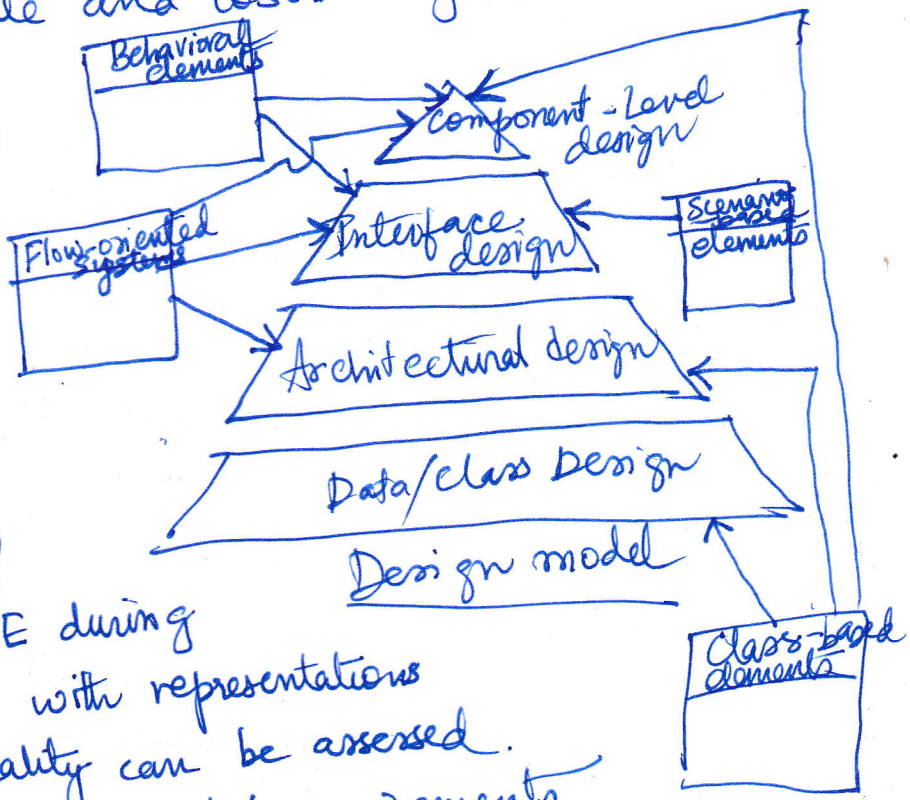
Focus of design: architecture, data structure, interfaces and components.

Design establishes quality and an estimation of whether the model can be implemented within the constraints, schedule and cost.



What's the output?
- data/class design
- Architectural design
- Interface design
- Component design

Why design is important?
Quality is fostered in SE during design. Design provides with representations of software where quality can be assessed.
Design is a place where stakeholder's requirements are accurately transformed into a software product.

Bad design leads to
- unstable system – small changes will lead to failure
- poor quality
- difficult to test
- Difficult to assess quality until very late in the process

What is design process?
Design Process: It is an iterative process that translates requirements into a "blueprint" for constructing software.

How to assess quality in design?
Use technical reviews (TR)

What are characteristics of a good design?

1. Should have a good architecture
   - use recognizable architectures or design patterns
   - components with good design characteristics
   - should be implemented in evolutionary process
2. Should be modular – logically partitioned into elements and sub-elements (systems)
3. Distinct representations of data, architecture, interfaces & components
4. Components should exhibit independent functional characteristics.
5. Interfaces should reduce complexity of connections between components and external environment
6. Design should be derivable from analysis of requirements
7. Design should be represented properly using a notation for effective communication.
8. data structures should be appropriate for classes and drawn from recognizable data patterns implementation

What are the FURPS quality attributes for design? developed by Hewlett & Packard

- Functionality: features and capabilities of system including security
- Usability: considers human factors, aesthetics, documentation etc.
- Reliability: frequency and severity of failure (MTTF)
- Performance: processing speed, response time, resource consumptions etc.
- Supportability: combines extensibility, adaptability, serviceability and maintainability.

