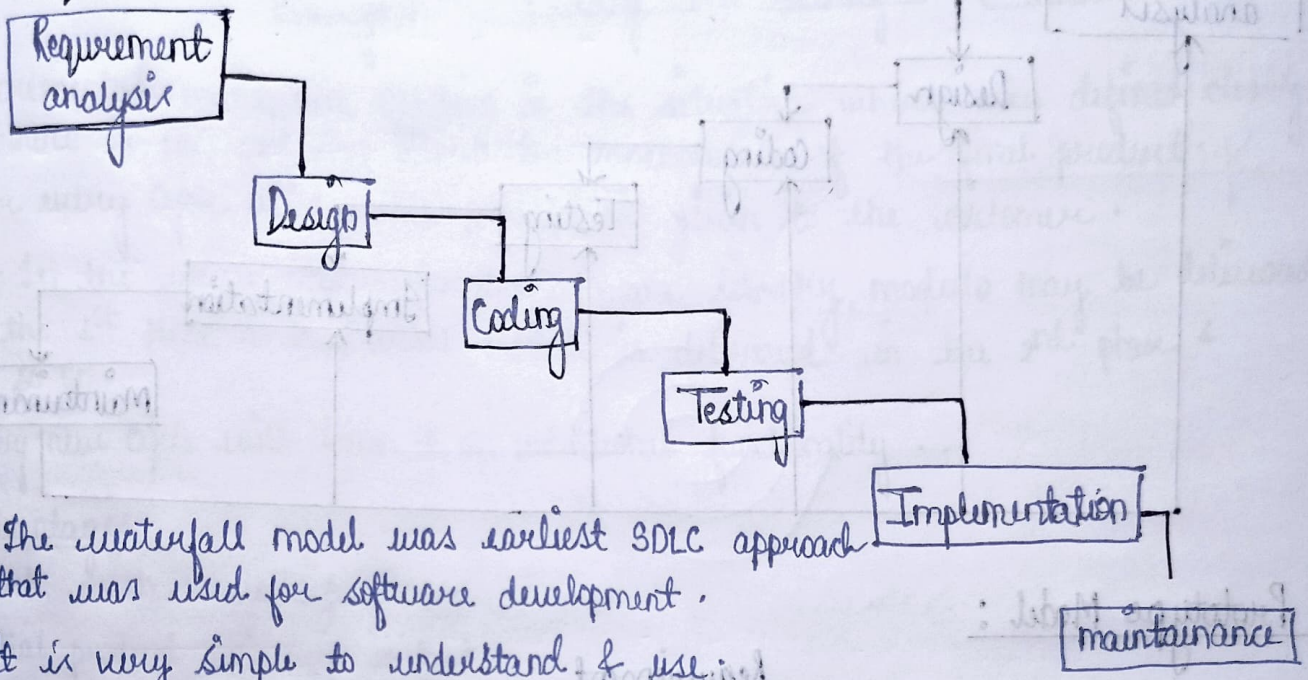
→ Why System is developed
→ What will be done
→ When will it be done
→ Who is responsible for a function
→ Where they are located
→ How will the job be done technically
→ How much of each resource is needed

SDLC

- Planning
- Feasibility study
  - ↳ economic feasibility
  - ↳ Technically feasible

- Requirement analysis
- S/w Design
- Coding
- Testing
- Implement & maintance

Waterfall (Linear Sequential Model / classical life cycle model)-



→ The waterfall model was earliest SDLC approach that was used for software development.

→ It is very simple to understand & use.

→ In this model, each phase must be completed before the next phase can begin.

→ Developed in 1970 by Winston Royce.

Q) When should we use this?

- Requirements are clear & fixed. (they may not changed)
- It is good to use this model when the terminology will understood.
- The project is short & cost is low.
- Risk is zero or minimum.

