Software Engineering
CSE - 320

24/01/2023

#⇒ **Software:** A software is a set of programs associated with documents to perform a specific task.

There are two types of softwares:
(i) Generic
(ii) Bespoke

① Generic - Software for public use or whatever we use in our laptops.
eg: Adobe.

② Bespoke - Software for a single user. It is developed as per user's requirement.

#⇒ **Software Engineering:** It is a systematic discipline cost effective approach/ method / mechanism to develop a software.

⇒ **The Role of Software Engineering:**
It work as a bridge between customer and programmer.

#⇒ **Phases of Development:**

① Requirement Analysis
② Design
③ Implementation
④ Testing
⑤ Evolution / Maintainence.

24/01/2023

Software Engineering Evolution:

1945 - 1965 : Origin of Software Engineering

1965 - 1985 : Software Crisis

1990 - 2000 : Internet

2000 - 2010 : Light weight

2010 ⟶ : AI Based

Software Development Methods:

Method → Work Strategy

(i) Waterfall Model:
(ii) Iterative / Increamental Model:
(iii) Azile Method:

Factors for the software crisis:

i) Over Budget
ii) Not delivering on time.
iii) Product is poor quality.
iv) Software product is not meeting the customer requirement.

30/01/2023

LIFE CYCLE MODEL :-

SDLC : Software Developer Life Cycle Model

(I) A software life cycle model is a discriptive and diagramatic approach to develop the software.

Software development methods are for all the models.

(II) It will identify all the activities to develop the software.

(III) It will divide the life cycle into phases.

Need of SDLC :

I) It helps to identify the inconsistency and duplication in the software development process.

II) It identify the entry and exit criteria for every phase in the software development process.

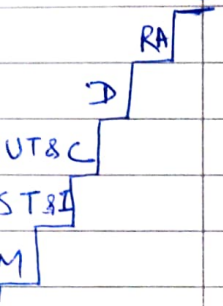
III) There are many life cycles has been proposed

- (a) Classical Waterfall Model
- (b) Iterative waterfall Model
- (c) Evolutionary model
- (d) Propotype Model
- (e) Spiral model

30/01/2023

(1) ⇒ Classical Waterfall Model:

There are six phase in the classical waterfall model:



ⓐ Feasibility Study
ⓑ Requirement Analysis & specification
ⓒ Design
ⓓ Coding & Unit testing
ⓔ System testing & integration.
ⓕ Maintainence.

ⓐ ⇒ **Feasibility Study:** The main aim of feasibility study is to determine the financially and technically information regarding the software.

ⓑ ⇒ **Requirement Analysis & specification:** The main aim of this phase is to understand the exact requirement of the customer and document that properly.

→ This phase contain two different activities: (i) Requirement Gathering and Analysis
(ii) Requirement Specification.

ⓒ **Design:** Design phase tranforms requirement specification into a suitable programming language.

During the design phase, software architecture is derived from SRS document.

30/01/2023

There are two approaches to design the software
(i) Traditional Approach
(ii) Object oriented Approach

(i) Traditional Approach: It consist of two activities (a) Structured Analysis
(b) Structured Design

(ii) Object Oriented Approach:
(a) We identify the object. after that
(b) we identify the relationship among object.

31/01/2023

(d) Coding and Unit Testing: Purpose of coding phase is to translate the software design into source code.
- Each module of the design phase is coded then each module is tested then each module is documented.

→ Purpose of unit testing is to test the individual module ie to check the module is working or not working.

(e) System testing and Integration: Different modules are integrated in a planned manner.
- Modules are almost never integrated in a one shot.
→ During each integration step partially integrated system is tested.

System Testing: Whenever all modules are successfully integrated then tested. The goal of system testing is to ensure that the developed system is working as per the SRS requirement

30/01/2023

There are two approaches to design the software
(i) Traditional Approach
(ii) Object oriented Approach

(i) Traditional Approach: It consist of two activities @ Structured Analysis
(b) Structured Design

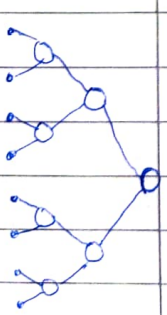
(ii) Object Oriented Approach:
@ We identify the object. after that
(b) we identify the relationship among object.

31/01/2023

(d) Coding and Unit Testing: Purpose of coding phase is to translate the software design into source code.
- Each module of the design phase is coded then each module is tested then each module is documented.

→ Purpose of unit testing is to test the individual module ie to check the module is working or not working.

