Lecture 6

- **Design Modelling 1**

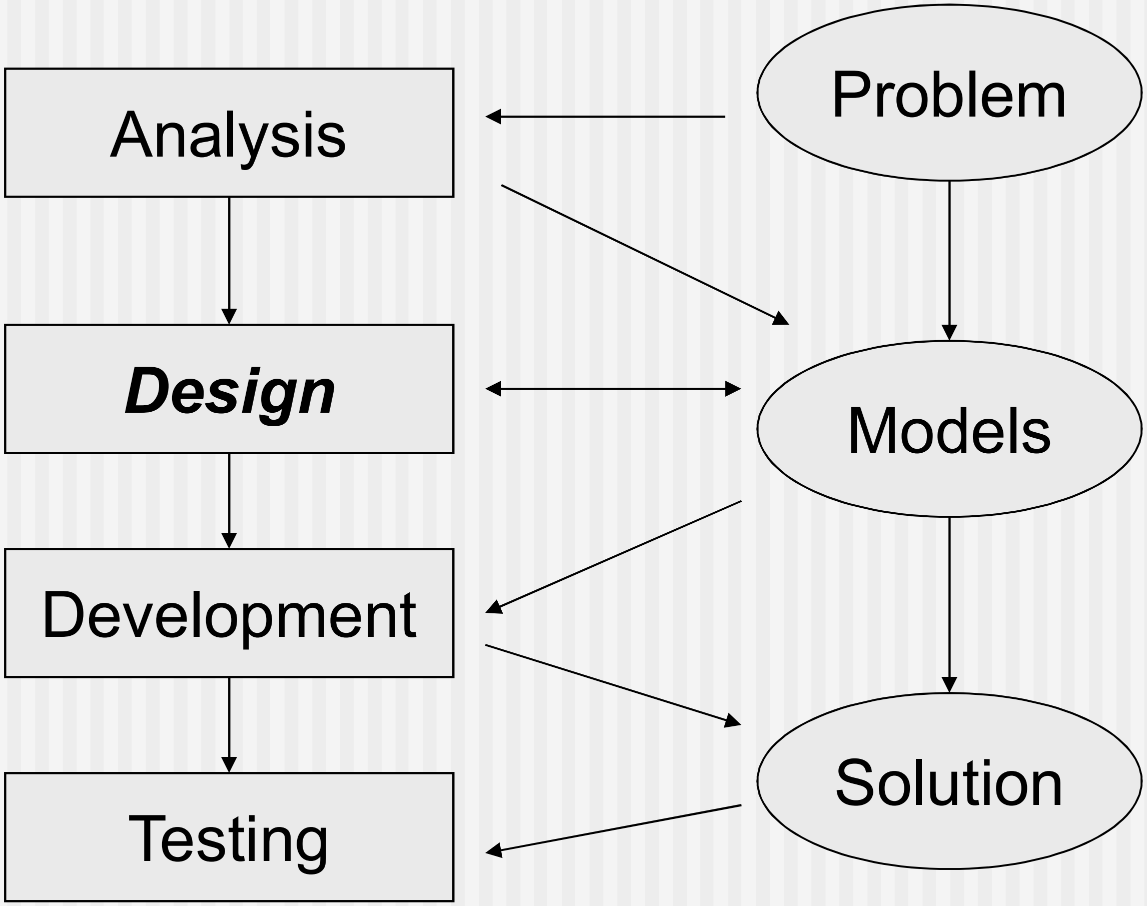
“How to create the representations of the software?”

Topics

- Elements of Design Model
- Data/Class Design
- Architectural Design
- Architectural Concepts

