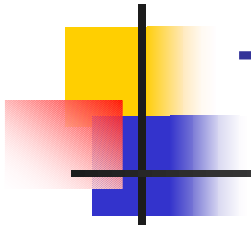
ICS 52
Introduction to Software Engineering

Lecture Notes for Fall Quarter, 2001

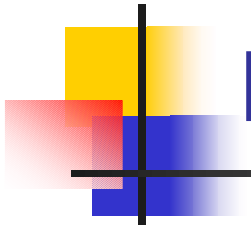
André van der Hoek

Lecture 2-1



Today's Lecture

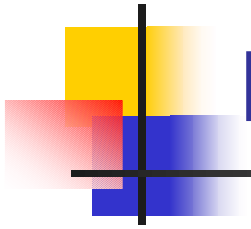
- Recurring and fundamental principles of software engineering
- An introduction to requirements



Recurring, Fundamental Principles

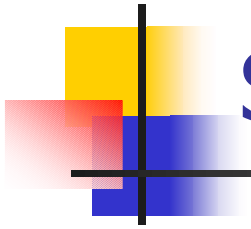
- Rigor and formality
- Separation of concerns
  - Modularity
  - Abstraction
- Anticipation of change
- Generality
- Incrementality

These principles apply to all aspects of software engineering



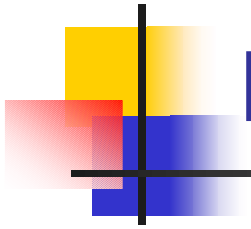
Rigor and Formality

- 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 confidentiality in products
- Formality is "rigor -- mathematically sound"
  - Often used for mission critical systems



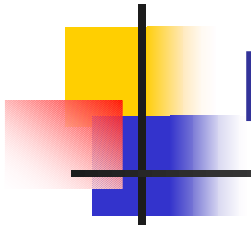
Separation of Concerns

- 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

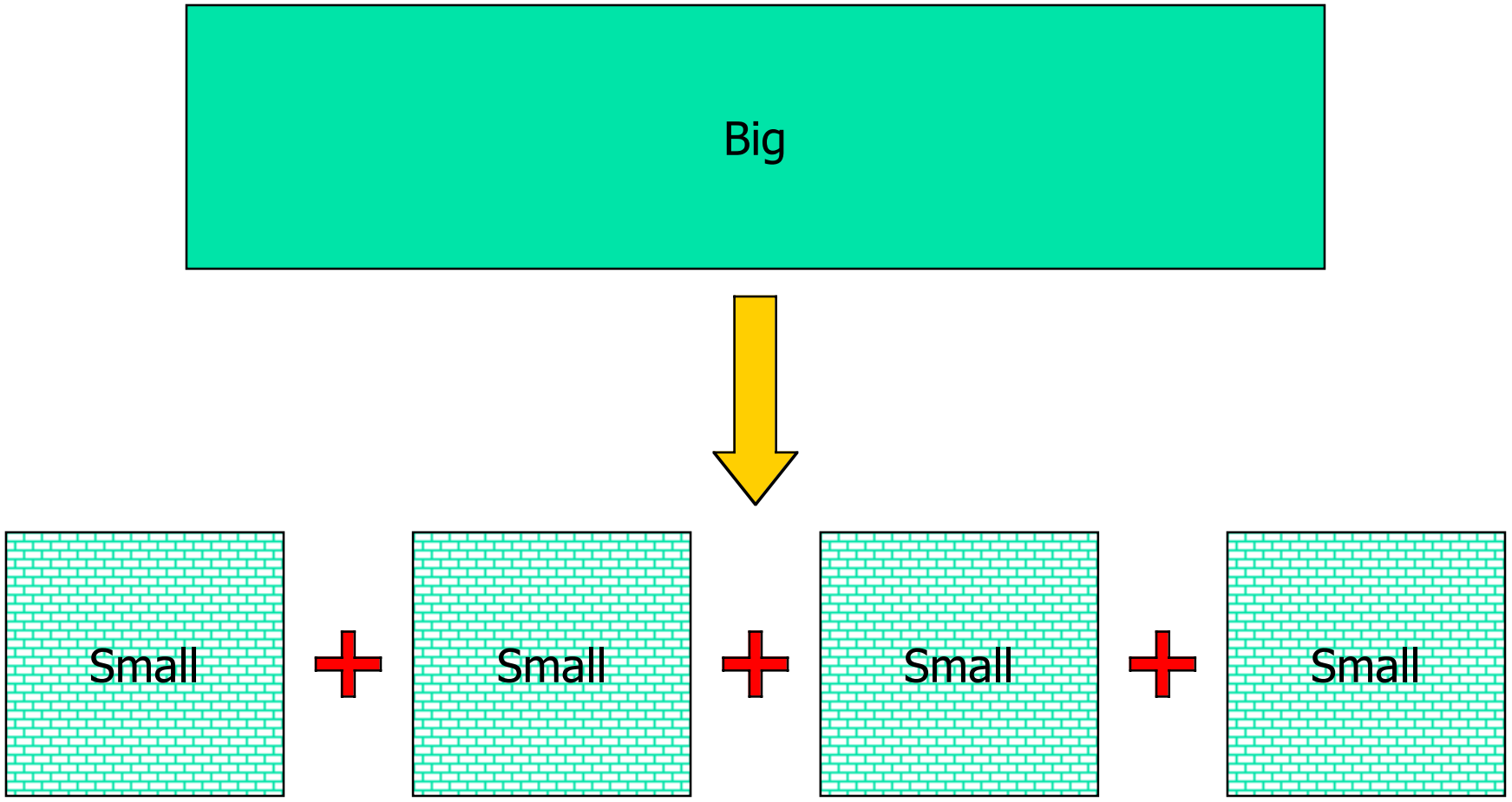
- Time
  - Requirements, design, implementation, testing, …
  - Dial, receive confirmation, connect, talk, …
- Qualities
  - Efficiency and user friendliness
  - Correctness and portability
- Views
  - Data flow and control flow
  - Management and development

