Introduction of Software Engineering

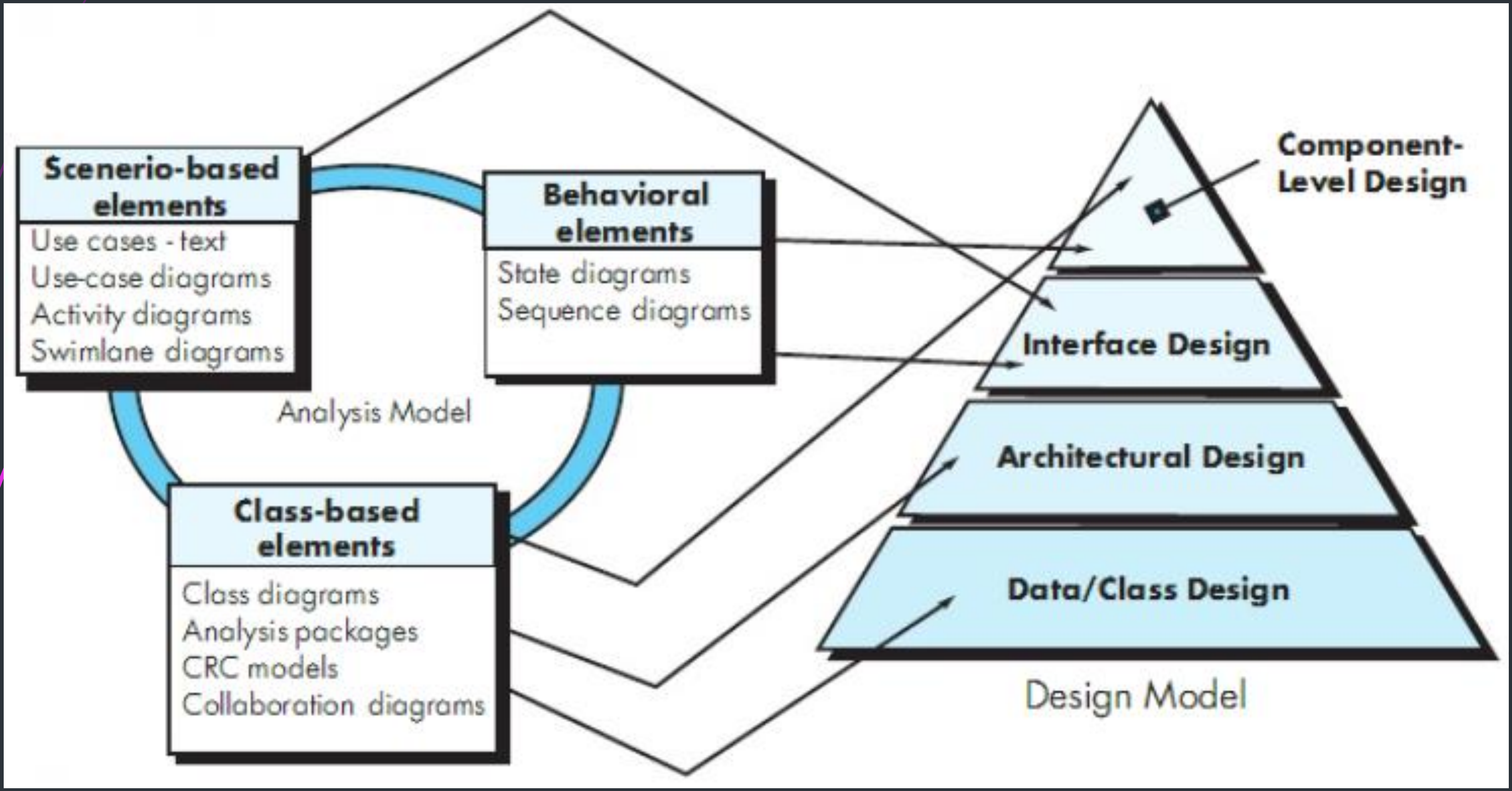
Chapter 7:

Design Modeling

VŨ THỊ TRÀ

CONTENTS

Translating from the requirement model into the design model



Guideline for the evaluation of a good design

- implement all of the explicit requirements.
- be a readable, understandable guide for developers and testers.
- provide a complete picture of the software, addressing the data, functional, and behavioral domains.

Quality Guidelines

1. exhibit an architecture.
2. be modular.
3. contain distinct representations of data, architecture, interfaces, and components.
4. lead to data structures that are appropriate for the classes.
5. lead to components that exhibit independent functional characteristics.
6. lead to interfaces that reduce the complexity of connections between components and with the external environments.
7. be derived using a repeatable method.
8. be represented using a notation that effectively communicates its meaning.

Quality Attributes

1. Functionality
2. Usability
3. Reliability
4. Performance
5. Supportability

Characteristics of Design Quality

- A mechanism for the translation of the requirements model into a design representation.
- A notation for representing functional components and their interfaces.
- Heuristics for refinement and partitioning.
- A guideline for quality assessment.

Generic Task Set for Design

1. Examine the information domain model and design appropriate data structures for data objects and their attributes.
2. Using the analysis model, select an architecture style (pattern) that is appropriate for the software.
3. Partition the analysis model into subsystems and allocate them within the architecture.
   - ✓ Be certain that each subsystem is functionally cohesive.
   - ✓ Design subsystem interfaces.
   - ✓ Allocate analysis classes or functions to each subsystems.

Generic Task Set for Design

4. Create a set of design classes or components:
   - ✓ Translate analysis class description into a design class.
   - ✓ Check each design class against design criteria; consider inheritance issues.
   - ✓ Evaluate or select design patterns for a design class or a subsystem.
   - ✓ Review design classes and revise as required.
5. Design any interface required with external systems or devices.

Generic Task Set for Design

6. Design the user interface:
    - ✓ Review results of task analysis.
    - ✓ Specify action sequence based on user scenarios.
    - ✓ Create behavior model of the interface.
    - ✓ Define interface objects and control mechanisms.
    - ✓ Review the interface design and revise as required.

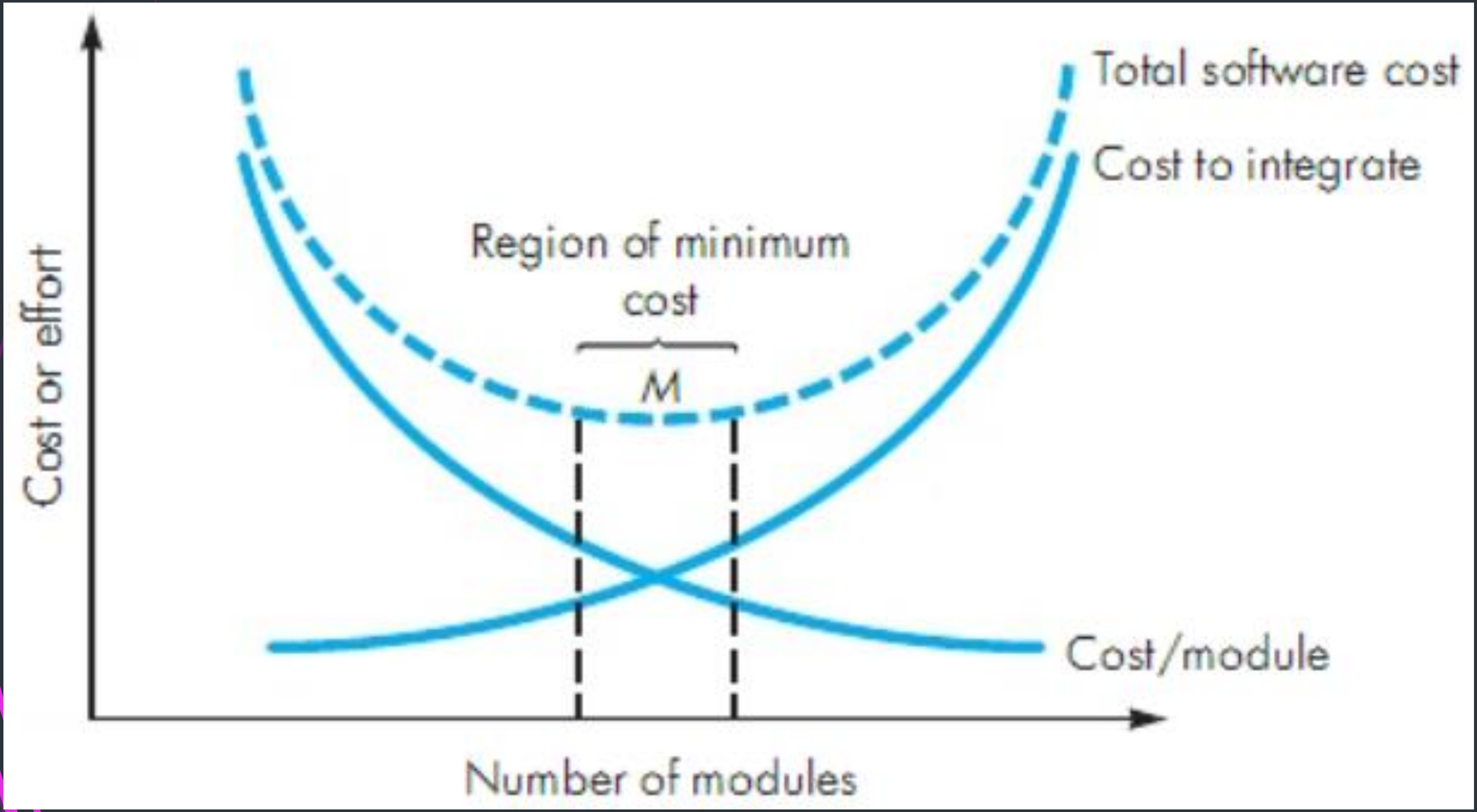
Generic Task Set for Design

7. Conduct component-level design:
   - ✓ Specify all algorithms at a relatively low level of abstraction.
   - ✓ Refine the interface of each component.
   - ✓ Define component-level data structures.
   - ✓ Review each component and correct all errors uncovered.
8. Develop a deployment model.

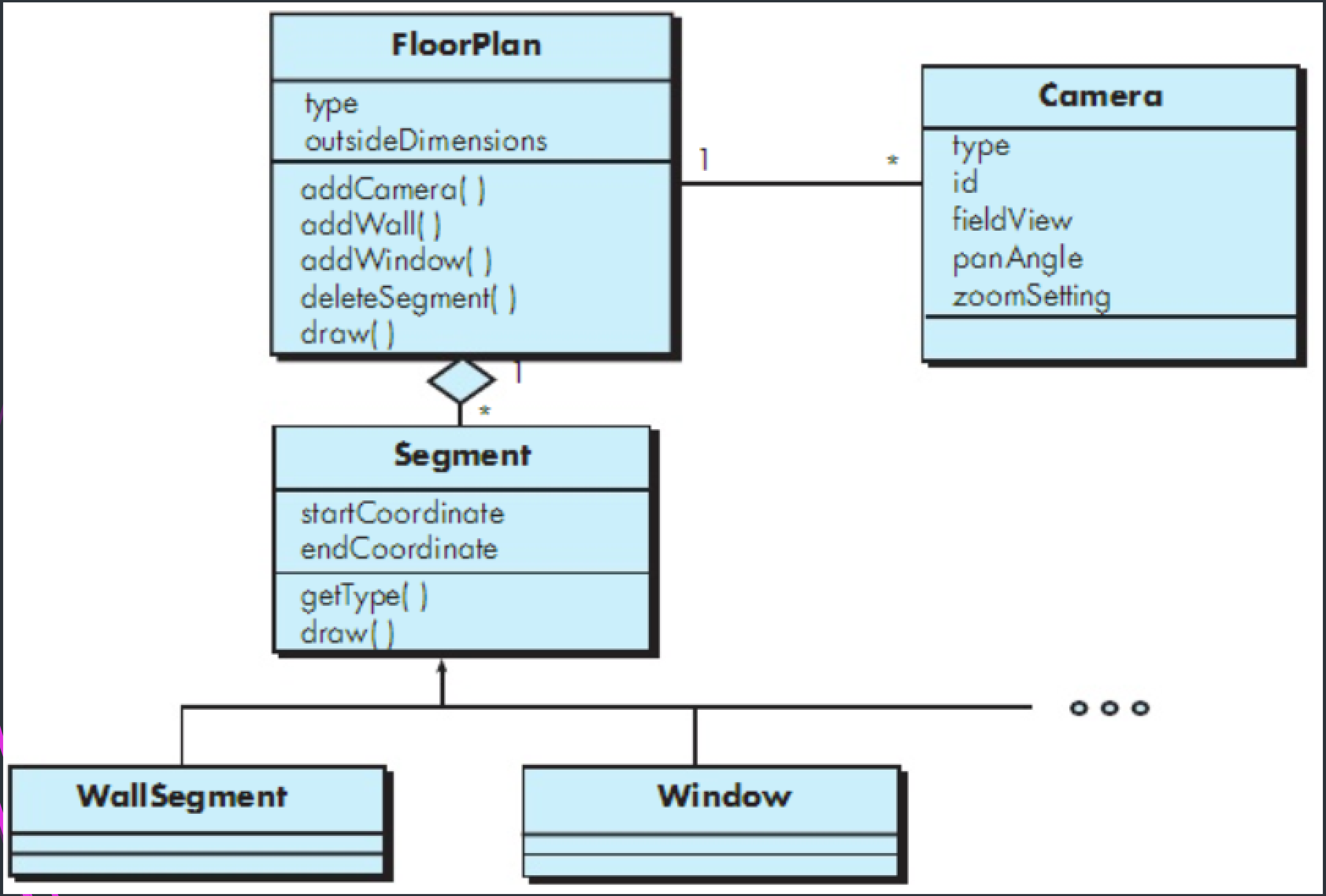
Design Concepts

1. Abstraction
2. Architecture
3. Patterns
4. Separation of concerns
5. Modularity
6. Information Hiding
7. Functional Independence
8. Refinement
9. Aspects
10. Refactoring
11. Object-oriented design concepts
12. Design classes
13. Dependency inversion
14. Design for test

