ICS 52: Introduction to Software Engineering

Fall Quarter 2004

Professor Richard N. Taylor

Lecture Notes

Week 2: Principles and Requirements Engineering

http://www.ics.uci.edu/~taylor/ICS_52_FQ04/syllabus.html

University of California, Irvine

Recurring, Fundamental Principles

- Rigor and formality
- Separation of concerns
  - Modularity
  - Abstraction
- Anticipation of change
- Generality
- Incrementality

These principles apply to all aspects of software engineering

Rigor and Formality

- Creativity often leads to imprecision and inaccuracy
  - Software development is a creative process
  - Software development can tolerate neither imprecision nor inaccuracy
- Rigor helps to…
  - …produce more reliable products
  - …control cost
  - …increase confidence in products
- Formality is “rigor -- mathematically sound”
  - Often used for mission critical systems

Separation of Concerns

- Trying to do too many things at the same time often leads to mistakes
  - Software development is comprised of many parallel tasks, goals, and responsibilities
  - Software development cannot tolerate mistakes
- Separation of concerns helps to…
  - …divide a problem into parts that can be dealt with separately
  - …create an understanding of how the parts depend on/relate to each other

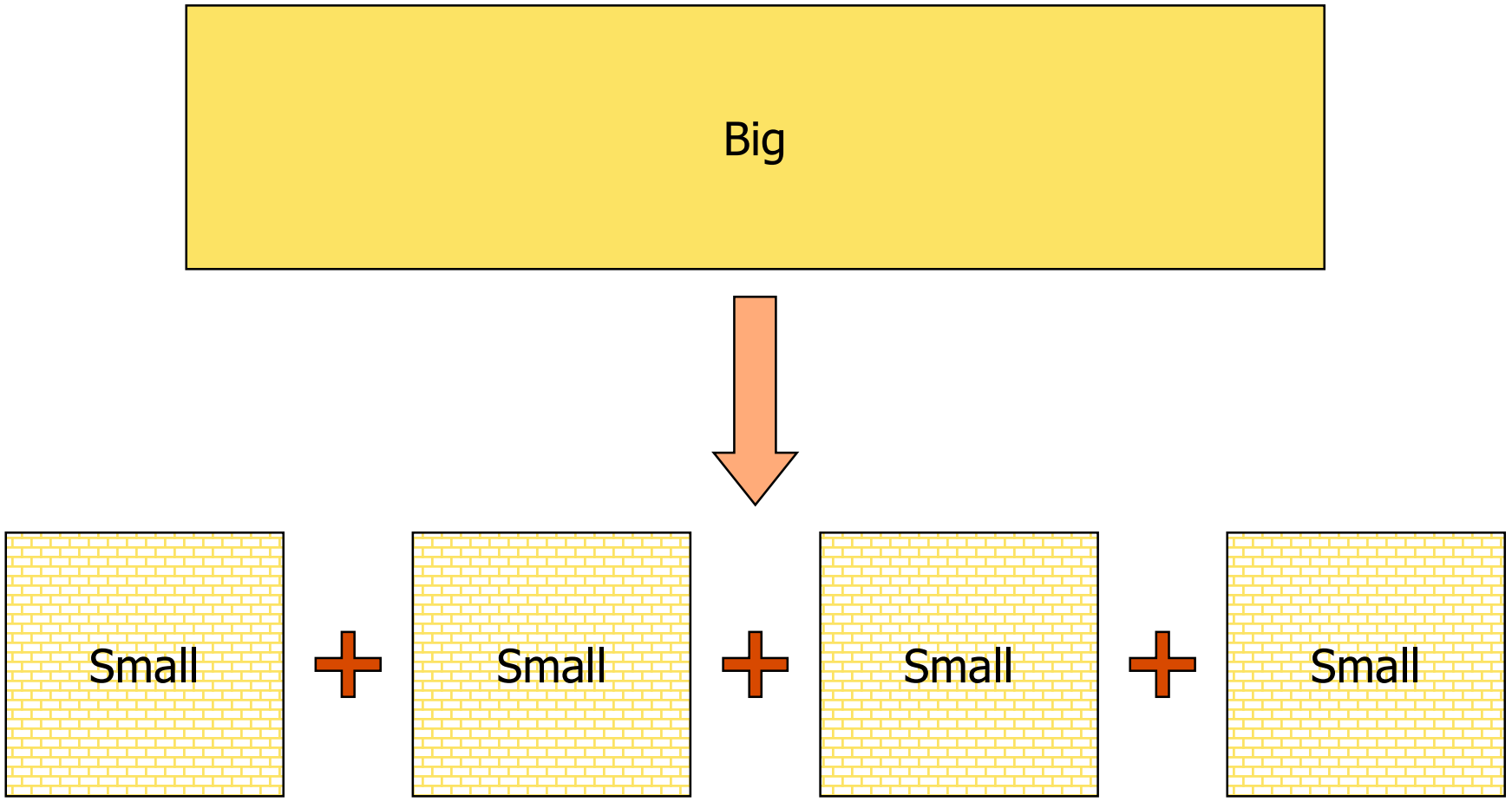
Example Dimensions of Separation

- Time
  - Requirements, design, implementation, testing, …
  - Dial, receive confirmation, connect, talk, …
- Qualities
  - Efficiency and user friendliness
  - Correctness and portability
- Views
  - Data flow and control flow
  - Management and development

Modularity

- Separation into individual, physical parts
  - Decomposability
    - » Divide and conquer
  - Composability
    - » Component assembly
    - » Reuse
  - Understanding
    - » Localization
- It is a particular type of separation of concerns
  - Divide and conquer “horizontally”
  - “Brick”-effect

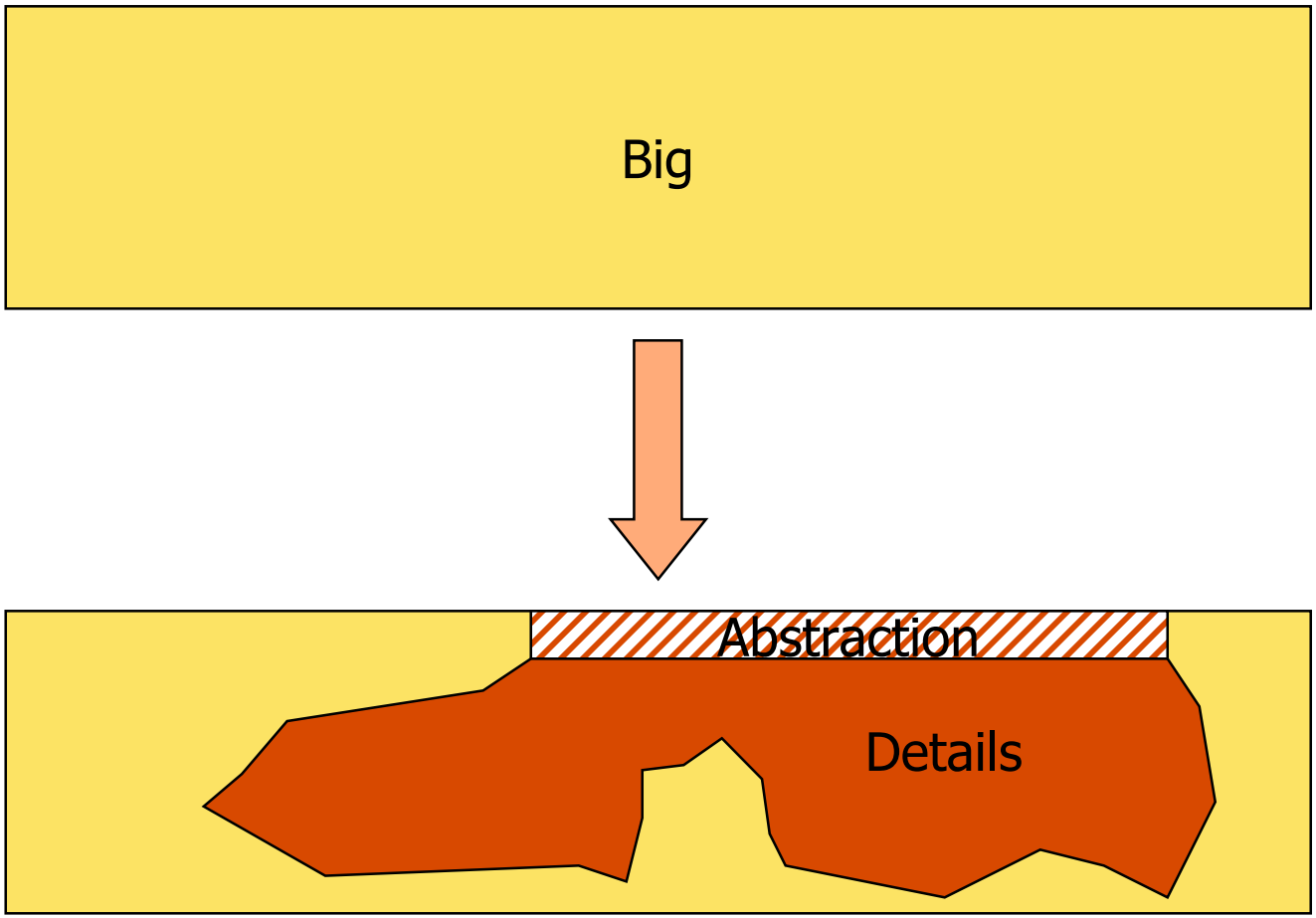
Modularity



Abstraction

- Separation into individual, logical parts
  - Relevant versus irrelevant details
    - » Use relevant details to solve task at hand
    - » Ignore irrelevant details
- Special case of separation of concerns
  - Divide and conquer “vertically”
  - “Iceberg”-effect