1. Elements of Design Model

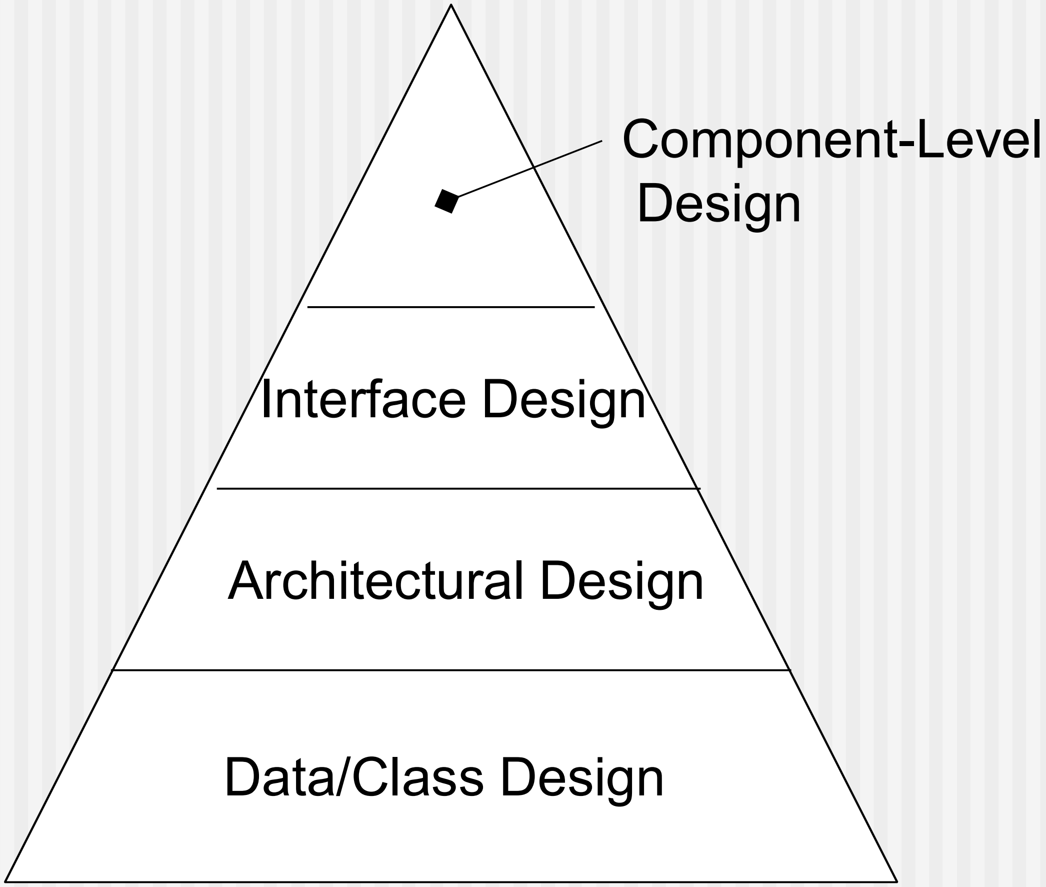
Design is The Second Task



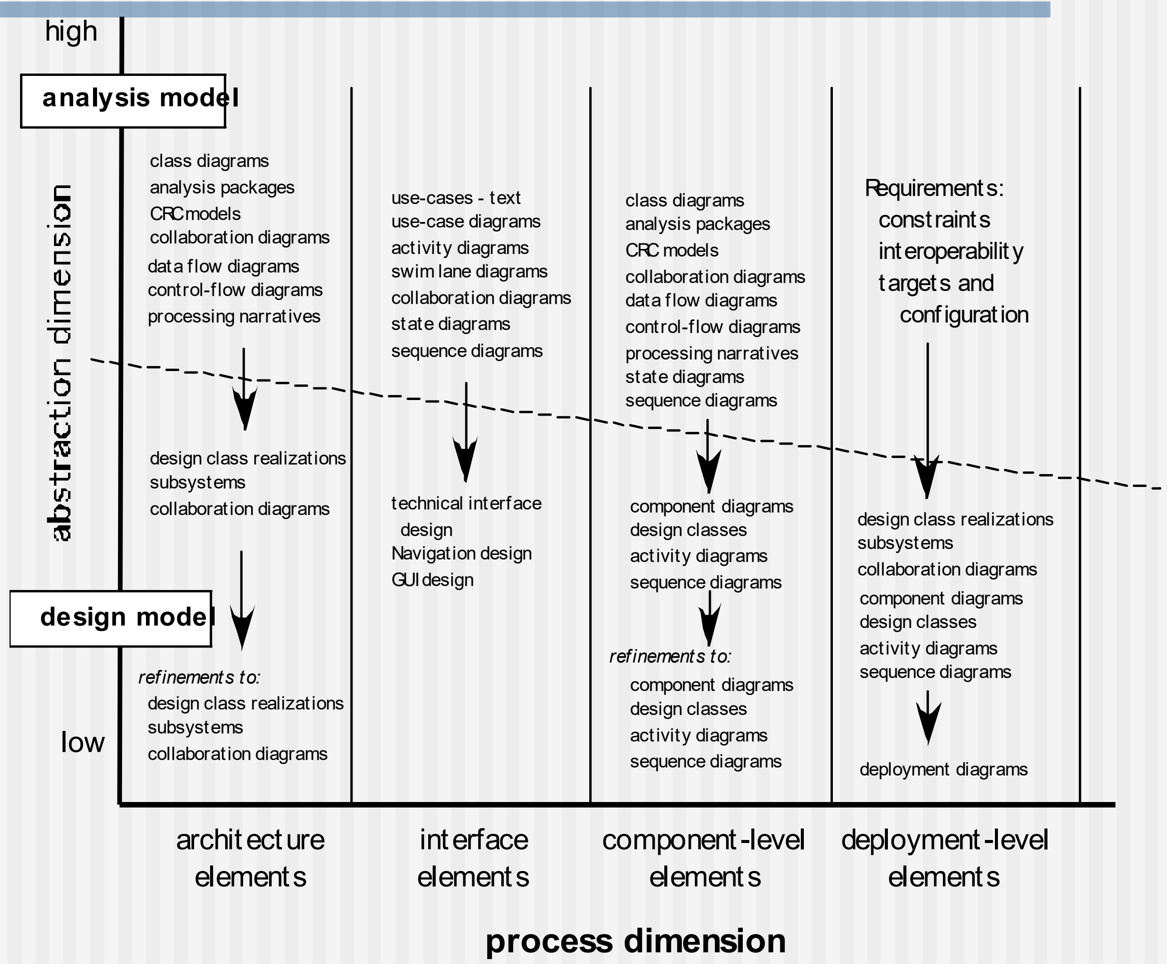
Designing Software

- From our understanding of the problem, we start building the software
- Translate the analysis model into the design model
- Map the information from the analysis model to the design representations - data design, architectural design, interface design, component-level design

The Design Model



The Design Model



high

analysis model

abstraction dimension

class diagrams
analysis packages
CRC models
collaboration diagrams
data flow diagrams
control-flow diagrams
processing narratives

use-cases - text
use-case diagrams
activity diagrams
swim lane diagrams
collaboration diagrams
state diagrams
sequence diagrams

class diagrams
analysis packages
CRC models
collaboration diagrams
data flow diagrams
control-flow diagrams
processing narratives
state diagrams
sequence diagrams

Requirements:
constraints
interoperability
targets and
configuration

design class realizations
subsystems
collaboration diagrams

technical interface
design
Navigation design
GUI design

component diagrams
design classes
activity diagrams
sequence diagrams

design class realizations
subsystems
collaboration diagrams
component diagrams
design classes
activity diagrams
sequence diagrams

design model

refinements to:
design class realizations
subsystems
collaboration diagrams

refinements to:
component diagrams
design classes
activity diagrams
sequence diagrams

deployment diagrams

low

architecture elements

interface elements

component-level elements

deployment-level elements

process dimension

These slides are designed to accompany *Software Engineering: A Practitioner’s Approach, 8/e* (McGraw-Hill, 2014) Slides copyright 2014 by Roger Pressman.

Design Model Elements

- Data elements
  - Data model --> data structures
  - Data model --> database architecture
- Architectural elements
  - Application domain
  - Analysis classes, their relationships, collaborations and behaviors are transformed into design realizations
  - Patterns and “styles” (Chapters 9 and 12)
- Interface elements
  - the user interface (UI)
  - external interfaces to other systems, devices, networks or other producers or consumers of information
  - internal interfaces between various design components.
- Component elements
- Deployment elements

Data Elements

- Data design – creates a model of data and/or information that is represented at a high level of abstraction
- The data model is then refined into progressively more implementation-specific representations that can be processed by the computer-based system

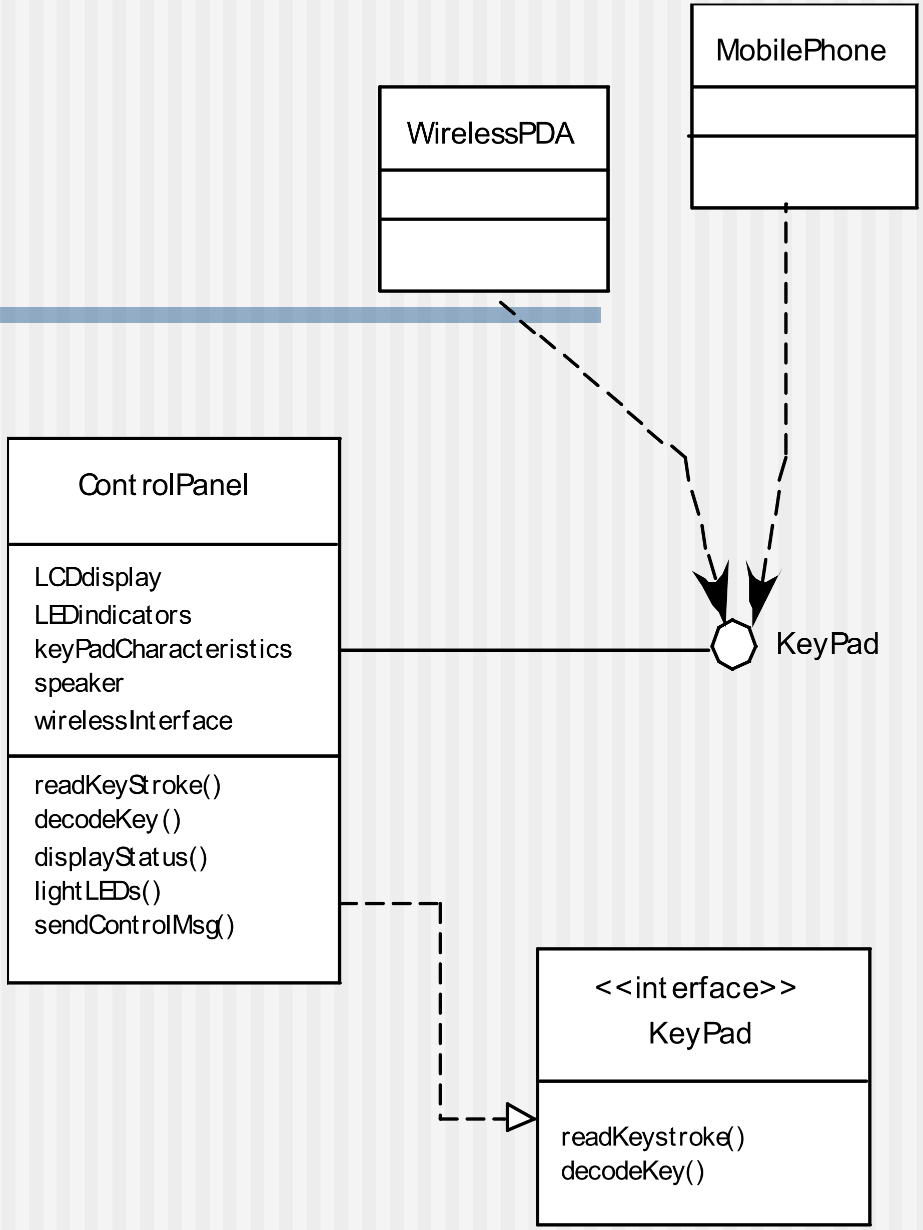
Architectural Elements

- The architectural model [Sha96] is derived from three sources:
  - information about the application domain for the software to be built;
  - specific requirements model elements such as data flow diagrams or analysis classes, their relationships and collaborations for the problem at hand, and
  - the availability of architectural patterns (Chapter 16) and styles (Chapter 13).

Interface 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
- Modeled using UML communication diagrams (called collaboration diagrams in UML 1.x)

Interface Elements

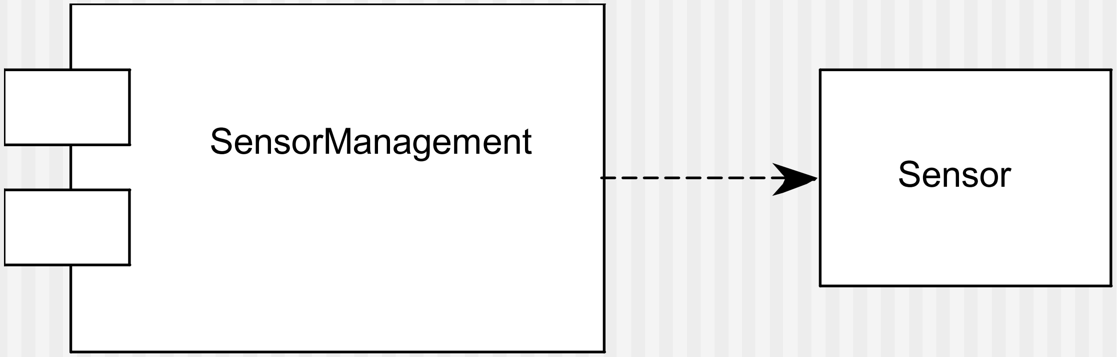


These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill, 2014) Slides copyright 2014 by Roger Pressman.

