ECE4574 – Large-Scale SW Development for Engineering Systems
Lecture 4 – Object-Oriented System Architecture and Design

Creed Jones, PhD

Course Updates

- Quiz 1 is TODAY! (7 PM to 1 AM Eastern)
- Homework 1 is due Friday, September 22

Topics for Today

- Software Design
  - Design Challenges
  - Key Design Concepts
  - Levels in a Software Design
  - Design Practices
  - Top Down and Bottom Up
  - Prototyping
- Software Architecture
  - Structure
  - Relationship to Non-functional requirements
  - Views of SW Architecture
  - Software Architect roles

SOFTWARE DESIGN

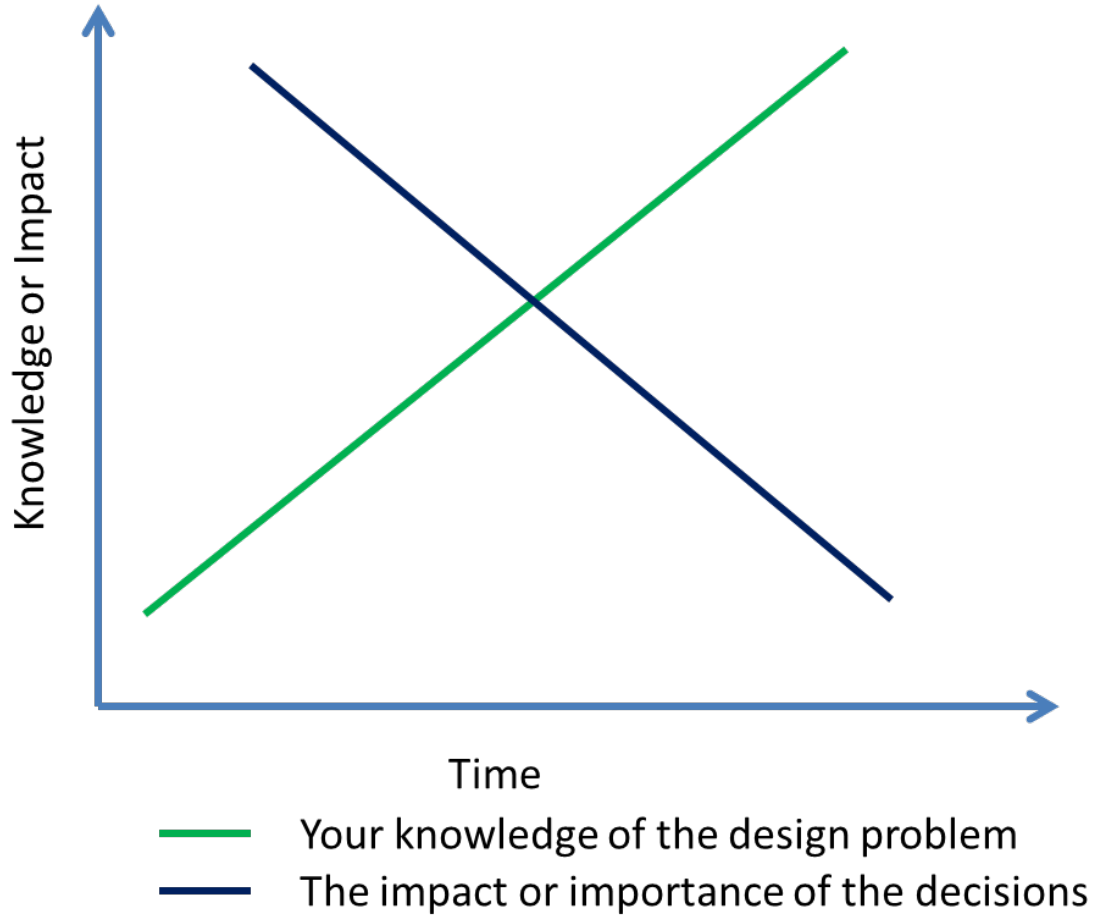
Design is a difficult, yet creative, process that is essential to do well if we are going to create good software

- Balancing tradeoffs and priorities
  - "Good, fast, cheap – pick any two"
- Nondeterministic
  - There is no formula
- Operating under constraints
  - timing, business rules, resources
- Emergent
  - we don't know all of the essential facts when we begin

Doing a good job of gathering and writing the requirements is hard but essential

- Decide on specific functional requirements
  - inputs, outputs, interfaces, tasks, data
  - timing, security, system resources
- Requirements must be usable and useful
  - Not too verbose
  - No gaps

As we design a system or process, we learn more about the specific problem – once we finally know enough, we are done



The key design concept in software is the managing of complexity

Software programs are among the most complex things ever designed by man

- Principles:

1. Avoid unnecessary complexity
2. Avoid single points of failure – both in the design and in the design team/process
3. Avoid excessive entanglement…

Desirable characteristics for a design

I. Minimal Complexity
   - as simple as possible to meet requirements

II. Ease of Maintenance

III. Loose Coupling
   - connections among different program parts are minimized

IV. Extensibility
   - enhancements don't require a major rework

V. Reusability
   - pieces are easy to use in other systems

VI. High "Fan-in"
   - a high number of classes make use of a given class

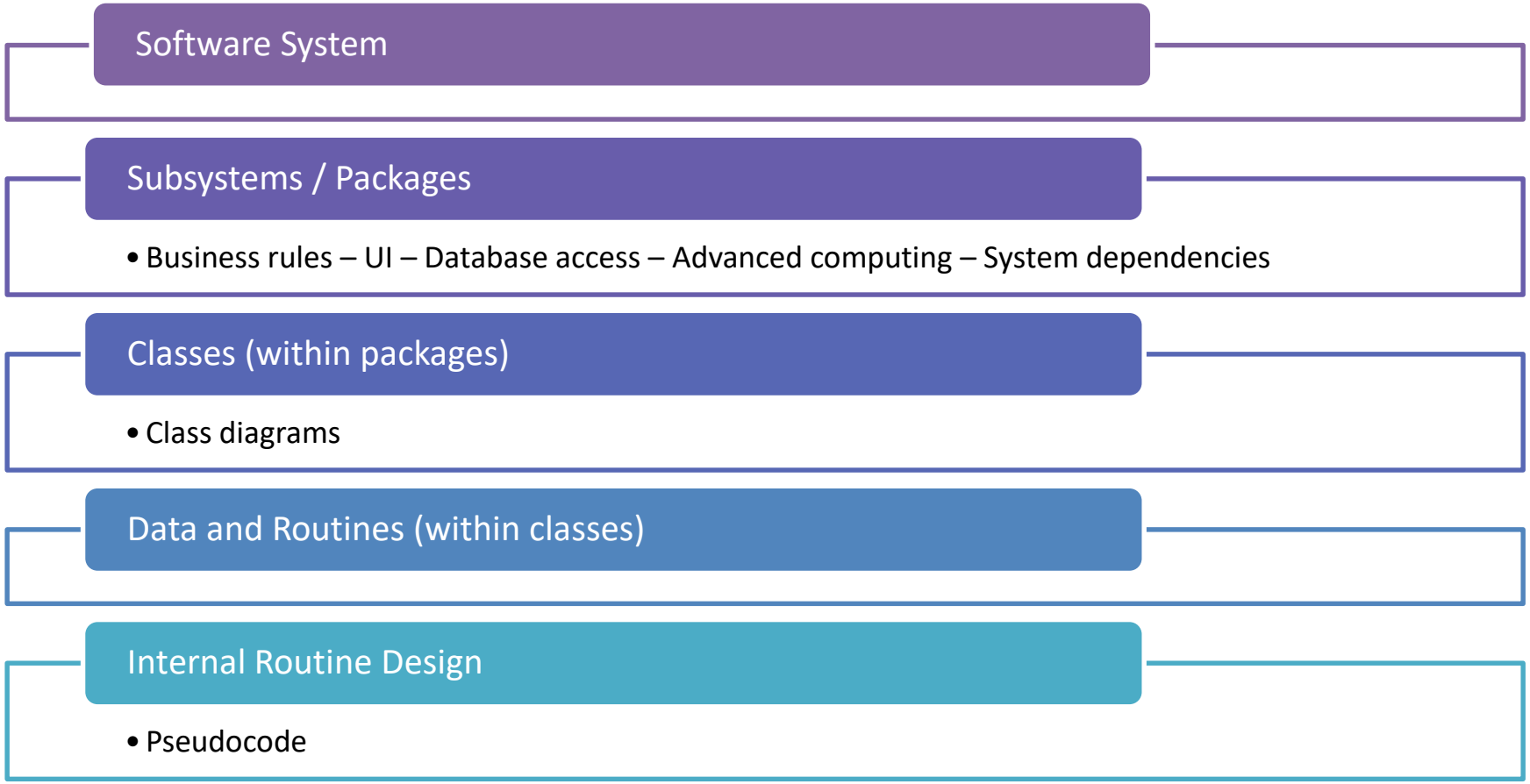
VII. Low to Medium "Fan-out"
   - a given class uses a modest number of other classes

VIII. Portability
   - easy to move to another platform

IX. Leanness
   - no superfluous parts

X. Stratification
   - design in levels of decomposition

A software design has a number of levels; keep the design separated into these levels



Design Practices

- Find Real-World Objects
- Form Consistent Abstractions
- Encapsulate Implementation Details
- Consider Inheritance – carefully
- Hide "Secrets" (complexity, algorithms, moving parts)
- Identify Areas Likely to Change
- Keep Couplings Loose (fewer and farther between)
- Look for Common Design Patterns

I believe that *decomposition* or dividing a Problem into Pieces is the Key Engineering Method

- Find the natural "seams" in the system and split into subsystems along those lines

- Determine interaction between the subsystems (maybe not in ultimate detail…)
- Then, design each subsystem

Top-Down or Bottom-Up?

