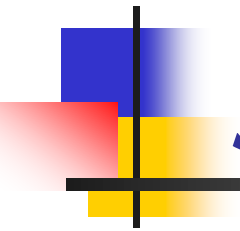
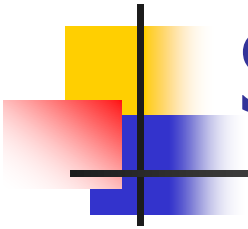


COMP 354: Introduction to Software Engineering

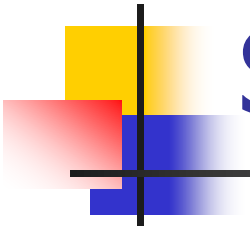
Design Concepts

Based on Chapter 9 of the textbook



Software Design

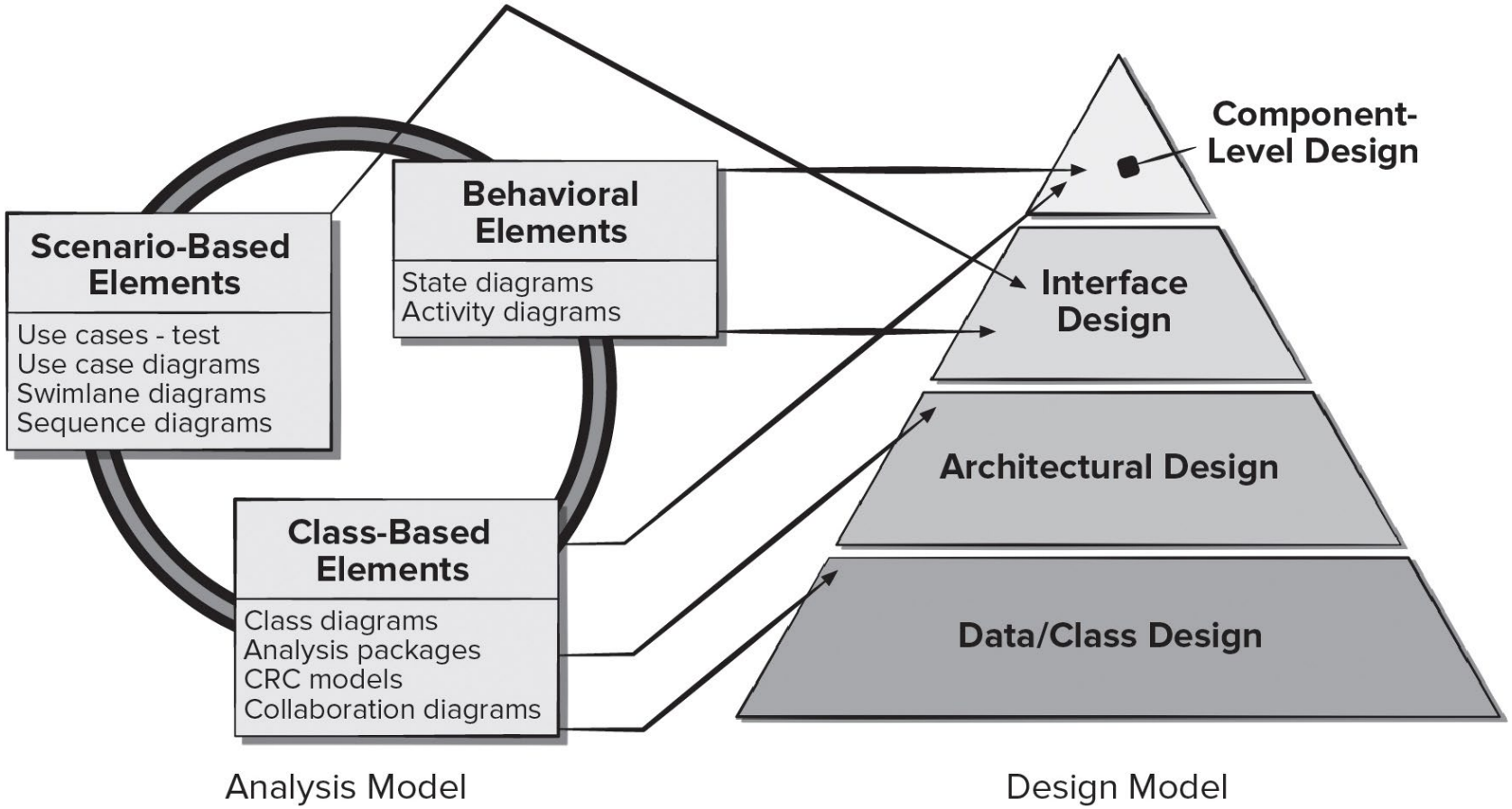
- Encompasses the set of principles, concepts, and practices that lead to the development of a high quality system or product.
- Design principles establish and overriding philosophy that guides the designer as the work is performed.
- Design concepts must be understood before the mechanics of design practice are applied.
- Software design practices change continuously as new methods, better analysis, and broader understanding evolve.