Component 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 UML component diagrams, UML activity diagrams, pseudocode (PDL), and sometimes flowcharts

Component Elements

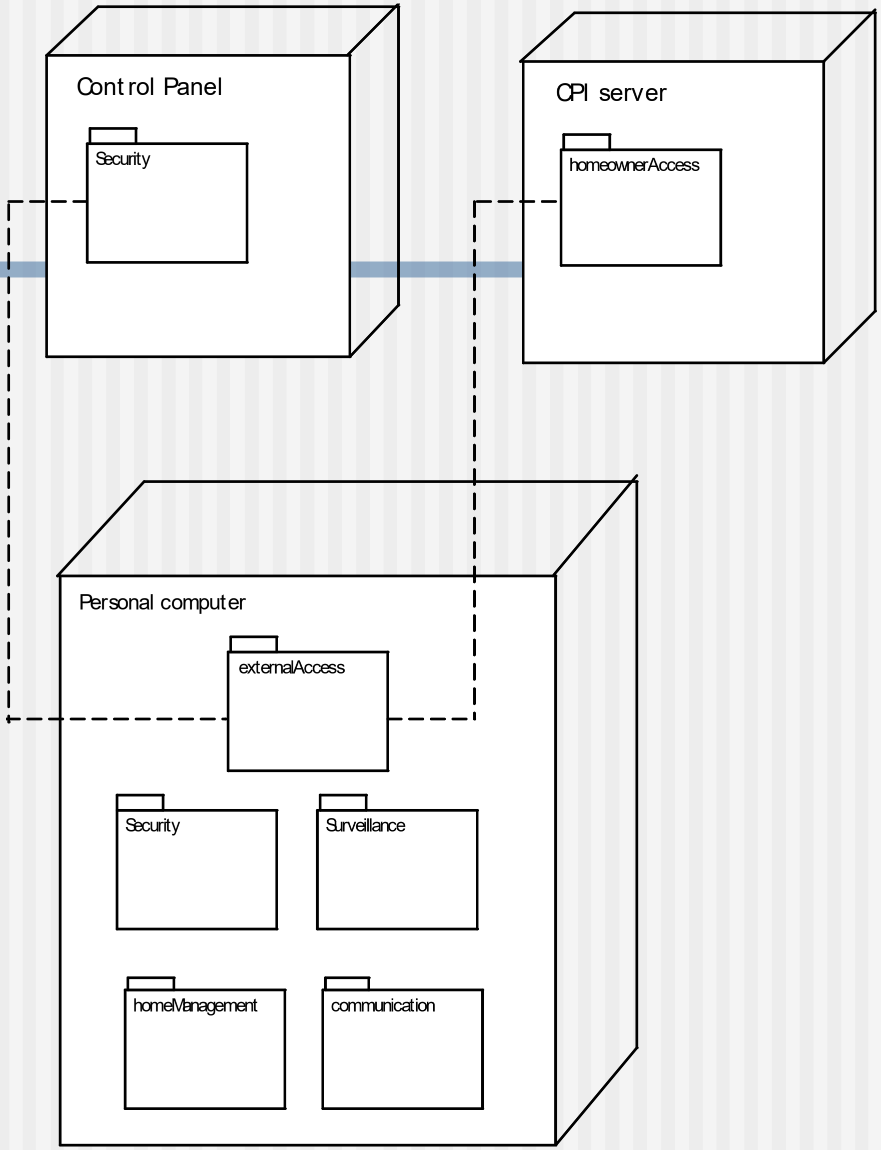


These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill, 2014) Slides copyright 2014 by Roger Pressman.

Deployment Elements

- Indicates how software functionality and subsystems will be allocated within the physical computing environment
- Modeled using UML deployment diagrams
- *Descriptor form* deployment diagrams show the computing environment but does not indicate configuration details
- *Instance form* deployment diagrams identifying specific named hardware configurations are developed during the latter stages of design

Deployment Elements



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill, 2014) Slides copyright 2014 by Roger Pressman.

2. Data/Class Design

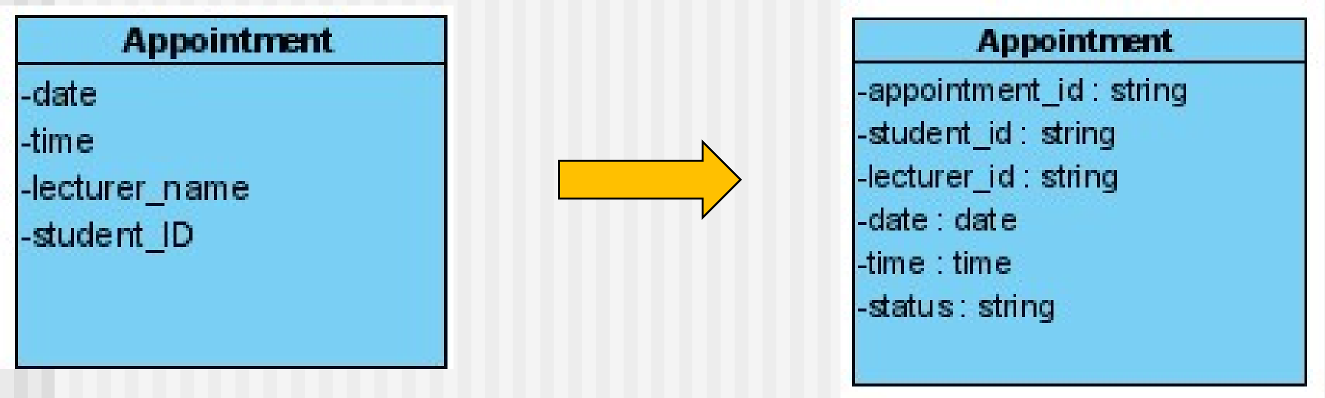
Data Design

- At program-component level, the design of data structures and the associated algorithms required to manipulate them is essential to the creation of high-quality applications
- At application level, the translation of a data model into a database is pivotal to achieving the business objectives of a system
- At business level, the collection of information stored in disparate databases and reorganized into a “data warehouse” enables data mining and knowledge discovery

Data Design

- Use logical representations of data objects (entity classes) identified during requirements definition and specification
- Well-designed data can lead to better program structure and modularity, and reduced procedural complexity
- Elaborate the classes with implementation details such as data types, processing attributes, relationships
- Create data dictionary to represent the database elements to be developed