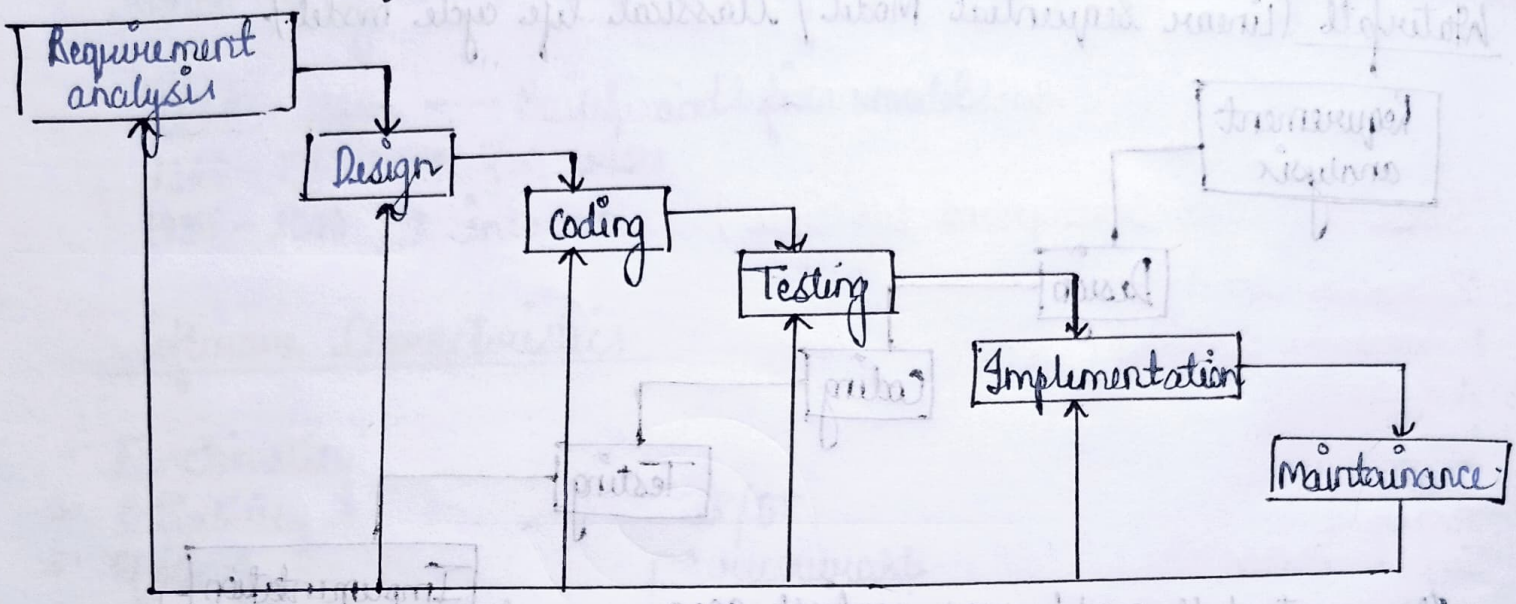
Advantages

- Simple & easy to understand
- Easy to manage
- Clearly defined stages.
- Well understood.
- Easy to arrange task.
- Works for smaller & low budget project.

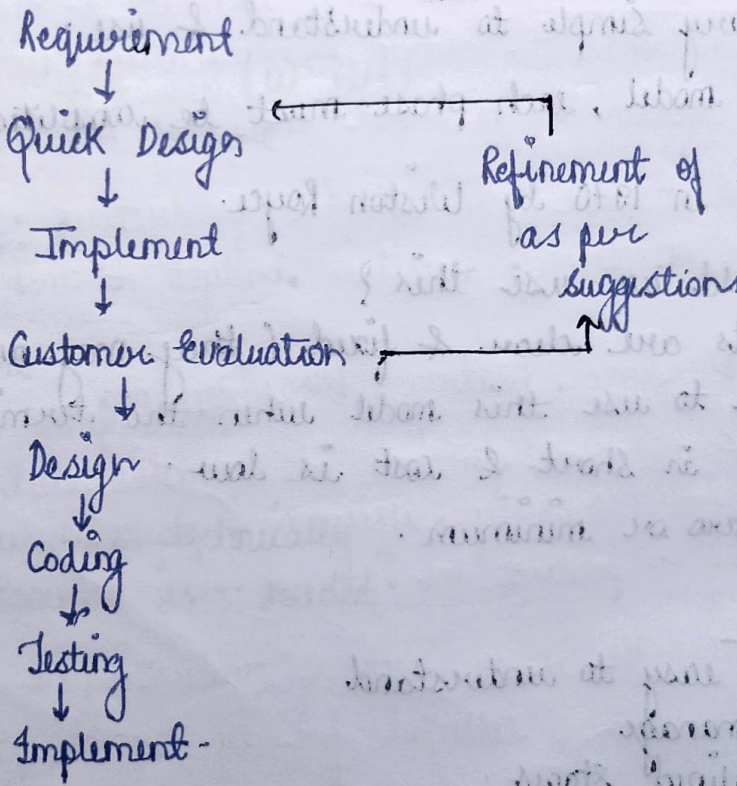
Problems :

Difficult to define all requirements at the beginning.
This model is not suitable for accomodating any change.
It does not scale up will to large project.

Iterative Waterfall Model.