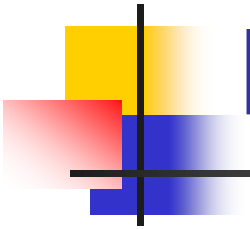


Software Engineering Design

- Data/Class design – transforms analysis classes into implementation classes and data structures.
- Architectural design – defines relationships among the major software structural elements.
- Interface design – defines how software elements, hardware elements, and end-users communicate.
- Component-level design – transforms structural elements into procedural descriptions of software components.

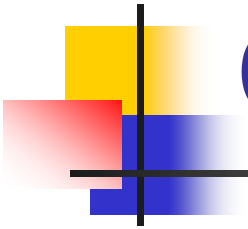
Mapping Requirements Model to Design Model





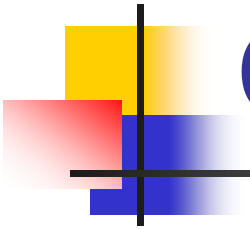
Design and Quality

- The design must implement all of the explicit requirements contained in the analysis model, and it must accommodate all of the implicit requirements desired by the customer.
- The design should be a readable, understandable guide for those who generate code and for those who test and subsequently support the software.
- The design should provide a complete picture of the software, addressing the data, functional, and behavioral domains from an implementation perspective.



Quality Guidelines

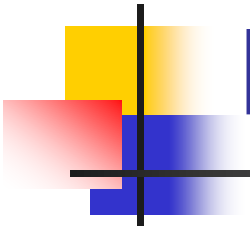
1. A design should exhibit an architecture (a) created using recognizable architectural styles or patterns, (b) composed of well designed components (c) implemented in an evolutionary fashion.
2. A design should be modular.
3. A design should contain distinct representations of data, architecture, interfaces, and components.
4. A design should lead to data structures that are drawn from recognizable data patterns.
5. A design should contain functionally independent components.
6. A design should lead to interfaces that reduce the complexity of connections between components and the external environment.
7. A design should be derived using a repeatable method that is driven by software requirements analysis.
8. A design should be represented using meaningful notation.



Common Design Characteristics

Each new software design methodology introduces unique heuristics and notions – yet they each contain:

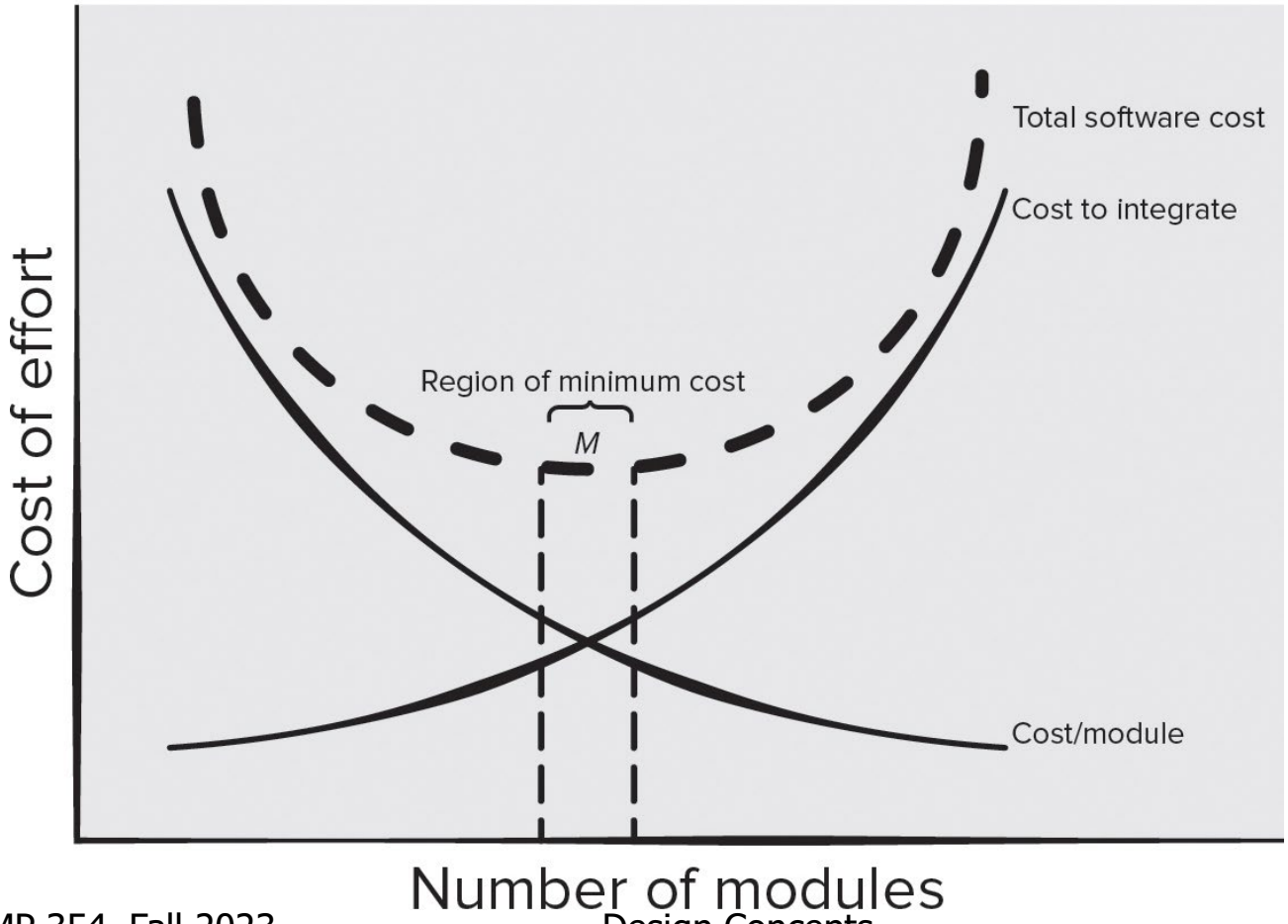
1. A mechanism for the translating the requirements model into a design representation.
2. A notation for representing functional components and their interfaces.
3. Heuristics for refinement and partitioning.
4. Guidelines for quality assessment.

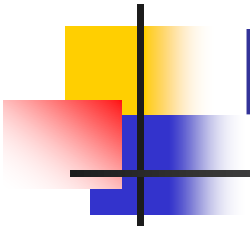


Design Concepts

- Abstraction – data (named collection of data describing data object), procedural (named sequence of instructions with specific and limited function).
- Architecture - overall structure or organization of software components, ways components interact, and structure of data used by components.
- Design Patterns - describe a design structure that solves a well-defined design problem within a specific context.
- Separation of concerns - any complex problem can be more easily handled if it is subdivided into pieces.
- Modularity—compartmentalization of data and function.