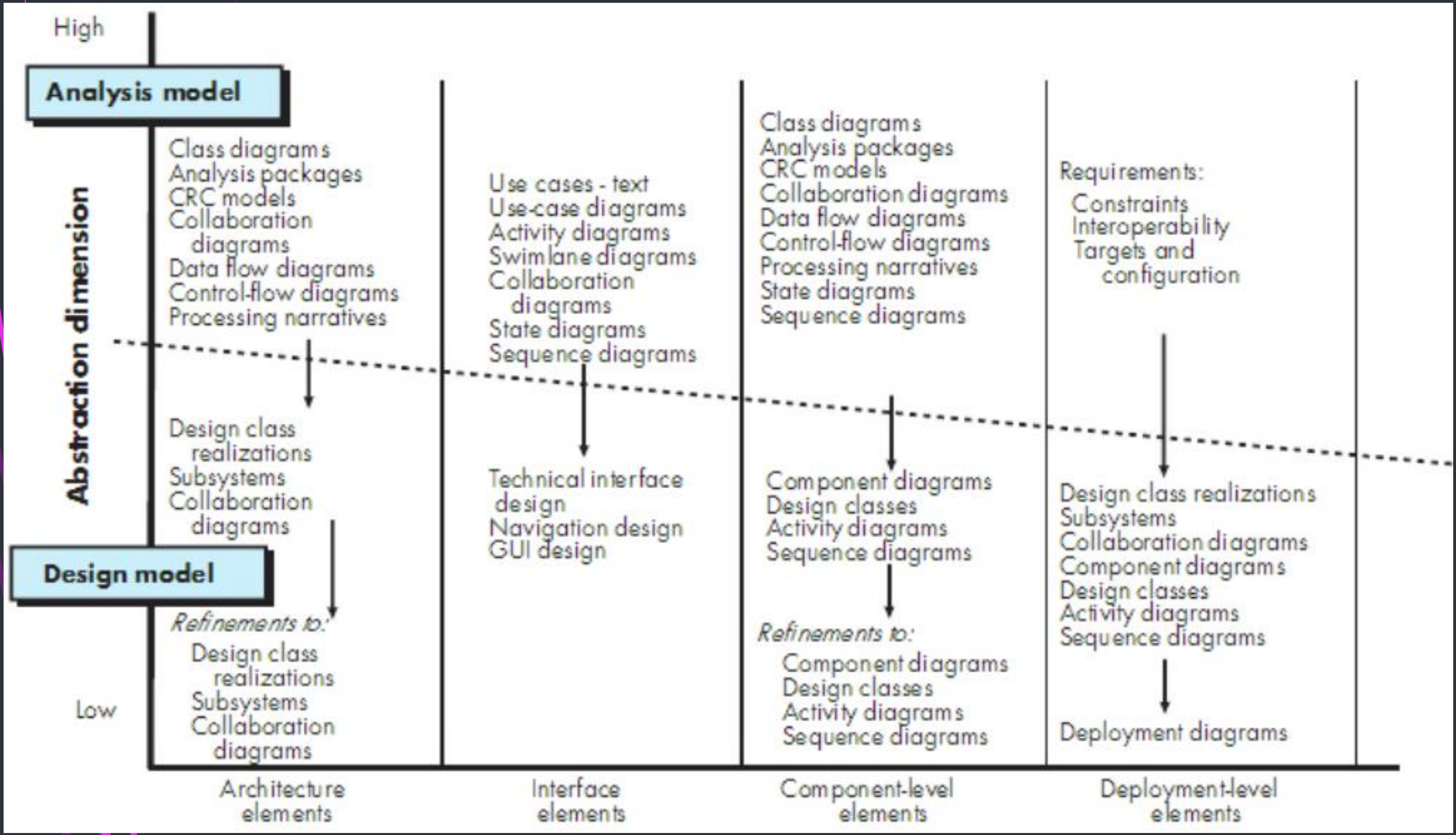
Modularity and software cost



Design class for FloorPlan



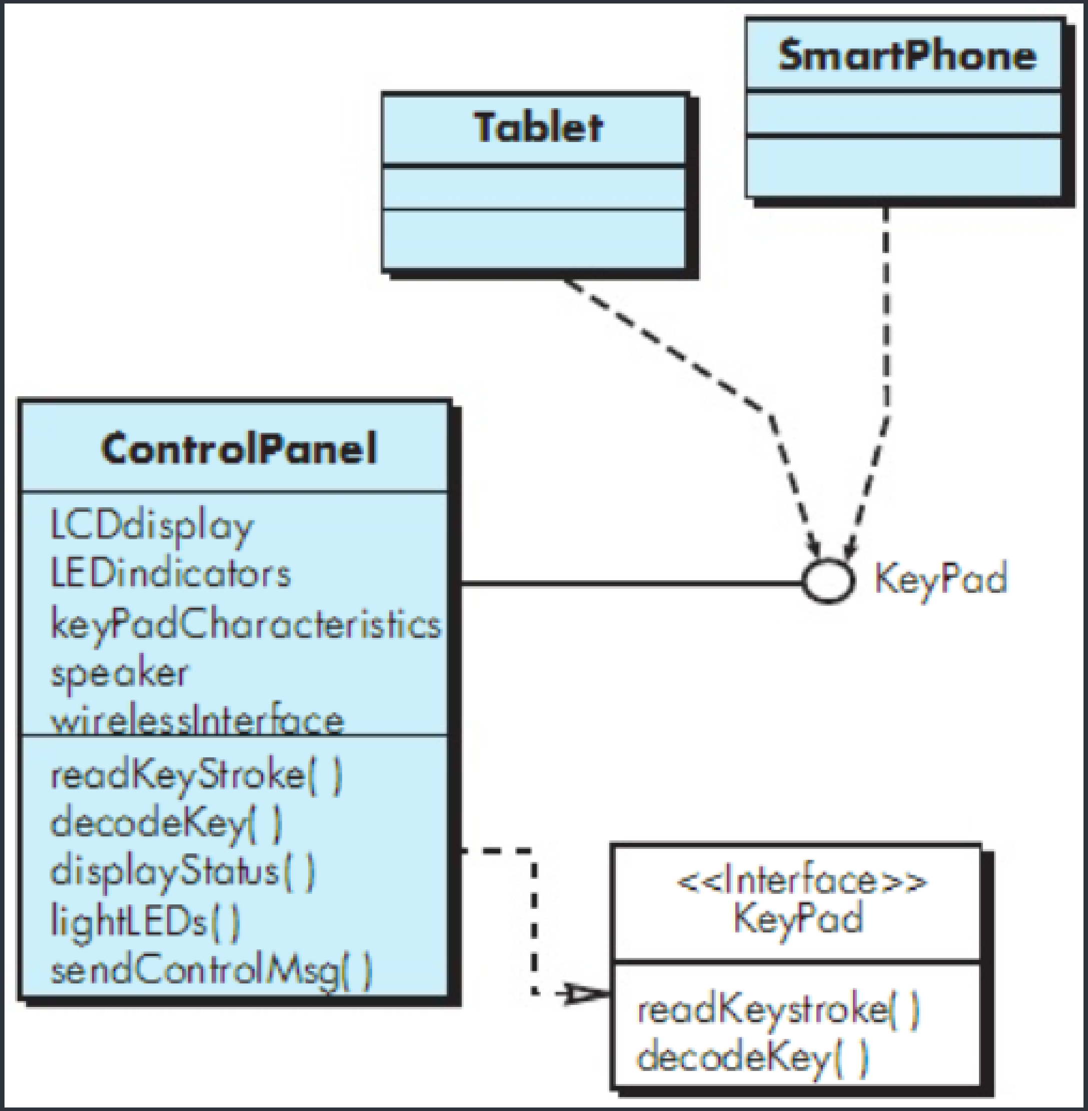
Dimensions of Design Model



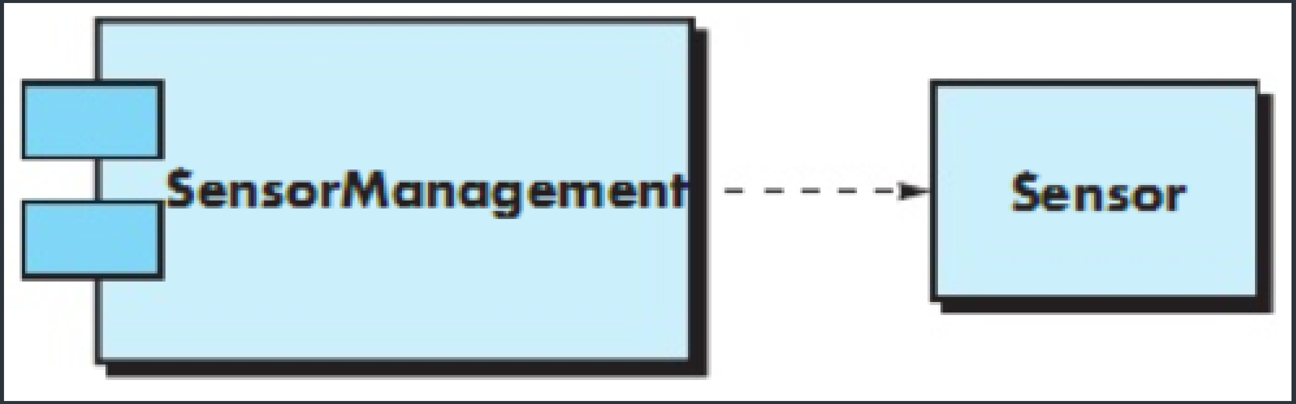
The Design Model

- Data design elements
- Architecture design elements
- Interface design elements
- Component-level design elements
- Deployment-level design elements

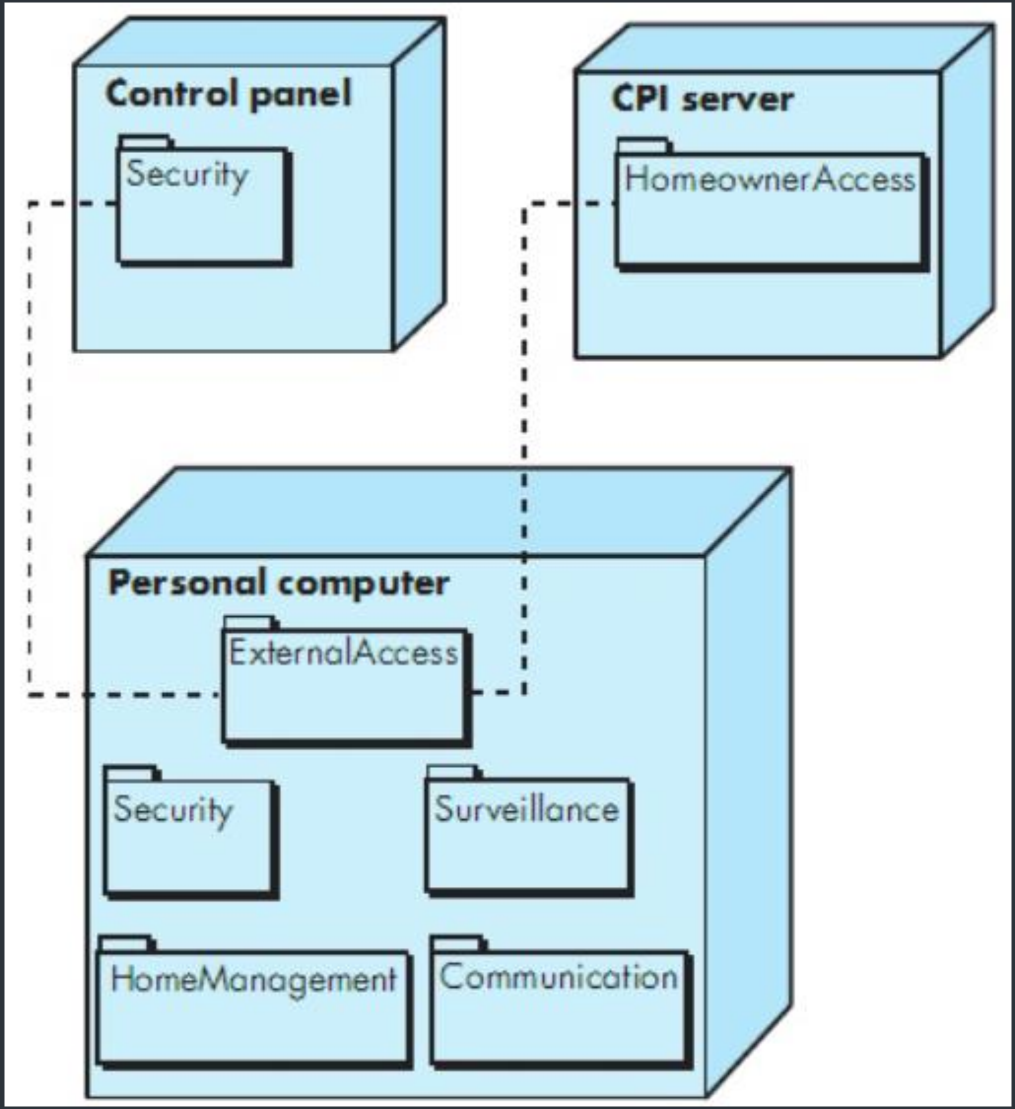
Interface representation for ControlPanel



A UML component diagram



A UML deployment diagram



CONTENTS

Distinguish between Architecture and Design

- Object – an instance of a class
- Design – an instance of an architecture
- Eg: I can design a network-centric software system in many different ways from the client-server architecture using either Java EE or .NET framework.
- Elements and structures – as part of architecture, the root of every design.