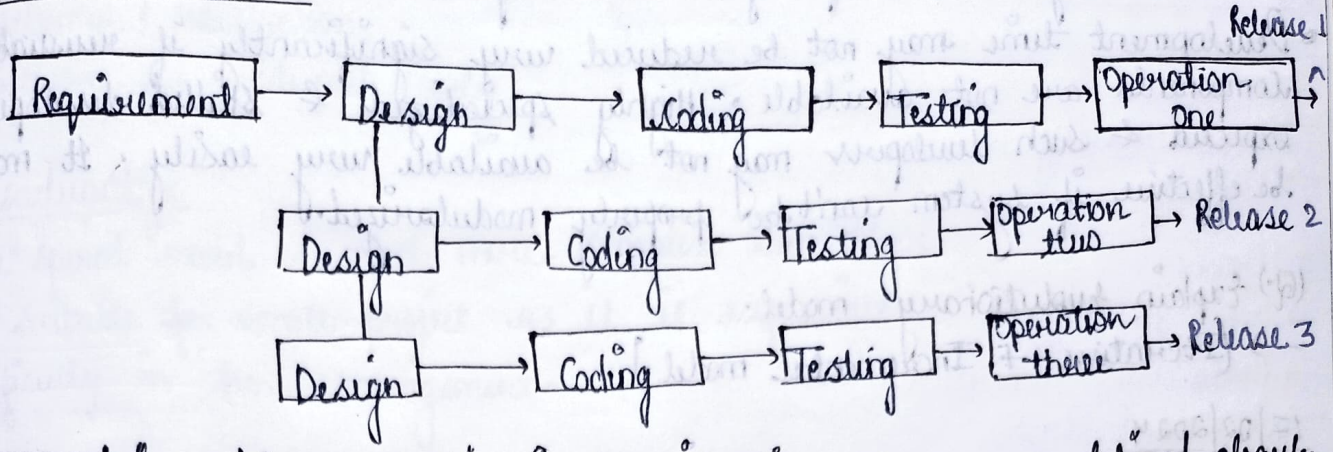
Prototype Model :



Features.

- It help in determining user requirements more deeply.
- At the time of actual product development customer feedback is available.

Incremental Model



- Incremental model are effective in the situation where are defined clearly & there is no confusion about the functionality of the final product.
- After every cycle a usuable product is given to the customer.

eg:- In the university automation software library module may be delivered in the 1st phase & examination module is delivered in the 2nd phase & so on.

- Every new cycle will have & an additional functionality.

Advantages

- Work with small size team
- Initial product delivery is fast.
- Can accomodate changes
- ~~Fast~~ Customer response or feedback is considered.

Disadvantages

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

Rapid Application Development Model (RAD)

- This model is an prototype model & was developed by IBM in 1980's & described in the book of James Martin entitled "Rapid Application Development" Here user involvement is essential from requirement phase to delivery of the product. The process is started with building a rapid prototype & is given to user for evaluation. The user feedback is obtained & prototype is refined. The process continues till the requirements are finalized.
- In this model quick initial views about the product are possible due to delivery of rapid prototype. The development time of the product may be reduced due to use of powerful development tools. If user can't be

involved throughout the life cycle. This may not be an appropriate model.

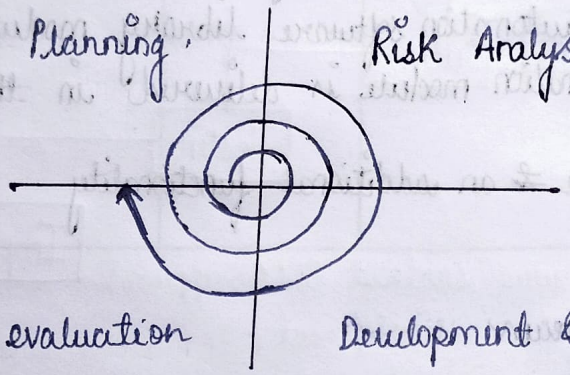
- Development time may not be reduced very significantly if reusable components are not available. Highly specialized & skilled developers are expected & such developers may not be available very easily. It may not be effective if system can't be properly modularized.

(Q.) Explain evolutionary model.

→ (Iterative + Incremental model).

17/02/2024

Spiral Model :-



- Spiral model is one of the most important software development life cycle model.
- This model was 1st described by Barry Boehm in 1986.
- Spiral model can handle large amount of risk.
- The spiral model has 4 phases :
  1.) Planning
  ii.) Risk analysis
  iii.) Development & testing
  iv.) Evaluation.
- Spiral model is used when the project is large & the amount of risk is large.

Eg. Used in ISRO, NASA.

Advantages.

1.) Best model for risk analysis & risk handling.
2.) Good for large project

3.) Flexibility in requirement.
4.) Customer satisfaction
5.) Software is produced early.