Abstraction



Anticipation of Change

- Not anticipating change often leads to high cost and unmanageable software
  - Software development deals with inherently changing requirements
  - Software development can tolerate neither high cost nor unmanageable software
- Anticipation of change helps to…
  - …create a software infrastructure that absorbs changes easily
  - …enhance reusability of components
  - …control cost in the long run

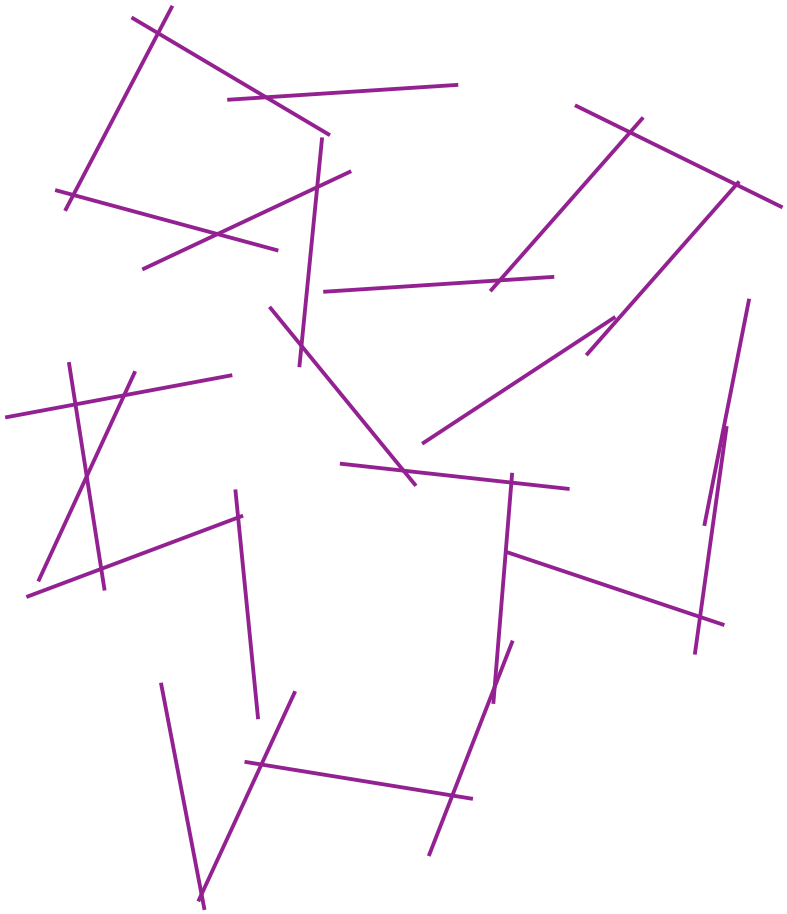
Generality

- Not generalizing often leads to continuous redevelopment of similar solutions
  - Software development involves building many similar kinds of software (components)
  - Software development cannot tolerate building the same thing over and over again
- Generality leads to…
  - …increased reusability
  - …increased reliability
  - …faster development
  - …reduced cost

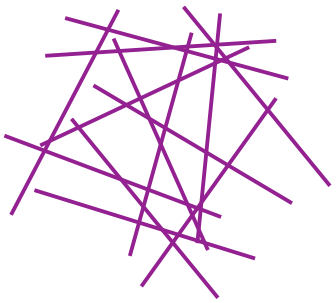
Incrementality

- Delivering a large product as a whole, and in one shot, often leads to dissatisfaction and a product that is “not quite right”
  - Software development typically delivers one final product
  - Software development cannot tolerate a product that is not quite right or dissatisfies the customer
- Incrementality leads to…
  - …the development of better products
  - …early identification of problems
  - …an increase in customer satisfaction
    - » Active involvement of customer