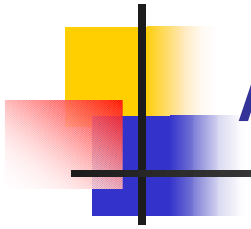


Modularity

- Separation into individual, physical parts
  - Decomposability
    - Divide and conquer
  - Composability
    - Component assembly
    - Reuse
  - Understanding
    - Localization
- Special case of separation of concerns
  - Divide and conquer “horizontally”
  - “Brick”-effect

Modularity

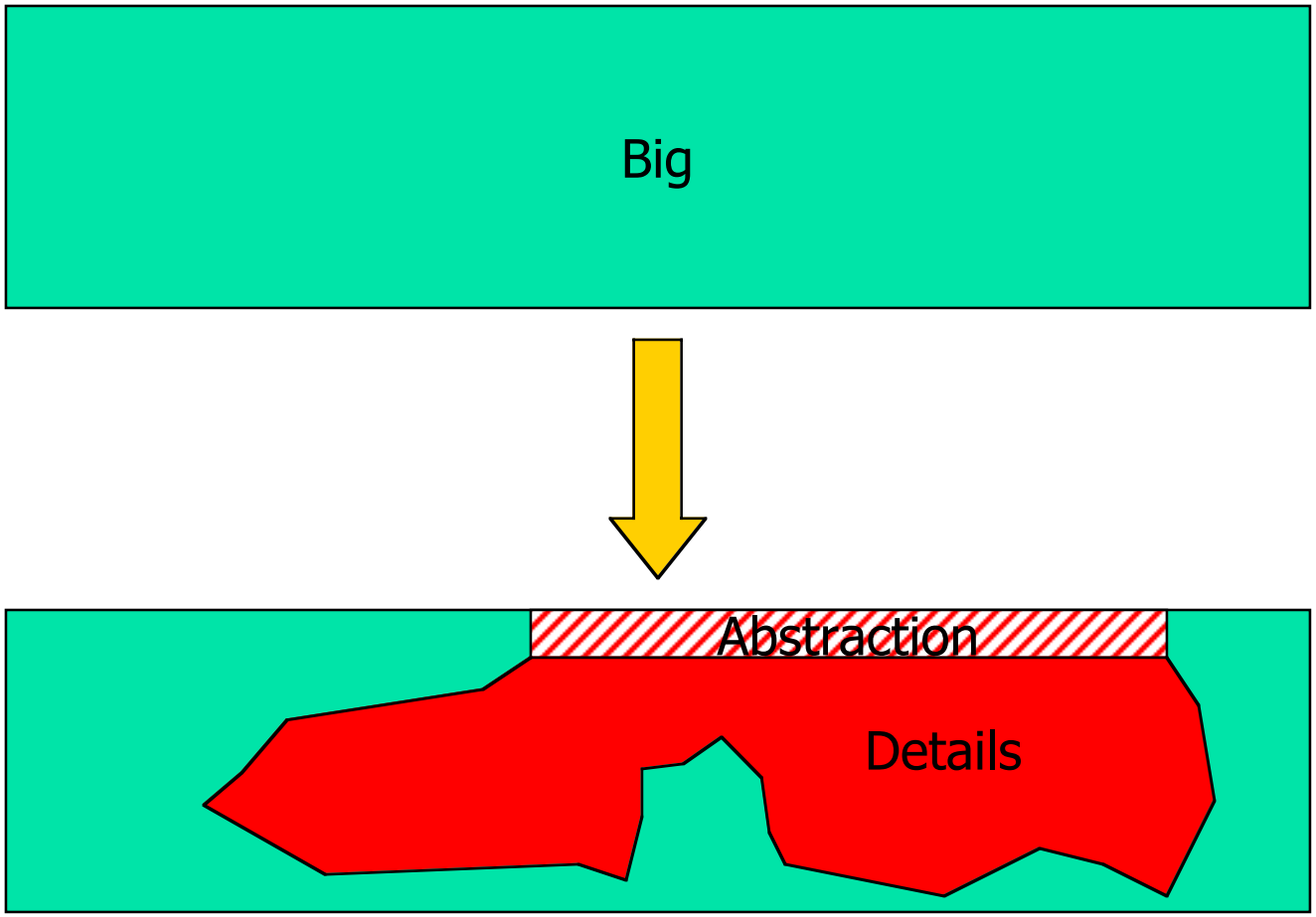


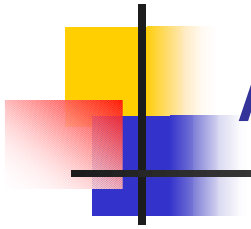


Abstraction

- 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

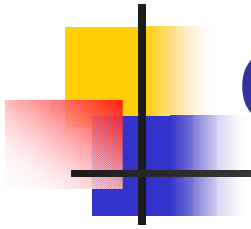
Abstraction





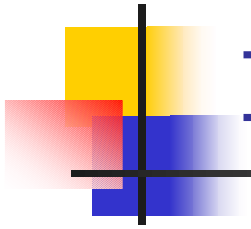
Anticipation of Change

- 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

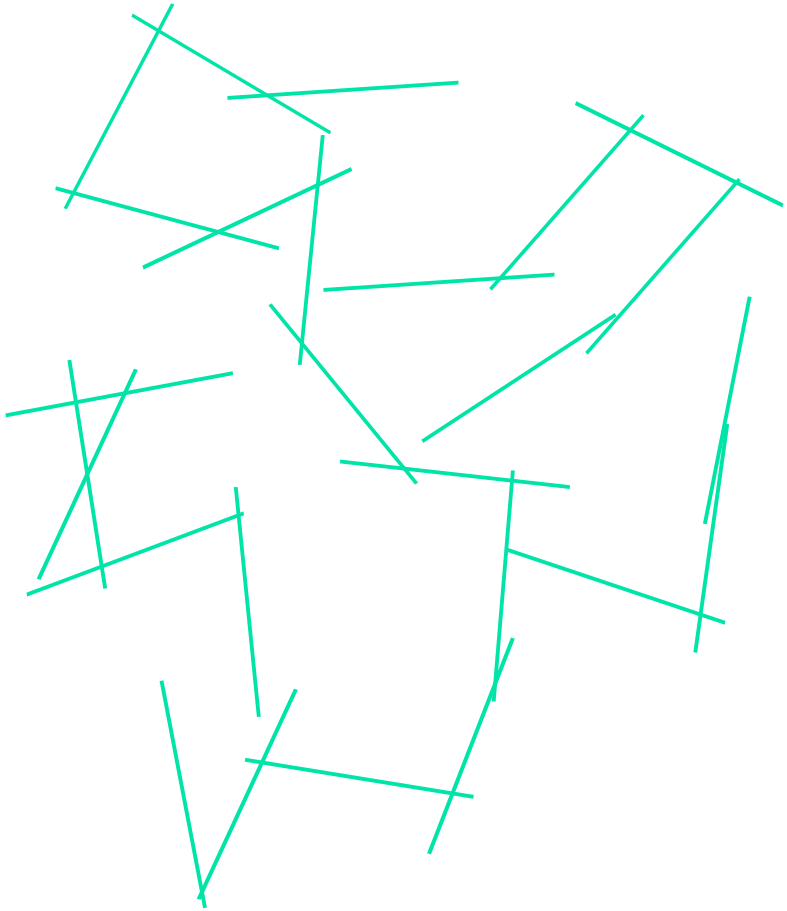
- Not generalizing often leads to continuous redevelopment of similar solutions
  - Software development involves building many similar kinds of software (components)
  - Software development cannot tolerate building the same thing over and over again
- Generality leads to…
  - …increased reusability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