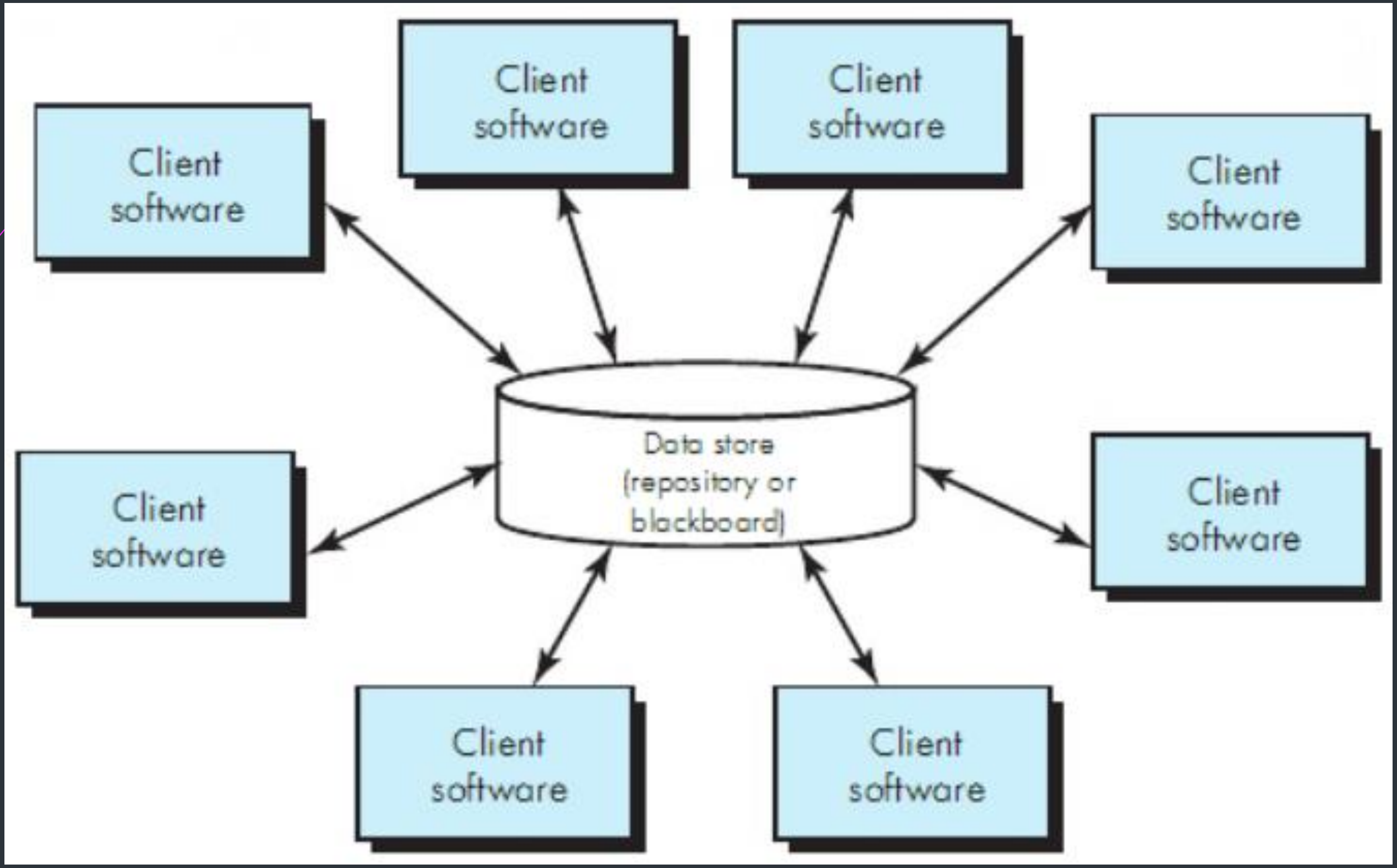
Architecture description

- A set of work that reflect different views of the system.
- Multiple metaphors:
  - ✓ Blueprint metaphor
  - ✓ Language metaphor
  - ✓ Decision metaphor
    - cost, usability, maintainability, performance
  - ✓ Literature metaphor

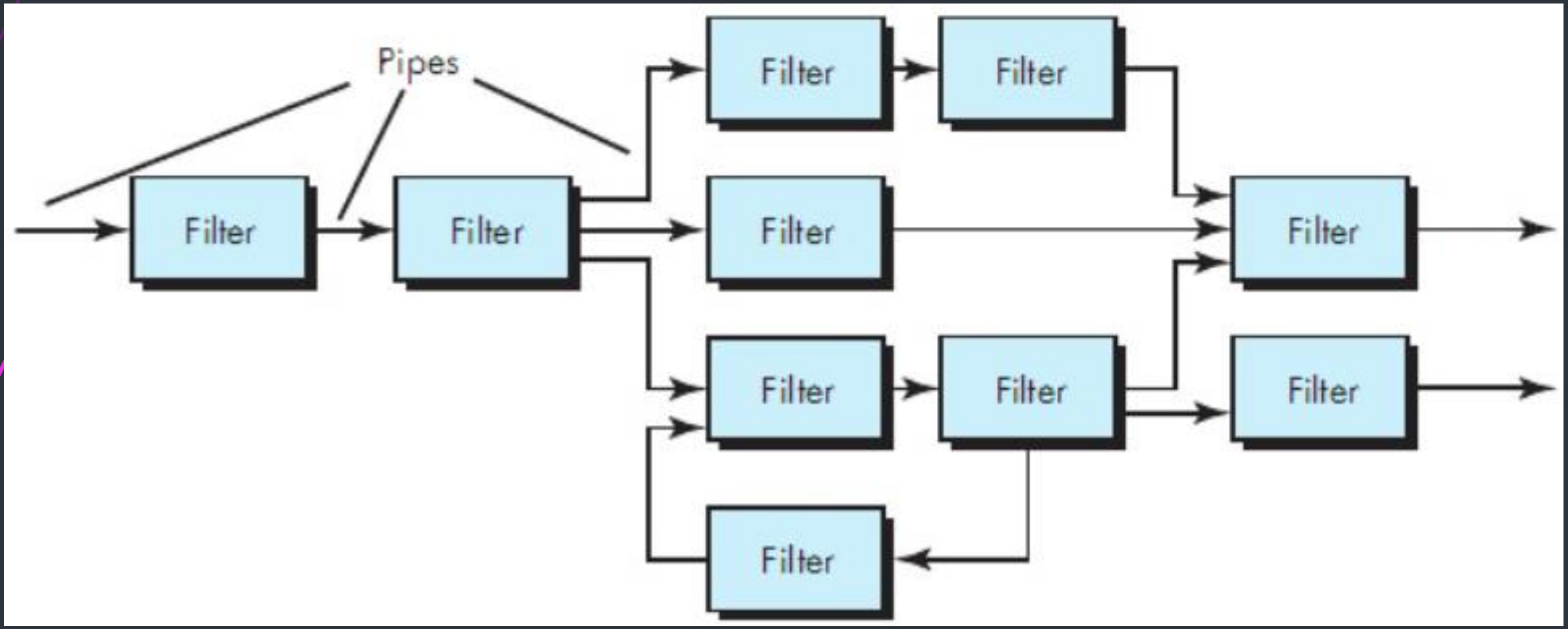
Taxonomy of Architecture Styles

- Data-centered architectures
- Data flow architectures
- Call and return architectures
  - ✓ Main program/ subprogram architectures
  - ✓ Remove procedure call architectures
- Object-oriented architectures
- Layered architectures