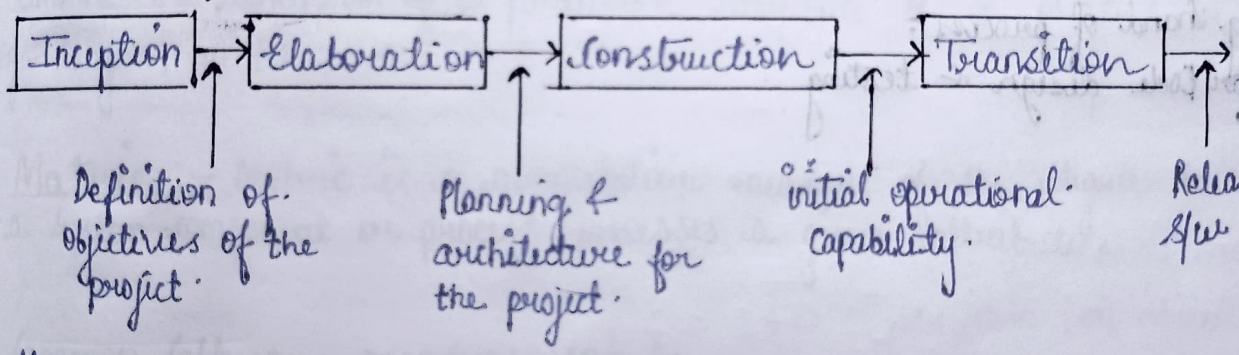
Disadvantage

1.) The spiral model is much more complex than other.
2.) Not suitable for small project as it is expansive.
3.) Difficulty in time management.

Win-Win Spiral Model.

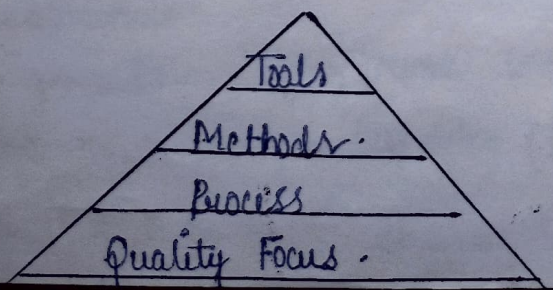
The win-win spiral model is a variation of the spiral model that focuses on ensuring that all stack holders in the project are satisfied with the outcome. It does this by incorporating a negotiation process into the planning & decision making steps of the spiral model.

The Unified Process



- Unified process is described by I Jacobson, G Booch, J Rambaugh in their book in 1998.
- It is a software engineering process with the goal of producing good quality maintainable software within specified time & budget. Unified process is an attempt to draw on the best features & characteristics of traditional software process model but implements many of the best principle of agile (flexible) software development.

Layered Approach



1) Quality focus.
→ Degree of goodness
→ Correctness
→ Maintainability
→ Usability.

2.) Process
→ What to do?
→ Deals with activities
→ Actions & task.

3.) Method
→ Deals with how to implement.
→ Communication
→ Requirement & design modeling analysis.
→ Use of programming language
→ Testing & Support.

4.) Tools
→ Helping hand of process.
→ Used for code design & testing

Unit-2
Measures

Measure: A measure provide a quantitative indication of the extend amount, dimension, capacity or size of some attribute of a product or process.

Types of measure

1.) Direct Measures (Internal Attributes)

eg :- Cost, effort, LOC, Speed, memory

2.) Indirect measure (External Attributes)

eg:- functionality, quality, complexity, efficiency, reliability, maintainability.

Measurement:- A measurement is an indication of size quantity amount or dimension of a particular attribute of a product or process.

eg - length, width.

Matric :- Matric is a quantitative measure of the degree to which a system component or process possesses a given attribute.

Common Software measurement :-

1.) **No. of line of code** : does not count blank space & comment.

- Easy to use
- Language & paramelve dependent
- Easy to compute.

2.) **Halstead S/w Science.**

n_1 = no. of distinct operator.
n_2 = no. of distinct operands.
N_1 = Total no. of operators.
N_2 = Total no. of operands.
$N = N_1 + N_2$ (Total length)
Program Vocablory $(n) = n_1 + n_2$

estimated length $(\hat{N}) = n_1 \log_2 n_1 + n_2 \log_2 n_2$

purity variation $(PR) = \frac{\hat{N}}{N}$

Volume $V = N \log_2 n$

Difficulty $(D) = \frac{n_1}{2} \times \frac{N_2}{n_2}$

program effort $(E) = D \times V$.

eg if (K > 2)
{
if (K > 3)
x = x + K ;
}

Distinct operator : if , (), {}, <, =, +, ;
Distinct operands : K, 2, 3, x.
$n_1 = 9$, $n_2 = 4$, $N_1 = 13$, $N_2 = 7$.

(Q.) if (x > 5)
{
x = x + 2 ;
if (x < 7).
{
x = 0 ;
}
}

Sol" Distinct operator :- if (, > ,), {, =, +, ; , <, }
Distinct operand :- x, 5, 2, 7, 0

$n_1 = 10$, $n_2 = 5$, $N_1 = 17$, $N_2 = 9$

Program vocabulary (n) = $n_1 + n_2 = 10 + 5 = 15$
Total length (N) = $N_1 + N_2 = 17 + 9 = 26$.
Estimated length ($\hat{N}$) = $n_1 \log n_1 + n_2 \log n_2$

$= \frac{10 \log 10}{0.301} + \frac{5 \log 5}{0.301}$.