Modularity and Software Cost

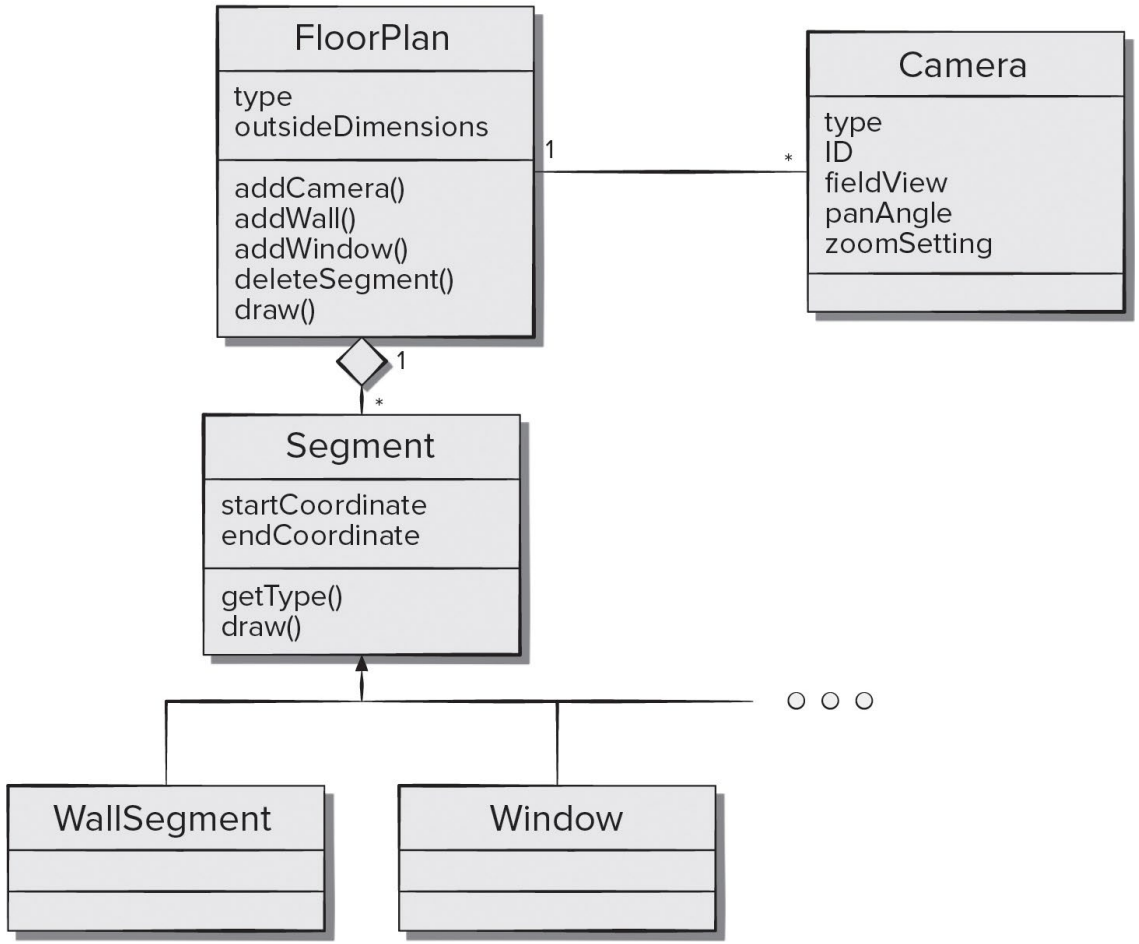


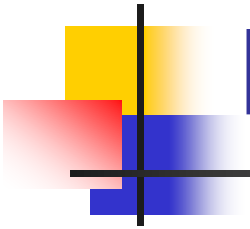


Design Concepts

- Information Hiding - controlled interfaces which define and enforces access to component procedural detail and any local data structure used by the component.
- Functional independence - single-minded (high cohesion) components with aversion to excessive interaction with other components (low coupling).
- Stepwise Refinement – incremental elaboration of detail for all abstractions.
- Refactoring—a reorganization technique that simplifies the design without changing functionality.
- Design Classes—provide design detail that will enable analysis classes to be implemented.

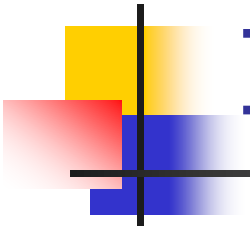
Design Class Example





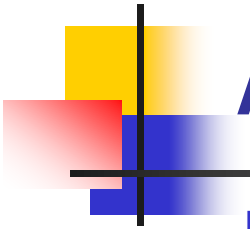
Design Class Characteristics

- Complete - includes all necessary attributes and methods) and sufficient (contains only those methods needed to achieve class intent).
- Primitiveness – each class method focuses on providing one service.
- High cohesion – small, focused, single-minded classes.
- Low coupling – class collaboration kept to minimum.



