Software Design 16/08/2023

- Both Modular and layered design must be present together in designing of a software. If any design misses to be applied, then future development or enhancements in software will become very complex and impossible to handle.
- The above is saying that - the design principles are related to each other, they are not isolated. Hence, various designs must be considered in order to develop a large and complex software so that future requirements as well as updates must be done easily.

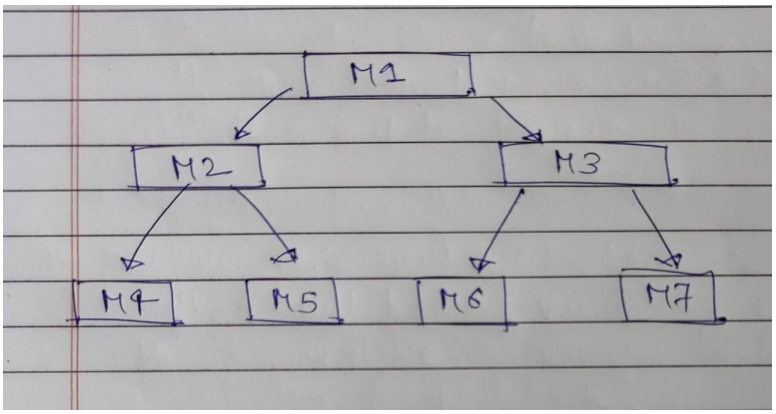
Visibility across modules

- Module A is visible to another module B if A directly or indirectly calls a B.

***Layering principle*:**

- *Modules at layer can only call the modules immediately below them.*
- The reverse must also be true:

  Modules at layer mustn't know about what is at the top.(This is what abstraction and encapsulation does)



Therefore, in the above diagram M1 cannot call M4, M5, M6, M7 directly, but it can call M2 and M3 directly. Also, M4 doesn't know about M1 or M2 or M3.

Abstraction

- Lower level modules → Generally, I/O and low-level functionalities.
- High-level modules → Manager modules.

Principle of abstraction:

Lower level modules cannot call higher level modules.

A module is unaware of the higher level modules. (doesn't know hoe to invoke it).

Why abstraction is required?

- The simple answer to the above question is that it is then very easy to add low-level functionality modules without affection the higher-level modules.
- Enhancements would be very easy then.
- Future considerations have to be made:
  - Lesser cost testing
  - More business
  - Lesser cost enhancement……..

High level design

- Also called *Software Architecture Development*.
- In the high level design, mapping between functions (fi) and modules (mj) are done.
- Mapping must be such that:
  - High cohesion.
  - Less coupling.
  - Neat hierarchy (layered design).

💡 Coupling occurs mainly in different levels of abstraction. Whenever there is calling, then always coupling will be there.

Object-oriented matrix (CK matrix)

This is the matrix which shows the relationships between modules qualitatively and not degree-wise.

Note that module relations cannot be expressed quantitatively.

Example:

- We cannot say that these two modules have 3.5 degree of coupling. Instead, we say that out of 5 levels of coupling, these have 3 levels of coupling.

Two approaches

- Functional oriented design
- Object-oriented design

These two techniques compliment each other but not competing each other. *Each technique is used at different levels of design in software development cycle.*

Dataflow diagrams, structure charts, and flowcharts are used in both approaches. But UML is only designed for Object oriented design.

Functional oriented:

- Started looking from high level view.
- Then refined into more detailed functions.
- Map to module structure.

💡 System state will be centralized in functional oriented designs. There will be some variables which are available outside or even everywhere.

Object-oriented design:

- System state is decentralized among the modules.
- Each object will manage its own state.
- Similar objects constitute a class and they communicate by message passing.

Note that object based design is called object enabled design.

Some fancy terms only:

- RMI means remote method invocation is just a invocation. But it is from remote side and not local.
- Interrogation is just a function call. Objects know each other state through interrogation(function call).