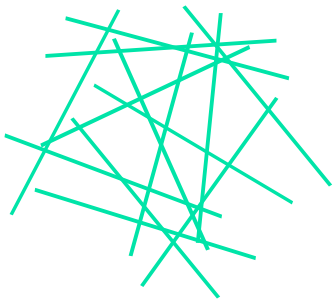
Incrementality

- 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

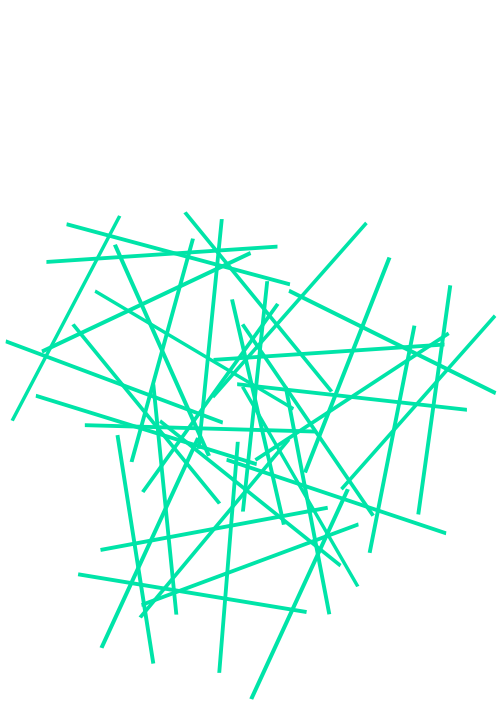
Cohesion



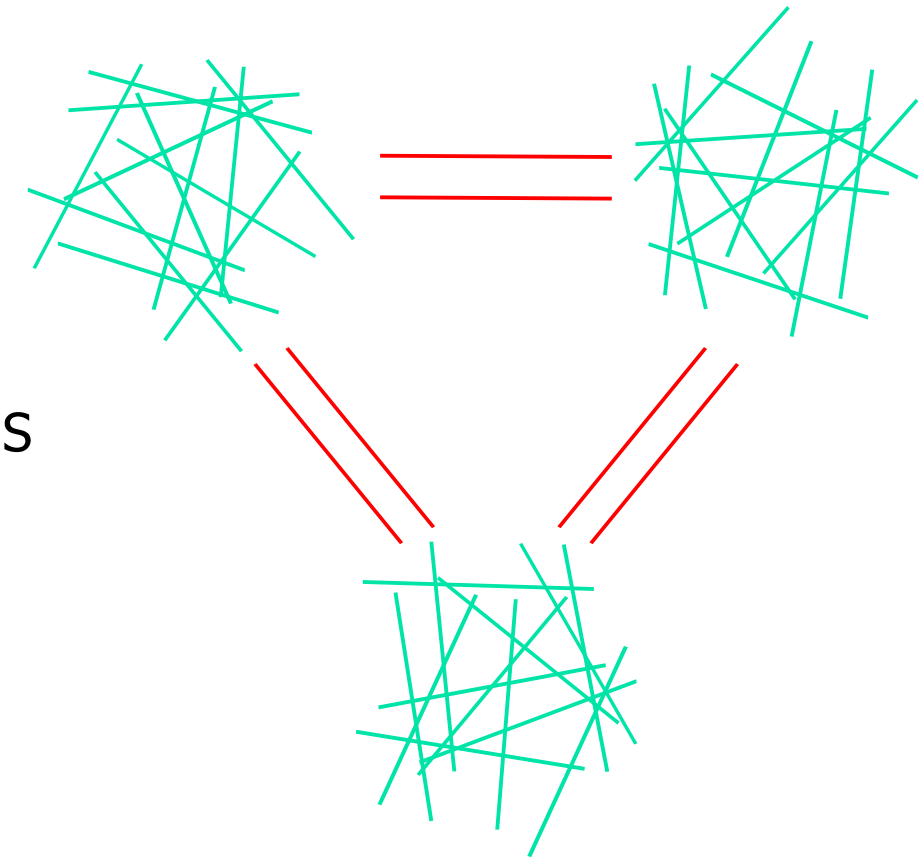