Information Hiding

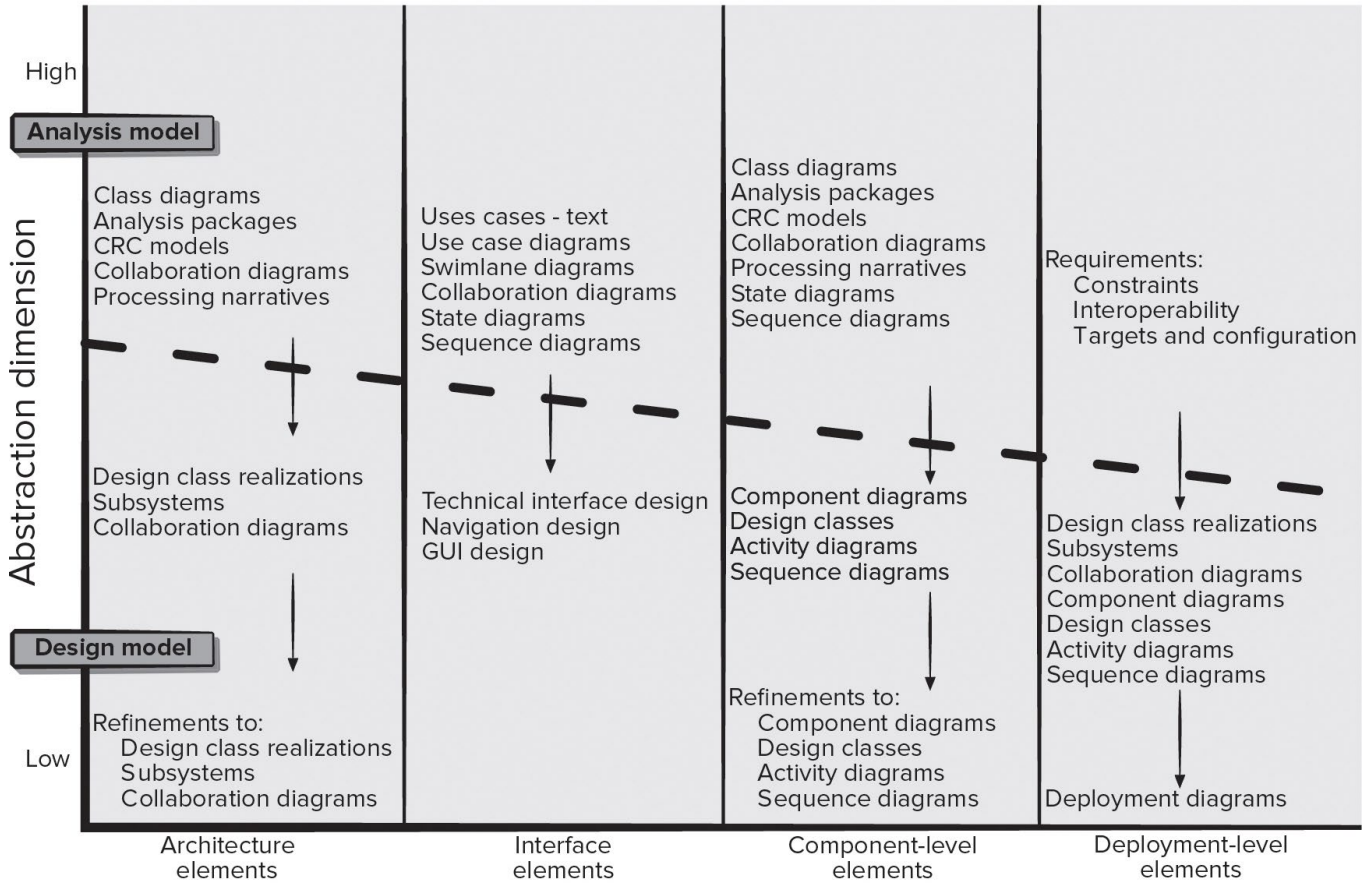
- Reduces the likelihood of "side effects."
- Limits the global impact of local design decisions.
- Emphasizes communication through controlled interfaces.
- Discourages the use of global data.
- Leads to encapsulation—an attribute of high quality design.
- Results in higher quality software.

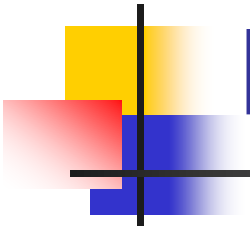


Architecture Properties

- Structural properties. This aspect of the architectural design representation defines the components of a system (for example, modules, objects, filters) and the manner components are packaged and interact with one another.
- Extra-functional properties. The architectural design description should address how the design architecture achieves requirements for performance, capacity, reliability, security, adaptability, and other characteristics.
- Families of related systems. The architectural design should draw upon repeatable patterns (building blocks) often encountered in the design of similar systems.