$= 10 + 3.4 = 33.22 + 11.29$

$\hat{N} = 13.4$ 44.51

Purity variation PR = $\frac{\hat{N}}{N} = \frac{13.4}{26} \cdot 0.51 = 1.71$

Volume (V) = $N \log n$

$= 26 \frac{\log 15}{0.301}$.

$= 30.5 = 101.3$

Difficulty (D) = $\frac{n_1}{2} \times \frac{N_2}{n_2} = \frac{10}{2} \times \frac{9}{5} = 9$.

∴ Program effort (E) = $D \times V = 9 * 301.5 = 274.5$ 913.5

20/02/2024

FPA (Function Point Analysis)

Measuring Software size interms of line of code is difficult. The no. of components is useful in predicting the no. of assembly line staff needed but it does not say anything about the func^n available in the finished product. When dealing with customer the manufacture talk in terms of func^n available & not in the terms of components.

- Func^n point measure functionalities from the user point of view i.e on the basis of what the user request & receive in return from the system.

Function point divide func^n in 5 unit:
Input : information entering the system.
Output : information leaving the system
Enquiries : request for instant access to info.
Internal logical files : info. held within the system.
External interface files : info. held by other system i.e used by the system being analysed.

For Calculating FPA.

Step1: Each func^n point is ranked according to the complexity.

Functional unit	Weighting Factor		
	Low	Avg	High
EI	3	4	6
EO	4	5	7
EQ	3	4	6
ILF	7	10	15
EIF	5	7	10

Step2: Now calculate unadjusted func^n point [UAF] by multiplying each func^n point by its corresponding weight factor.

$$UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} x_{ij} * w_{ij}$$

(w_{ij} → weighting factor.)

Step3: Calculate final func^n point.

$$FP = UFP * CAF$$ (complexity adjustment factor).

by default = 14 question.

$$CAF = 0.65 + 0.01 \sum f_i$$

Scaling from 0 – no influence 3 – Avg 5 – essential.
1 – incidental 4 – Significant

Q.) Consider the project with the following functional unit :-
no. of user enquiries = 35 , no. of user files = 6 , no. of external interfaces = 4. No. of user ~~= 40~~ output = 40, no. of user input = 50.
Assume all complexity adjustment factor & writing factors are average. Compute the func^n point for the project.

Sol^n $UFP = \sum_{i=1}^{5} \sum_{j=1}^{3} z_{ij} * w_{ij}$

$= 50 \times 4 + 40 \times 5 + 35 \times 4 + 6 \times 10 + 4 \times 7.$

$= 200 + 200 + 140 + 60 + 28.$

$= 628.$

$CAF = 0.65 + 0.01 \sum fi$

$= 0.65 + 0.01 (14 \times 3)$

$= 0.65 + 0.42$

$= 1.07.$

$FP = UFP \times CAF$

$= 628 \times 1.07.$

$= 672.$

Q.) An application has the following :
10 external i/p, 12 external o/p, 20 low (low) internal logical files, 15 high (high) external interface files, 12 average external enquiries & a value of complexity adjustment factor is 1.10. What are the unadjusted func^n point & final point.

Sol^n $CAF = 1.10.$

$UFP = 10 \times 3 + 12 \times 7 + 20 \times 7 + 15 \times 10 + 12 \times 4$

$= 30 + 84 + 140 + 150 + 48.$

$\therefore UFP. = 452$

$FP = UFP \times CAF.$

$\therefore FP. = 497.2$

22/02/2024

COCOMO Model (Constructive Cost estimation model.)

- Cocomo model is one of the very famous model which is used to estimate the cost of the project. Cocomo model was proposed by Boehm in 1981.
- According to Boehm project cost estimation should be done through 3 ways
  1. Basic Cocomo
  2. Intermediate Cocomo
  3. Complete/Detailed Cocomo.

1.) Basic Cocomo

- The basic Cocomo gives an approximate estimate of the project parameters, it means it predict the effort & cost of the project.

The following formula is used to estimate the cost in this model:

$$\text{Effort } (E) = a * (KLOC)^b \text{ PM}$$

PM → person month. KLOC → project size.

$$\text{Development Time } (D) = C\,(E)^d \text{ months.}$$

$$\text{Average staff size } (SS) = \frac{E}{D} \text{ person}$$

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

Range of Line of code (LOC)	Project	a	b	c	d
(2-50)	Organic	2.4	1.05	2.5	0.38
(50-300)	Semidetached	3.0	1.12	2.5	0.35
(above 300)	Embedded	3.6	1.20	2.5	0.32

Organic – (2 – 50 KLOC) - data processing system, simple inventory man^n system
Semidetached – (50 – 300 KLOC) – DBMS, difficult inventory management system.
Embedded – (more than 300 KLOC) – ATM, air traffic control, Banking Software real time project.

Q.) Suppose that a project was estimated to be 400 KLOC calculate the effort & development time for each of the 3 mode i.e organic, semidetached & embedded.