Data Design



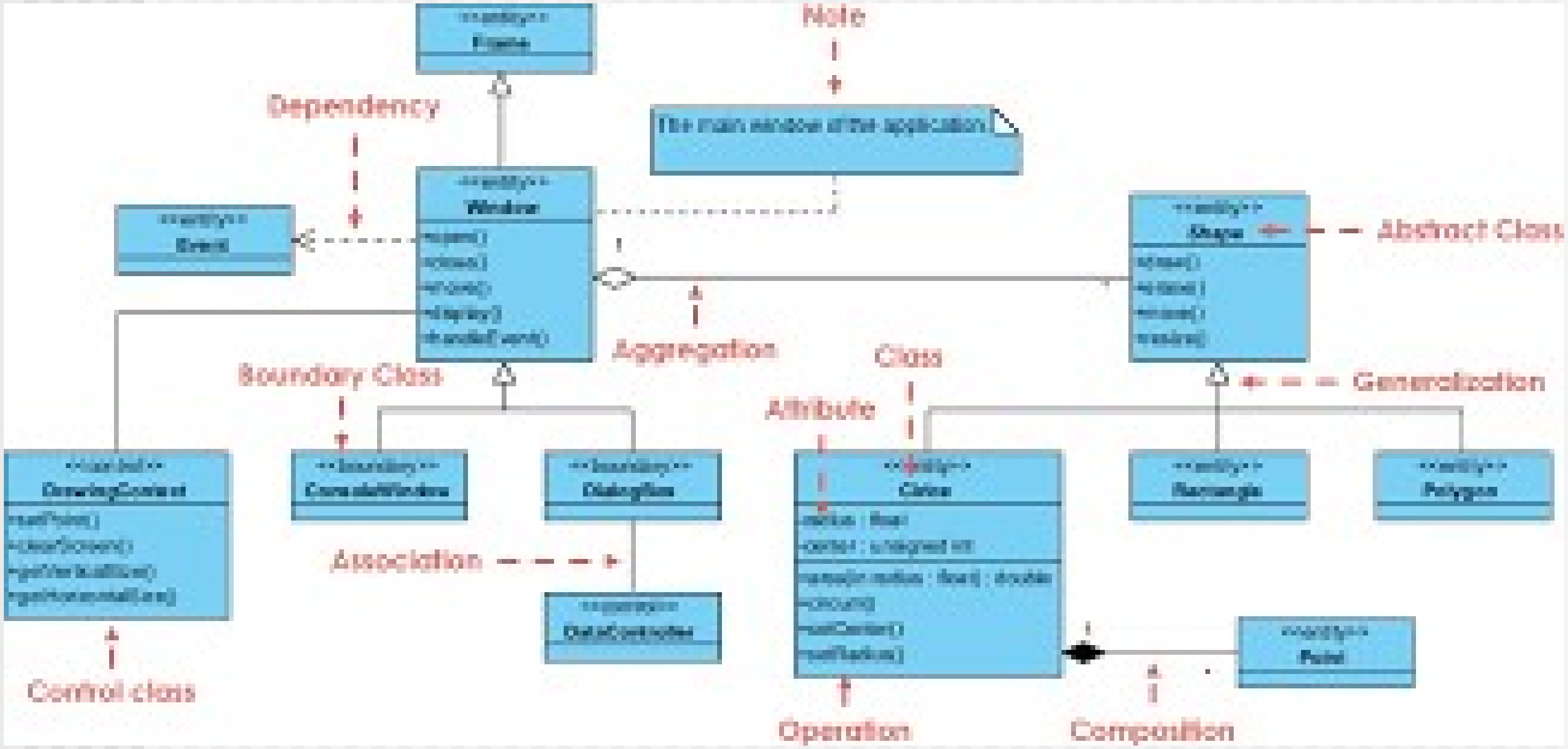
Appointment

-date
-time
-lecturer_name
-student_ID

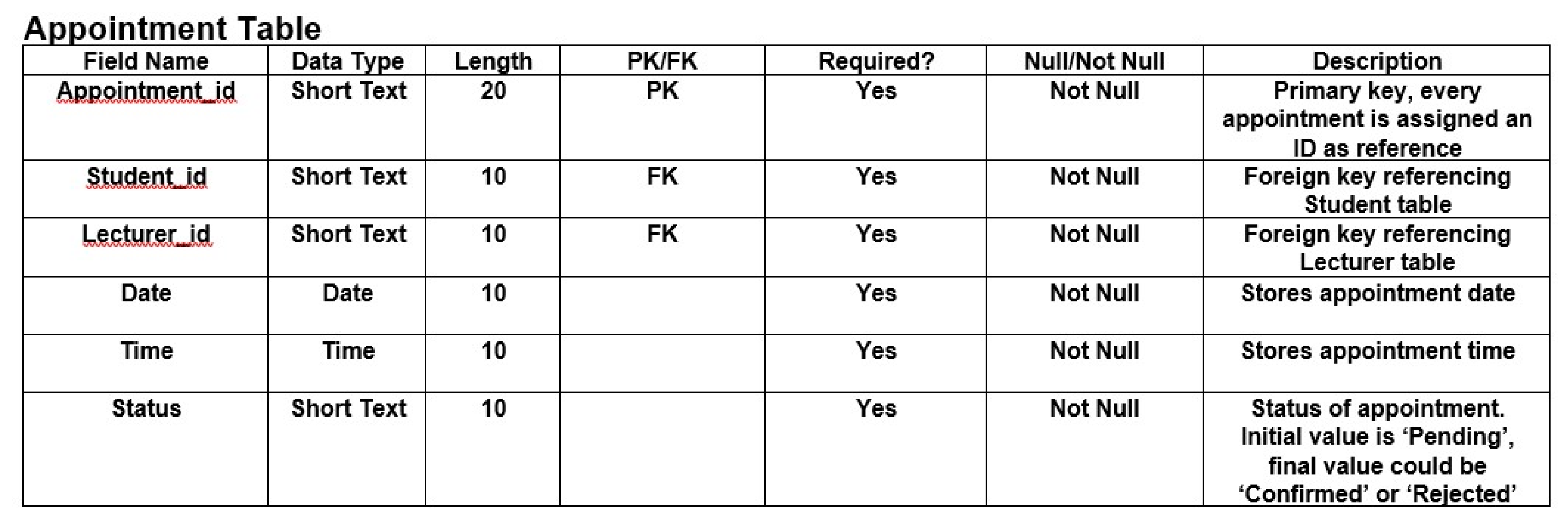
Appointment

-appointment_id : string
-student_id : string
-lecturer_id : string
-date : date
-time : time
-status : string

Data/Class Design Example



Data Dictionary



Appointment Table

Field Name	Data Type	Length	PK/FK	Required?	Null/Not Null	Description
Appointment_id	Short Text	20	PK	Yes	Not Null	Primary key, every appointment is assigned an ID as reference
Student_id	Short Text	10	FK	Yes	Not Null	Foreign key referencing Student table
Lecturer_id	Short Text	10	FK	Yes	Not Null	Foreign key referencing Lecturer table
Date	Date	10		Yes	Not Null	Stores appointment date
Time	Time	10		Yes	Not Null	Stores appointment time
Status	Short Text	10		Yes	Not Null	Status of appointment. Initial value is ‘Pending’, final value could be ‘Confirmed’ or ‘Rejected’

Data Structure Design

- Processing requirements may involve use of data structures such as arrays, structures/records, stacks, etc.
- Describe structure and its use
- Describe contents in similar way to database tables/files

3. Architectural Design

Why Architecture?

The architecture is not the operational software. Rather, it is a representation that enables a software engineer to:

(1) analyze the effectiveness of the design in meeting its stated requirements,

(2) consider architectural alternatives at a stage when making design changes is still relatively easy, and

(3) reduce the risks associated with the construction of the software.

Why is Architecture Important?

- Representations of software architecture are an enabler for communication between all parties (stakeholders) interested in the development of a computer-based system.
- The architecture highlights early design decisions that will have a profound impact on all software engineering work that follows and, as important, on the ultimate success of the system as an operational entity.
- Architecture “constitutes a relatively small, intellectually graspable mode of how the system is structured and how its components work together” [BAS03].

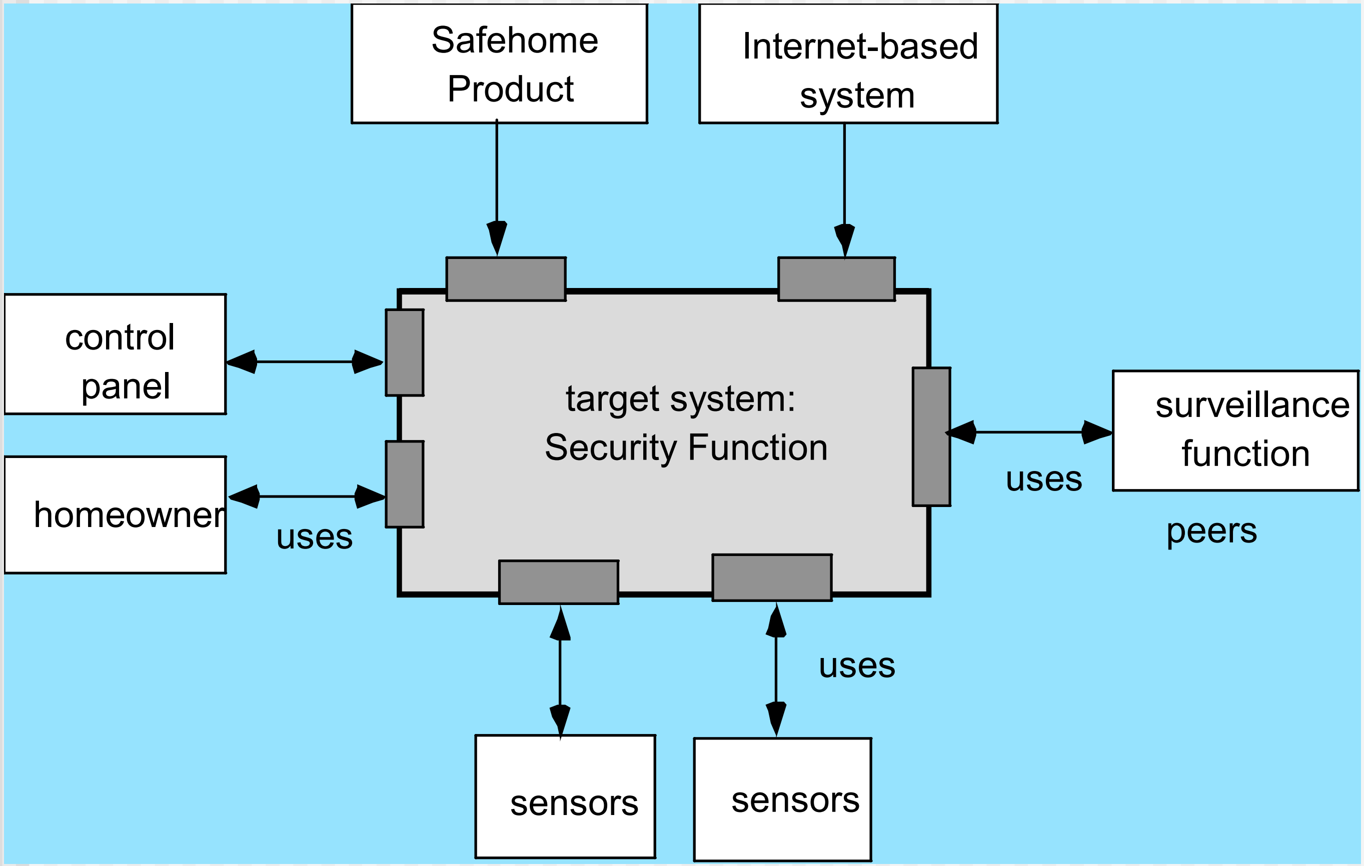
Architectural Descriptions

- The IEEE Computer Society has proposed IEEE-Std-1471-2000, *Recommended Practice for Architectural Description of Software-Intensive System,* [IEE00]
  - to establish a conceptual framework and vocabulary for use during the design of software architecture,
  - to provide detailed guidelines for representing an architectural description, and
  - to encourage sound architectural design practices.
- The IEEE Standard defines an *architectural description* (AD) as a "a collection of products to document an architecture."
  - The description itself is represented using multiple views, where each *view* is "a representation of a whole system from the perspective of a related set of [stakeholder] concerns."