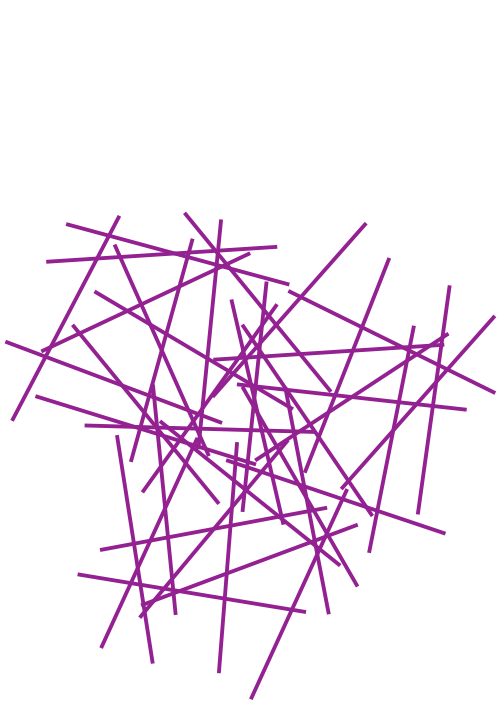
Cohesion



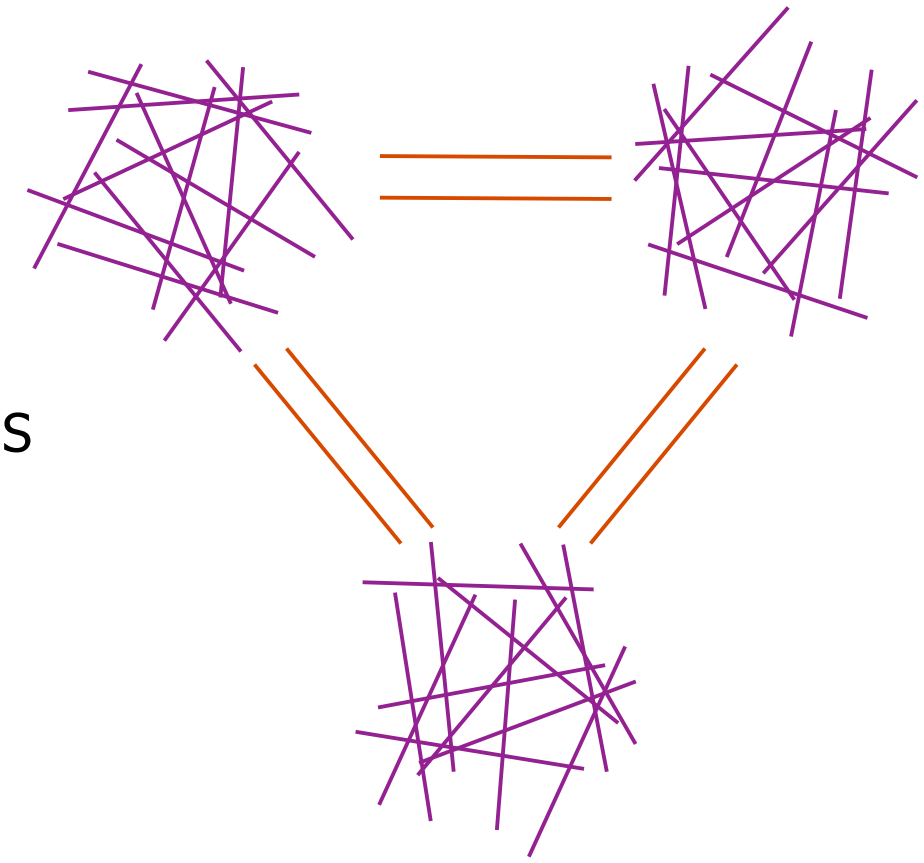
VERSUS



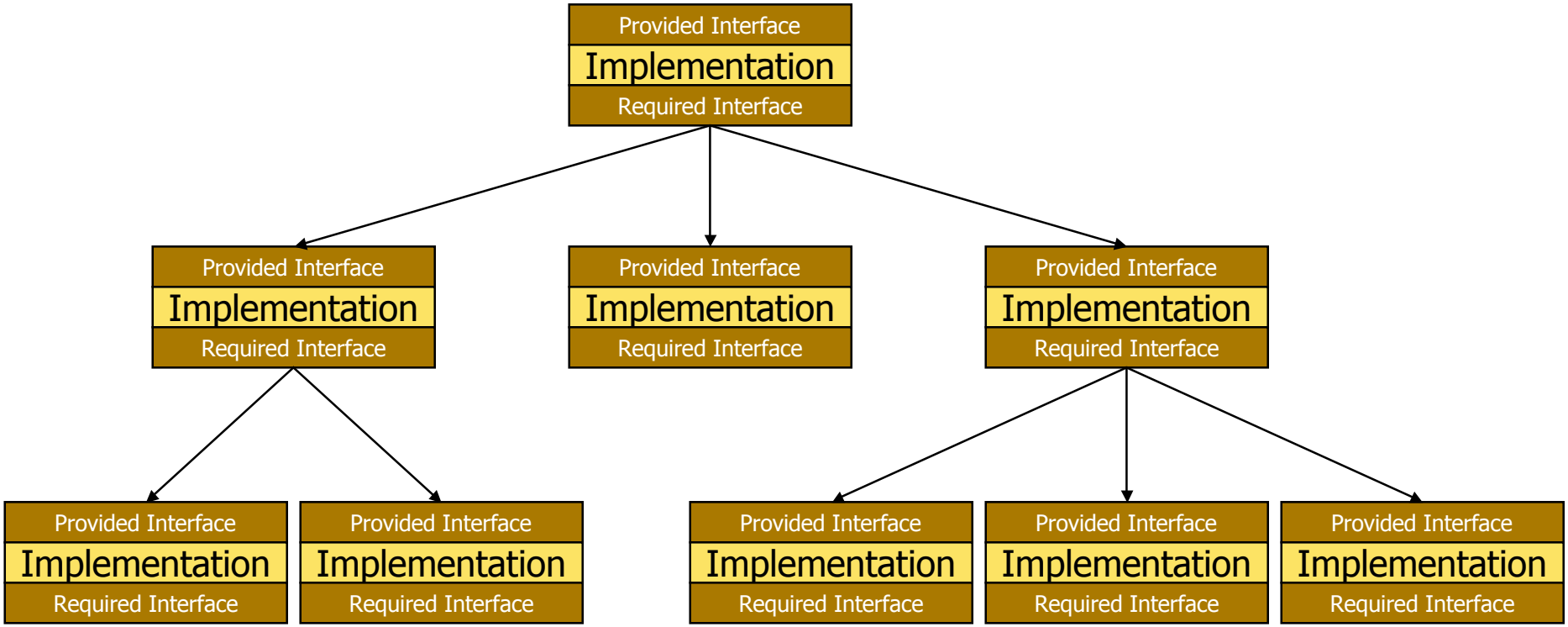
Coupling



VERSUS

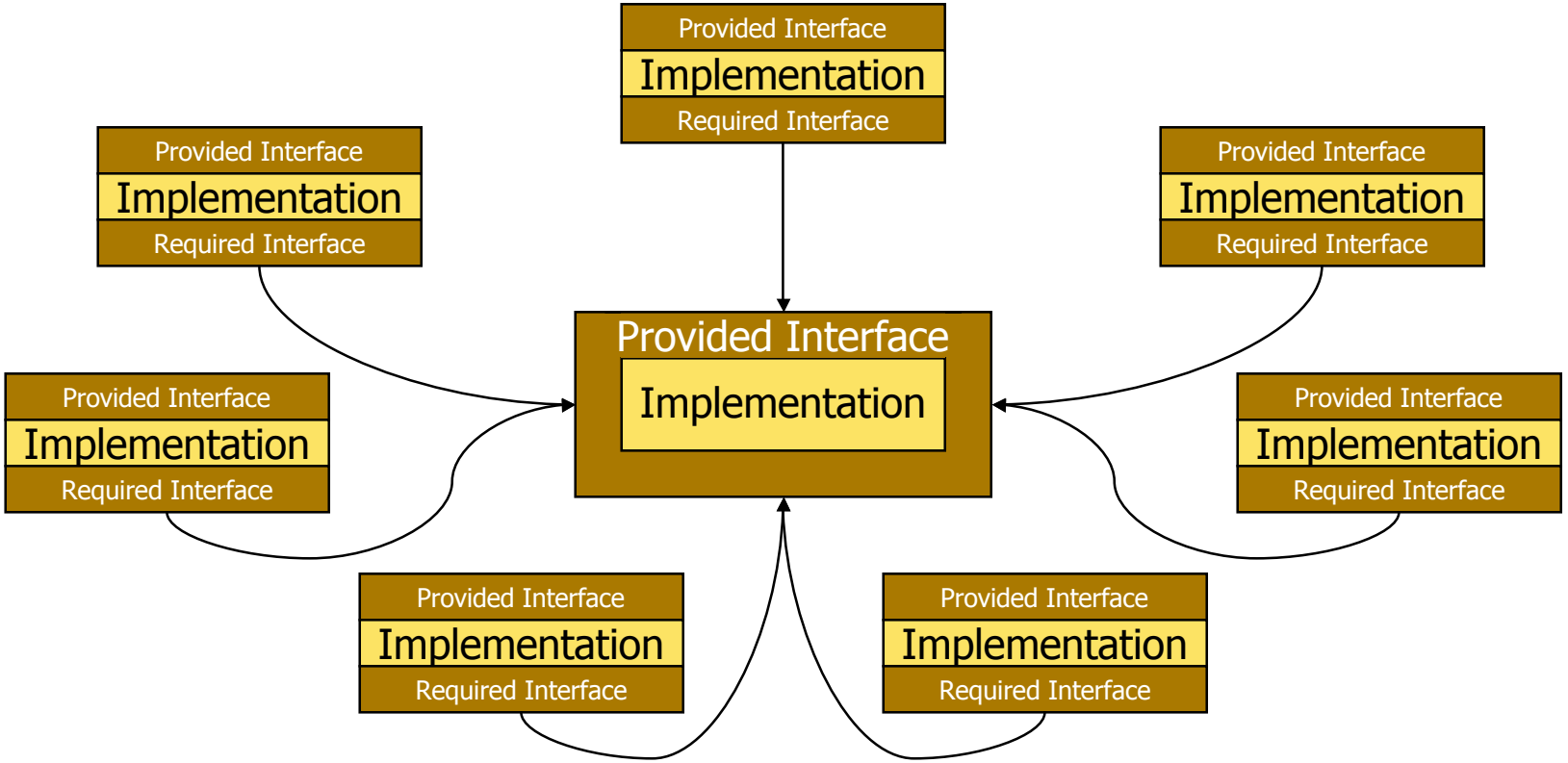


A Good Separation of Concerns, 1



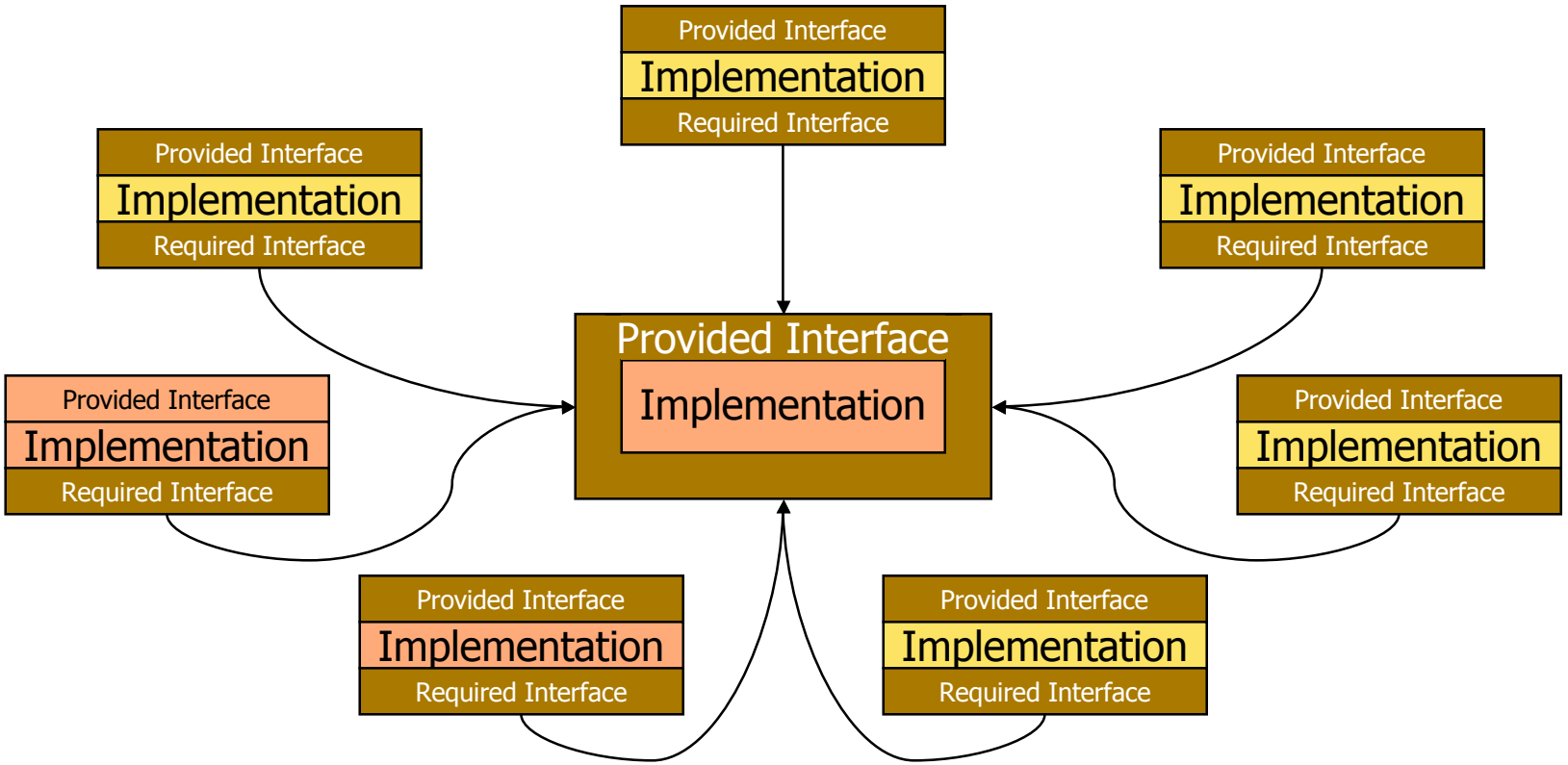
Abstraction through the use of provided/required interfaces
Modularity through the use of components
Low coupling through the use of hierarchies
High cohesion through the use of coherent implementations