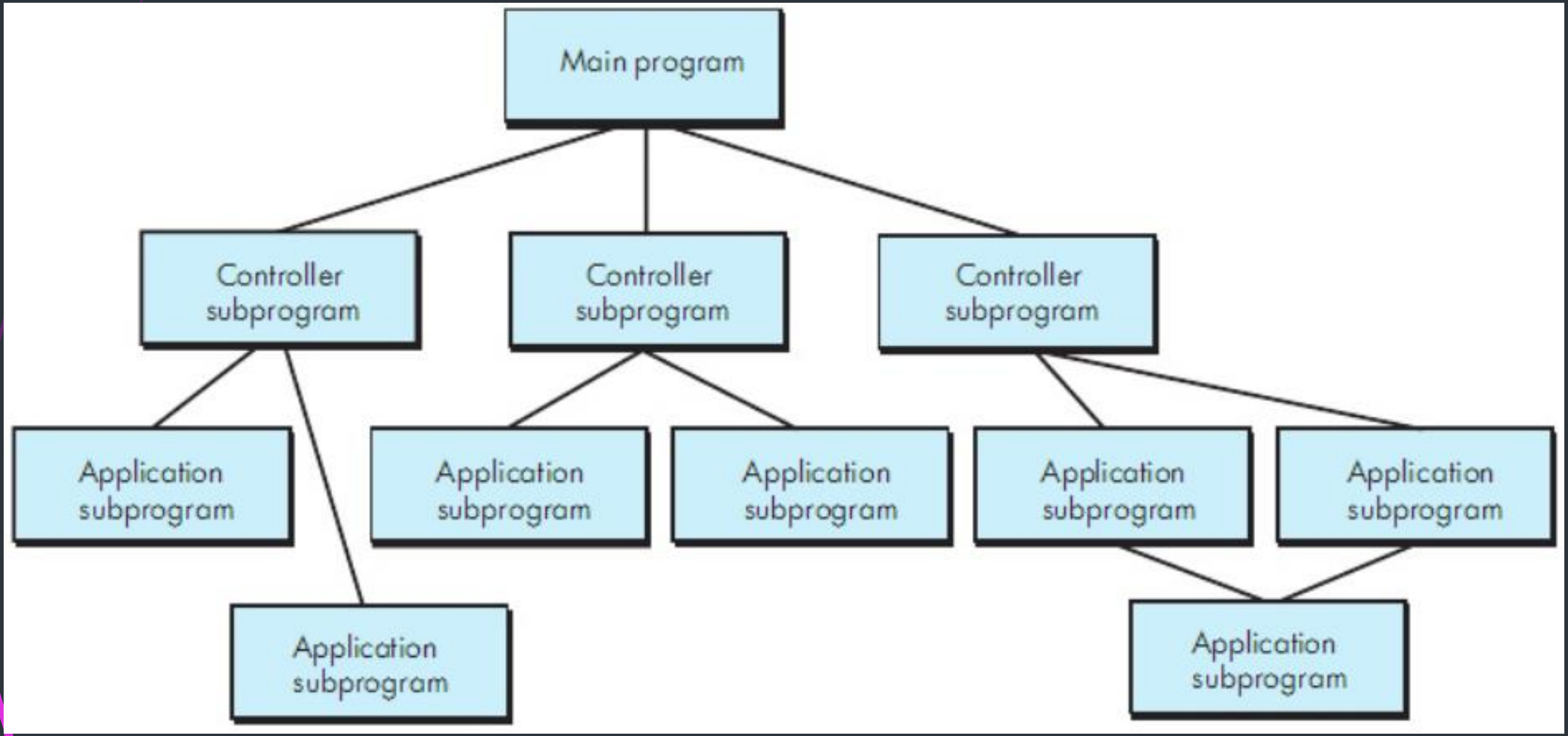
Data-centered architecture



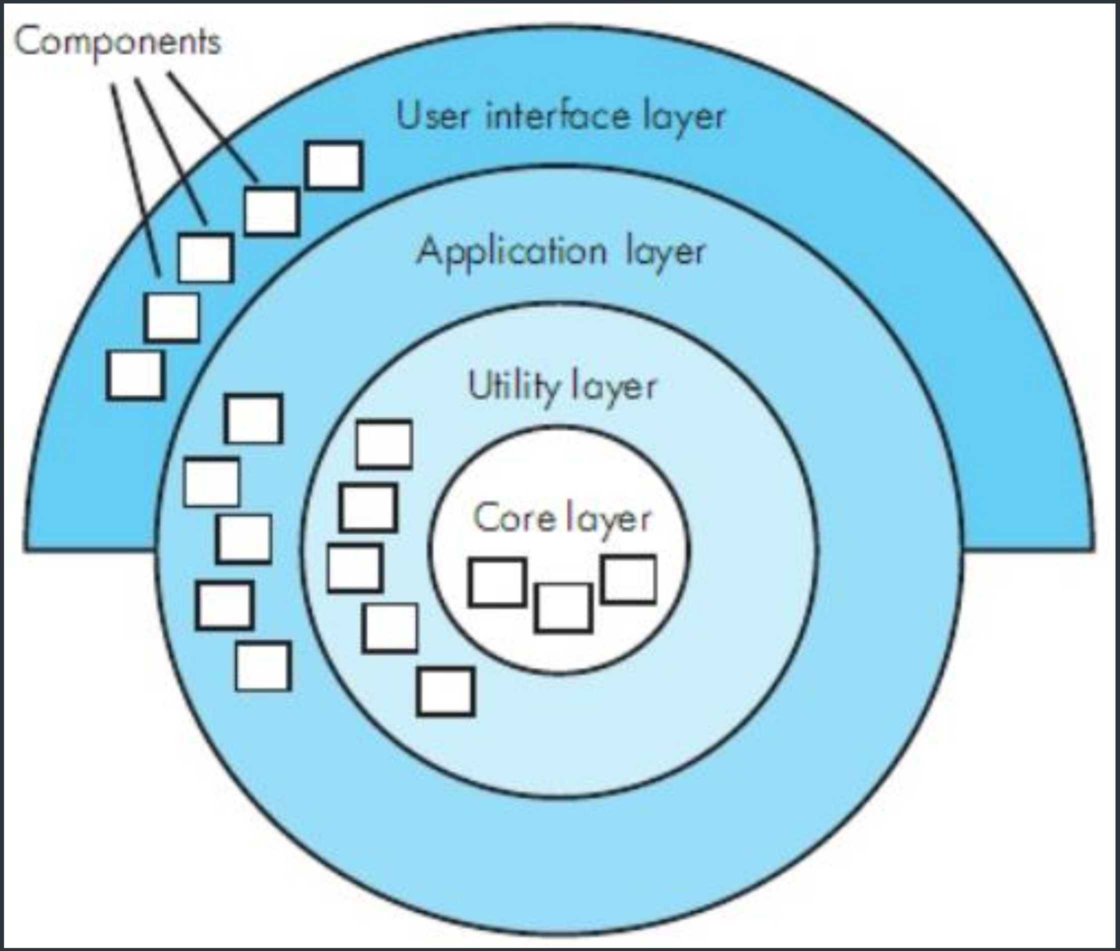
Data flow architecture



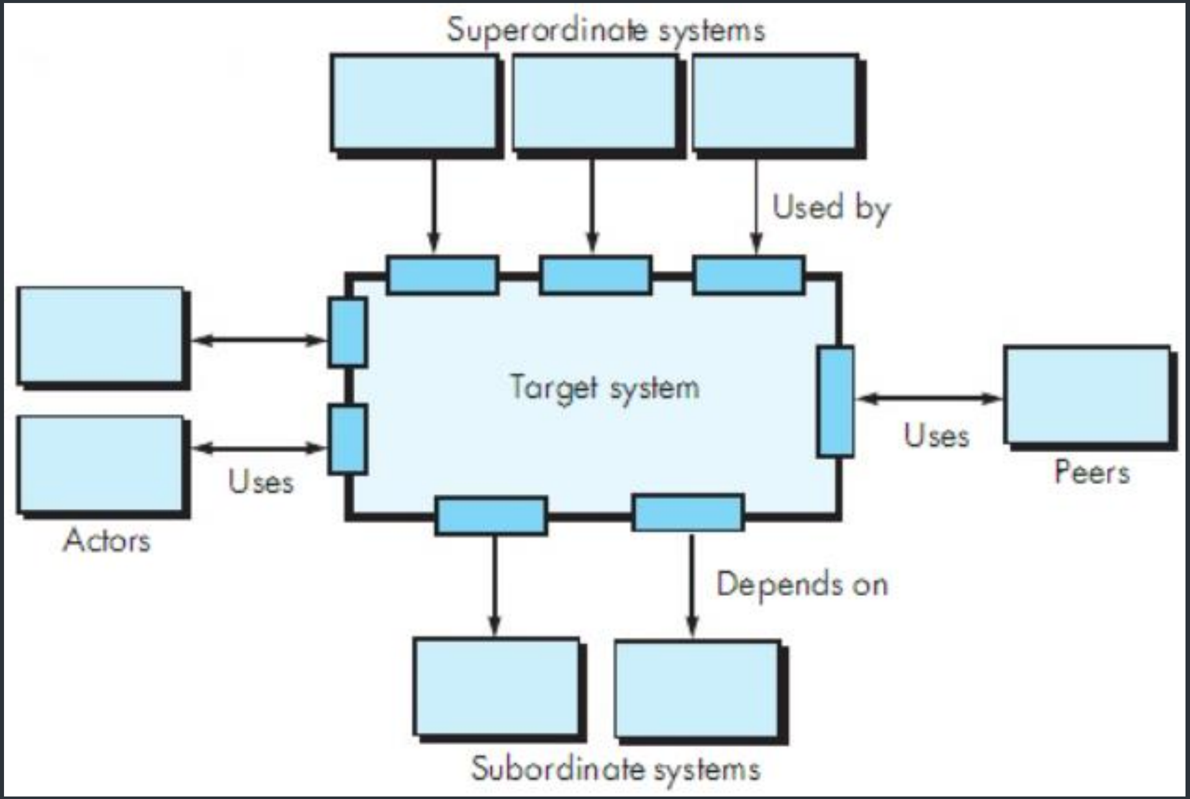
Main program/ subprogram architecture



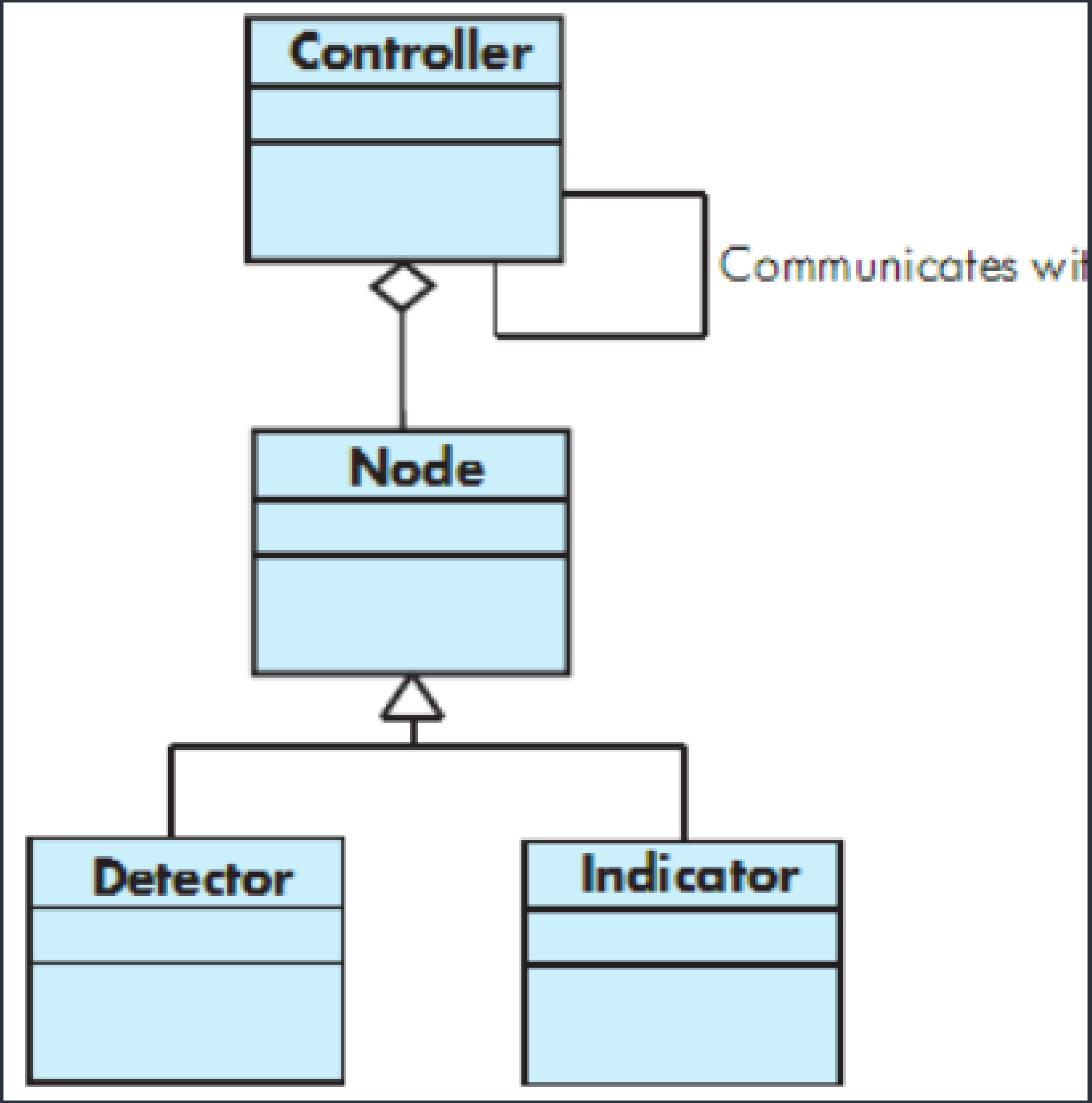
Layered Architecture



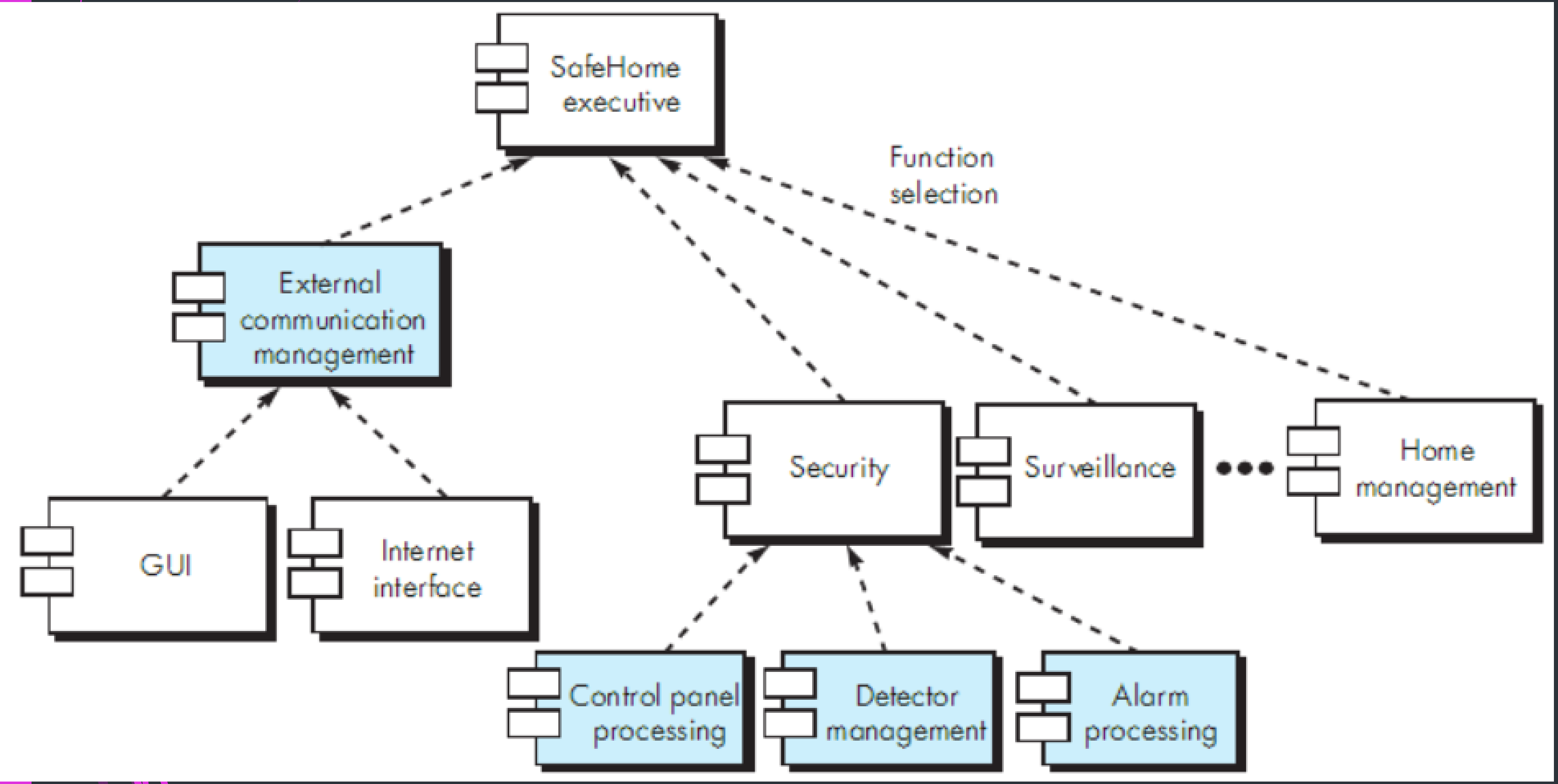
Architecture context diagram



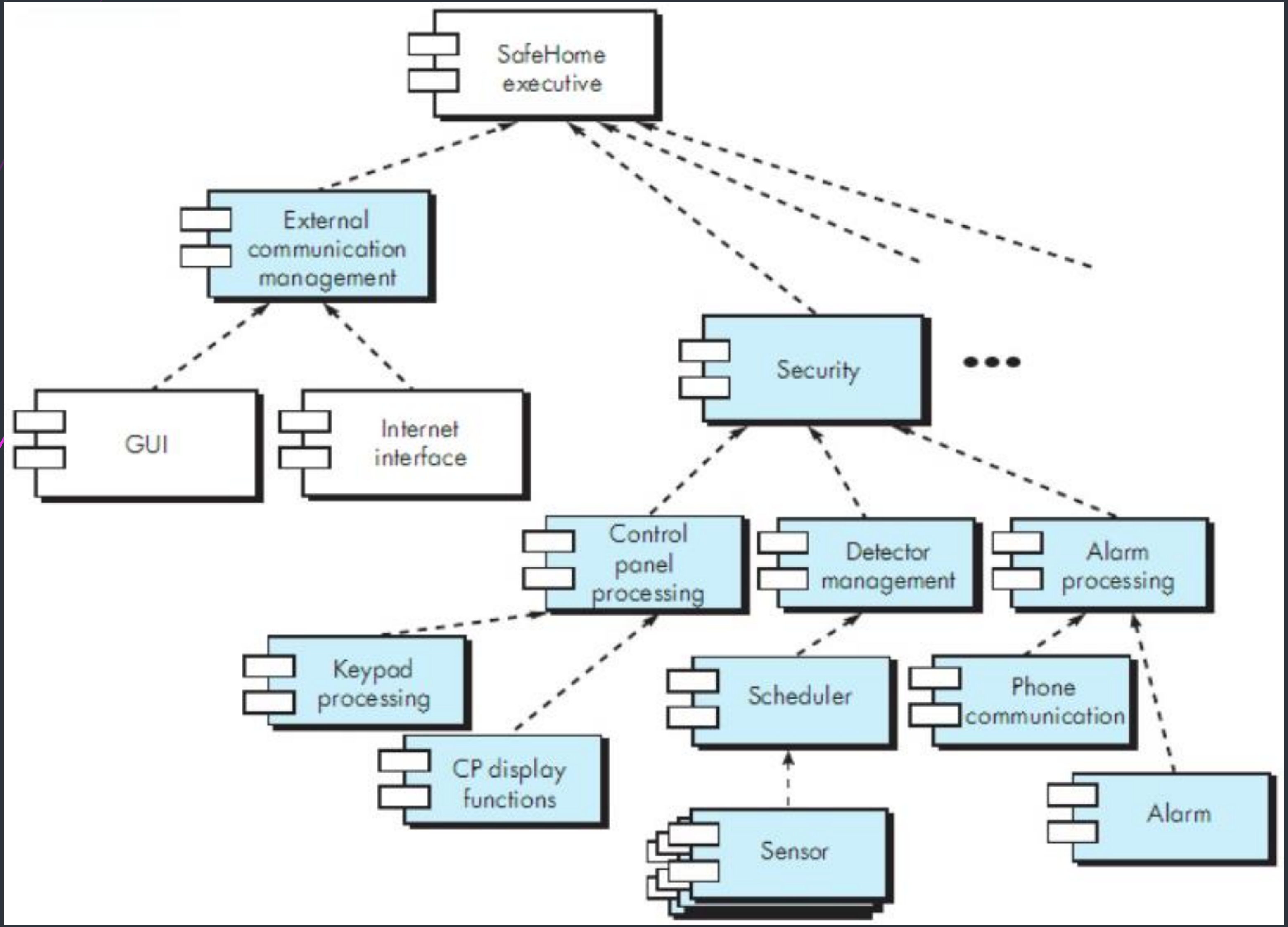
UML relationships for SafeHome security function archetypes