Data/class design transforms class models into design classes and required data structures to implement the software

Architectural design defines relationships between major structural elements of software, the architectural styles and design patterns to achieve the requirements of system. It also defines the constraints for implementing the architecture

Interface design : defines how software communicates with other existing systems and with humans who use it. It is primarily concerned with data/control flow and behavior of system.

Component-level design : Transform structural elements of architecture into procedures/processing of software components. Class models, flow models & behavior models serve as inputs to component design.

What are design concepts ?

Design Concepts : These are the fundamental concepts that every designer (software) must be aware of that were identified over a period of time

how to remember?
(PAID FARM)

- Abstraction
- Architecture
- Patterns
- Modularity
- Information hiding
- Functional Independence
- Refactoring
- Design Classes

Abstraction:

- Procedural abstraction (behavior)
  - A sequence of instructions that have specific and limited function
- Data abstraction (Structure)
  - A named collection of data that describes a data object

Architecture: Structure or organization of program components, how they interact, and structure of data used by the components.

Architectural patterns are used to solve common design problems.

eg: Event-driven architecture, Pipe & filter architecture, Layered architecture etc.

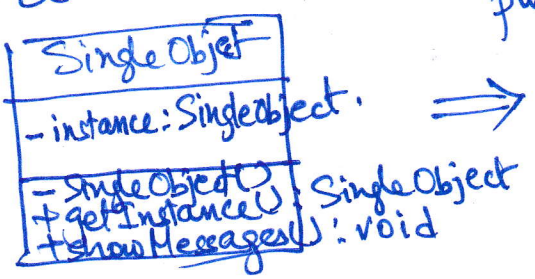
What are types of architectural models?

- Structural models
- Framework models
- Dynamic models (behavior)
- Process models
- Functional models

Architectural Description Languages (ADL) have been developed to represent these models.

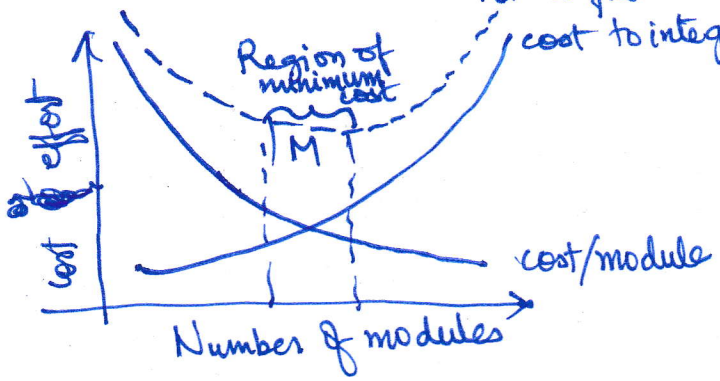
Patterns: Design pattern is a design structure that solves a commonly occuring design problems within a specific context and the consequences of using it (trade offs) & forces.

eg: Singleton Pattern: This pattern can be applied when a programmer needs only one instance of a class




```
public class SingleObject {
    private static SingleObject instance = new SingleObject();
    private SingleObject() { }
    public static SingleObject getInstance()
    {
        return instance;
    }
}
```

Modularity: Software is divided into separately named and addressable components, called modules. These modules together (integrated) satisfy requirements.
Thus, modularity engenders integration i.e, these modules must be integrated. This effort to integrate is the overhead associated with modularity.



Information Hiding: Hide details of data structure and ~~design~~ procedural processing behind a module interface. Knowledge of these need not be known to users of the module.
Greatest benefit is the easy modification of code during testing and maintenance as inadvertant errors are not easily propogated to other modules.

Functional Independence: It is a result of modularity abstraction and information hiding.
Modules have "single minded" function and have an "aversion" to excessive interaction with others.
Assessed based on two quality criteria
- cohesion: degree to which a module focuses on one thing.
- coupling: degree to which a module is connected to others modules and outside world

Design classes:

Analysis classes are refined into design classes so that they can be implemented. New design classes may be implemented to support business solutions.