VERSUS



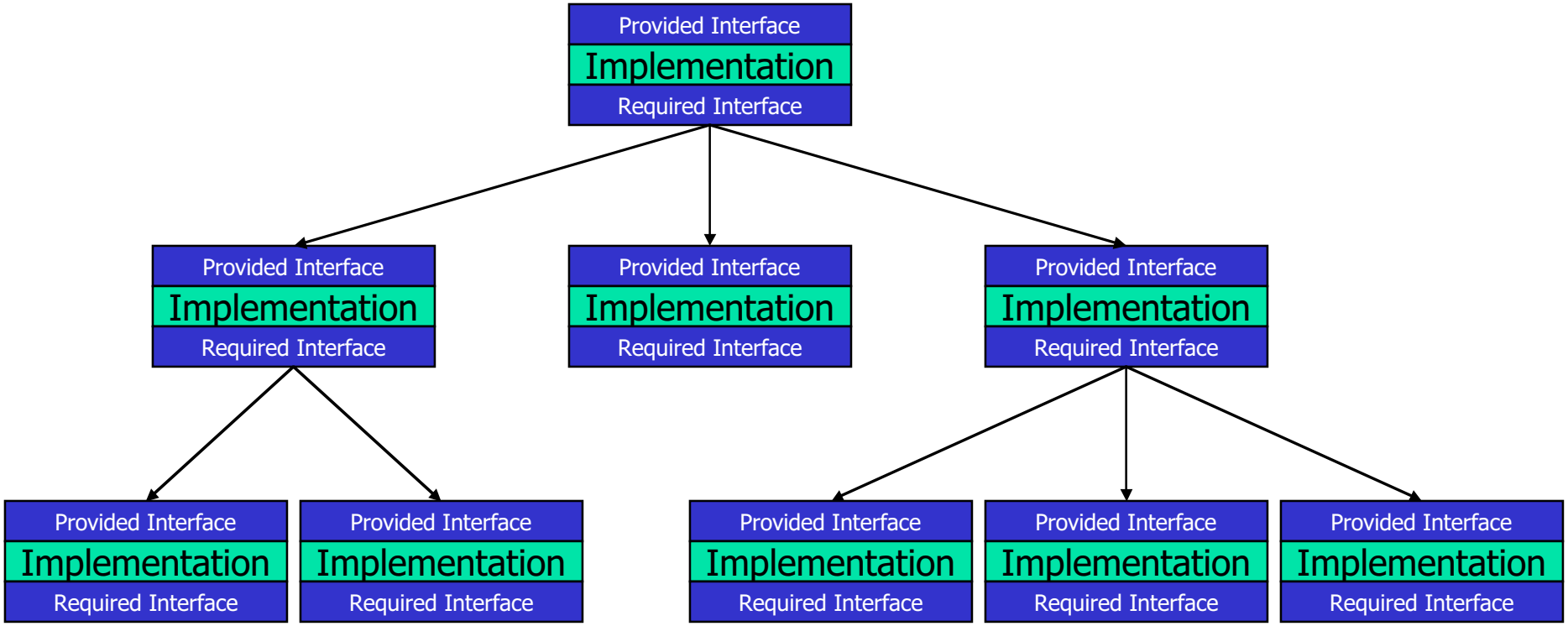
Coupling



VERSUS

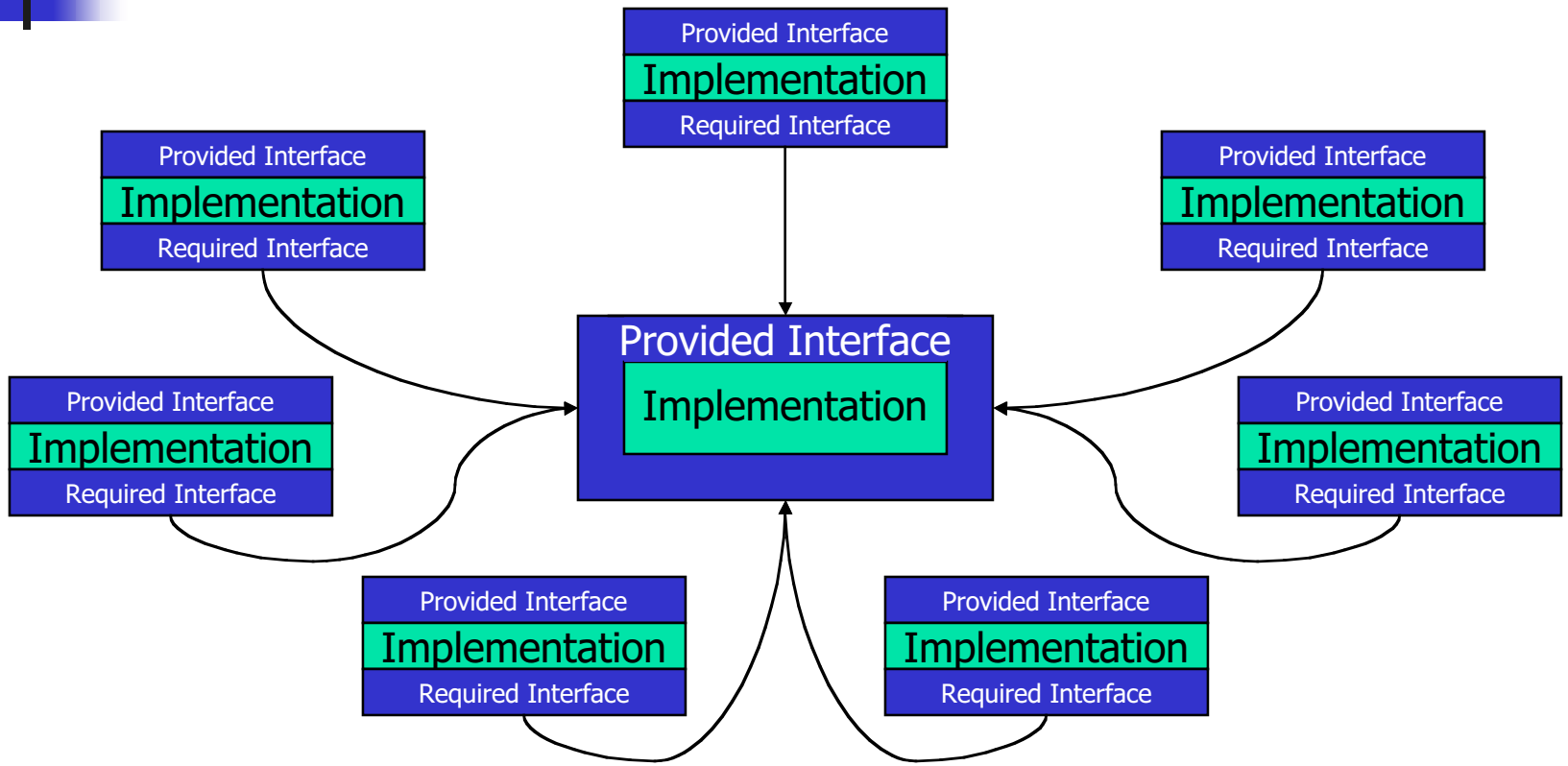


A Good Separation of Concerns, 1



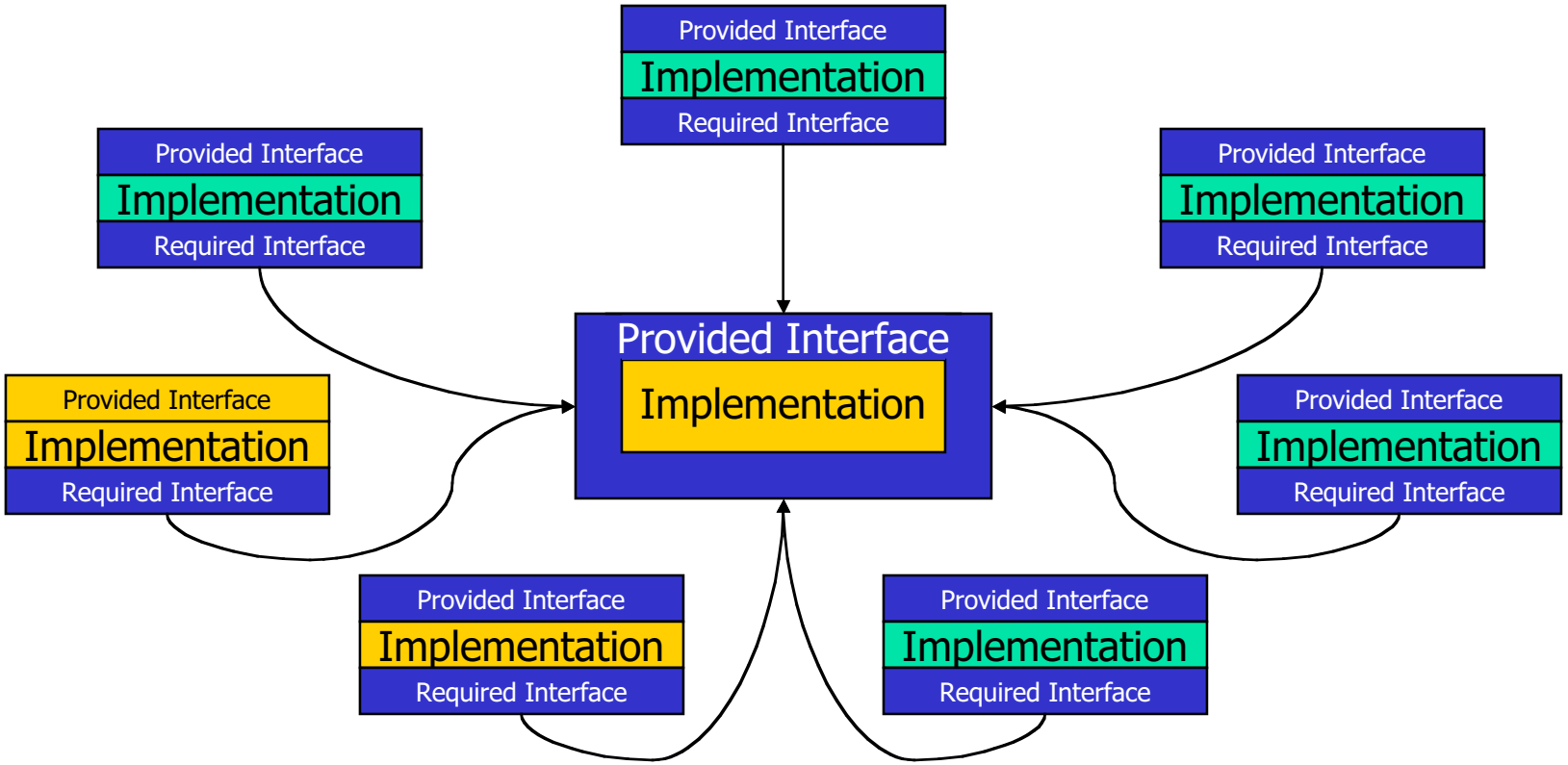
Abstraction through the use of provided/required interfaces
Modularity through the use of components
Low coupling through the use of hierarchies
High cohesion through the use of coherent implementations