Overall architectural structure for SafeHome with top-level components

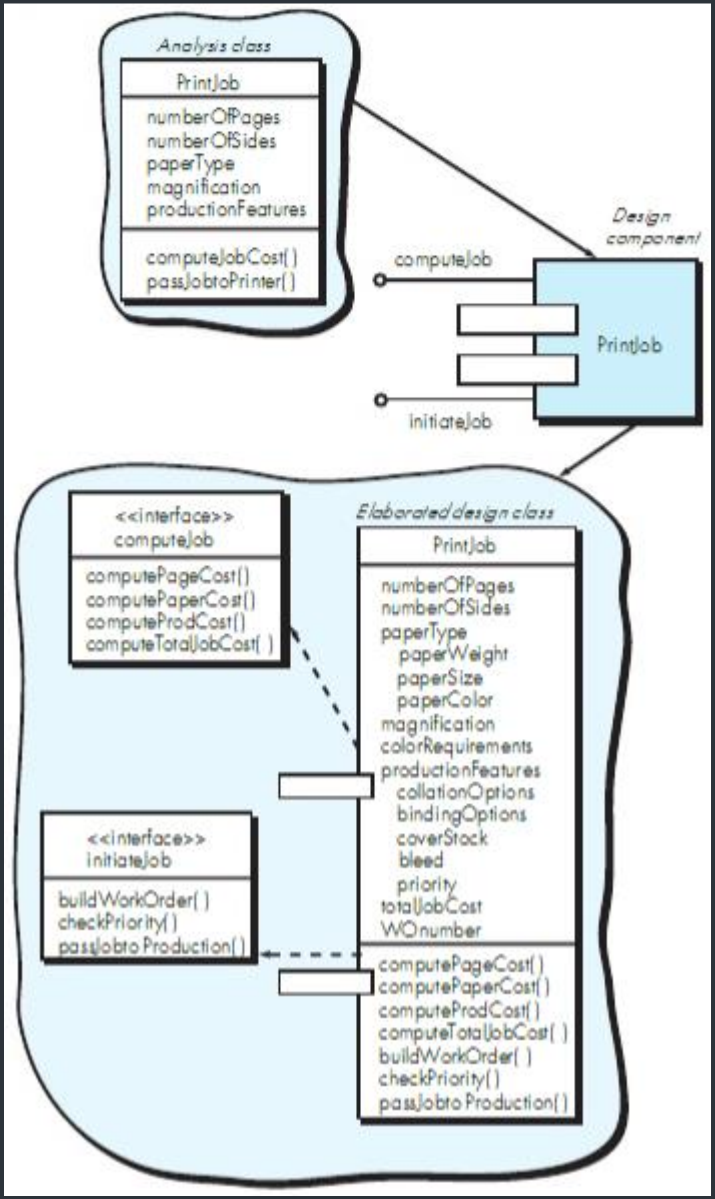


An instantiation of the security function with component elaboration

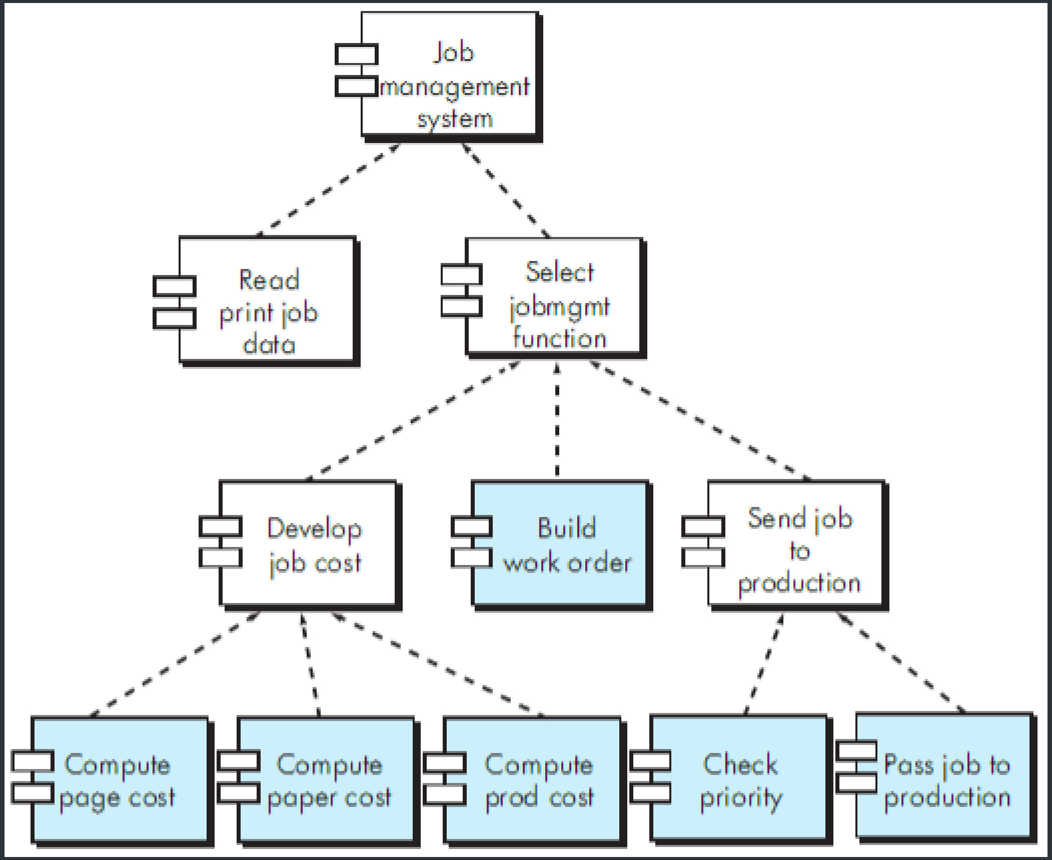


CONTENTS

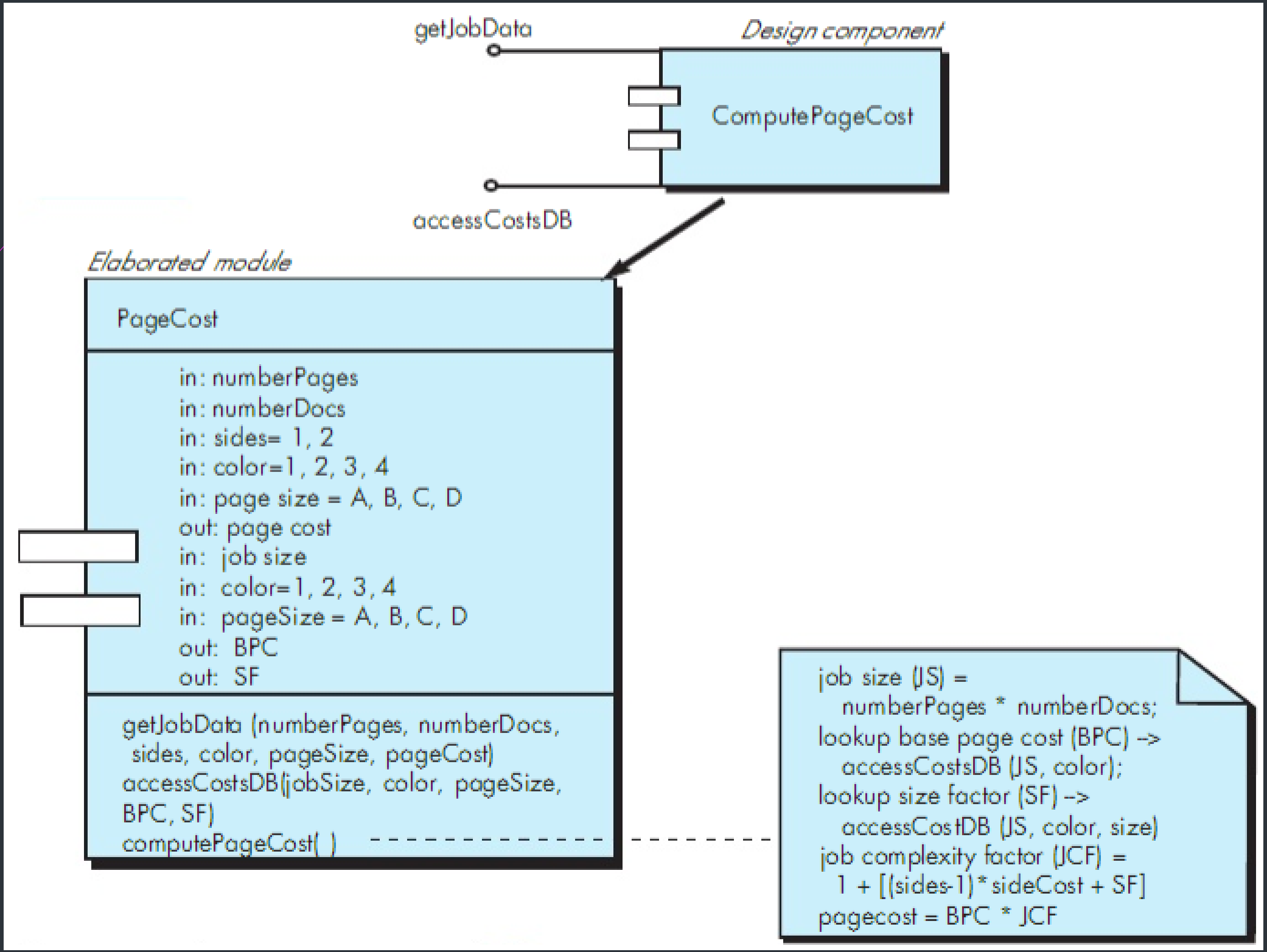
Elaboration of a design component



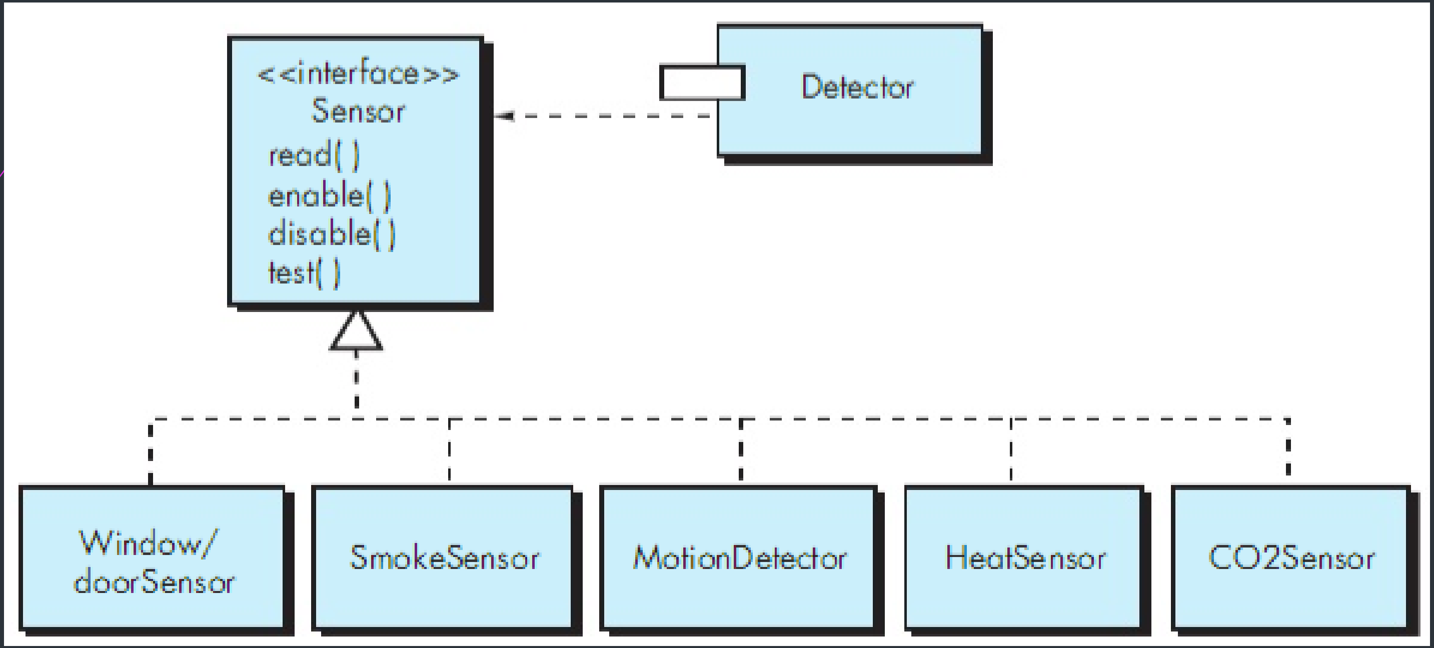
Structure chart for a traditional system