Sol? Organic

Effort (E) = $a * (KLOC)^b$ PM

= $2.4 * (400\ KLOC)^{1.05}$ PM

= 2.4×539.7

Effort (E) = 1295.3 PM

Development Time (D) = $C(E)^d$ months

= $2.5 (1295.3)^{0.38}$

= 2.5×15.2

= 38.1 Months.

Semidetached

Effort (E) = $a * (KLOC)^b$ PM

= $3 \times (400)^{1.12}$ PM

= 3×450.9

= 1352.7 PM

Development Time (D) = $C(E)^d$ months

= $2.5 (1352.7)^{0.35}$

= 2.5×12.4

= 31.1 months

Embedded

Effort (E) = $a * (KLOC)^b$ PM

= $3.6 \times (400)^{1.20}$

= 4772.8 PM

Development Time (D) = $C(E)^d$ months

= $2.5 (4772.8)^{0.32}$

= 37.5 months.

(Q.) A project size of 200 KLOC is to be developed. Software development team has average experience on similar type of project. The project schedule is not very tight. Calculate the effort development time avg. staff size & productivity of the project.

Sol: KLOC = 200 — Semidetached.

Effort = $a * (KLOC)^b$ PM

= $3 * (200)^{1.12}$ PM

$\therefore (E) = 667.07$ PM

Development Time (D) = $c (E)^d$ months

= $2.5 (667.07)^{0.35}$

$\therefore D = 24.34$ months.

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

= $\frac{667.07}{24.34}$

= 27.4

$\therefore SS = 28$ persons.

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

= $\frac{200}{667.07}$

$\therefore P = 0.29$ KLOC/PM

23/02/2024

2.) Intermediate Cocomo

Project	a	b	c	d
Organic	3.2	1.05	2.5	0.38
Semidetached	3.0	1.12	2.5	0.35
Embedded	2.8	1.20	2.5	0

This is an extension of basic Cocomo model. It uses 15 additional predictors considering its environment called cost drivers to estimate a value or cost. Cost drivers are used to adjust the nominal cost of a project to the actual project environment hence increasing the accuracy of the estimate.

The cost drivers are grouped into 4 categories

1.) Product attributes :-

a) Required software reliability (RELY)
b) Database size (DATA)
c) Product Complexity. (CPLX)

2.) Computer attributes :-

a) Execution time constraint (TIME)
b) Main storage constraint (STOR)
c) Virtual machine (VIRT)
d) Computer turn around time. (TURN)

3.) Personal attribute :-

a) Analyst capability (ACAP)
b) Application experience (AEXP)
c) Programmer Capability (PCAP)
d) Virtual machine experience (VEXP)
e) Programming language experience (LEXP)

4) Project attributes :-

a) Modern programming practices (MODP)
b) Use of software tools (TOOL)
c) Required development schedule (SCED).

Each cost driver is rated for a given project environment the rating uses a scale: very low, low, nominal, high, very high, extra high which describes to what extent the cost drivers applies to the project being estimated. The multiplying factors for all 15 cost drivers are

multiplied to get the effort adjustment factor (EAF).
The intermediate COCOMO equations are :-

$$E = a(KLOC)^b \times EAF$$
$$D = c(E)^d$$

Cost Drivers	Ratings					
	Very low	Low	Nominal	High	Very high	Extra high
Product Attributes						
RELY	0.75	0.88	1.00	1.15	1.40	—
DATA	—	0.95	1.00	1.08	1.16	—
CPLX	0.70	0.85	1.00	1.15	1.30	1.65
Computer attributes						
TTME	—	—	1.00	1.11	1.30	1.66
STOR	—	—	1.00	1.06	1.21	1.56
VIRT	—	0.87	1.00	1.15	1.30	
TURN	—	0.87	1.00	1.07	1.15	
Personnel Attributes						
ACAP	1.46	1.19	1.00	0.86	0.71	—
AEXP	1.29	1.13	1.00	0.91	0.82	—
PCAP	1.42	1.17	1.00	0.86	0.70	—
VEXP	1.21	1.10	1.00	0.90	—	—
LEXP	1.14	1.07	1.00	0.95	—	—
Project Attributes						
MODP	1.24	1.10	1.00	0.91	0.82	—
TOOL	1.24	1.10	1.00	0.91	0.83	—
SCED	1.23	1.08	1.00	1.04	1.10	—

27/07/2024

Q) A new project with estimated 400 KLOC embedded system has to be developed. Project manager has a choice of hiring from two pools of developers,

Very highly capable with little experience ~~when~~ in programming lang. being used or developers of low quality but a lot of experience with programming language. What is the impact of ~~high~~ hiring all developers from one or other pool.

Given :-

Sol^n Case I : Very highly capable with little experience in programming language

$$E = a (KLOC)^b \times EAF$$
$$= 2.8 \times (400)^{1.20}$$
$$\therefore E = 3712 \text{ PM}$$

Case I : $EAF = 0.82 \times 1.14$
$= 0.9348$.