(e) System testing and Integration: Different modules are integrated in a planned manner.
- Modules are almost never integrated in a one shot.
→ During each integration step partially integrated system is tested.



System Testing: Whenever all modules are successfully integrated then tested. The goal of system testing is to ensure that the developed system is working as per the SRS requirement

31/01/2023

(f) **Maintainance:** This phase requires much more efforts as compare to the other phases.

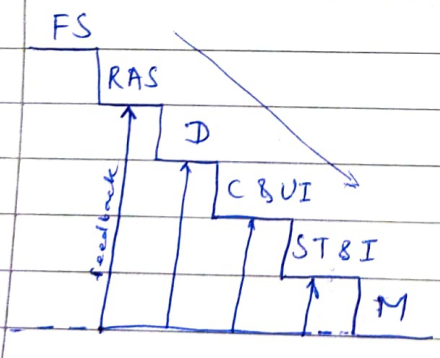
- There are three types of maintainance
  1. Corrective Maintainance
  2. Perfective Maintainance
  3. Adaptive Maintainance

① **Corrective M.:** to correct the errors, which were not discovered during the development process.

② **Perfective M.:** to improve the functionalities of the system, maintainance is required is called perfective maintainance

③ **Adaptive M.:** Port the software to a new environment.

(2.) **ITERATIVE WATERFALL MODEL:**



- There is a feedback concept in the Iterative waterfall Model
- Once a defects is detected we need to go backe to the phase, where it was introduced

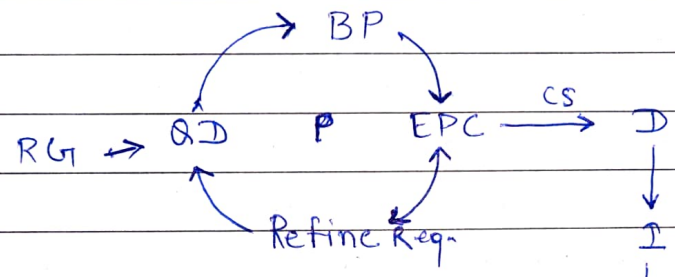
→ **Phase Containment of errors:** The principle of detecting errors as close to it point of introduction as possible.

31/01/2023

(3) PROTYPE MODEL:

Pheses for the protype model:

i) Requirement Gathering
ii) Prototype Phase
iii) Design
iv) Implement
v) Test
vi) Maintainance



QD → Quick Design
EPC → Evaluation of Protype from Customer
BP → Build Prototype

- Before start the actual development, first we work on a prototype; when propotype is completed then evaluated by the customer, when customer satified then we start the design phase.
- Prototype model is suitable for large projects.

→ Reason to develop the prototype model:

i) It is impossible to get it right the first time.

- The developed protype is submitted to the customer for the evaluation.

02/02/2023

- Requirement Analysis and specifications phase become redundent
- Even construction of a working prototype model involves some additional cost.
- Overall cost is less.

4. EVOLUTIONARY MODEL:

This model is a combination of increm-ental model or iterative model.

- The system is broken down into several module which can be incrementally implemented and delivered.
- First develop the core module of the software.
- The initial product skelton is refined into the increasing levels of capability.

→ Advantages of Evolutionary Model:-

i) User get a chance to experiment with a partially developed system.
ii) It helps to find the exact user requirements.

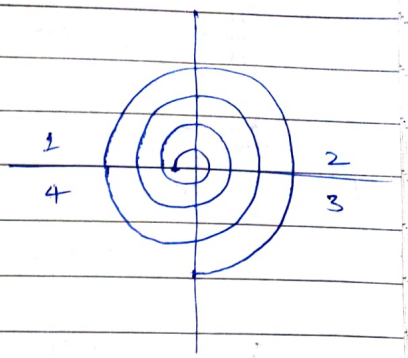
→ Disadvantages :-

i) Often it is difficult to sub divide the problems into functional unit.

- This model is useful for very large projects (which require times like five years, 10 years.)

02/02/2023

5. SPIRAL MODEL:



Quad 1) → Determine the Objective.
Quad 2) → Identify the risk & resolve.
Quad 3) → Develop the next level of product.
Quad 4) → Evaluation from the customer.

