CH-5

"SOFTWARE DESIGN"

The SRS document developed during requirement gathering & analysis phase is converted into DESIGN DOCUMENT during the design phase. The set of selected requirements are modelled into some design and then implemented by using some programming language.

OUTCOME OF DESIGN PROCESS

The following items are designed and documented during the design phase:

1.) Different modules required : The different modules in the software must be clearly identified and each module should perform a well-defined task. Each module should be named according to the function performed by it.

2.) Control Relationships among Modules : Control relationship refers to the shifting of control from one module to the other due to nested calling of modules in one another.

3.) Interfaces among Modules : Whenever one module calls the other module, it passes a list of parameters to the called module. The function of an interface is to keep track of the list of parameters passed & interpret them according to their function.

We can broadly classify the design activities into two parts

1. High level design ② ~~Do~~ Detailed design.

- identification of diff modules
- control relationship. → outcome of high level is called Program structure or software

data structure and algo. → outcome of the detail design is called module specification

4) Data structures of the individual mo

This part comes under the detailed process. The data structures are required by each module to store the data and results calculated. So it is very important to find an appropriate data structure for each module.

5) Algorithms / Techniques to be used.

After the modules and their respective data structures are found, we have to design a proper algorithm by which they can perform the given task. These algorithm should be simple, concise (brief) and accurate.

CHARACTERISTICS OF A GOOD SOFTWARE DESIGN.

The software design the tool that is used as a base to further implement the project through coding. A software design should be developed in such a way that it can efficiently provide solution to different problem domains. A good software design have following properties:

1) Correctness.

A good s/w design must be accurate i.e. it should correctly implement all the functionality of system.

ility.

A good design should be very clear & simple to implement. An understandable s/w design is also easy to maintain and update.

) Efficiency.

Efficiency refers to the capability of a s/w in terms of memory, time and cost optimization.

) Maintainability.

The customer needs always keep on changing even after the development of the software product. So a s/w design should be easy to change and update.

UNDERSTANDABILITY CONCEPT OF A DESIGN

We usually say that a good s/w design must be very clear and easy to understand. But how we can make it understandable?

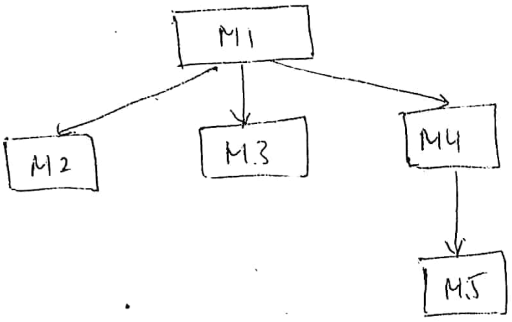
Following steps can be taken to increase understandability of a s/w design solution:

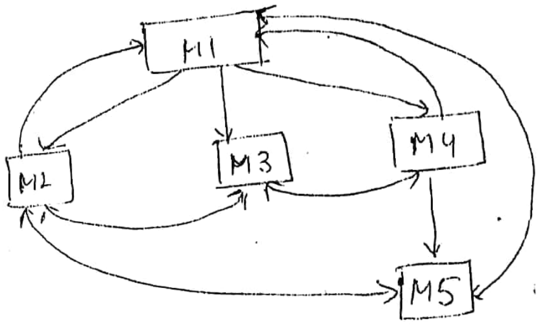
1. We can use the layered and modular structure to develop a s/w design.
2. We should assign meaningful (& unique) names to the different components of a design according to the function performed by them.
3. It should use decomposition & abstraction in order to keep the design simple and less complex.

☆ Implementation of modular and layered design.

1) MODULARITY.

Modularity uses the concept of decomposition of a problem into various modules. Decomposing the problem helps to reduce the complexity and to understand each module individually. This is very much similar to Divide and Conquer Rule.





(a) NEAT MODULARITY

→ It provide clear division of all the modules.

→ The module from lower level could not call a module back from higher level & thus is a well-organised design.

(b) POOR MODULARITY.

→ It represents very chaotic organisation of modules.

→ Here any module can call a module from any level. It do not provide control of the function calling mechanism.

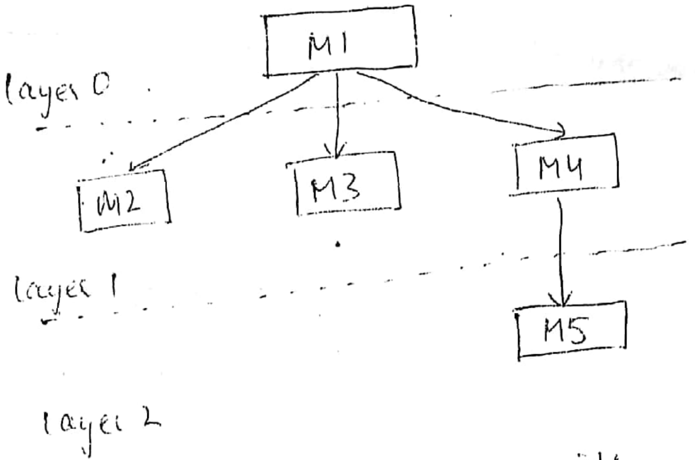
A design solution is called highly modular, if the different modules in the solution have high cohesion and their inter-modular coupling are low.

[we will study COHESION & COUPLING after a while.]