A Good Separation of Concerns, 2



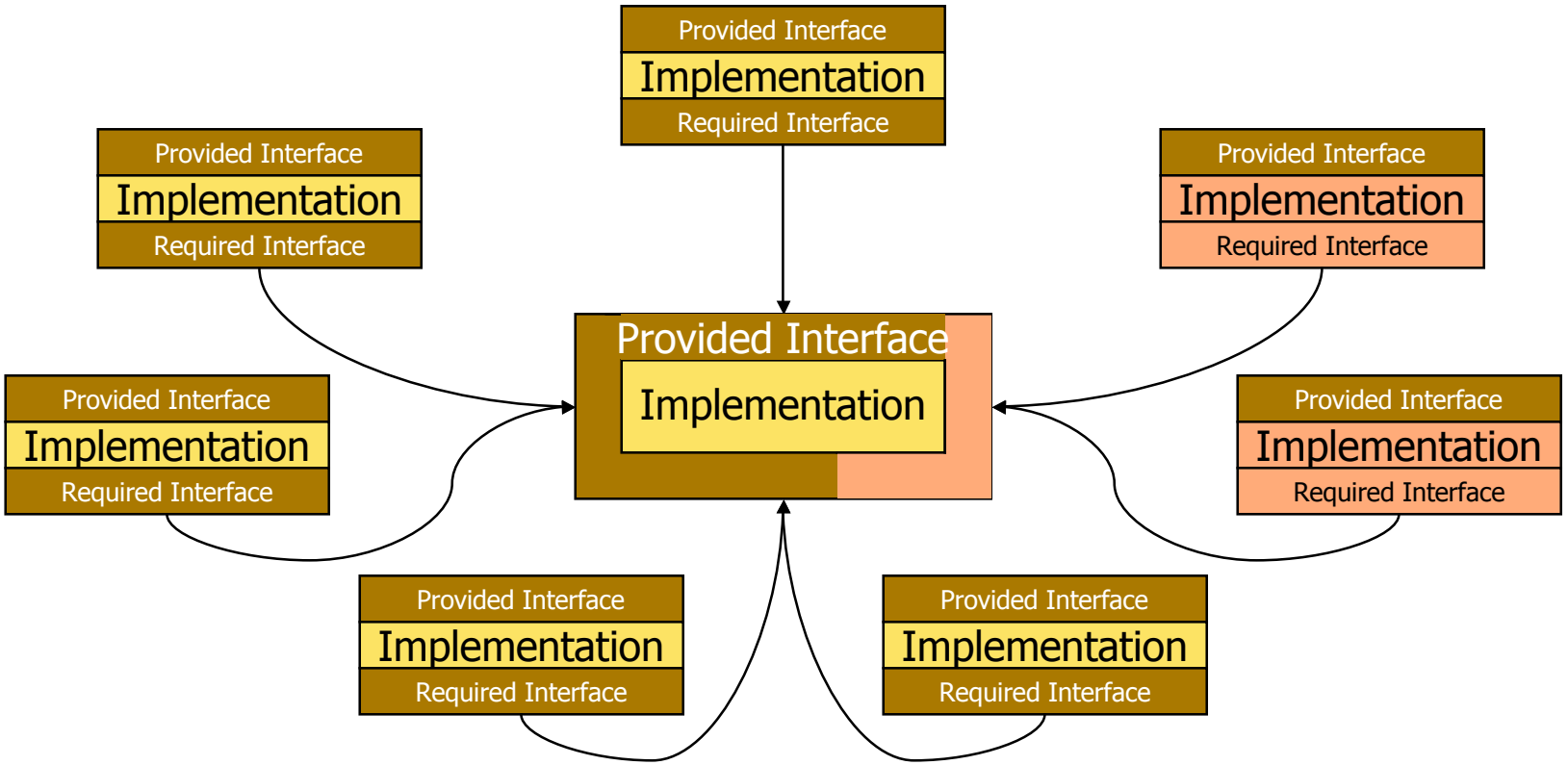
Abstraction through the use of provided/required interfaces
Modularity through the use of components
Low coupling through the use of a central "blackboard"
High cohesion through the use of coherent implementations