Types of design classes

- User Interface (UI) classes
- Business Domain classes
- Process classes
- Persistent classes
- System classes

Characteristics of good design class

- Primitiveness: Class should provide only one service of implementing a
- High Cohesion: Class should have a small, focused set of responsibilities and single-mindedly applies attributes and methods to implement them.
- Low Coupling: Communication among classes should be minimized. A class should have limited knowledge about classes in other subsystems (Law of Demeter). This law suggests that a class should send messages only to neighboring classes.

Refactoring: Design activity (adopted from Agile method) simplifies design (or code) of component without changing its function or behavior.

The Design Model

What is a design model? What are its four major elements?

The design model consists of data design, architectural design, interface design, component-level design and deployment-level design models.

The four major elements of design model are:

- data design
- architecture design
- components design
- interface design

Along the process dimension (as development moves from activity to activity) the architectural elements, followed by interface elements, followed by component-level elements are developed. This may not be necessarily sequential. Until these models are fully developed, deployment model is delayed.

Data design: It leads to model of data modeling database At application-level, data design consists of At component-level data design models data structures that are required to implement local data objects

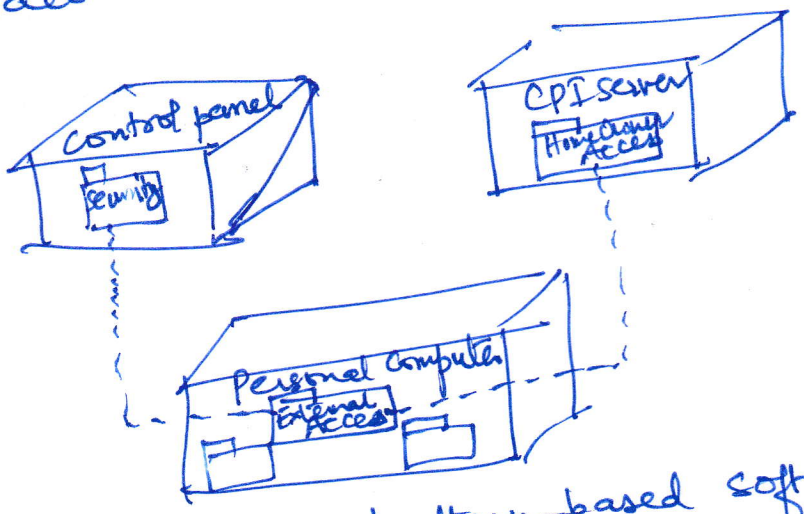
Architectural design: This is equivalent to the floor plan of a house. It gives overall view of the systems, subsystems, components and their relationships of a software system

Interface Design elements: This is equivalent to detailed drawings of doors, windows and external utilities to a house. Interface design focuses on user interface, interfaces to components and interfaces to systems external to the application.

"An Interface is a externally visible specifier for externally visible [public] operations of class, component or other classifier without specification of internal structure."

Component-level design : This is equivalent to detailed drawings of each room in the house.
It defines the data structures for all data objects and the details of algorithms to process that data.
It also specifies interface to other component operations.
UML component diagrams are drawn to show this model

Deployment-level design : this focuses on how software functionality and subsystems are allocated to physical computing environment.
UML deployment diagrams are shown to model ~~this diagram~~ it.



What is meant by pattern-based software design?

Pattern-based software design:
Design patterns solve ~~provide solution design issues~~ for existing or commonly occuring design problems.
Thus SEngineer should reuse these patterns whenever possible instead of reinventing them or create a new solution.

Description (Template) of Design Pattern

- Pattern name
- Intent
- Motivation
- Applicability
- Structure
- Participants
- Collaborations
- Consequences
- Related patterns

What are the types of design patterns

— Architectural patterns: Patterns at system, subsystem and component level that solve some design problem.

— Design patterns: ~~A~~ this is a component-level pattern that solves some design problem.

— Idioms / "Coding patterns": They are language specific patterns that implement algorithmic element of a component, a specific interface or communication mechanism These patterns are specific to a component.

Frameworks: A framework is not a pattern but an incomplete skeletal framework with plug-in points that enable it to be adopted to some specific problem domain, in order to solve some design problems.