- Spiral model is proposed by **Boehm** in **1988**.
- Each loop of the spiral model represents a phase of the software process.
- ★ There is no fixed number of phases.
- The team must decide how to structure the project into phases.
- Start work using some generic model. ~~is loop~~
- Each loop in the spiral having four quadrant.
- The object of the first quadrant is to identify the objective of the object.
- Objective of second quadrant is risk assesment and reduction of the risk.
- Objective of the third quadrant is to develop the next level of product & validation.
- Objective of fourth quadrant is to get evaluated by the customer

09/02/2023

Comparision of different life cycle Model:

Iterative Waterfall Model:
→ This model is widely used
→ This model is suitable for well understood problems

Prototype Model:
→ This model is suitable for not well understood problems.

Evolutionary Model:
→ This model is suitable for large problems.

Spiral Model:
→ [illegible]

Requirement Analysis and Specification:-
Many projects fail because of they start implementing the system without determining whether they are building what the customer really wants.

Goal of Req. Analysis & Specification phase:-
① Fully understand the user requirements
② Remove the inconsistency & anomalies,
③ Documents requirement properly SRS.

④ Req. Analysis & Specification contains two different activities:
① Requirement gathering & analysis
② Specification

09/02/2023

→ The way to gather the requirements:
1. Observation of existing system.
2. Study the existing procedure.
3. Discussion with the customer and user.
4. Analyse of what need to be done.

→ In the case of absent any working system, a lot of imagination and creativity are required.

Some desirable attributes of a good system analyst:
1. Good interaction skills.
2. Imagination and creativity.
3. Experience

Inconsistent Requirement:
1. Some part of the requirement:
2. Contradict with the some other part.

Eg: One customer say "turn of heater, when the temprature is greater than 100°C"
Second customer say "turn of the heater when the temprature is less than 100°C".

Incomplete Requirement:
1. Some requirements have been omitted and customer provided half of the requirements (need full requirement)

for example: The analyst has not recorded (i) when the tempr. falls 90°C. turn on heater.
(ii) Turn off water shower

13/02/2023

⇒ **Requirement Analysis**: It involves:
(i) Obtaining a clear in depth understanding of the product.
(ii) Remove the all ambiguities and remove the inconsistency.

⇒ **Several points need to be understand by the system analyst**:—
i) What is the problem?
ii) Why it is important?
iii) What is the solution of the problem?
iv) Which type of challenges are there?

→ Main aim of Requirement Specification:
i) Systematically organize the requirements
ii) Document requirements properly rated

→ **SRS Document usefulness**:—
i) Statement of user needs
ii) Contract document
iii) Reference document
iv) Definition for the implementation

→ **SRS Document**:— The SRS document is known as Blackbox specification.
→ The system is considered as the blackbox whose internal details are ~~not found~~ known to the user.
→ External details are visible to the customer.

13/02/2023

→ SRS document concentrates on what need to be done and carefully avoid the solution aspects.

→ SRS document work as a contract between customer and development team.

→ Properties Of A Good SRS :-

i) It should be concise.
ii) It should be specified what the system must do.
iii) Easy to change.
iv) It should be consistent.
v) It should be complete.
vi) It should be traceble
vii) It should be verifyable.

→ SRS Document Parts :-

i) Functional requirement
ii) Non-functional req.
iii) Constraints.

→ Example for functional requirements

Function=Search Book
Input : Author name
Output : Book details with locations.

14/02/2023

Functional Requirements	Non-functional Requirements
Check Balance	→ Security
Deposit cash	→ Efficiency
Transfer funds	→ Usability
	→ Safety
change PIN	→ Performance
Transaction history	→ Reliability
Bill payment	→ Availability
	→ Maintainability
Deposit cheque	→ Portability
Loan payment	
Credit card payment	

Verbs	Attributes
Mandatory	Not mandatory
Captured in use case	Captured in quality attribute scenario
Product feature	Product properties
Easy to capture	Difficult to capture.

⇒ **Constraints**: It describe things that the system should or should not do.

Some examples of constrains:

i) → Hardware to be used.

ii) → Operating system
→ or DBMS to be used

iii) → Capabilities of I/O devices.

iv) → Standard compliance

v) → Data representation
→ by the interfaced system.

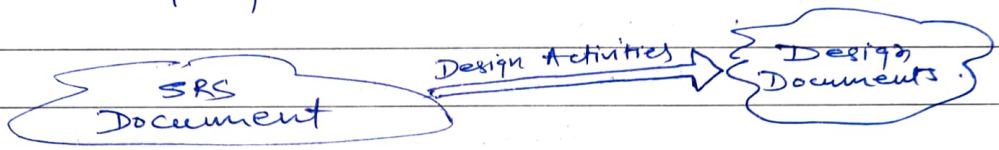
14/02/2023

→ Examples for Bad SRS documents:
(i) Unstructured specifications
(ii) Forward References
(iii) Overspecification
(iv) Contradiction.