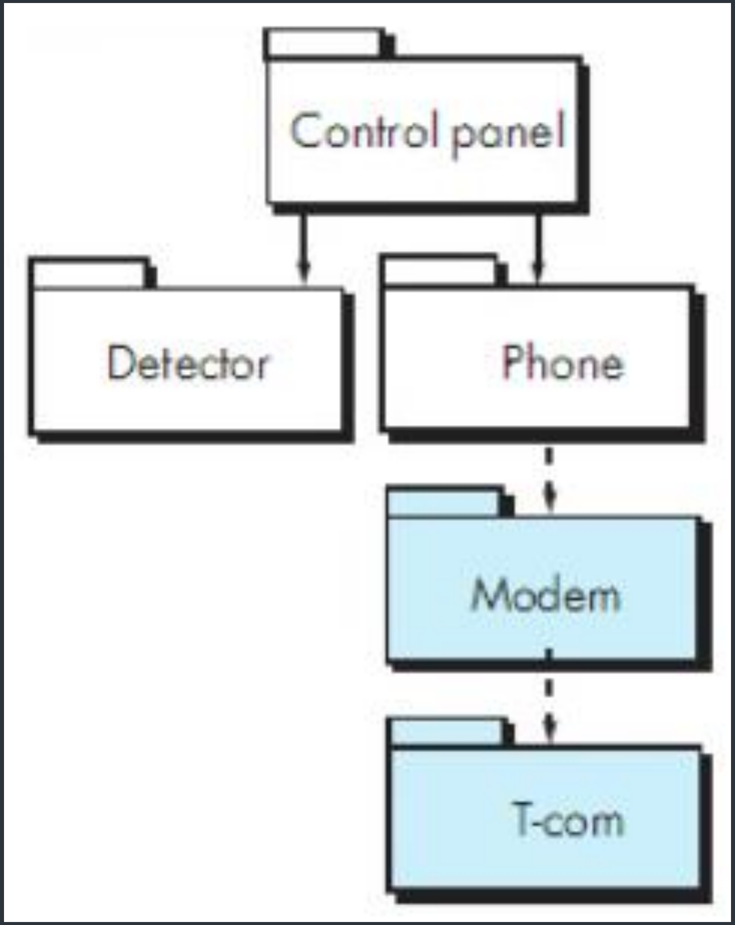
Component-level design for ComputePageCost


getJobData
Design component
ComputePageCost
accessCostsDB
Elaborated module
PageCost
in: numberPages
in: numberDocs
in: sides= 1, 2
in: color=1, 2, 3, 4
in: page size = A, B, C, D
out: page cost
in: job size
in: color=1, 2, 3, 4
in: pageSize = A, B, C, D
out: BPC
out: SF
getJobData (numberPages, numberDocs, sides, color, pageSize, pageCost)
accessCostsDB(jobSize, color, pageSize, BPC, SF)
computePageCost( )
job size (JS) = numberPages * numberDocs;
lookup base page cost (BPC) --> accessCostsDB (JS, color);
lookup size factor (SF) --> accessCostDB (JS, color, size)
job complexity factor (JCF) = 1 + [(sides-1)* sideCost + SF]
pagecost = BPC * JCF

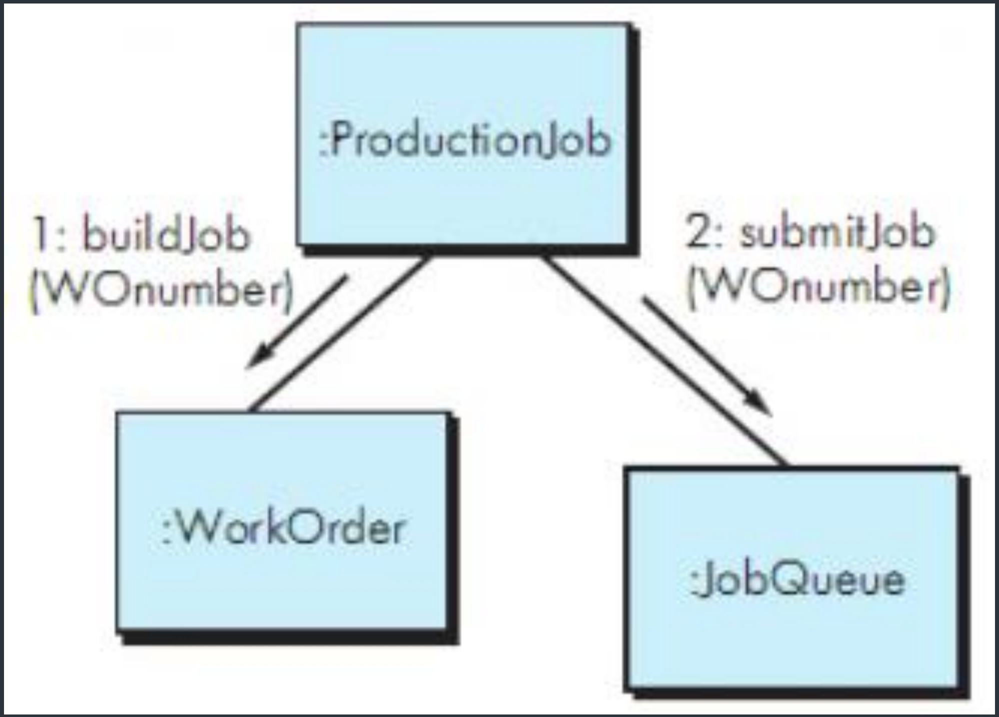
Following the OCP (Open-Closed Principle)



