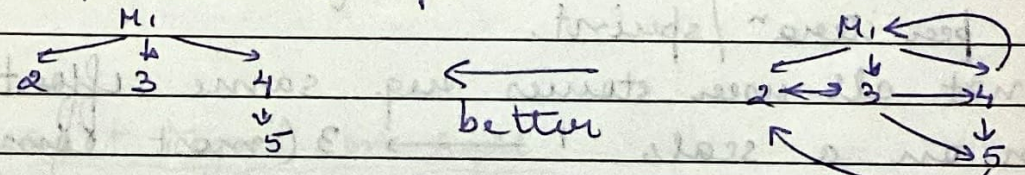
Week-5

L1 Software Design

- outcomes of design process – components, interfaces among diff. comp., data structures of individual comp., algos. req. to implement indi. comp.
- software architecture – high level software design
- Online portal comp. → catalogue mgmt., inventory mgmt., payment tracking, etc.
- Charac. of good software design —
1. correctness – implements all functionalities of the sys.
2. efficiency – ensures that resources are managed well
3. maintainability – easy to Δ.
4. understandable – by all in dev. team.

L2 Design Modularity

Eg.

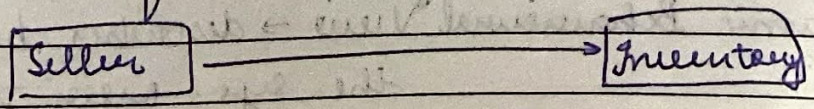


- highly coupled modules share a lot of data.
- When all functions in a module perform a single objective, the module is said to have a good cohesion.
- Coupling → measure of degree of interacⁿ btw. 2 modules
- Cohesion → measure of how funcⁿˢ in a module coop. together to perform a single objective.
- Good design → High cohesion, low coupling
- Modular → Problem has been decomposed into a set of modules that have only ltd. interaction with each other.

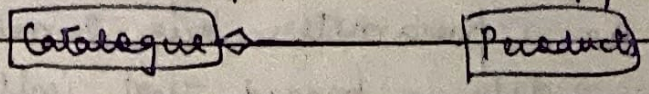
- Data Coupling → Communica" using primitive data type passed as a parameter btw 2 module
- Control Coupling → data is passed that influences the int. logic of a module. Eg. flags, switches.
- Common coupling → modules share global data.
- Content Coupling → modules refers to internals of other module. Rare in Python, Java (PL)
- Functional Cohesion → diff. func"s of the module coop. to comp. a single task.
- Sequential Cohesion → diff. func" of module execute in a seq. Output goes as input.
- Communicational Cohesion → if all func"s of the module refer to or update the same data structu
- Procedural cohesion → related by seq., but weak towards entirely diff. purposes.
- Coincidental cohesion → meaningless relationships.

L3 Object Oriented Design: Basic Concepts

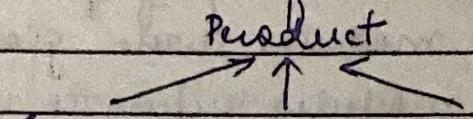
- Object → key building block. Working of a software in terms of interacting objects. It represents a real-world entity. → data abstrac".
  → data, methods, encapsula" (data + methods)
- Class → template for object crea", all obj. possessing similar attributes & methods constitute a class.
  Eg. electronics, food, cloth → objs. of type product.
  ↓ Relationships
- Association → take each other's help to perform some func".



- Composition – Represent whole/part relationships



- Inheritance – extend features of an existing class



- Dependency – If class B depends on class A, if any Δs are made to class A, Δs have to made to class B as well.
- Design → provides structure to a software sys. Outcome of the design process → diff. modules req. & relation among modules. Communicating the design to other members is impt.

L4 Unified Modelling (UML) Diag.

- Modelling – Creating an ext., explicit representⁿ of the sys. to be built. UML → helps represent the software design via multiple views & greater level of detail.
- Impt. views of a software sys —
  1. structural view
  2. dynamic behavioural view
- Structural View (Class Diag.) → structure / comp. of the software sys. & relationships. Describes logical parts of the sys. – classes, data & funcⁿ.
- Class Diag. → describes the struc. of the sys., describes system's classes, attributes, operⁿˢ, relationship among obj.
- Dynamic Behavioural View → describes behaviour of the sys. over time.

(1) State Machine View – models diff. states of an obj of a class.
(2) Activity View – models flow of control among computational activities.
(3) Interaction View → seq. of messages exchanges among parts of a sys.

- purposes of modelling –
(i) Vehicle for communication & idea generations
(ii) guide development of softwares
(iii) Close correspondence with the implementation
(iv) generate code from models

1.5 VeriSim
- learning event.
- Verifying designs by simulating scenarios
- develop an integrated understanding of class & seq. diag.
- construct a state diag. which models the scenarios