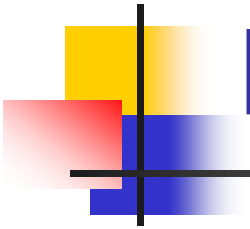
Design Model





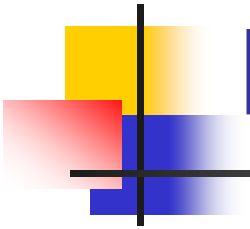
Design Modeling Principles

- **Principle 1**. Design should be traceable to the requirements model.
- **Principle 2**. Always consider the architecture of the system to be built.
- **Principle 3**. Design of data is as important as design of processing functions.
- **Principle 4**. Interfaces (both internal and external) must be designed with care.
- **Principle 5**. User interface design should be tuned to the needs of the end-user and stress ease of use.



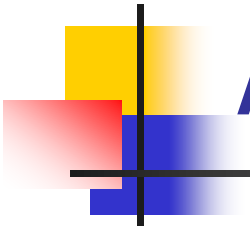
Design Modeling Principles

- **Principle 6**. Component-level design should be functionally independent.
- **Principle 7**. Components should be loosely coupled to each other than the environment.
- **Principle 8**. Design representations (models) should be easily understandable.
- **Principle 9**. The design should be developed iteratively.
- **Principle 10**. Creation of a design model does not preclude using an agile approach.



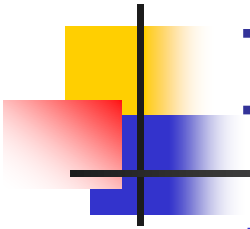
Data Design Elements

- Data model – data objects and database architectures.
  - Examines data objects independently of processing.
  - Focuses attention on the data domain.
  - Creates a model at the customer's level of abstraction.
  - Indicates how data objects relate to one another.
- Data object can be an external entity, a thing, an event, a place, a role, an organizational unit, or a structure.
- Data objects contain a set of attributes that act as an quality, characteristic, or descriptor of the object.
- Data objects may be connected to one another in many different ways.



Architectural Design Elements

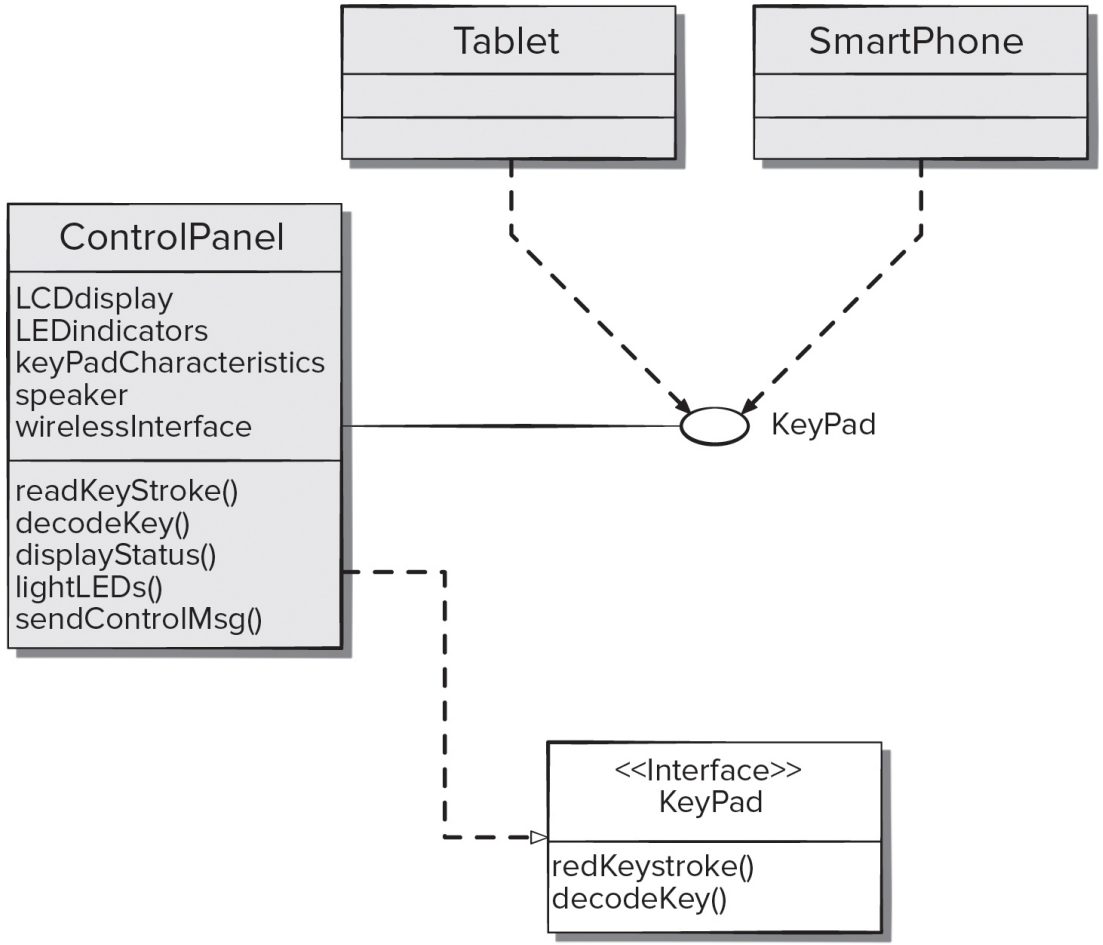
- Architectural design for software - equivalent to the floor plan for a house.
- The architectural model is derived from three sources:
  - Information about the application domain for the software to be built.
  - Specific requirements model elements such as data flow analysis classes and their relationships (collaborations) for the problem at hand, and
  - Availability of architectural patterns and styles.