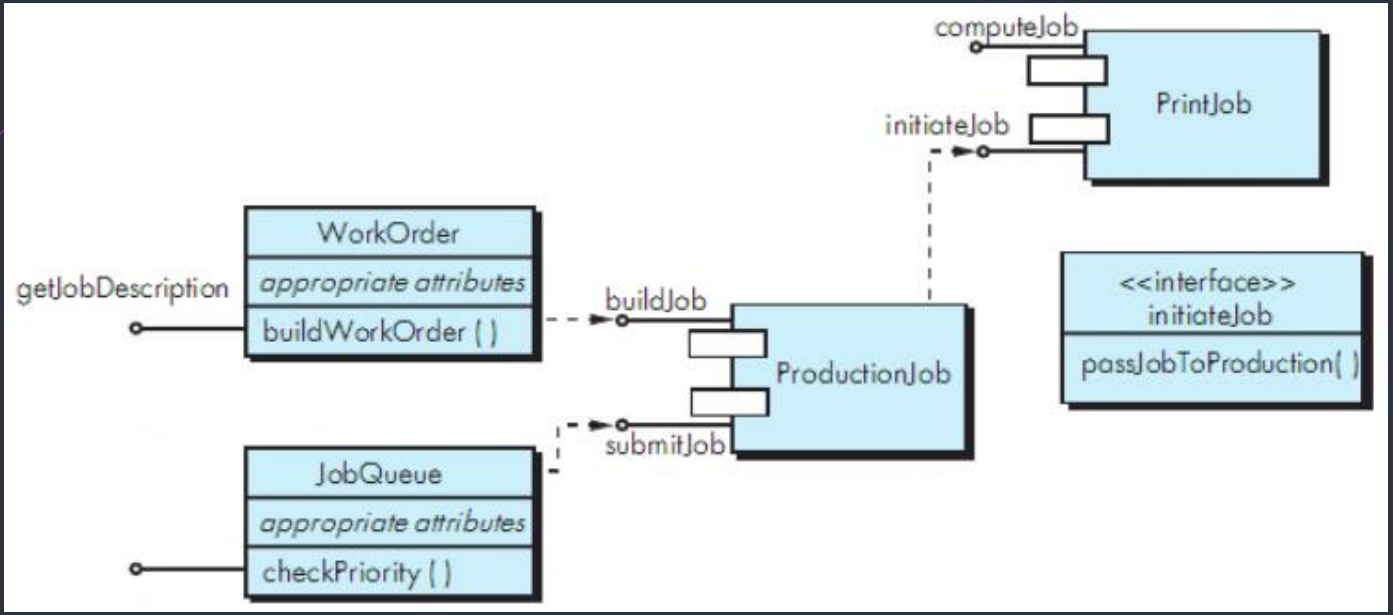
Layer cohesion



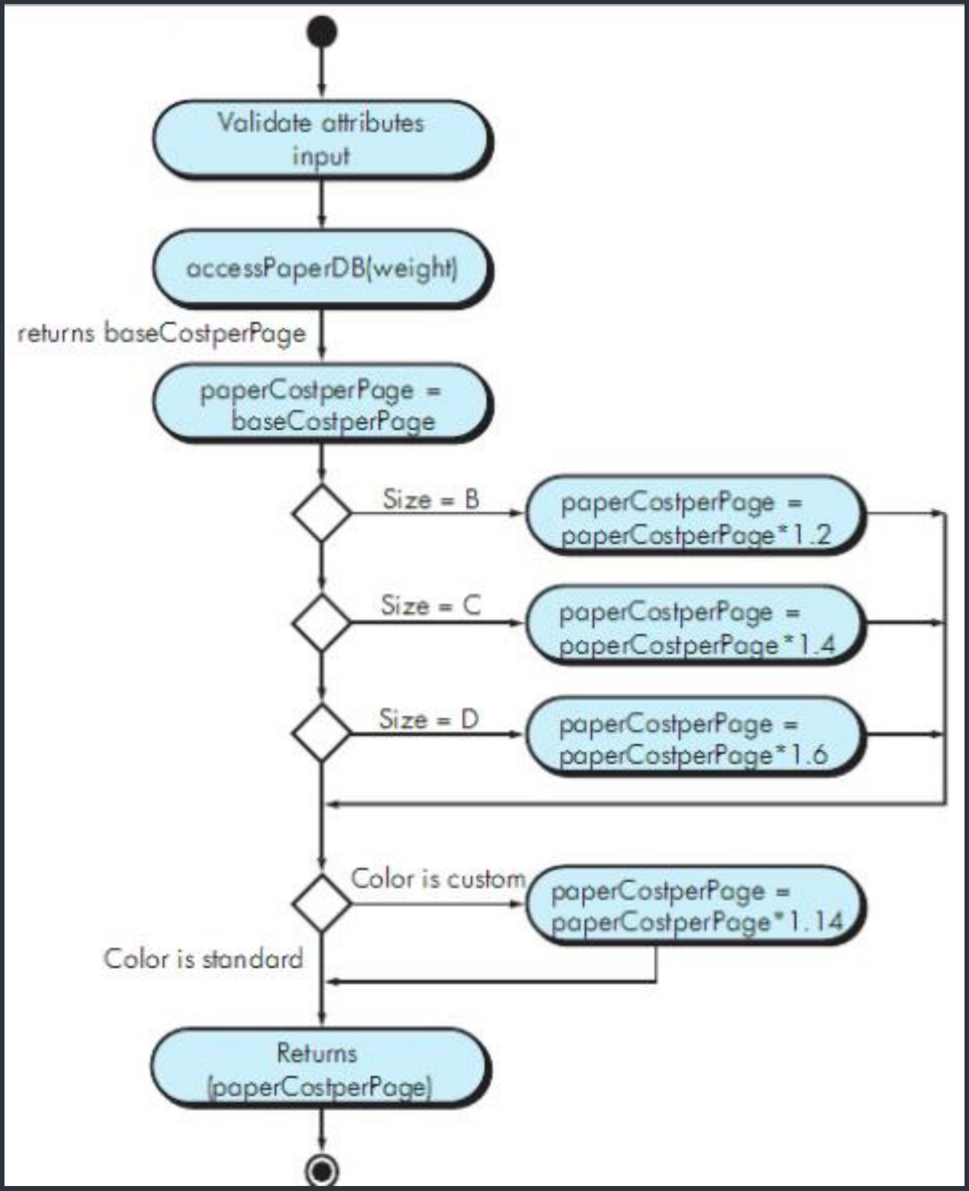
Collaboration diagram with messaging



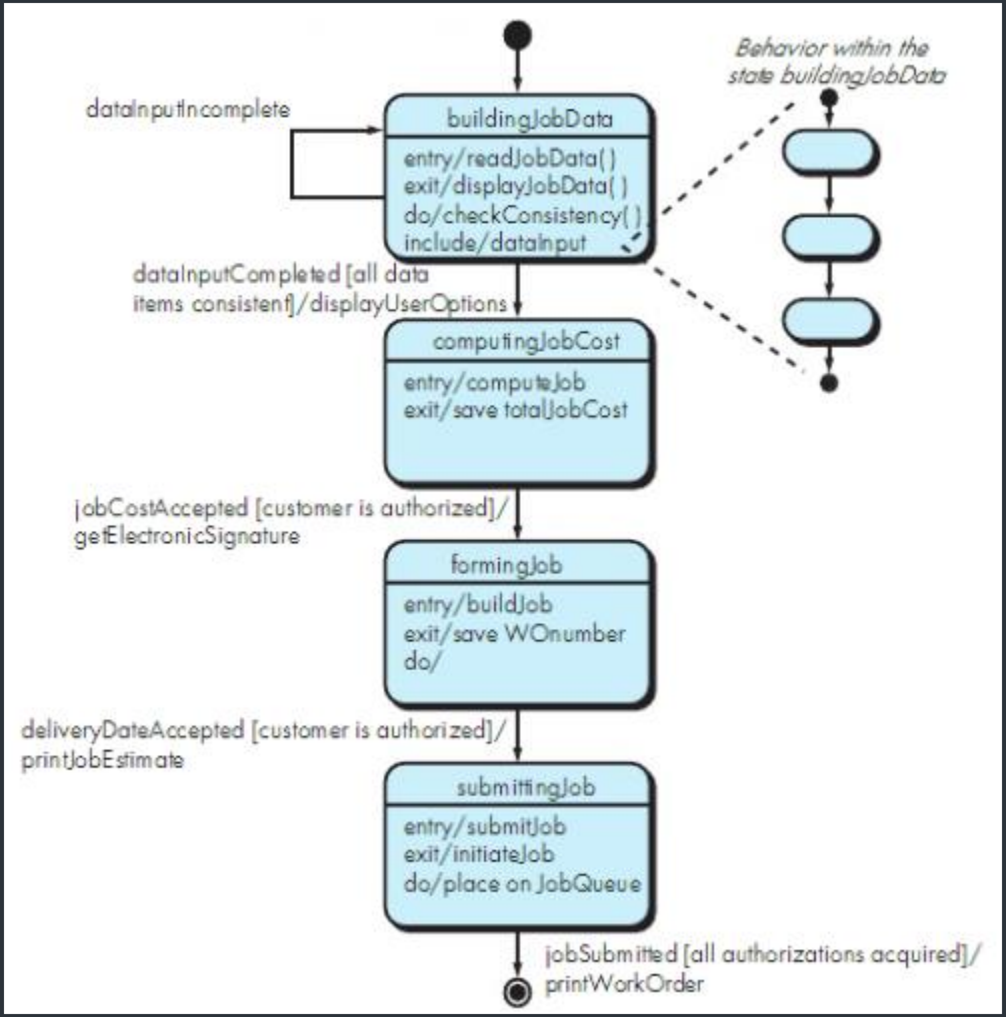
Refactoring interface and class definition for PrintJob



UML diagram for comput-PaperCost()


Validate attributes input
accessPaperDB(weight)
returns baseCostperPage
paperCostperPage = baseCostperPage
Size = B
paperCostperPage = paperCostperPage*1.2
Size = C
paperCostperPage = paperCostperPage*1.4
Size = D
paperCostperPage = paperCostperPage*1.6
Color is custom
paperCostperPage = paperCostperPage*1.14
Color is standard
Returns (paperCostperPage)

State chart fragment for PrintJob class

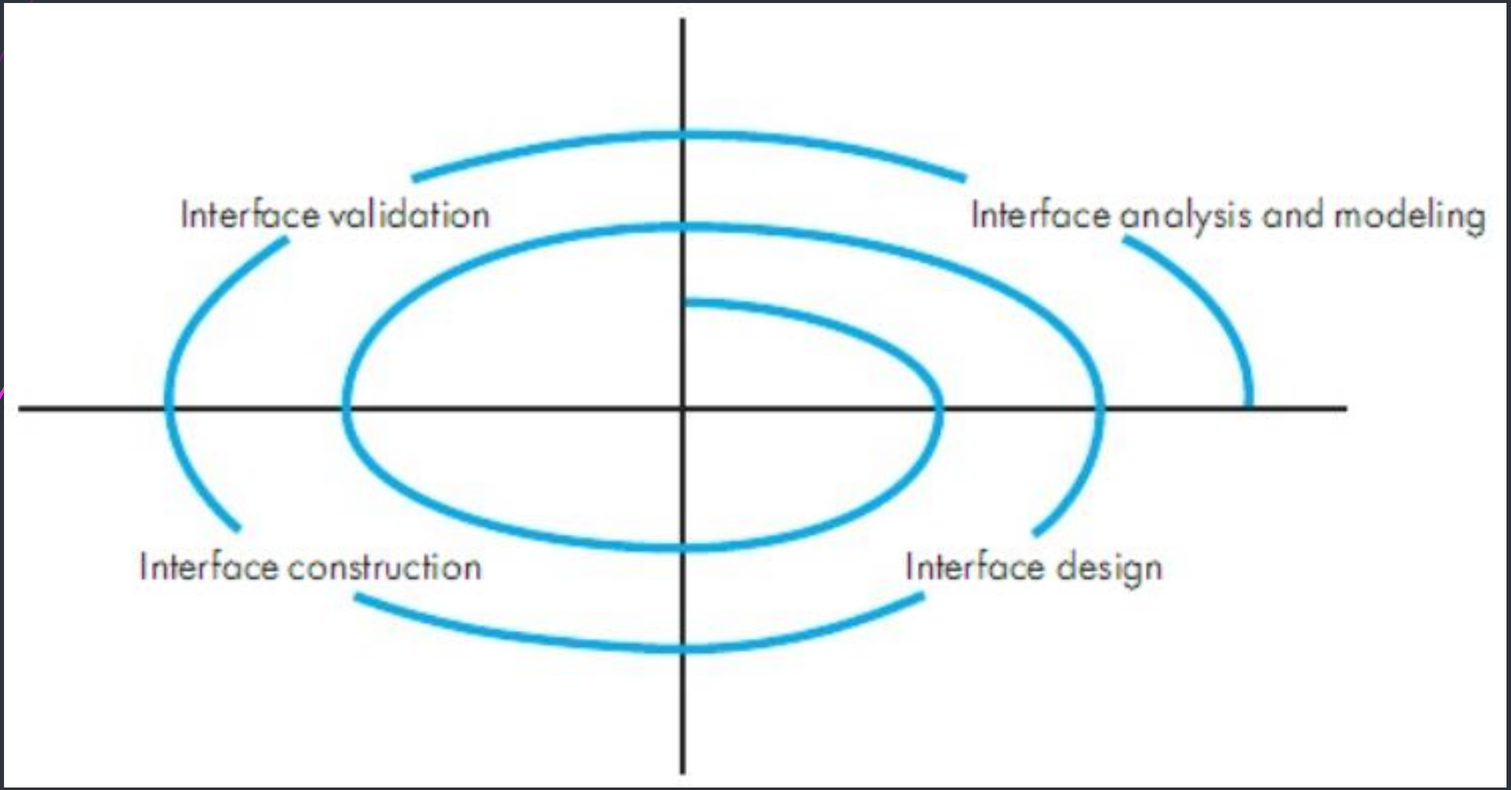

Behavior within the state buildingJobData
dataInputIncomplete
buildingJobData
entry/readJobData( )
exit/displayJobData( )
do/checkConsistency( )
include/dataInput
dataInputCompleted [all data items consistent]/displayUserOptions
computingJobCost
entry/computeJob
exit/save totalJobCost
jobCostAccepted [customer is authorized]/ getElectronicSignature
formingJob
entry/buildJob
exit/save WOnumber
do/
deliveryDateAccepted [customer is authorized]/ printJobEstimate
submittingJob
entry/submitJob
exit/initiateJob
do/place on JobQueue
jobSubmitted [all authorizations acquired]/ printWorkOrder

CONTENTS

Three Golden Rules

1. Place the user in control
2. Reduce the user's memory load
3. Make the interface consistent

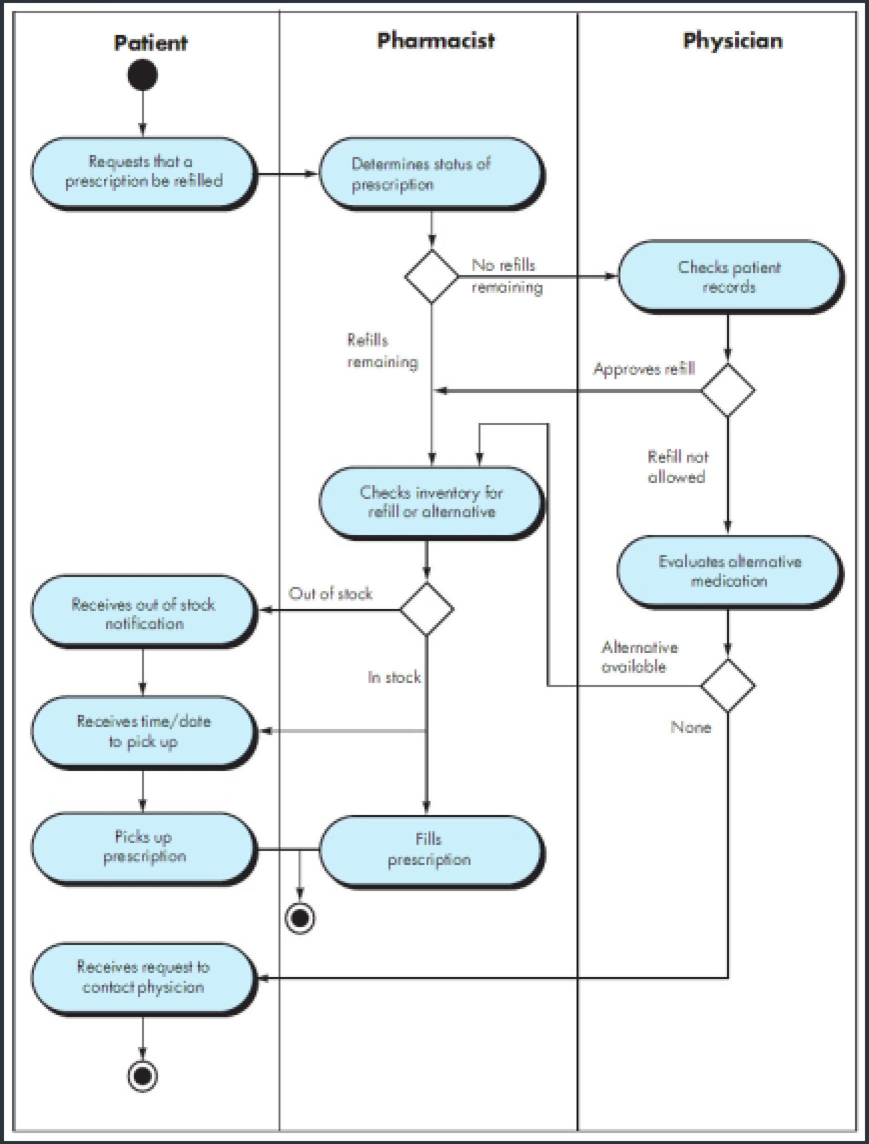
User Interface Design Process



Interface Analysis and Modeling

- User Case
- Task Elaboration
- Object Elaboration
- Workflow Analysis
- Hierarchical Representation

Swimlance diagram for prescription refill function


Patient
Pharmacist
Physician
Requests that a prescription be refilled
Determines status of prescription
No refills remaining
Checks patient records
Refills remaining
Approves refill
Refill not allowed
Checks inventory for refill or alternative
Evaluates alternative medication
Out of stock
Receives out of stock notification
Alternative available
In stock
None
Receives time/date to pick up
Picks up prescription
Fills prescription
Receives request to contact physician

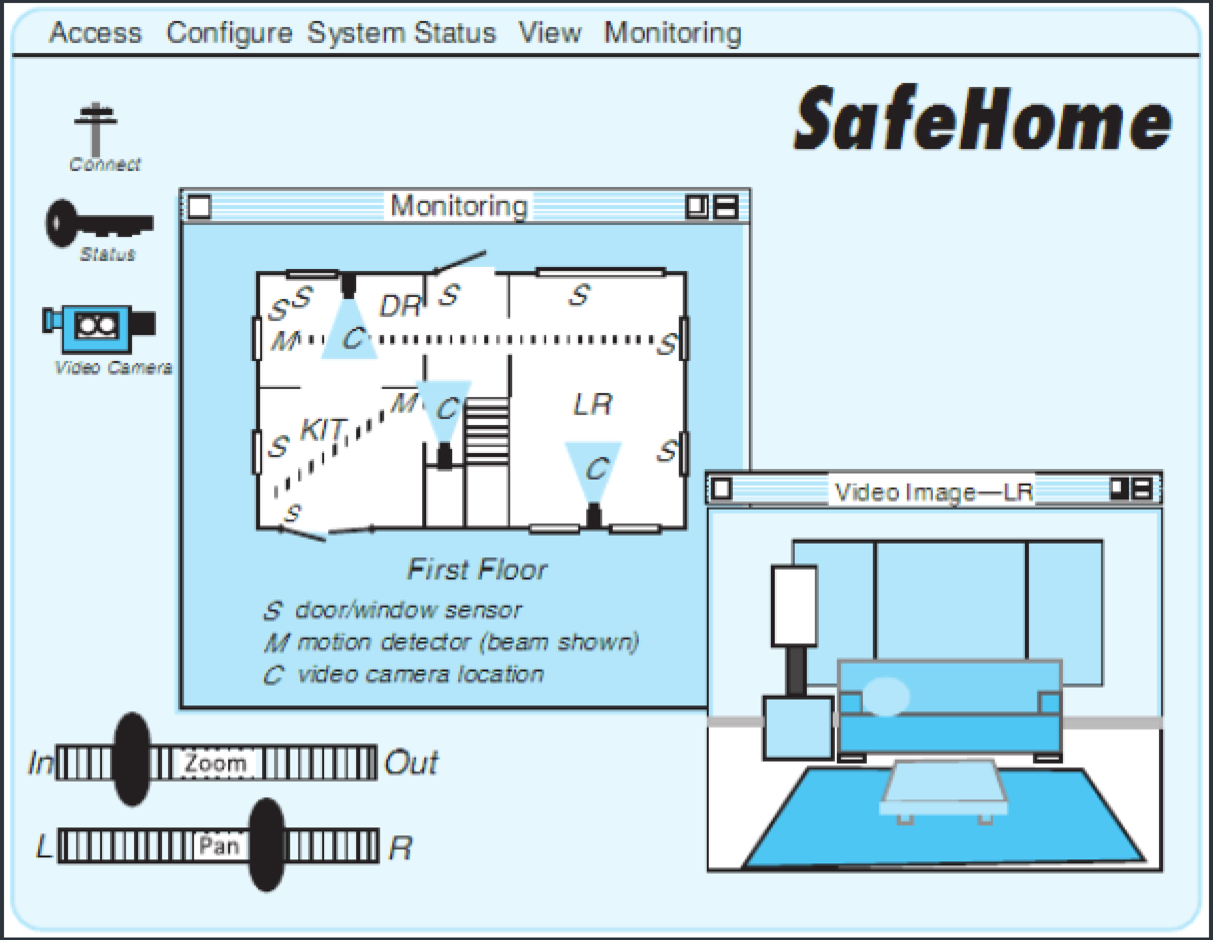
Interface Design Steps

1. Defined interface objects and actions (operations).
2. Identify events (user actions) that will cause the state of user interface to change.
3. Depict representation of each state.
4. Indicate how the user interprets each state from information provided through the interface.