Benefit 1: Anticipating Change



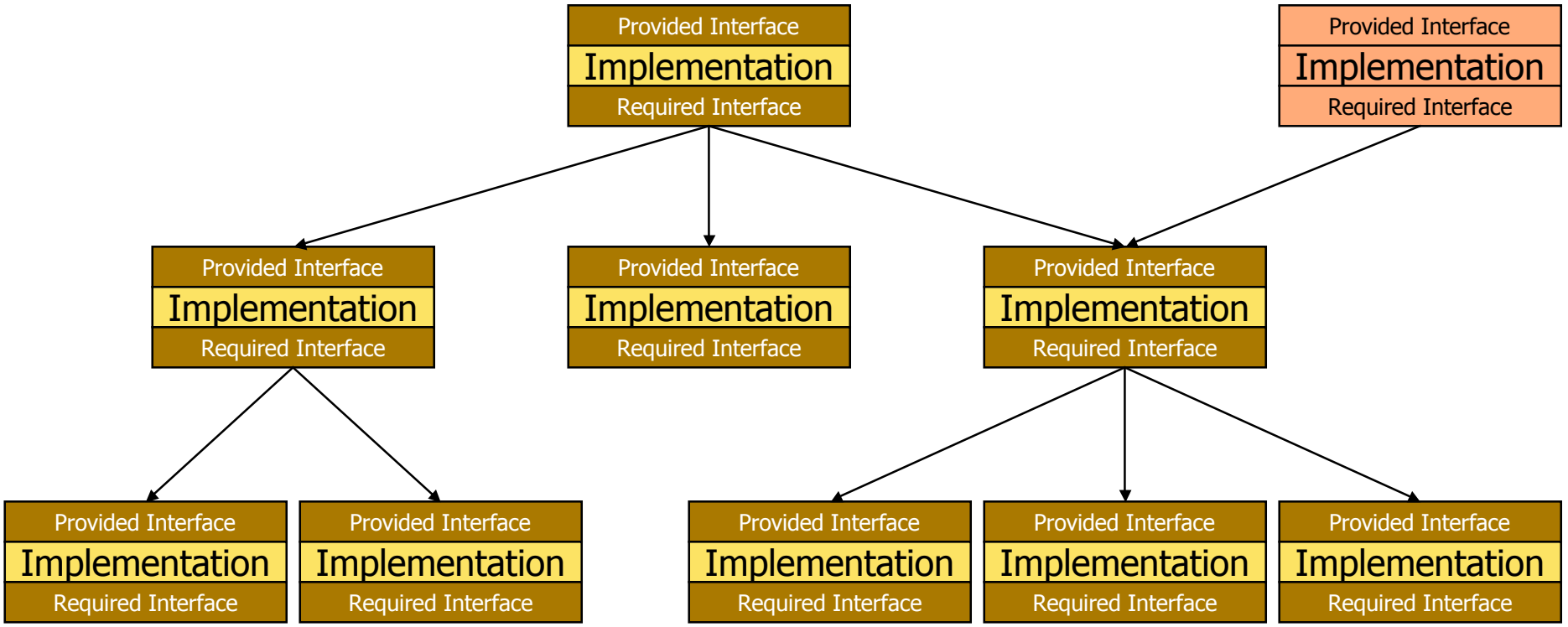
Separating concerns anticipates change

Benefit 1: Anticipating Change



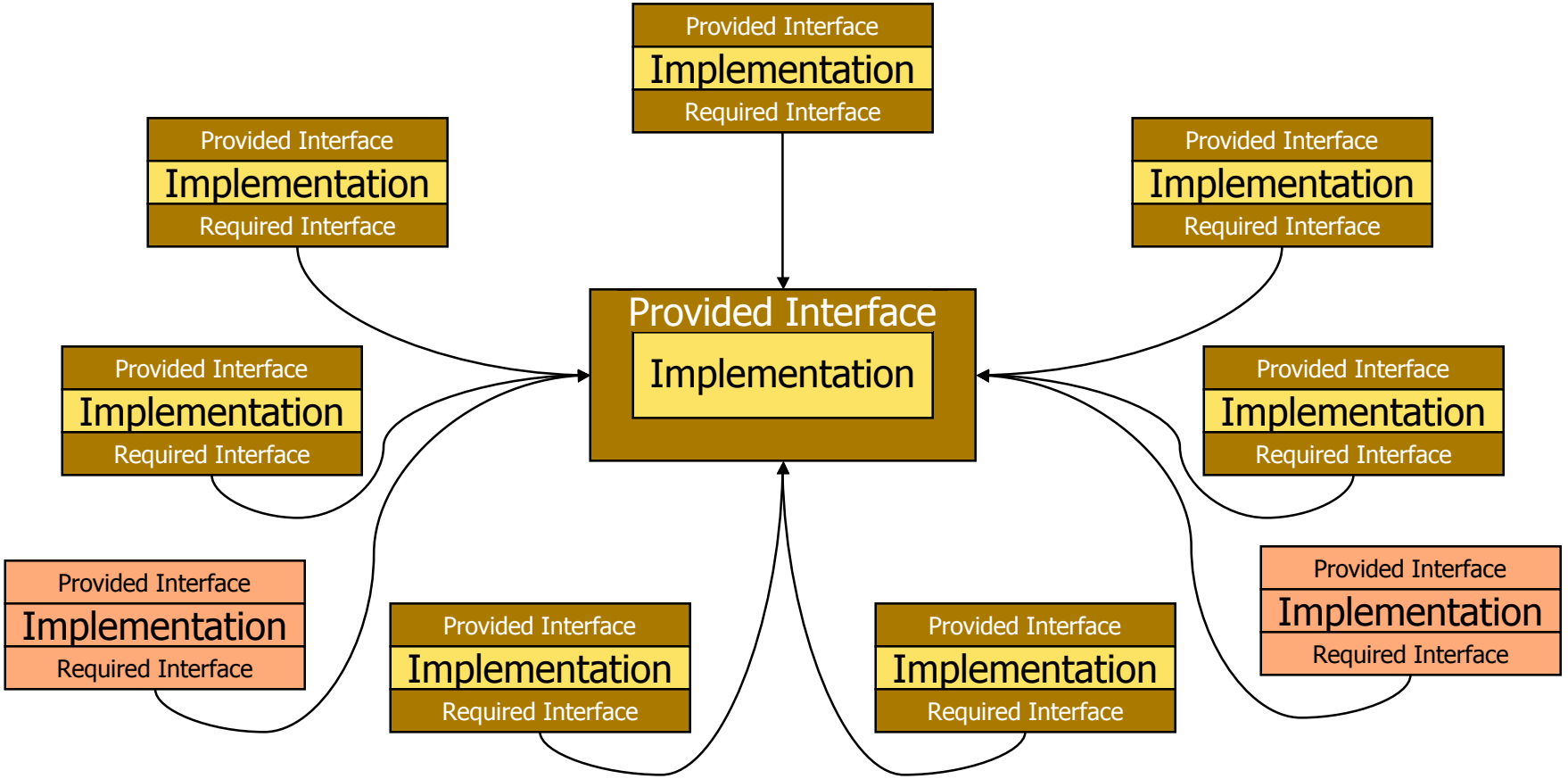
Separating concerns anticipates change

Benefit 2: Promoting Generality



Separating concerns promotes generality

Benefit 3: Facilitating Incrementality



Separating concerns facilitates incrementality

Recurring, Fundamental Principles

- Rigor and formality
- Separation of concerns
  - Modularity
  - Abstraction
- Anticipation of change
- Generality
- Incrementality

These principles apply to all aspects of software engineering

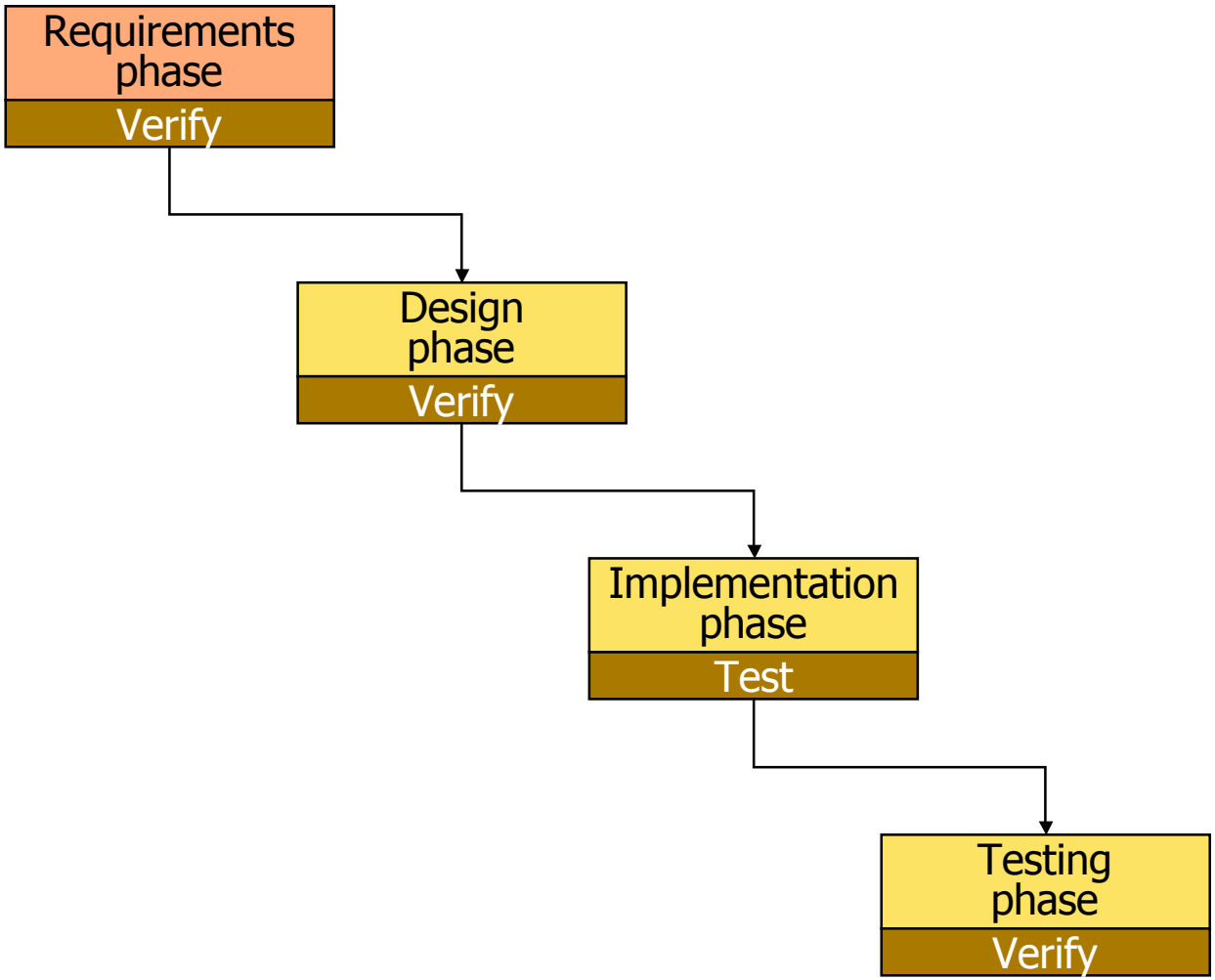
But Hey, This is a Postmodern Society

- If you think about what these software engineering principles are all about, you'll see they rest on deeper and more universal values:
  - Honesty
    - » If a specification says "X", then it should mean "X"
  - Living up to your commitments
    - » What is in a module's specification should be supported fully by its implementation
  - Faithfulness
    - » If you made the promise and other modules were built upon that promise, don't change your mind later on
  - Straightforwardness
    - » Say what you mean (and mean what you say)
  - Not being two-faced
    - » One specification for all customers
- The beauty of software is that the consequences of violating these principles are quickly seen and felt; there is indeed justice.
- So ditch the gutless, relativistic, self-centered ooze you get from society when you enter a software project (in fact, just ditch that junk period!)

Software Engineering Also Rests on Two Fundamental Understandings

- The characteristics of the *created* (software, in all its various forms)
  - Malleable
  - Intangible
  - Ultimately grounded in a precise, unforgiving medium: the computer
- The characteristics of the *creators* (people)
  - Fallible
  - Inconsistent
  - Imperfect
  - Unable to consistently meet the required standard of perfection

ICS 52 Life Cycle



Requirements Phase

- Terminology
  - Requirements analysis/engineering
    - » Activity of unearthing a customer's needs
  - Requirements specification
    - » Document describing a customer's needs

Requirements Analysis

- System engineering versus software engineering
  - What role does software play within the full solution?
  - Trend: software is everywhere
- Contract model versus participatory design
  - Contract: carefully specify requirements, then contract out the development
  - Participatory: customers, users, and software development staff work together throughout the life cycle
- Analyzing for a marketplace

Techniques for Requirements Analysis

- Interview customer
- Create use cases/scenarios
- Prototype solutions
- Observe customer
- Identify important objects/roles/functions
- Perform research
- Construct glossaries