$\therefore E = 3712 \times 0.9348 = 3469.9$

$\therefore D = 2.5 (3469.9)^{0.32} = 33.9$

Case II : Developer one (AEXP) of low quality but lot of experience (LEXP) with programming language being used (high)

$EAF = (1.29) \times (0.95)$
$= 1.22$

$\therefore E = 3712 \times 1.22 = 4549.05$.

$D = 2.5 (C (E)^d$.

$= 2.5 (4549.05)^{0.32}$

$D = 37.02$

$\therefore E = 4549.05$, $D = 37.02$.

3.) Complete / Detailed Cocomo OR Cocomo II

The complite cocomo model is an extention of the intermediate cocomo model. This model is phase sensitive. It does not depend on any phase it is used to calculate the amount of effort required to complete each phase.

Phases :

i. Planning & Requirement
ii. System Design
iii. Detailed Design
iv. Coding
v. Testing
vi. Cost Constructive model

Cocomo -II is the revised version of the original cocomo. The model is depend onto ~~two~~ life cycle. It also provides a quantitative framework & set of tools & techniques for evaluating the effect of software technology or improvement on spw life cycle cost & schedule.

28/02/2024

Reverse Engineering :- (Backward Engineering)

- Software reverse engineering is also back engineering or backward engineering. It is the process of recovering & understand the basic method, design & the requirement specification of legacy software product.
- First we analyse the already created spw source code after that recreating it so we can find out the original code with some improvement. ~~It is a process of~~
- It is a process of rethink, restructure & rebuild our legacy software.
- Code → module specification → design → Requirement analysis.

Unit-3

Software Requirement Specification

SRS :- is a description of a software system to be developed. The goal of this phase is to clearly understand the customer requirement & to systematically organise the requirement into a specification document.

★ Imp ★

Characteristics of SRS

1.) **Consistency** :-

(i) No conflict b/w the requirement.

(ii) Every requirement must be specified using a standard terminology.

2.) **Correct** :-

(i) What is stated is exactly what is desired.

(ii) Expected functionality matches the requirement that are present in SRS.

3.) **Unambiguous** :-

(i) Every stated requirement has only one unique meaning.

(ii) SRS language can be used.

4.) **Complete** :-

(i) It includes all detailed specification, diagrams, functional & non-functional requirements & constraints.

5.) **Traceable** :-

Origin of each requirement should be clear.

6.) **Verifiable** :

SRS is verifiable if & only if each requirement is verifiable.

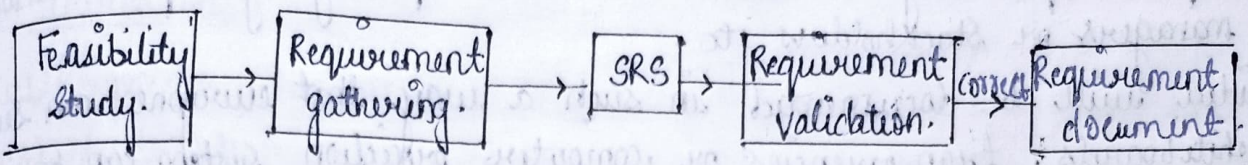
7.) **Modifiable** :

It is easy to make changes in SRS, retaining its structure plus correctly modification.

8.) **Ranked** : Ranked for stability or importance.

01/03/2024

Requirement Engineering Process:



Requirement Gathering / Elicitation Technique

Requirement elicitation is the process of discovering the requirement of a system. It is basically collecting requirement from stack holders, users & customers by conducting meeting, interviews, questionnaires, brainstorming sessions, facilitated application specification technique (FAST), quality function deployment, & use case approach.

1.) Interview

After recieving the problem statement from the customer the first step is to arrange a meeting with the customer. During the meeting or interview both the parties would like to understand each other. Normally specialized developers often called requirement engineers interact with the customer. The objective of conducting an interview is to understand the customer expectation from the software.

Interview may be open or structured. In open-ended interview there is no preset agenda context free questions may be asked to understand the problem.

For example: For a result management system requirement engineer may ask who is the controller of examination?, who has requested for such a software?, who will explain the manual system? etc.

In structured interview agenda of fairly open question is prepared. Sometimes a proper questionnair is designed for the interview. Interview may be started with simple question.

2) Brain storming sessions:

Brainstorming is a group technique that may be used during requirement elicitation to understand the requirements. The group discussion may lead to new ideas quickly & help to promote creative thinking. Brainstorming has

become very popular & is being used by most of the companies. It promotes creative thinking, generate new ideas & provide platform to share views. This group technique may be carried out with specialized group like actual users, managers or stakeholders etc.
Every idea will be documented in such a way that everyone can see it. Whiteboards, transparencies or computer projection system can be used to make it visible to every participant. After the session a detailed report will be prepared.

06/03/2024

3) FAST (Facilitated Application Specification):-
This approach encourages the creation of a joint team of customer & developer who works together to understand correct set of requirements. A facilitator can be a customer, a developer or an outsider controls the meeting.