Architectural Design

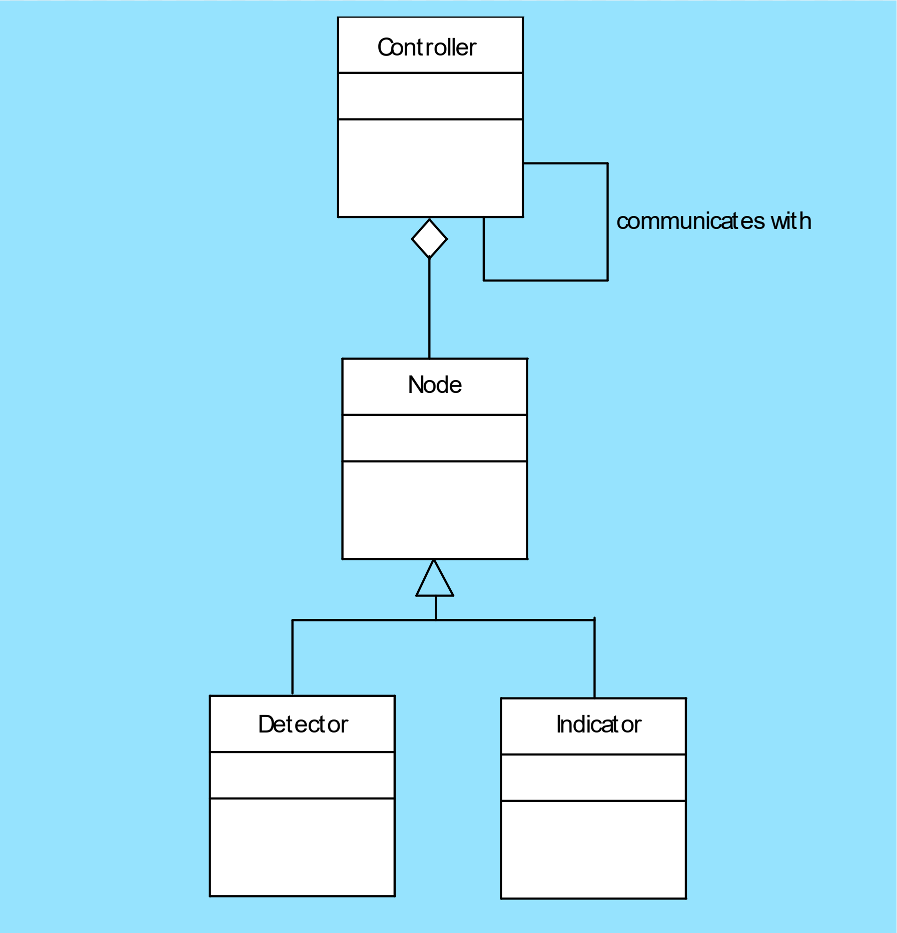
- The software must be placed into context
  - the design should define the external entities (other systems, devices, people) that the software interacts with and the nature of the interaction
- A set of architectural archetypes should be identified
  - An *archetype* is an abstraction (similar to a class) that represents one element of system behavior
- The designer specifies the structure of the system by defining and refining software components that implement each archetype

Architectural Context



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill, 2014). Slides copyright 2014 by Roger Pressman.