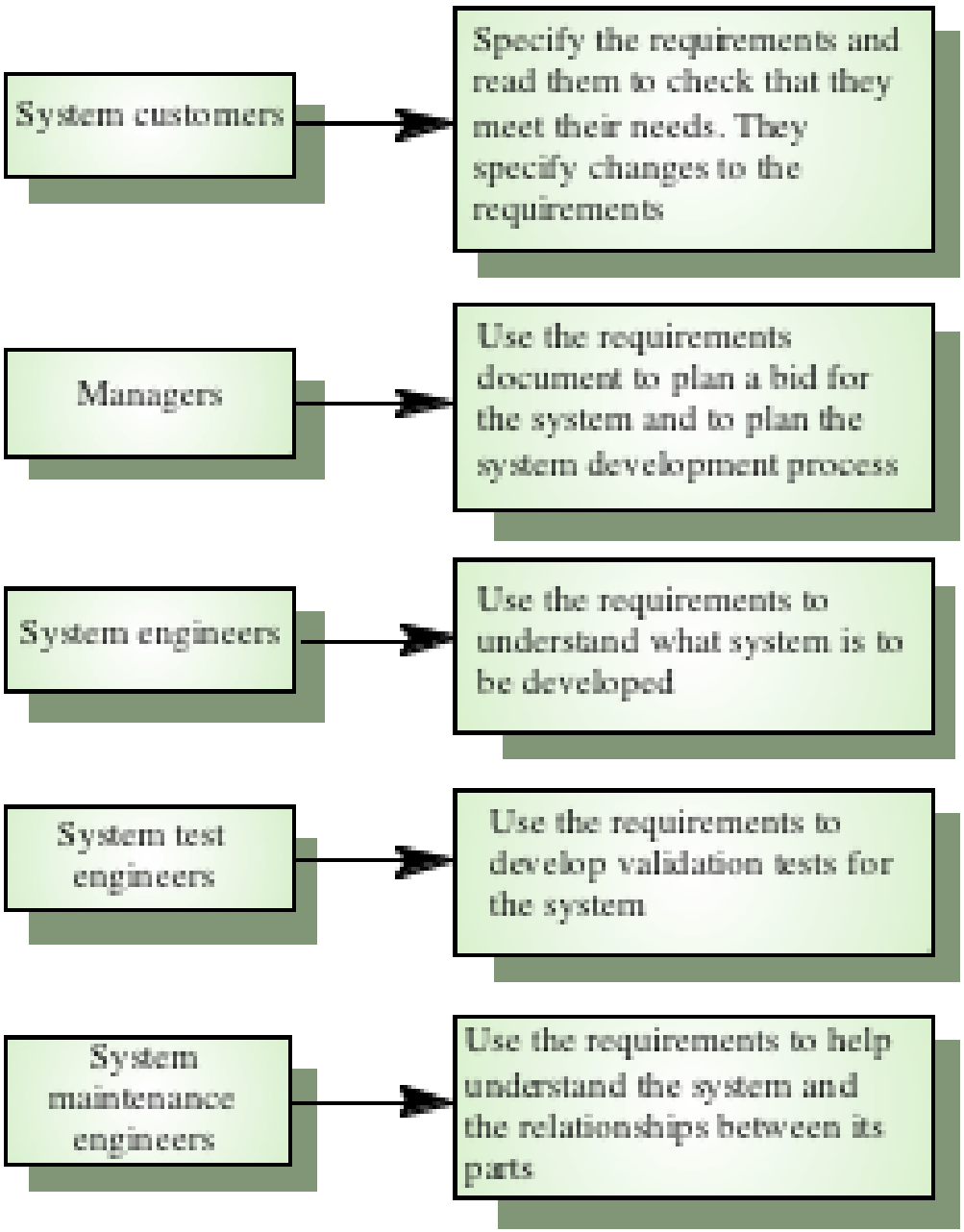
Use the principles

Requirements Specification

- Serves as the fundamental reference point between customer and software producer
- Defines capabilities to be provided without saying how they should be provided
  - Defines the “what”
  - Does not define the “how”
- Defines environmental requirements on the software to guide the implementers
  - Platforms
  - Implementation language(s)
- Defines software qualities

Requirements Specification (the Document)

- Purpose
  - Serve as the fundamental reference point between builder and buyer/"consumer " (contract)
  - Define capabilities to be provided, without saying how they should be provided
  - Define constraints on the software
    - » e.g. performance, platforms, language
- Characteristics
  - Unambiguous
    - » Requires precise, well-defined notations
  - Complete: any system that satisfies it is acceptable
  - Consistent
    - » There should be no conflicts or contradictions in the descriptions of the system facilities
  - Verifiable (testable)
  - No implementation bias (external properties only)
    - » "One model, many realizations"



Users of a requirements document

Lifecycle Considerations

- Serve as basis for future contracts
- Reduce future modification costs
  - Identify items likely to change
  - Identify fundamental assumptions
- Structure document to make future changes easy
  - e.g. have a single location where all concepts are defined

Recurring, Fundamental Principles

- Rigor and formality
- Separation of concerns
  - Modularity
  - Abstraction
- Anticipation of change
- Generality
- Incrementality

These principles apply to all aspects of software engineering

Requirements Volatility

	Customer Doesn't Care	Customer Cares	
		Measurable	Unmeasurable
Observable to Users	Requirement likely to change	Requirement	Goal
Not Observable to Users	Implementation detail	Constraint	

Figure 4–1: Matrix of Requirements Terminology

Source: David Alex Lamb, Software Engineering, Planning for Change
Prentice Hall, 1988

Structure of a Requirements Specification

- Introduction
- Executive summary
- Application context and Functional requirements
- Environmental requirements
- Software qualities
- Other requirements
- Time schedule
- Potential risks
- Future changes
- Glossary
- Reference documents

Content of a Requirements Specification

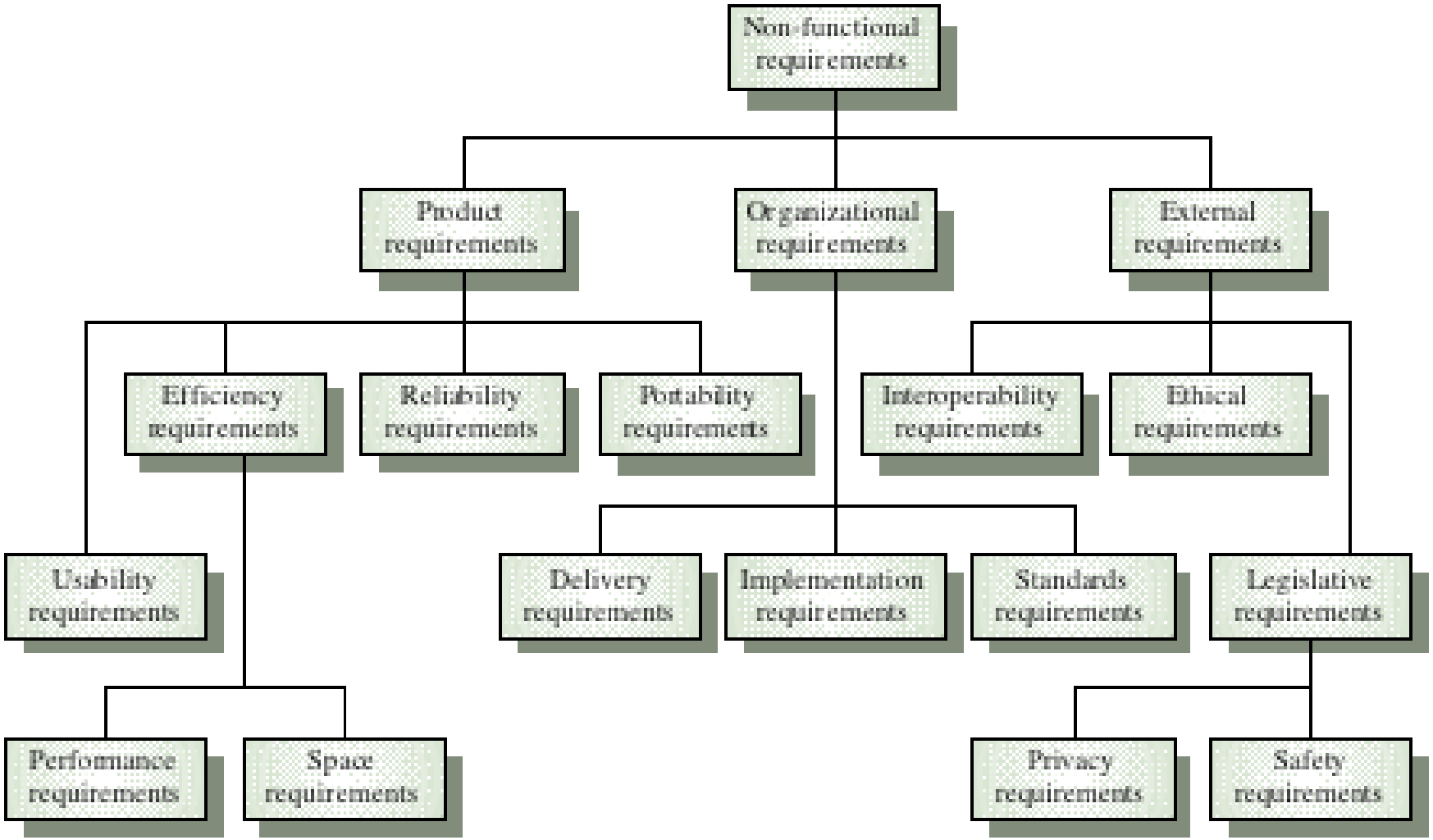
- Application context
  - Describe the situations in which the software will be used. How will the situation change as a result of introducing the software system?
  - Identify all things (objects, processes, other software, hardware, people) that the system may, or will, affect.
  - Develop an abstraction for each of those things, characterizing their properties/behavior which are relevant to the software system. ("World model.")
  - How might this context change?
- Functional requirements ("features")
  - Identify all concepts (objects) that the system provides to the users.
  - Develop an abstraction for each of those concepts, characterizing their properties and functions which are relevant to the user.
    - » What is the system supposed to do?
    - » What is supposed to happen when something goes wrong?

Object-oriented Analysis

Contents of a Requirements Specification, cont..