4) QFD (Quality Function Deployment):-
It emphasises to encorporate the voice of the customer with important then according to customer a value indicating a value of importance is assigned to each requirement thus the customer determine the importance of each requirement on a scale 1 to 5 as number 5 shows very important, no. 4 shows important, no. 3 not important, no. 2 shows not important, no. 1 unrealistic.

5) JAD (Joint Application Development)
- It is defined as a structural approach in which user, manager & analyst work together for several days in a series of intensive meeting to specify or retrieve system requirements.
- JAD is similar to brain storming, JAD produces high level specific software models including data function & behaviour. It includes DFDs, ER diagrams.
- JAD is a user requirement elicitation process that involve the system owner & end user in the design & development of an application through a succession of collaboration workshops called JAD sessions.

07/03/2024

SRS Structure (SRS Document)

1.) Introduction

(1.1) Purpose :- why we make s/w. eg University Management System

(1.2) Intended Audience :- who all are going to use s/w eg student, facality etc

(1.3) Scope : Future Scope

(1.4) Definations :- Those who all are involved in the System eg HOD, Director etc

(1.5) References :- Online media etc.

2.) Overall Description

(2.1) User Interfaces :- Interface for student.

(2.2) System Interfaces :- Server apache, HF http, https etc.

(2.3) Constraints / Assumption / Dependencies :- eg constraint on password, age > 18 etc

(2.4) User Characteristics :- Student, facality those who all are involved.

(2.5) Hardware Interfaces :- Hardware requirements.

3.) System Features and Requirements.

(3.1) Functional Requirement :-

(3.2) Use Cases :- pictorial representation are used.

(3.3) External Interface requirement :- Third party involvement eg Online payment.

(3.4) Logical database requirement :- where the data is stored.

(3.5) Non Functional Requirement :- eg Safety, security, quality, portability, reliability

4) Delivery for approval :- approval req. to deliver the Software.

Functional Requirement :- These are related to the expectation from the intended software they describe what the software has to do. They are also called product features. Sometimes functional requirements may also specify what the software should not do.
It is related to working.

Non Functional Requirements :- are mostly quality requirements that tell how well the software does, what it has to do.

- Non functional requirements that are specially important to the user include specifications of desired performance, availability, reliability, usability & flexibility.
- Non functional requirement for developers are maintainability, portability & testability (other than working)

Data Flow Diagrams (DFD's)

- DFD's are used widely for modelling the requirements.
- DFD show the flow of data through a system. The system may be a company, an Organization, a software system etc.

Standard symbols for DFD's

1.) ⟶ : represents dataflow

2.) ▭ : a source of system inputs or system output / Entity

3.) ◯ : represents process

4.) ▯▯ or ═══ : for data/file storage

eg. Online shopping

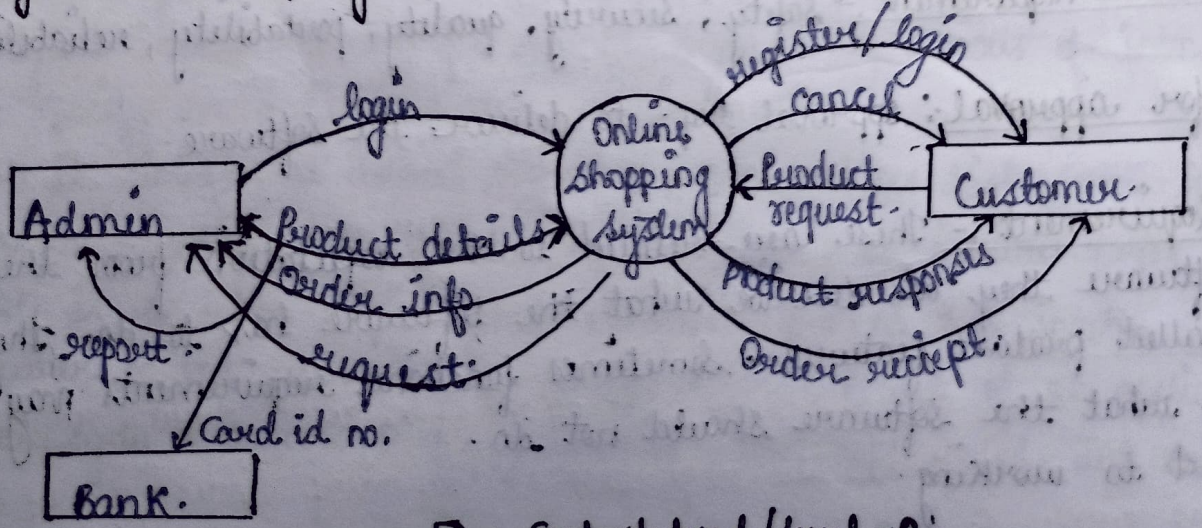


Fig: Context level / level-0