A Good Separation of Concerns, 2



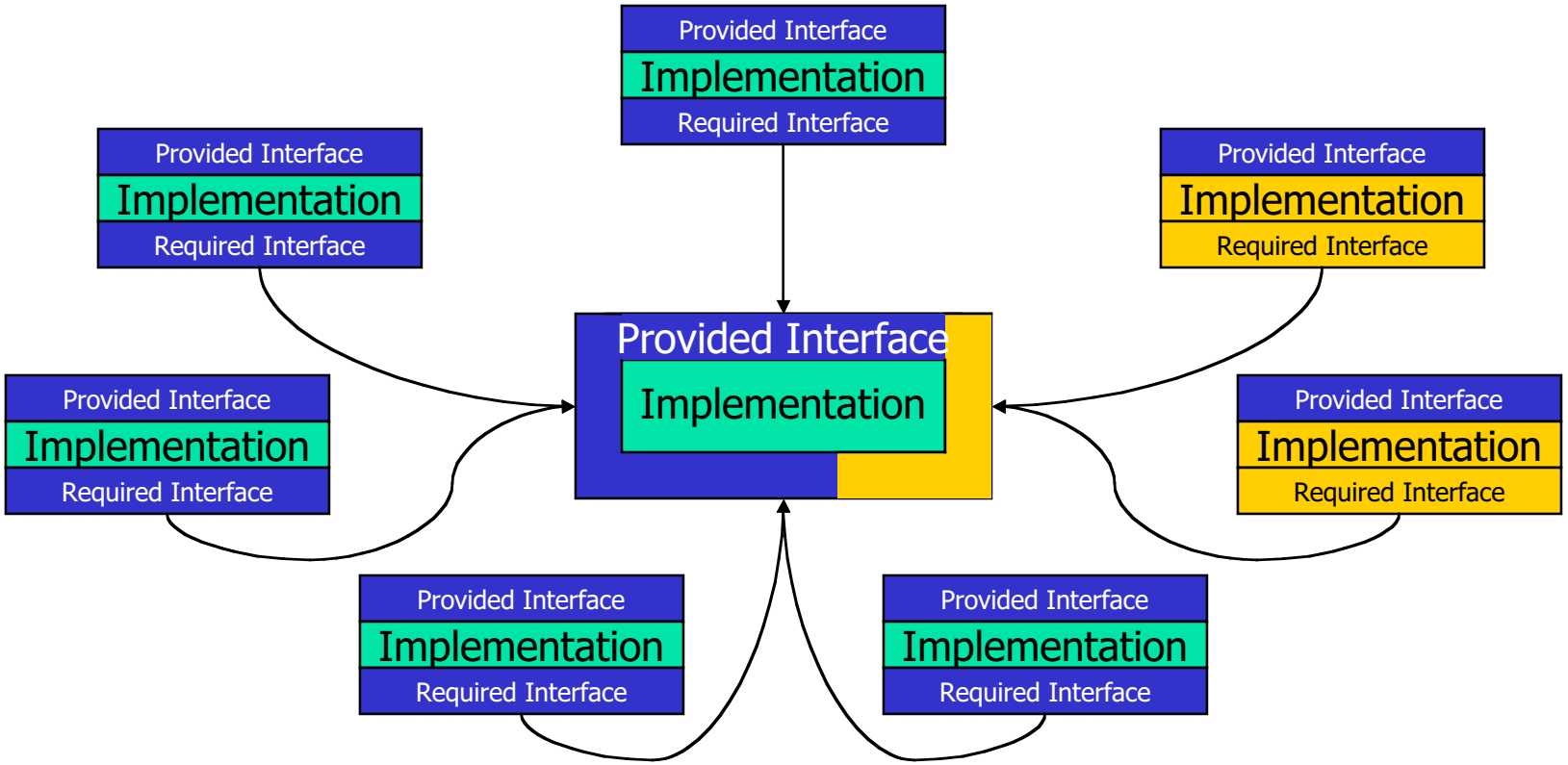
Abstraction through the use of provided/required interfaces
Modularity through the use of components
Low coupling through the use of a central "blackboard"
High cohesion through the use of coherent implementations