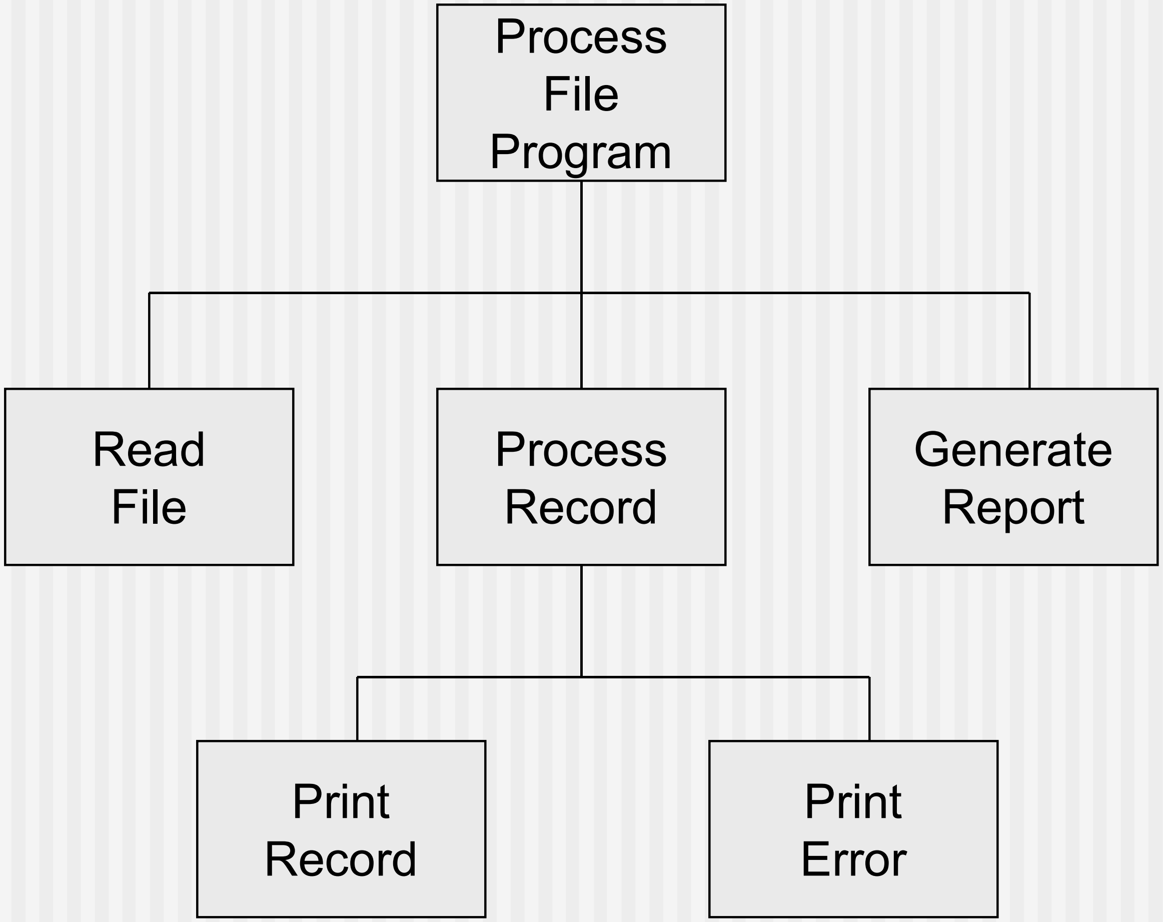
Archetypes



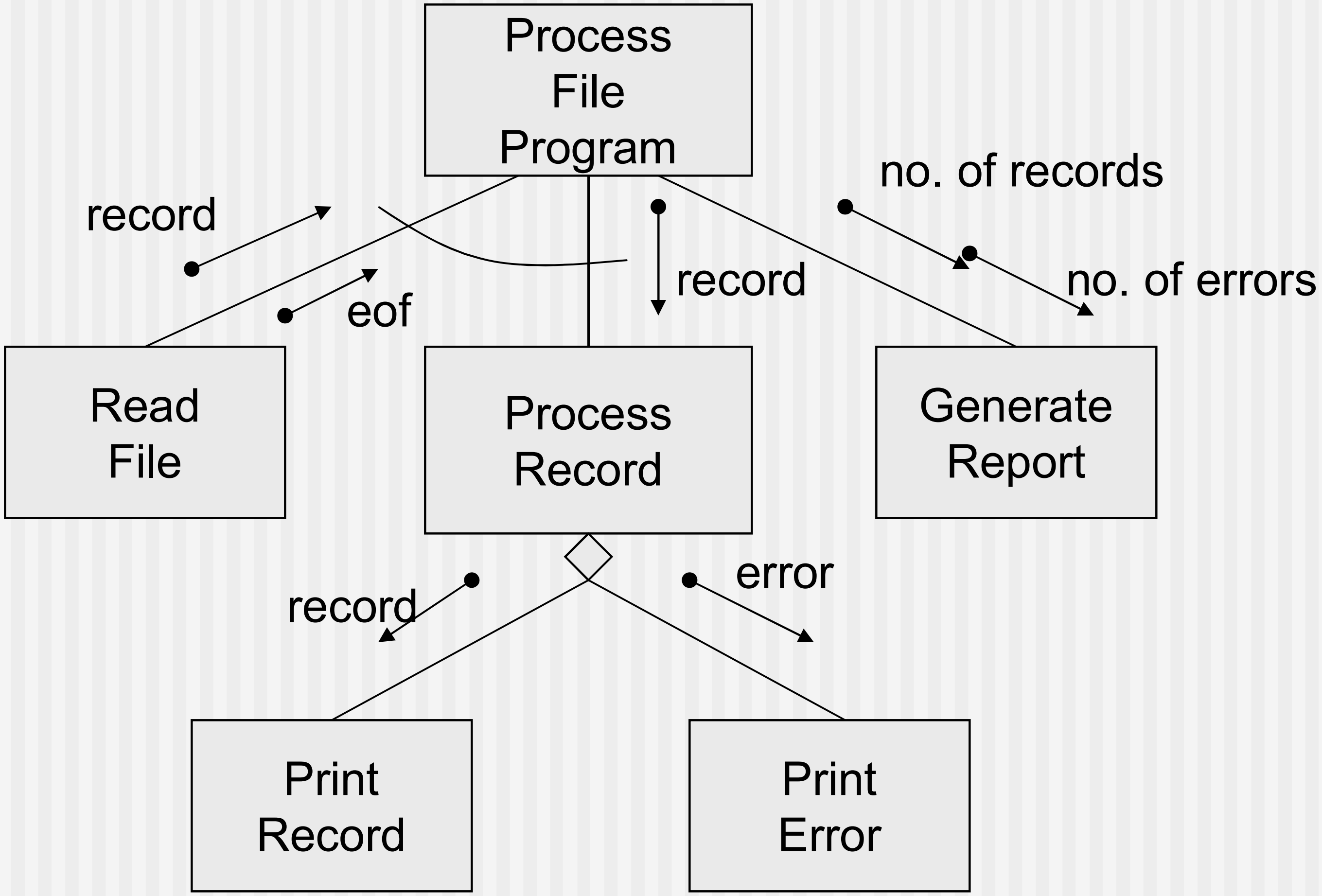
Architectural Structure

- Develop a modular program structure and represent the control relationship between modules
- Can be represented by
  - Hierarchy of modules
  - Structure chart
  - Component structure