Interface Design Elements

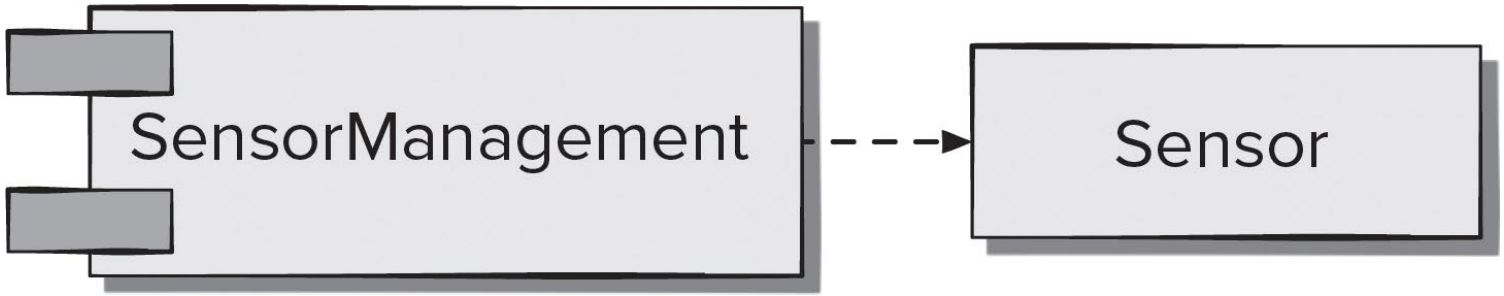
- Interface is a set of operations that describes the externally observable behavior of a class and provides access to its public operations.
- Important elements:
  - User interface (UI).
  - External interfaces to other systems.
  - Internal interfaces between various design components.
- UI or User Experience (UX) is a major engineering action to ensure the creation on usable software products.
- Internal and external interfaces should incorporate both error checking and appropriate security features.

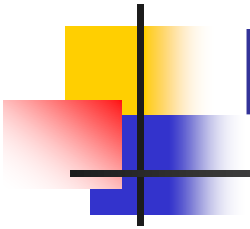
Interface Model for Control Panel



Component-Level Design Elements

- Describes the internal detail of each software component.
- Defines:
  - Data structures for all local data objects.
  - Algorithmic detail for all component processing functions.
  - Interface that allows access to all component operations.
- Modeled using U M L component diagrams.





Deployment Design Elements

- Indicates how software functionality and subsystems will be allocated within the physical computing environment.
- Modeled using U M L deployment diagrams.
  - Descriptor form deployment diagrams - show the computing environment but does not indicate configuration details.
  - Instance form deployment diagrams - identify specific hardware configurations and are developed in the latter stages of design.

UML
Deployment Instance Diagram