16/02/2023

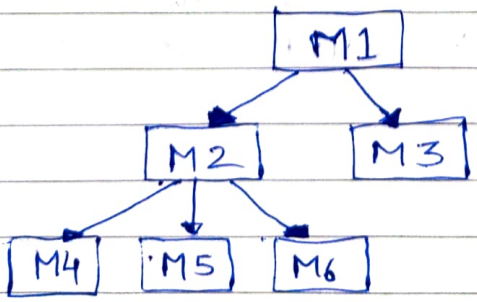
2. SOFTWARE DESIGN:

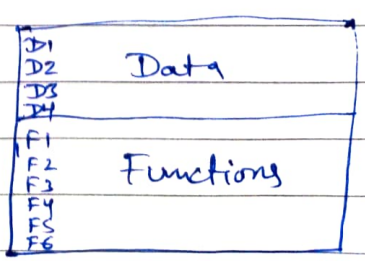
Design phase transform SRS document into a form easily implementable in some programming language.



⇒ Items uses in design phase:
i) module structure,
ii) control relationship among the modules.
iii) interface among different modules, data
iv) data structures of individual modules,
v) algorithms for individual modules.

→ Module Structure: → A module consist of:
i) several functions
ii) associated data structures





16/12/2023

→ Good Software Design:-

(i) Seldom arrived through a single step procedure

(ii) But through series of steps and iterations

→ Design activities are usually classified into two stages:-

i) Preliminary (or high level) design.

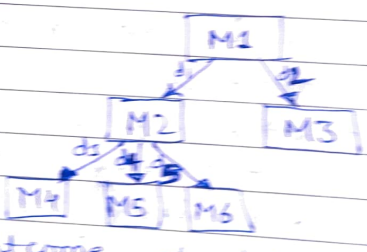
ii) detailed design

(i) High Level Design:-

a) modules

b) control relationship among modules

c) interface among modules



The outcome of high level design:
program structure (or software architecture)

→ Several notations available to represent high level design:

→ Usually a tree like structure called STRUCTURE CHART is used.

16/12/2023

→ Good Software Designs:

(i) Seldom arrived through a single step procedure

(ii) But through series of steps and iterations.

→ Design activities are usually classified into two stages:

1) Preliminary (or high level) design.

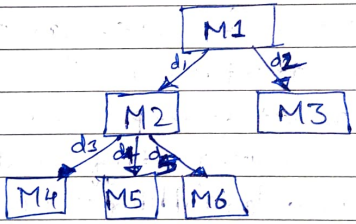
11) detailed design

(i) High Level Design:

a) modules

b) control relationship among modules

c) interface among modules.



The outcome of high level design:
program structure (or software architecture)

→ Several notations available to represent high level design.

→ Usually a tree like structure called STRUCTURE CHART is used.

16/02/2023

For each modules, design:
1) data structure
11) algorithms

#) Outcome of detailed design:
① module specification

20/02/2023

For each module consist of
→ data structures and algorithms.

#) Methods for software design:
① Procedural (function oriented)
② Object oriented

#) Good and Bad Design:
There is no unique way to design a software, it depends on the development team, so different designers have different solutions.

#) Good Design:
① Should implement all the functionalities correct.
② Easy to understandable
③ Easy to maintainable

i) Understandability:
a) Use consistent and meaningful names.
b) Make use of abstraction and decomposition principle.

20/02/2023

-> There are two ways to design a module
   i) Modular design
   ii) Layered design

i) Modularity: Design solutions should consist of a clearly decomposed set of modules is known as modularity.

-> Different modules should be neatly arranged in a hierarchy like a tree diagram.

3124RC41

-> A modularity is a fundamental attribute of any good design

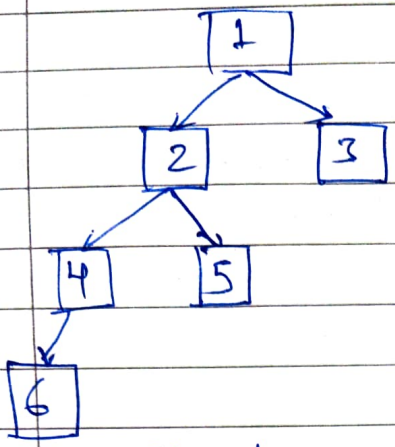
-> Decomposition of any problem into modules in such a way so that modules are independent to each other and use divide & concern approach.