- Performance requirements: speed, space
- Environmental requirements: platform, language, ...
- Subsets/supersets
- Expected changes and fundamental assumptions
- Definitions; reference documents

Non-functional requirement types



World Model (OOA) versus Simple Input/Output Characterizations as Reqt.s Specs

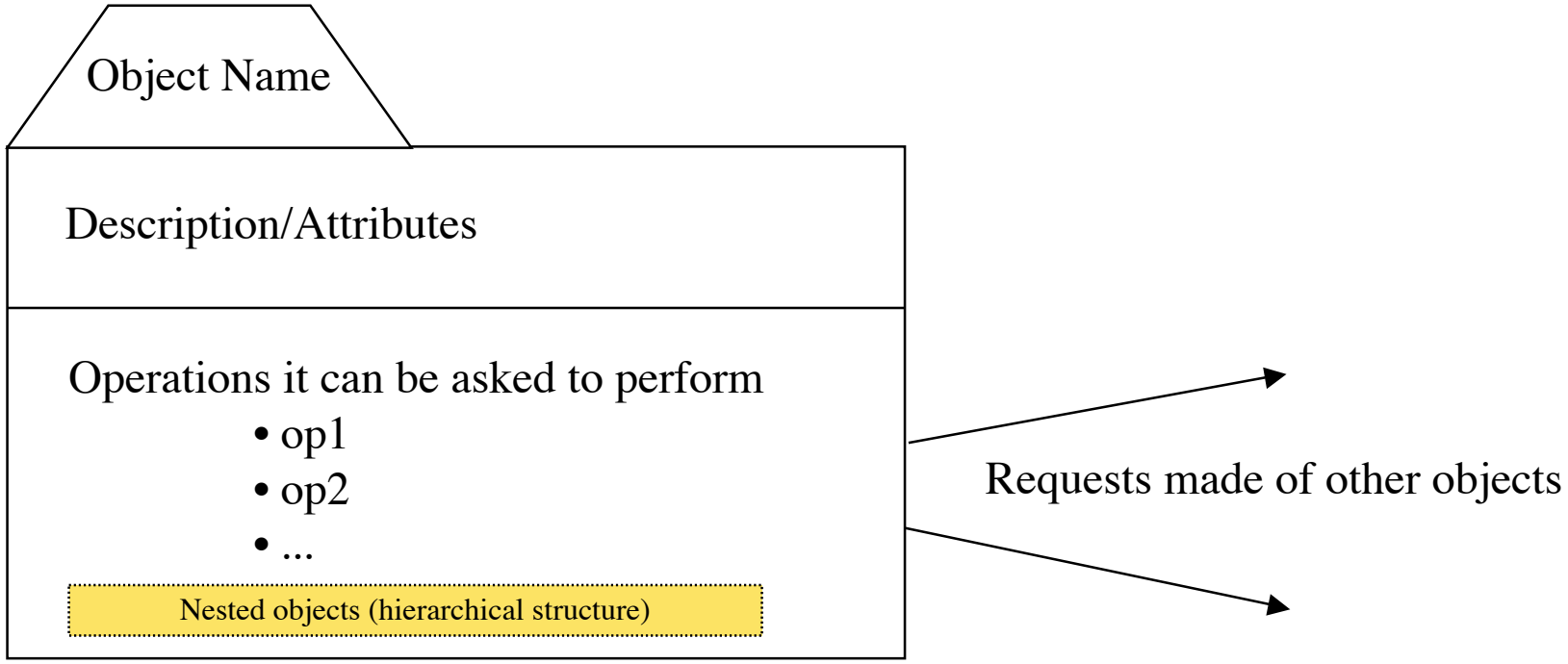
- The application context may change because of extrinsic factors
- The software system modifies the usage context

- I/O is only meaningful in a specific context
- "Input" and "output" may not be simple concepts
  - Cruise control systems: many sensors, complex conditions, and timing constraints only understandable in the application context

Techniques for Requirements Analysis

- Conduct interviews
- Build and evaluate prototypes
- Construct glossaries
- Separate concerns
- Focus on structure
  - Abstraction and hierarchical decomposition
- Use precise notation (be careful with diagrams!)
- Ask yourself:
  - Is it testable? Complete? Consistent?

Canonical Diagram for Requirements **Objects**



Note: this will not be the appropriate notation for all application contexts!

Mailing List Manager

Mailing Address

A place where mail can be delivered.
Name, Title, Street, City, State, ZipCode.

Operations:
(1) change any of the specified attributes to have a particular value.
(2) read any or all of the attributes
(3) create/delete address

Note: are the values to the "puts" or received from the "gets" strings? Only strings?

Mailing List

A list of Mailing_Address objects.
Name (of list)

Operations:
(1) Add Mailing_Address to list
(2) Delete Mailing_Address from list
(3) Sort list
(4) "Print" list

Note: What about querying the list to see if a particular address --- or part of one -- is already a member?

Storage

An indexed set of places where chunks of ASCII data can be stored. Number of indices, size of data currently stored in each index

Operations:
(1) Fetch data at index
(2) Store data at index

Mailing List Set Ops

Supports manipulation of multiple mailing lists.

Operations:
(1) Union of two lists
(2) Intersection of two lists
(3) Subtraction of one list from another

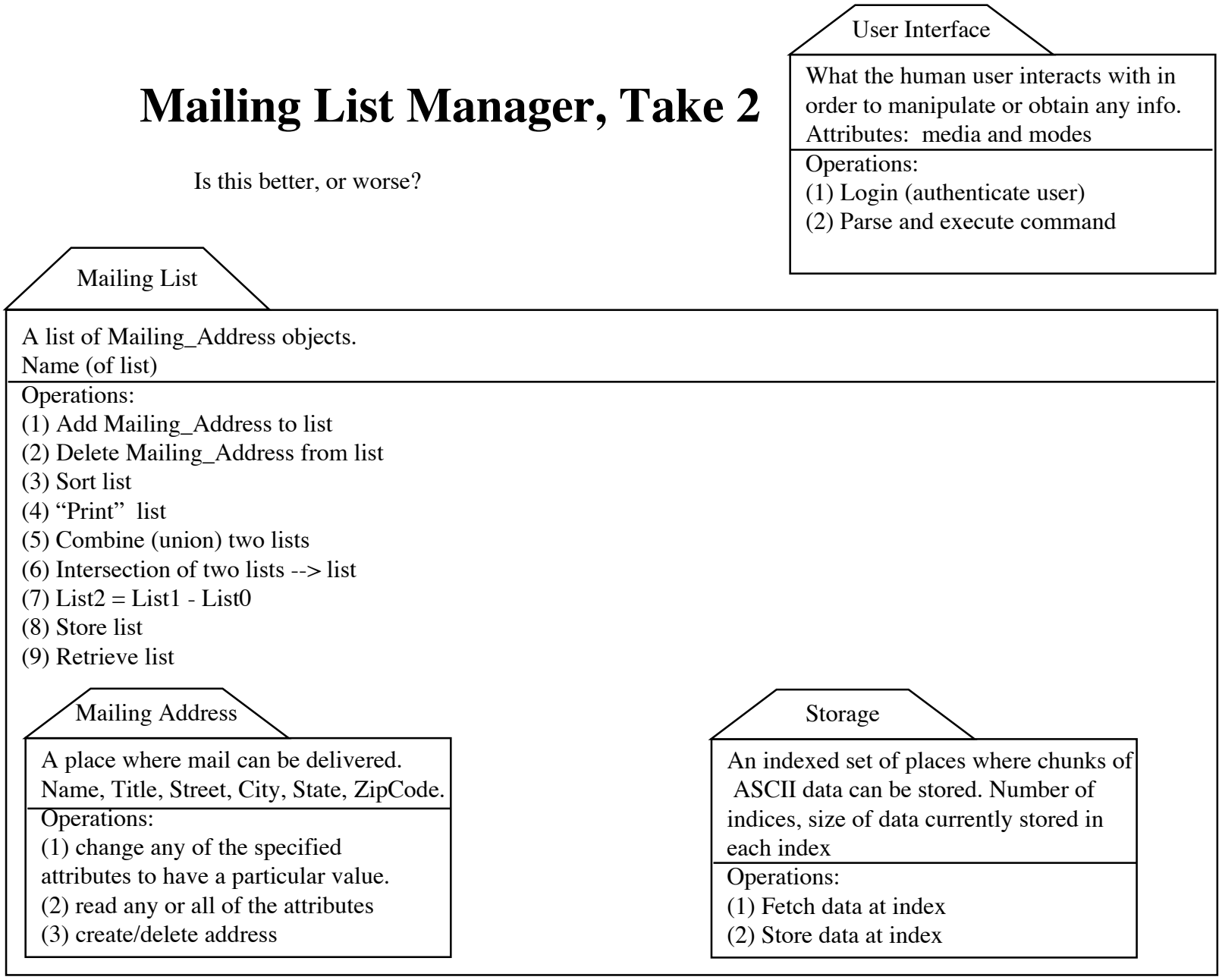
User Interface

What the human user interacts with in order to manipulate or obtain any info.
Attributes: media and modes