Benefit 1: Anticipating Change



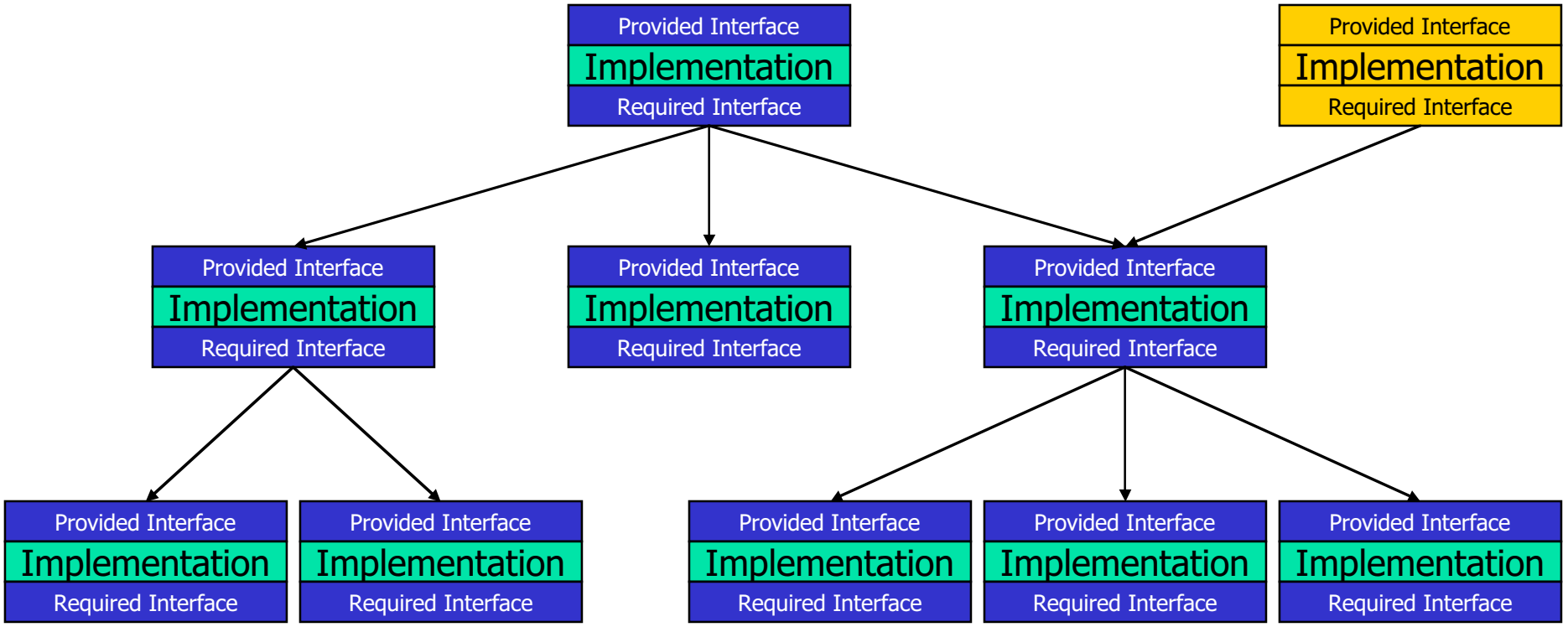
Separating concerns anticipates change

Benefit 1: Anticipating Change



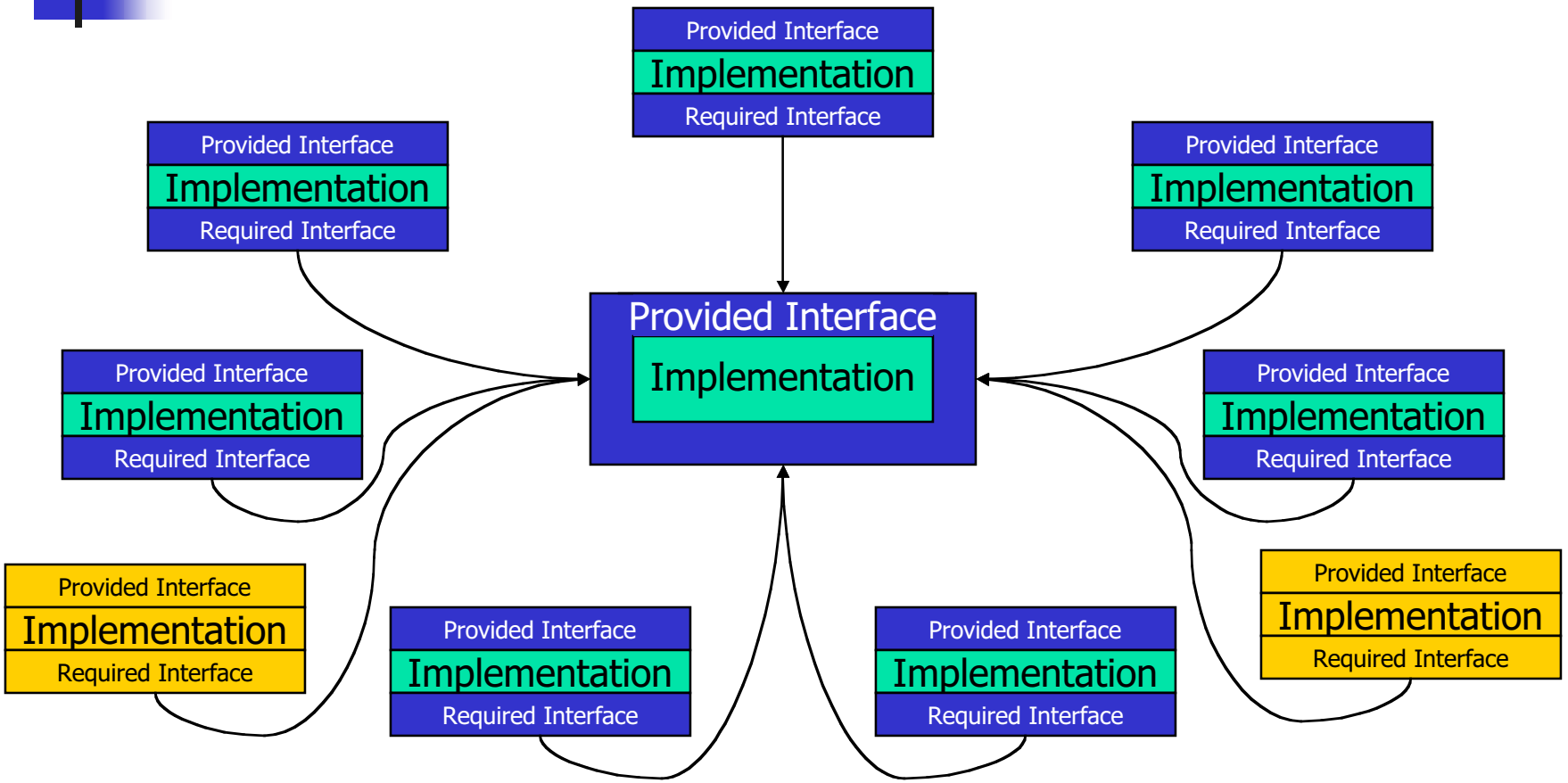
Separating concerns anticipates change

Benefit 2: Promoting Generality

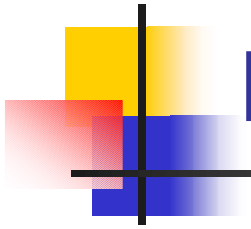


Separating concerns promotes generality

Benefit 3: Facilitating Incrementality



Separating concerns facilitates incrementality

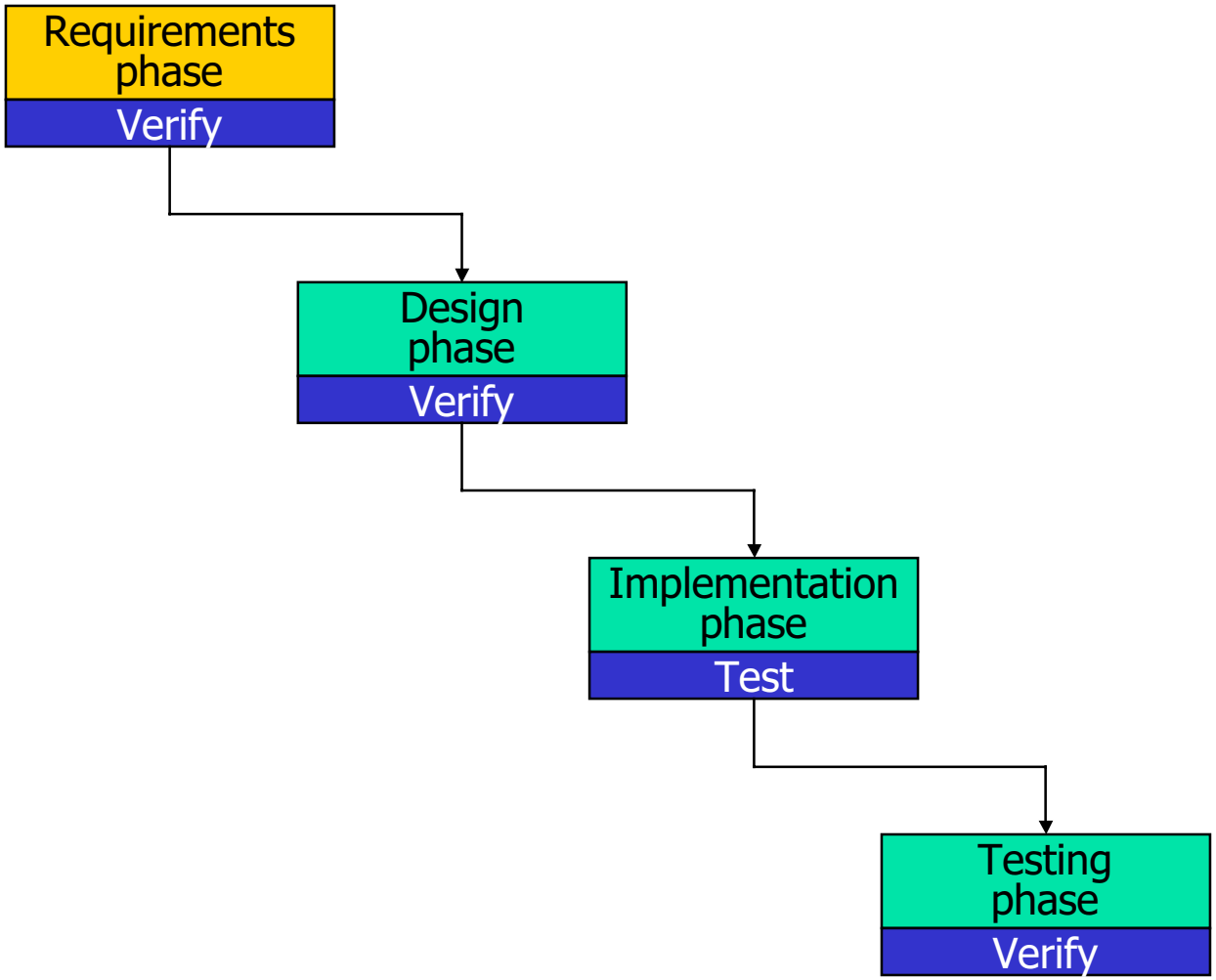


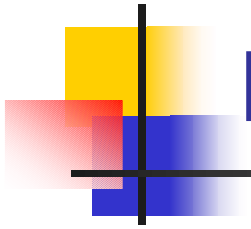
Recurring, Fundamental Principles

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

These principles apply to all aspects of software engineering

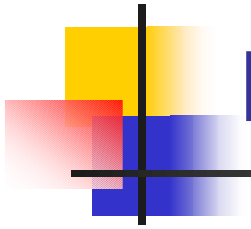
ICS 52 Life Cycle





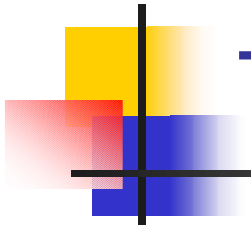
Requirements Phase

- Terminology
  - Requirements analysis/engineering
    - Activity of unearthing a customer's needs
  - Requirements specification
    - Document describing a customer's needs
- Note: requirements address what a customer needs, not what a customer wants
  - A customer often does not know what they want
  - Time-lag between initial desire and future need
  - Long and arduous, sometimes educational, process



Requirements Analysis

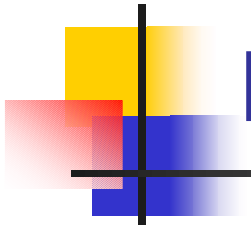
- System engineering versus software engineering
  - What role does software play within the full solution?
  - Trend: software is everywhere
    - Even in computer chips (TransMeta)
- Contract model versus participatory design
  - Contract: carefully specify requirements, then contract out the development
  - Participatory: customers, users, and software development staff work together throughout the life cycle