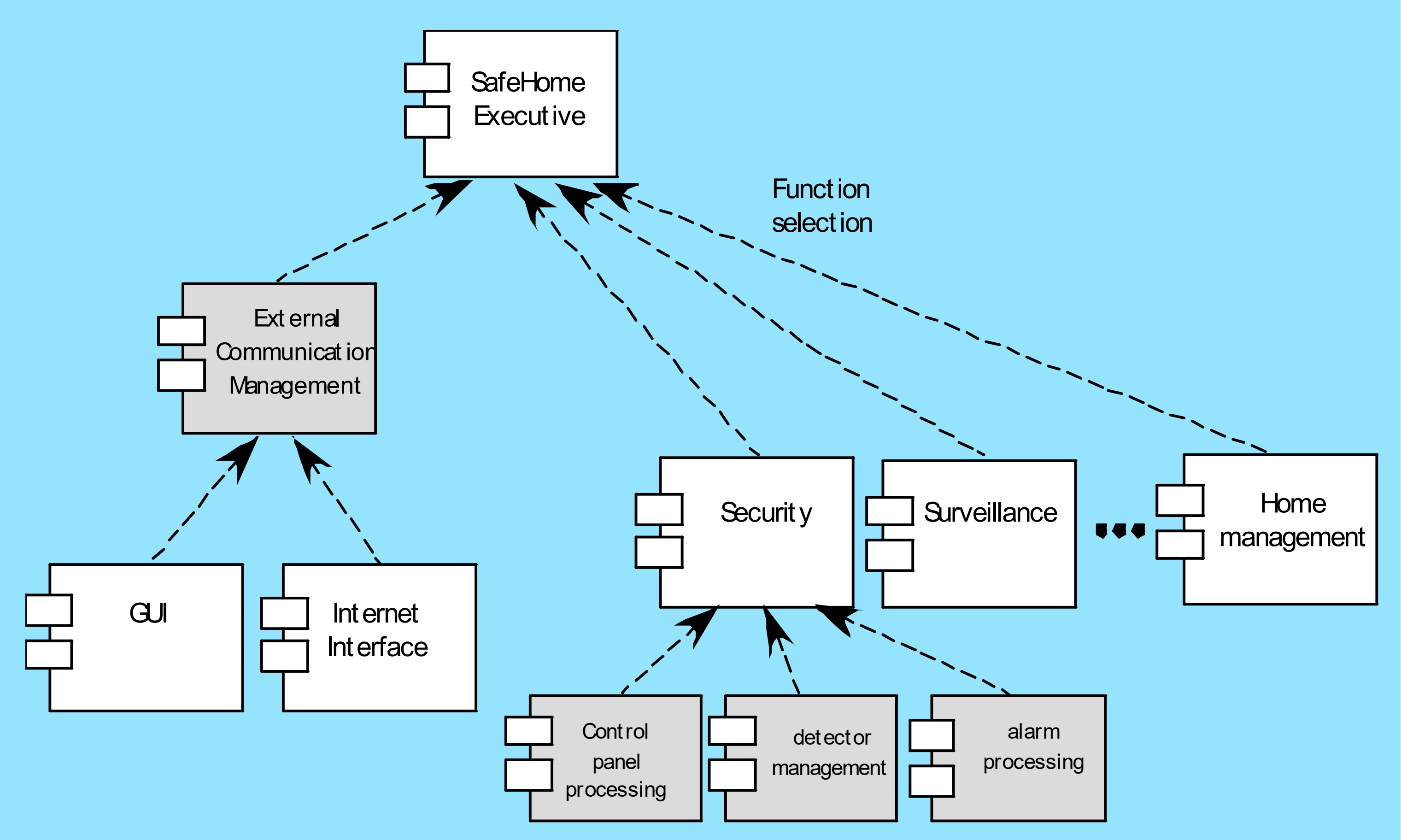
Hierarchy of Modules



Structure Chart

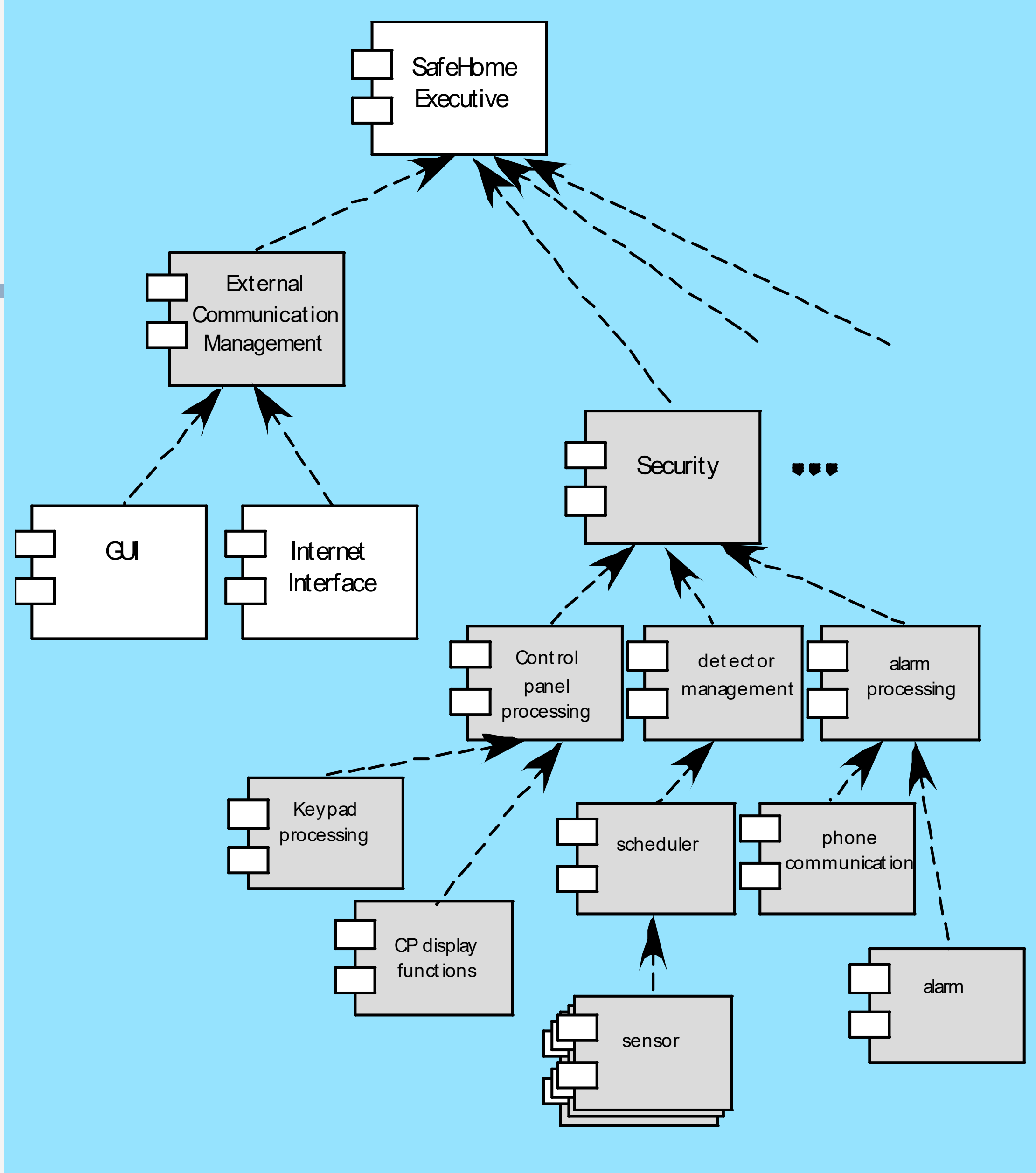


Component Structure



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill, 2014). Slides copyright 2014 by Roger Pressman.

Refined Component Structure



Architecture Reviews

- Assess the ability of the software architecture to meet the systems quality requirements and identify potential risks
- Have the potential to reduce project costs by detecting design problems early
- Often make use of experience-based reviews, prototype evaluation, and scenario reviews, and checklists

4. Architectural Concepts

Architectural Considerations

- Economy – The best software is uncluttered and relies on abstraction to reduce unnecessary detail.
- Visibility – Architectural decisions and the reasons for them should be obvious to software engineers who examine the model at a later time.
- Spacing – Separation of concerns in a design without introducing hidden dependencies.
- Symmetry – Architectural symmetry implies that a system is consistent and balanced in its attributes.
- Emergence – Emergent, self-organized behavior and control.

Architectural Complexity

- the overall complexity of a proposed architecture is assessed by considering the dependencies between components within the architecture [Zha98]
  - *Sharing dependencies* represent dependence relationships among consumers who use the same resource or producers who produce for the same consumers.
  - *Flow dependencies* represent dependence relationships between producers and consumers of resources.
  - *Constrained dependencies* represent constraints on the relative flow of control among a set of activities.

Architectural Genres

- *Genre* implies a specific category within the overall software domain.
- Within each category, you encounter a number of subcategories.
  - For example, within the genre of *buildings*, you would encounter the following general styles: houses, condos, apartment buildings, office buildings, industrial building, warehouses, and so on.
  - Within each general style, more specific styles might apply. Each style would have a structure that can be described using a set of predictable patterns.

Architectural Styles

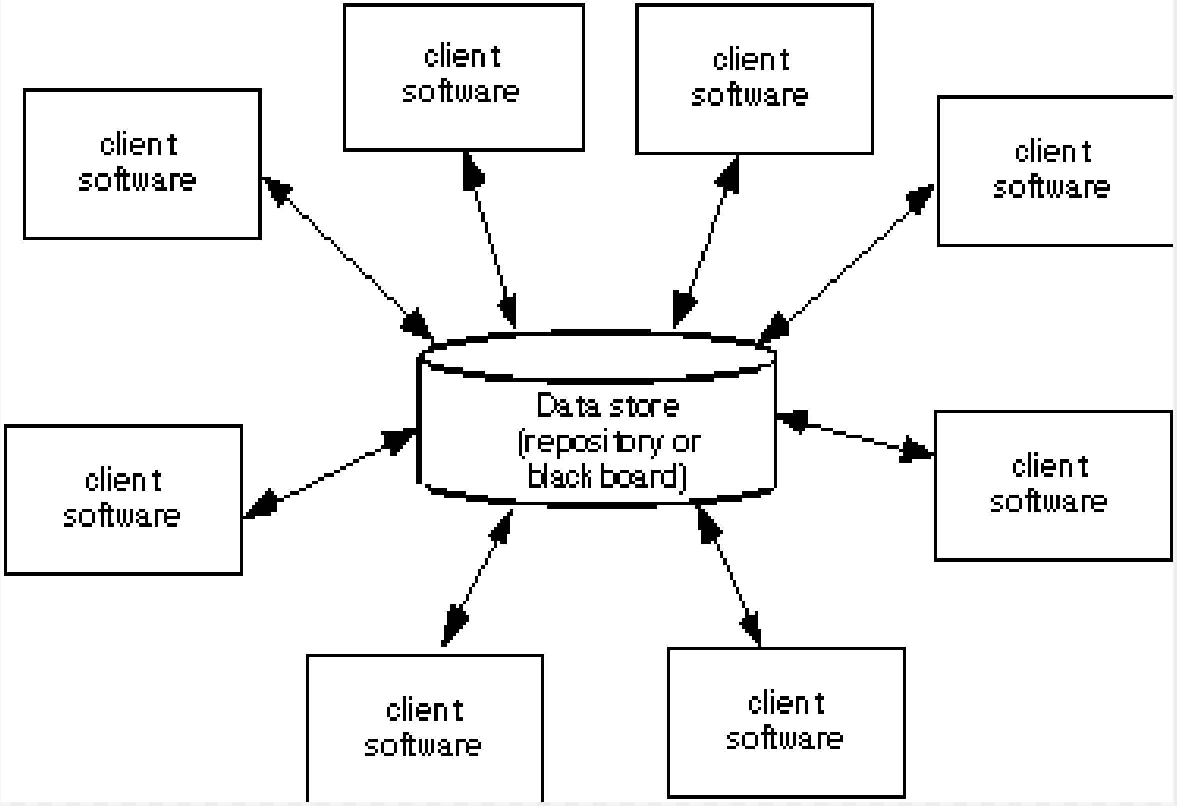
Each style describes a system category that encompasses:

- (1) a **set of components** (e.g., a database, computational modules) that perform a function required by a system,
- (2) a **set of connectors** that enable "communication, coordination and cooperation" among components,
- (3) **constraints** that define how components can be integrated to form the system, and
- (4) **semantic models** that enable a designer to understand the overall properties of a system by analyzing the known properties of its constituent parts.

Architectural Styles

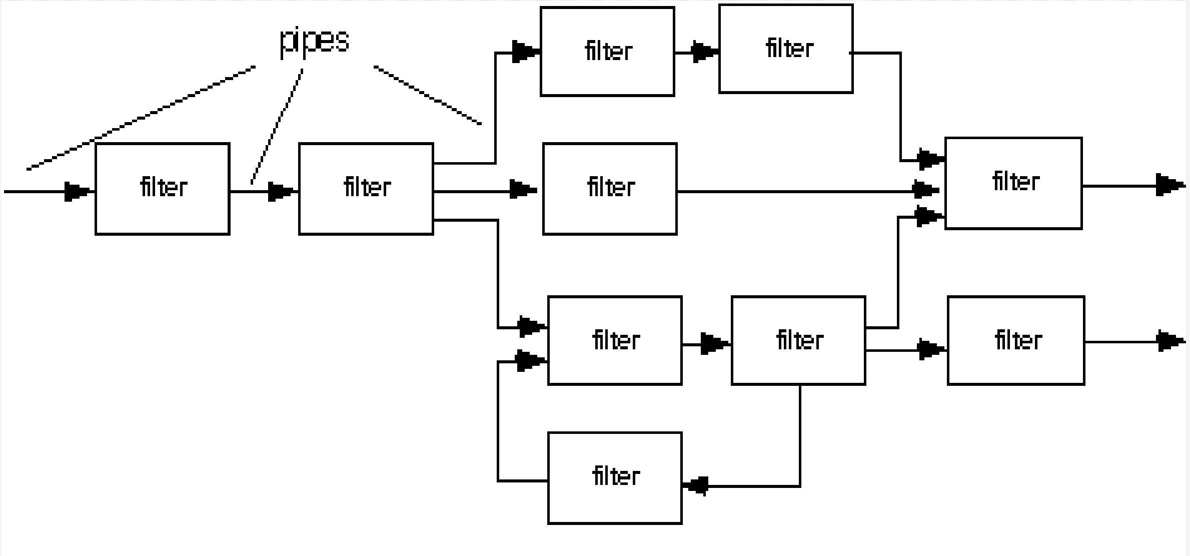
- Data-centered architectures
- Data flow architectures
- Call and return architectures
- Object-oriented architectures
- Layered architectures

Data-Centered Architecture

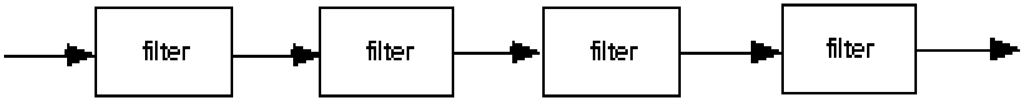


These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill, 2014). Slides copyright 2014 by Roger Pressman.