-> If the modules are independent, you can easily understand the modules.

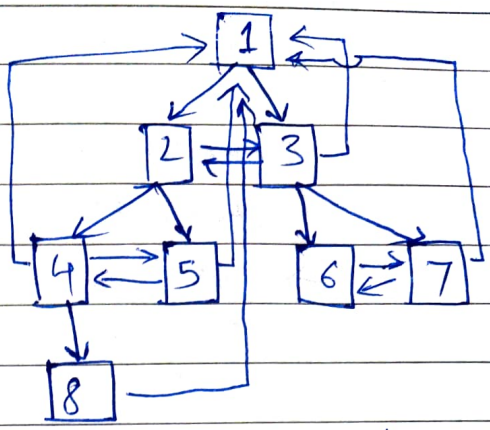
-> Reduce the complexity.

→ Layered

28/02/2023



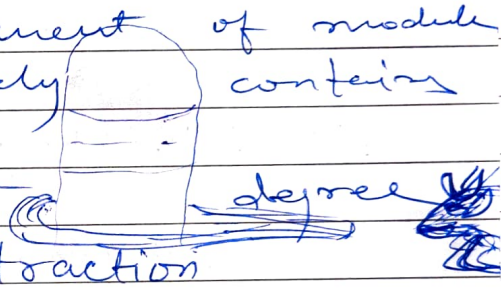
Clearly decomposed



Not clearly decomposed

--> Layered : Neat arrangement of module in a hierarchy contains two part

(1) Low fanout
(2) Control Abstraction



(1) Low fan out : fan out means measure of the no. of the module directly control by given module.

- In technical term, module should display high cohesion and low coupling.

21/02/2023

Cohesion and coupling:

④ Cohesion is a measure of function strength of a module.
→ A cohesive module perform a single task or function.

⇒ Coupling between two modules:
→ a measure of the degree of interdependence or interaction between the two modules.

⇒ Advantages of functional Independence:

i) Better understanding and good design.
ii) Complexity of design is reduced
iii) Different modules easily understood in isolation
iv) modules are independent.
v) A functionally ~~no~~ independent module can be easily take out and reused in a different program.

⇒ Neat Hierarchy:

Control hierarchy represents organization of modules, control hierarchy is also called program structure.

21/02/2023

→ Characteristic of a Module structure.

① Depth: no of levels of control.
② Width: overall span of control.
③ Fan Out: a measure of the no. of modules directly controlled by given modules.
④ Fan in: indicates how many modules directly invoke a given module.

-> High fan-in represents code reuse and is in general encouraged.

→ Layered Design:

- A design having modules:
  - with high fan out numbers is not a good design
  - A module having high fan out lacks cohesion.

-> A module that control another module said to be subordinate to it.

→ Goodness of Design:

- A design having modules with high fan out numbers is not a good design.
- A module having high fan-out lacks cohesion.

→ Visibility and Layering:

A module A is said to be visible by another module B, if A directly calls B

The layering principle require modules at a layer can call only the modules immediately below it.

21/02/2023

Cohesion and coupling:

④ Cohesion is a measure of function strength of a module.
→ A cohesive module perform a single task or function.

⇒ Coupling between two modules:
→ a measure of the degree of interdependence or interaction between the two modules.

⇒ Advantages of functional Independence:

i) Better understanding and good design.
ii) Complexity of design is reduced
iii) Different modules easily understood in isolation
iv) modules are independent.
v) A functionally ~~Un~~independent module can be easily take out and reused in a different program.

⇒ Neat Hierarchy: Control hierarchy represents organization of modules, control hierarchy is also called program structure.

21/02/2023

→ Characteristic of a Module Structure.

① Depth: no of levels of control.
② Width: overall span of control.
③ Fan Out: a measure of the no. of modules directly controlled by given modules.
④ Fan in: indicates how many modules directly invoke a given module.

-> High fan-in represents code reuse and is in general encouraged.

→ Layered Design:

- A design having modules:
- with high fan out numbers is not a good design
- A module having high fan out lacks Cohesion.

-> A module that control another module said to be subordinate to it.

→ Goodness of Design:

- A design having modules with high fan out numbers is not a good design.
- A module having high fan-out lacks cohesion.

→ Visibility and Layering:

A module A is said to be visible by another module B,
If A directly calls B

The layering principle require module at a layer can call only the modules immediately below it.