Techniques for Requirements Analysis

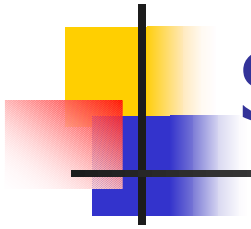
- Interview customer
- Create use cases/scenarios
- Prototype solutions
- Observe customer
- Identify important objects/roles/functions
- Perform research
- Construct glossaries
- Question yourself

Use the principles



Requirements Specification

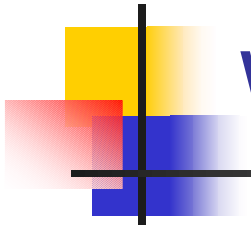
- Serves as the fundamental reference point between customer and software producer
- Defines capabilities to be provided without saying how they should be provided
  - Defines the "what"
  - Does not define the "how"
- Defines environmental requirements on the software to guide the implementers
  - Platforms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



Software Qualities

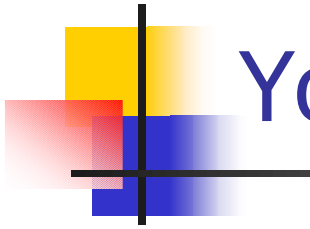
- Correctness
- Reliability
- Robustness
- Performance
- User friendliness
- Verifiability
- Maintainability
- Repairability
- Safety
- Evolvability
- Reusability
- Portability
- Understandability
- Interoperability
- Productivity
- Size
- Timeliness
- Visibility

These qualities often conflict with each other



Why Spend a Lot of Time?

- A requirements specification is *the* source for all future steps in the software life cycle
  - Lays the basis for a mutual understanding
    - Consumer (what they get)
    - Software producer (what they build)
  - Identifies fundamental assumptions
  - Potential basis for future contracts
- Better get it right
  - Upon delivery, some software is actually rejected by customers
- Changes are cheap
  - Better make them now rather than later



Your Tasks

- Read and study slides of this lecture
- Read and study Chapter 2 and Chapter 3 of Ghezzi, Jazayeri, and Mandrioli