Data Flow Architecture



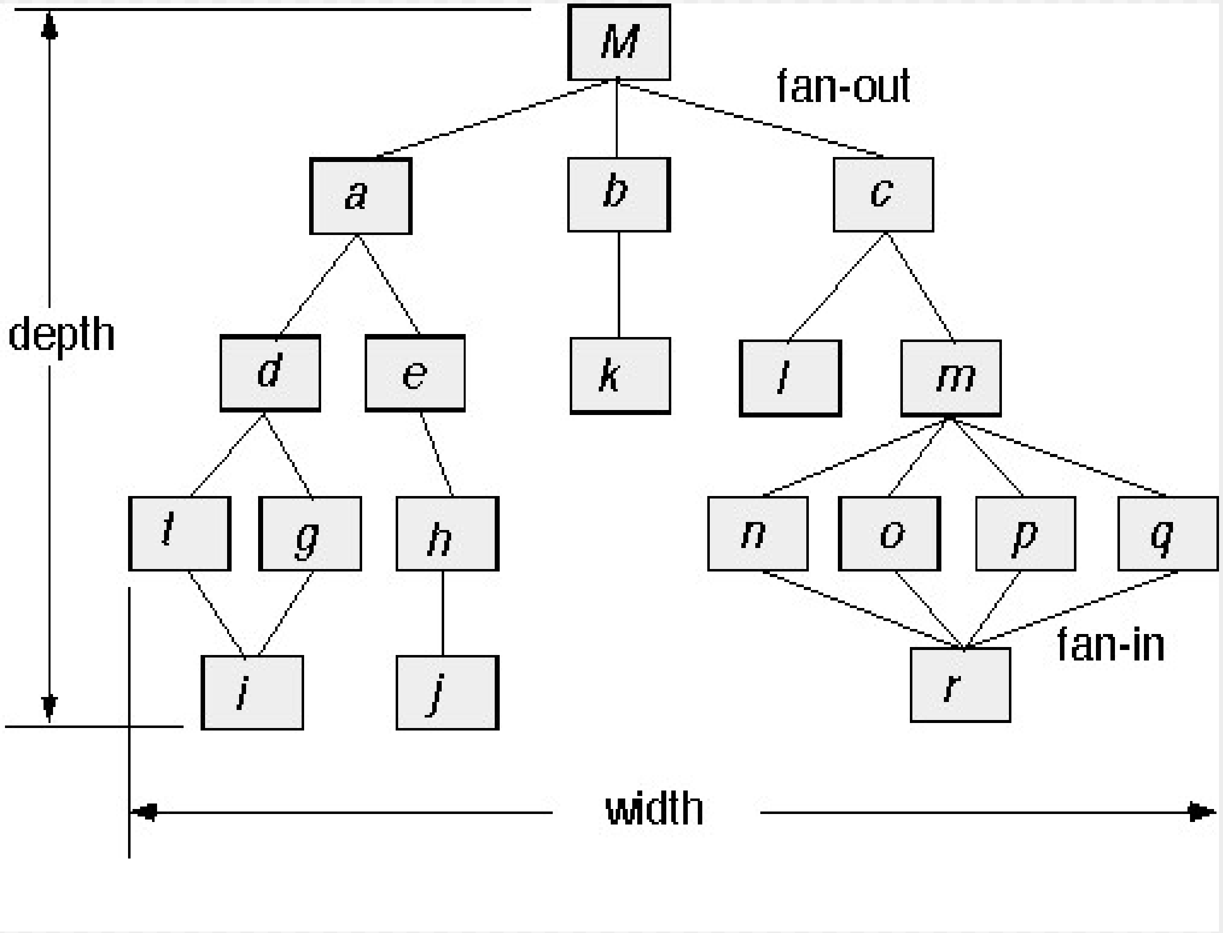
(a) pipes and filters



(b) batch sequential

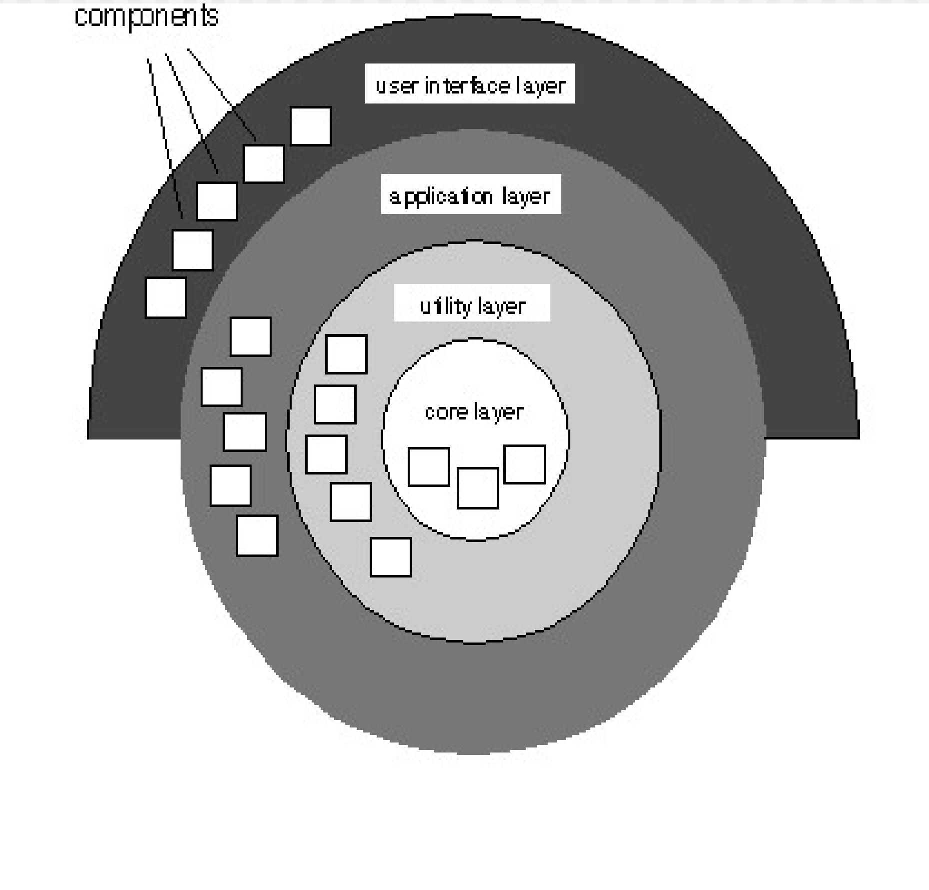
These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill, 2014). Slides copyright 2014 by Roger Pressman.

Call and Return Architecture



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill, 2014). Slides copyright 2014 by Roger Pressman.

Layered Architecture



These slides are designed to accompany *Software Engineering: A Practitioner's Approach, 8/e* (McGraw-Hill, 2014). Slides copyright 2014 by Roger Pressman.

Architectural Patterns

- Concurrency—applications must handle multiple tasks in a manner that simulates parallelism
  - *operating system process management* pattern
  - *task scheduler* pattern
- Persistence—Data persists if it survives past the execution of the process that created it. Two patterns are common:
  - a *database management system* pattern that applies the storage and retrieval capability of a DBMS to the application architecture
  - an *application level persistence* pattern that builds persistence features into the application architecture
- Distribution— the manner in which systems or components within systems communicate with one another in a distributed environment
  - A *broker* acts as a 'middle-man' between the client component and a server component.

ADL

- *Architectural description language* (ADL) provides a semantics and syntax for describing a software architecture
- Provide the designer with the ability to:
  - decompose architectural components
  - compose individual components into larger architectural blocks and
  - represent interfaces (connection mechanisms) between components.

Agility and Architecture

- To avoid rework, user stories are used to create and evolve an architectural model (walking skeleton) before coding
- Hybrid models which allow software architects contributing users stories to the evolving storyboard
- Well run agile projects include delivery of work products during each sprint
- Reviewing code emerging from the sprint can be a useful form of architectural review