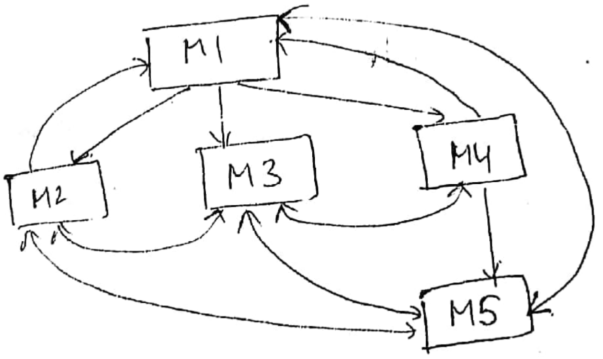
Layered Design.

A layered design is one that implements control abstraction i.e. the lower layer is unaware of the functions performed at higher layer. A lower layer is not allowed to call a function from higher layer. The whole design is organised in such a way that the control need not to be transffered from bottom to top. This is done to keep the design simple and easy to debug.

In the layered design, function calls are represented diagramatically in the form of a tree.



(a) Layered Design with good control abstraction



(b) Layered Design with poor control abstraction.

COHESION — within any

Cohesion is the functional strength of a module i.e. the degree to which different functions of the module cooperate to work for a single objective. Cohesion exists within the functions of the same module. A module should be highly cohesive to perform its function efficiently.

efficiency

TYPES OF COHESION.

Coincidental	Logical	Temporal	Procedural	Communi-cational	Sequential	Functional.

Low ——————————————→ High

1) Coincidental Cohesion.

A module is having coincidental cohesion if the tasks performed by its functions relate to each other very loosely.

2) Logical Cohesion.

A module is logically cohesive if all its functions perform similar logic / operation such as error handling, data output etc.

3) Temporal Cohesion.

A module possess temporal cohesion if the functions inside module execute at the same given time.

Procedural Cohesion.

In this type the functions or components of a module are executed one after the other in a sequence. These functions may perform entirely different tasks.

5) Communicational Cohesion.

All the functions use or refer to the same data structure to store the values.

6) Sequential Cohesion.

A module is sequentially cohesive if all its ~~modules~~ functions are executed in a sequence and output of one function becomes input to the next function.

7) Functional Cohesion.

A module possess functional cohesion if different functions are related with each other to complete a single task. ex: To calculate salary of an employee, many functions work together - computeOvertime(), computeWorkHours(), computeDeductions() etc.

COUPLING outside dept

Coupling is the bonding between two different modules i.e. degree of interdependance. It is an intermodular relation whereas cohesion is intramodular (within same module) relationship.

Coupling rises due to :-

a) two modules frequently call each other and pass large amount of data.

b) when two modules share some data or uses same data structures.

TYPES OF COUPLING :

Data	Stamp	Control	Common	Content

Low ⟶ High

① Data Coupling.

Two modules are data coupled if they communicate by passing parameters to each other in the form of primary data items :- integer, float, character etc.

② Stamp Coupling.

If two modules share a composite data item i.e. a combination of two or more items under one value, then they are said to be stamp coupled.

Ex: DOB → day + month + year.

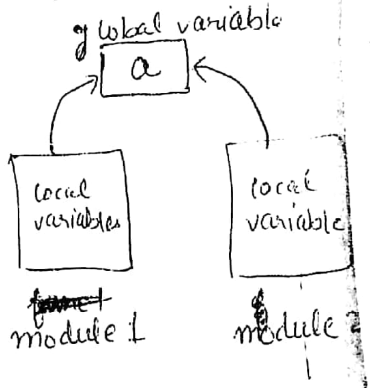
Address → street + town / city + state.

③ Control Coupling.

It occurs when one module decides and give instructions for the execution of other module.

④ Common Coupling.

If two modules share some global data item, they are commonly coupled.



⑤ Content Coupling.

Content coupling exists b/w 2 modules if they share a common piece of code i.e. they use the same logic or function while execution.

FUNCTIONAL INDEPENDANCE

A module that is highly cohesive and also has low coupling with other modules is said to be functionally independant of other modules.

ADVANTAGES

(1) Error Isolation.

Functional independance reduces the chances of error propagation from one module to another because ~~they~~ their interaction with each other is very less due to low coupling. Thus errors in one module do not effect the functionality of the other.

(2) Scope of Reuse.

A module performs a very well-defined task in a system and its interfacing with other modules is also low/weak in functional independance, so it can easily be taken out & reused in other similar applications.

(3) Understandability.

In functional dependance the modules are independant from each other so they can be understood in isolation. It reduces the complexity of the system to a great extent.

APPROACHES TO S/W DESIGN.

The approaches followed to design a software solution can be broadly classified into following categories:-

(1) Function-oriented Design : This is a conventional method and uses a centrallized database which is shared by all modules.

(2) Object-oriented Design : In OOD, the system is a collection of objects, used to model real world entities. each object has its own copy of data items and manage it. These objects are used to call the methods.

	FUNCTIONAL DESIGN APPROACH	OOD APPROACH
a)	It uses functions to execute the logic.	It uses objects to call the functions & then fire logic.
b)	It is a conventional method.	It is a modren approach & is still evolving.
c)	This is a top-down decomposition approach.	This a bottom-up unification approach.
d)	The data is centrallized & then shared by all the modules.	The system is decentrallized as each object has its own copy of data & no global data is shared.
e)	It uses structured analysis and design to model the requirements.	It uses Object Oriented Analysis and Design to model the user requirements. ex: UML Diagrams.