Homeowner Task List

- *Accesses* the SafeHome system
- *Enters* an ID and password to allow remote access
- *Checks* system status
- *Arms* or *disarms* SafeHome system
- *Displays* floor plan and sensor locations
- *Displays* zones on floor plan
- *Changes* zones on floor plan
- *Displays* video camera locations on floor plan
- *Selects* video camera for viewing
- *Views* video images (four frames per second)
- *Pans* or *zooms* the video camera

Preliminary screen layout

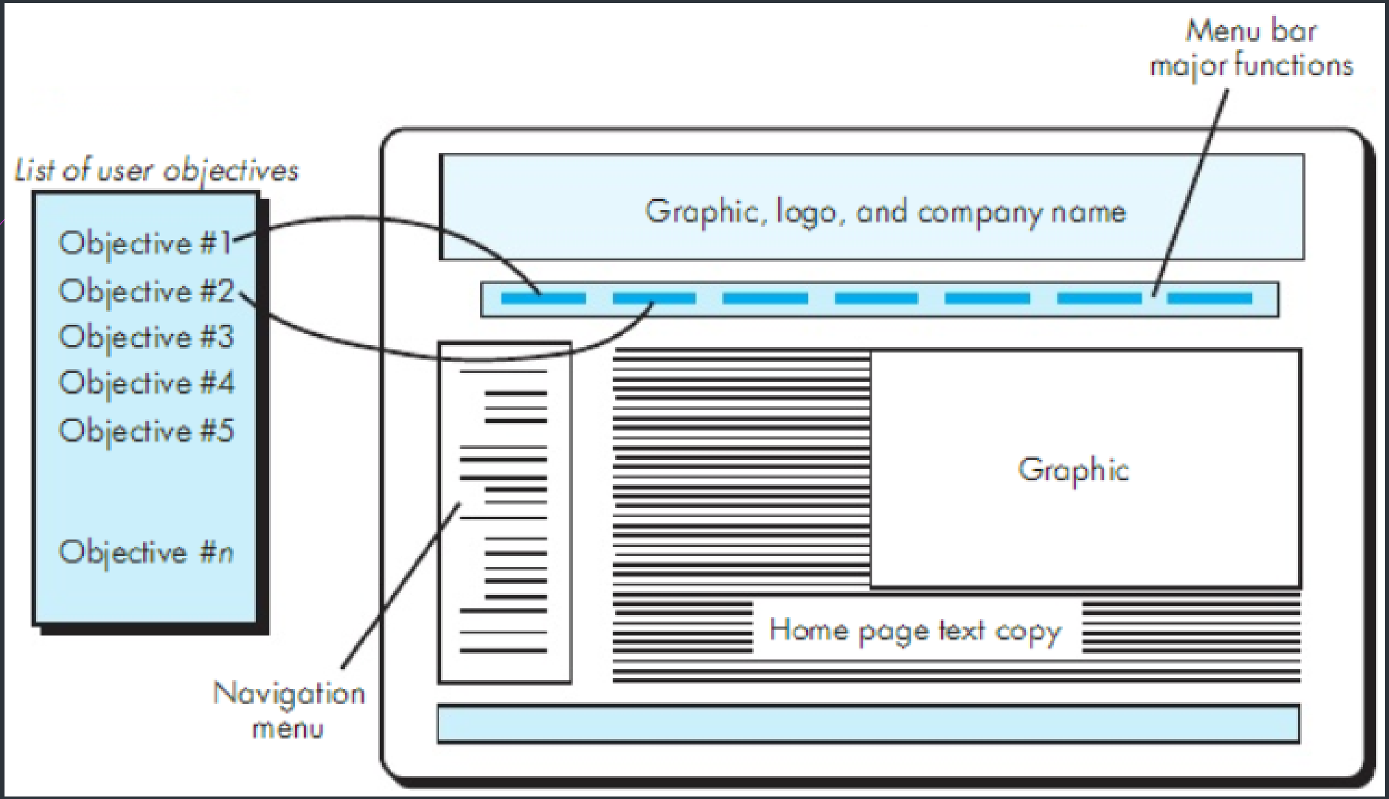


Design Issues

- Response Time
- Help Facilitates
- Error Handling
- Menu and Command Labeling
- Application Accessibility
- Internationalization

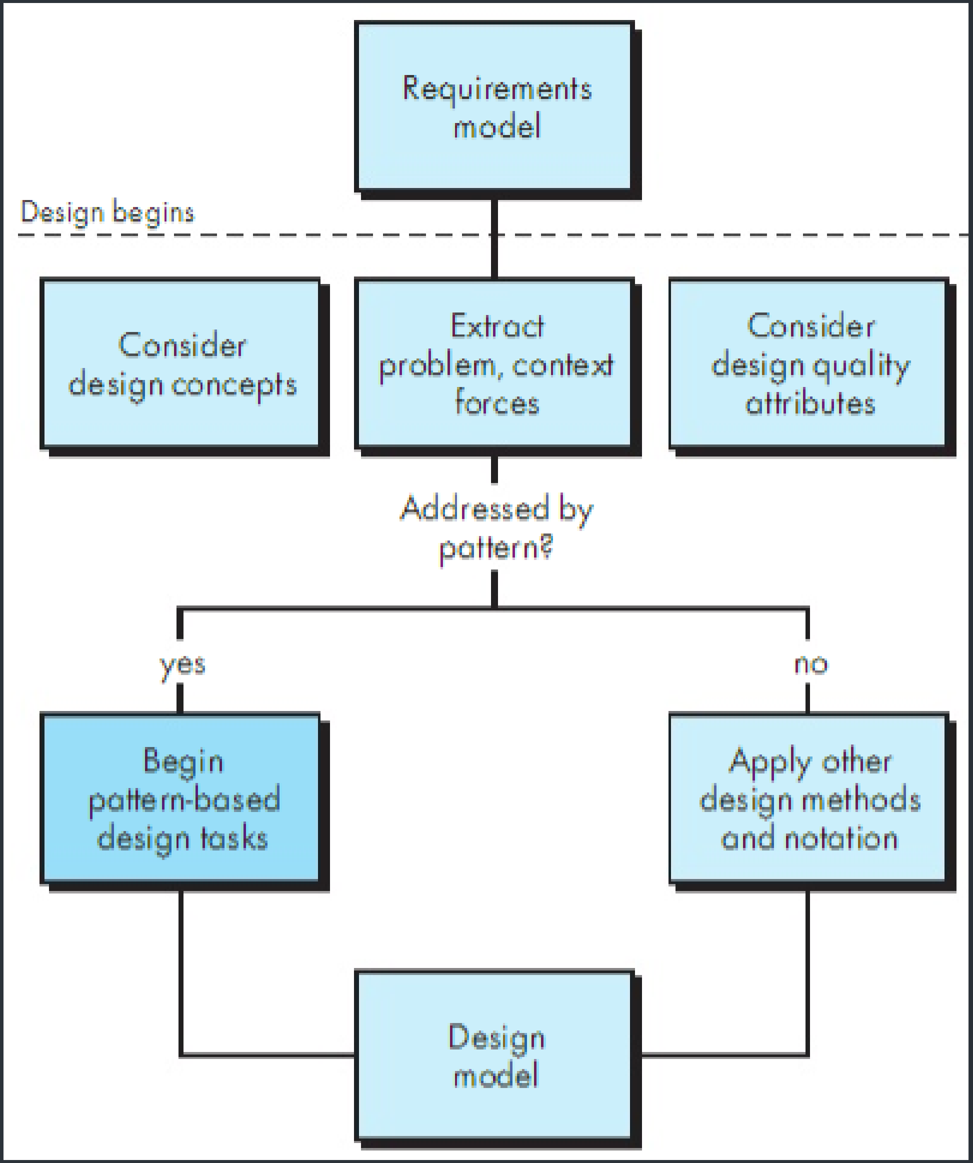
Interface Validation

The interface design evaluation cycle

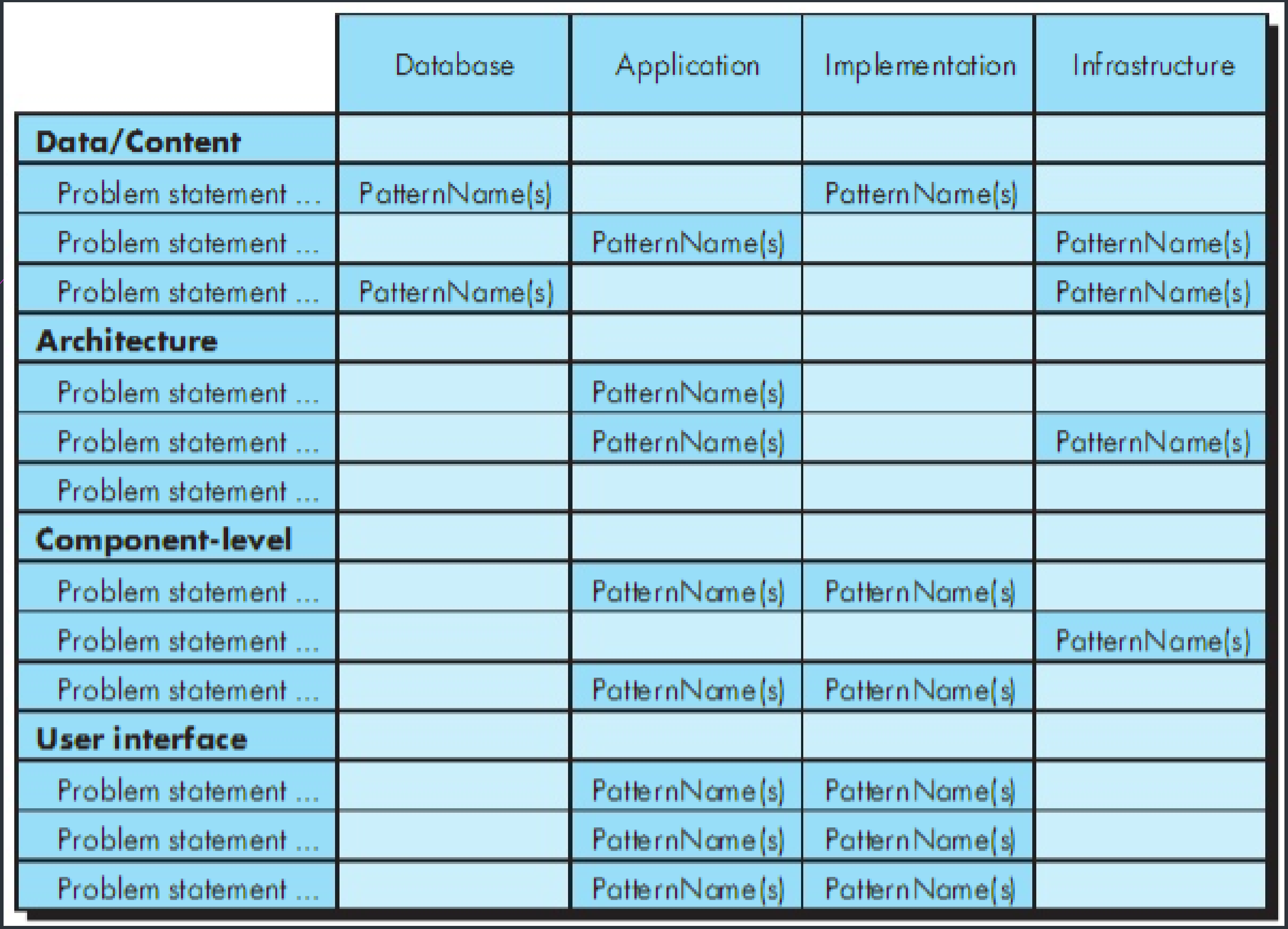


CONTENTS

Pattern-based design in context


Requirements model
Design begins
Consider design concepts
Extract problem, context forces
Consider design quality attributes
Addressed by pattern?
yes
no
Begin pattern-based design tasks
Apply other design methods and notation
Design model

Pattern-organization table



	Database	Application	Implementation	Infrastructure
Data/Content				
Problem statement ...	PatternName(s)		PatternName(s)	
Problem statement ...		PatternName(s)		PatternName(s)
Problem statement ...	PatternName(s)			PatternName(s)
Architecture				
Problem statement ...		PatternName(s)		
Problem statement ...		PatternName(s)		PatternName(s)
Problem statement ...				
Component-level				
Problem statement ...		PatternName(s)	PatternName(s)	
Problem statement ...				PatternName(s)
Problem statement ...		PatternName(s)	PatternName(s)	
User interface				
Problem statement ...		PatternName(s)	PatternName(s)	
Problem statement ...		PatternName(s)	PatternName(s)	
Problem statement ...		PatternName(s)	PatternName(s)	