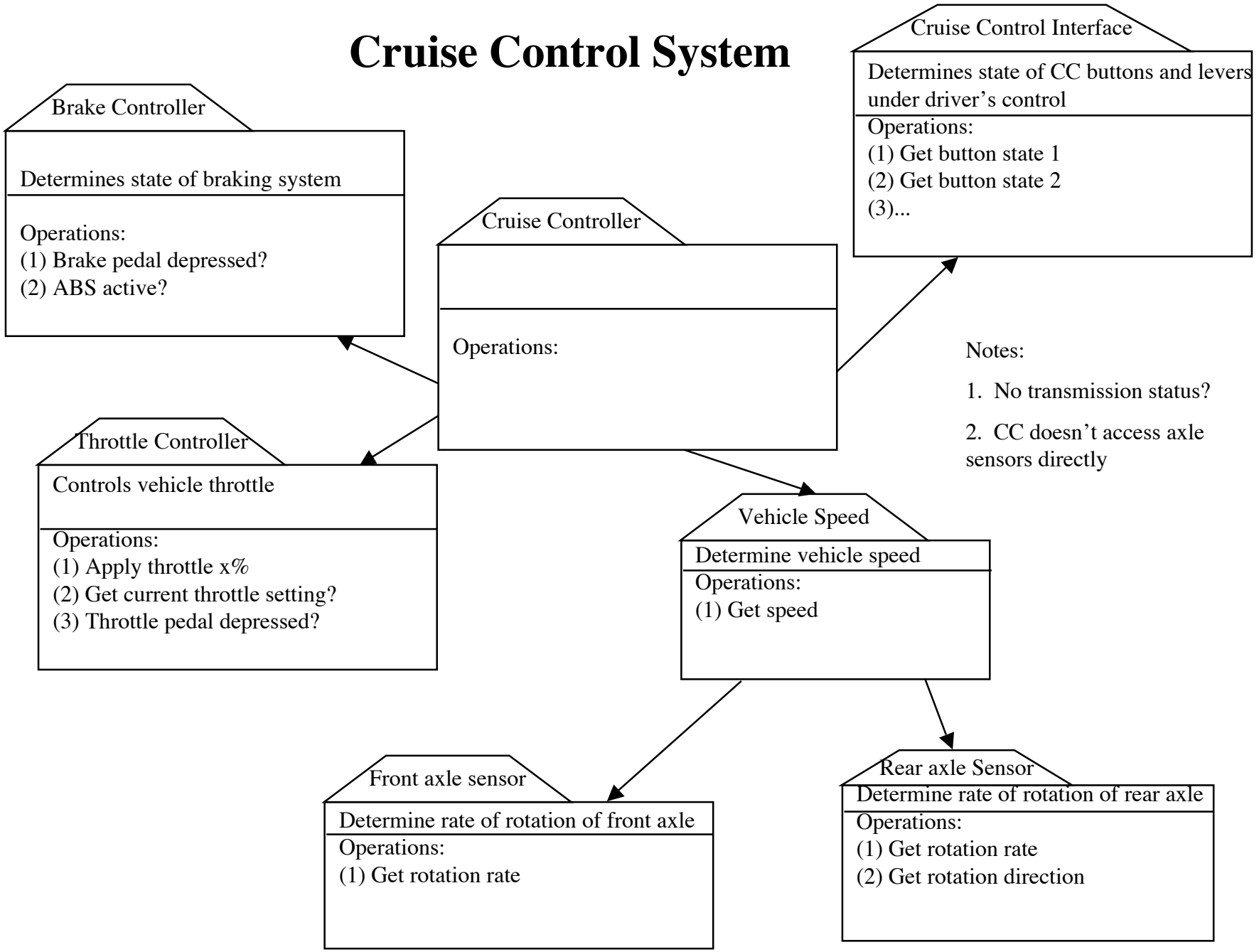
Operations:
(1) Login (authenticate user)
(2) Parse and execute command

Mailing List Manager, Take 2

Is this better, or worse?



Cruise Control System



Different Circumstances, Different Techniques

- Finite state machines
  - telephony examples
- Numerical systems
  - e.g. matrix inversion package

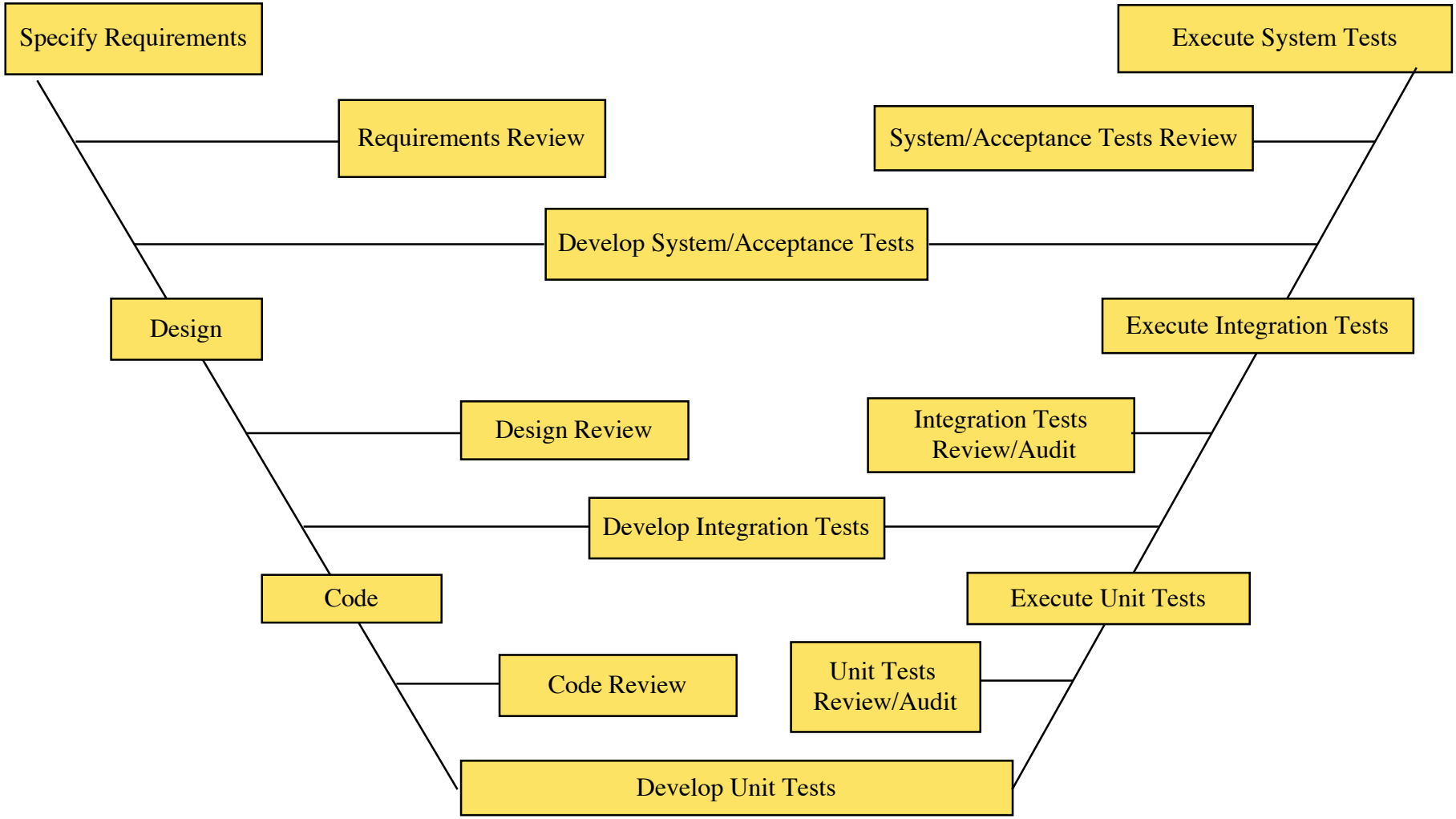
Glossary

- You'd think this was too simple and obvious to talk about…
  - Here's what the University of British Columbia School of Architecture has in their glossary for defining "geometry":
    - Once considered both science and art, the classical discipline of geometry was simultaneously a symbolic vessel and the Instrument for generating architectural form. Power and authority were communicated through ideal forms, and architects and masons used machines to inscribe these ideal forms into their monuments. The intervening centuries of revolution, capitalism, mass production, and distraction have gradually stripped geometry of its magical attributes, and today geometry no longer continues to be a primary conveyor of meaning. Geometry is now both technology and translator, an instrument that rationalizes number through a set of digitally calculated mathematical principles projected into an infinite number of points. This system provides a great deal of flexibility to the process of determining form since It also furnishes to architects the means to correlate and fabricate surfaces, edges, and forms with great precision.
- The objective is precision. The objective is clarity. The objective is disambiguation.

Acceptance Test Plan

- An operational way of determining consistency between the requirements specification and the delivered system
- If the system passes the tests demanded by this plan, then the buyer has no (legal) basis for complaint
- Develop a plan for conducting test to examine
  - Functional properties
  - Performance properties
  - Adherence to constraints
  - Subsets
- Representative technique: Property/test matrix: for each test case, what properties/behaviors will be demonstrated?

V-Model of Development and Testing Activities



Incremental Development of Tests

- Acceptance test plan (and tests): develop during requirements analysis
- Integration test plan (and test): develop during system architecture and detailed design specification
- Unit test plan (and tests): develop during implementation

ICS 52 Requirements Analysis Exercise

- Develop a requirements specification and acceptance test plan for the class project
- TA is the customer rep