- Do we start with the grand overall design, identify subsystems, then classes, and so on?
  - Set the bounds of the rest of the design
  - Consider all key factors
- Or do we start with the specific things that need to be done and work up, defining a hierarchy?
  - Allows us to get started more readily
  - Explore areas of greatest technical risk

- Iterate (there is room for TD and BU)

Prototyping (or "Experimental Design") can help us address areas of risk

- When we write a problem statement, it's a great time to think of risky areas of our design
  - Connecting to new systems or interfaces
  - External components being used for the first time
  - Hardware interfaces
  - Complex calculations
  - Stressing system resources in any way
  - Throughput or latency requirements

- Do some simple code to explore the proposed design
- Consider addressing tough requirements in the first sprint

VIRGINIA TECH

BRADLEY DEPARTMENT OF ELECTRICAL & COMPUTER ENGINEERING

SOFTWARE ARCHITECTURE

Software Architecture has to do with the entire system, thinking in large pieces

- Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

  - *IEEE*

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

  - *Bass and Clements, 2003*

Architecture Defines Structure

- Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

  - *IEEE*

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

  - *Bass and Clements, 2003*

Components are distinct software subsystems (one or more classes, libraries, etc…)

- Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

  - *IEEE*

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

  - *Bass and Clements, 2003*

Key point: Choosing an Architecture for a problem determines the relationships (collaborations) among the components

- Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

  - *IEEE*

- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.

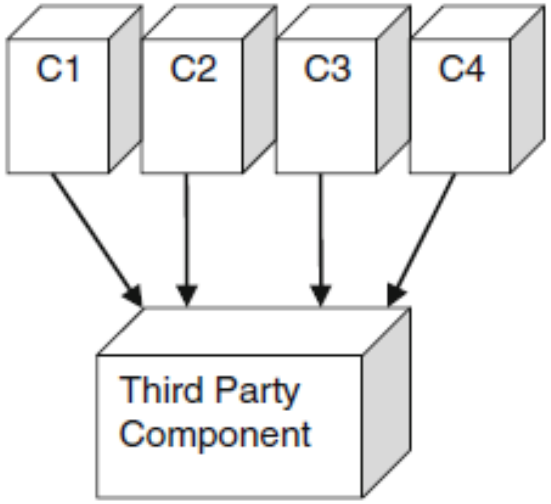
  - *Bass and Clements, 2003*

The Architecture also determines how the system talks to its surroundings (user, network, etc.)

- Architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.
  - *IEEE*

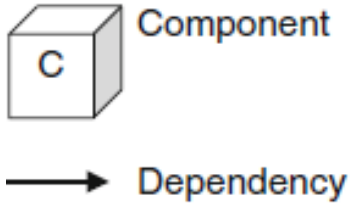
- The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them.
  - *Bass and Clements, 2003*

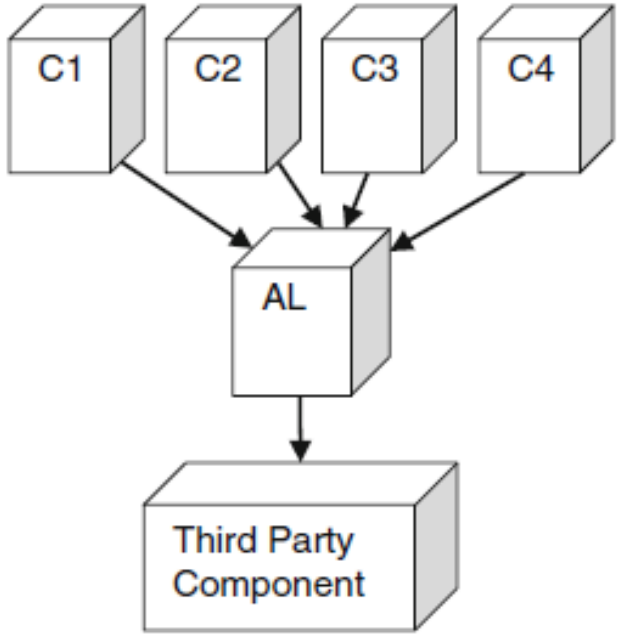
Architectural choices can determine adherence to good SW practices like, in this case, loose coupling (low dependency)



Four components are directly dependent on a third party component. If the third party component is replaced with a new component with a different interface, changes to each component are likely.

Diagram Key

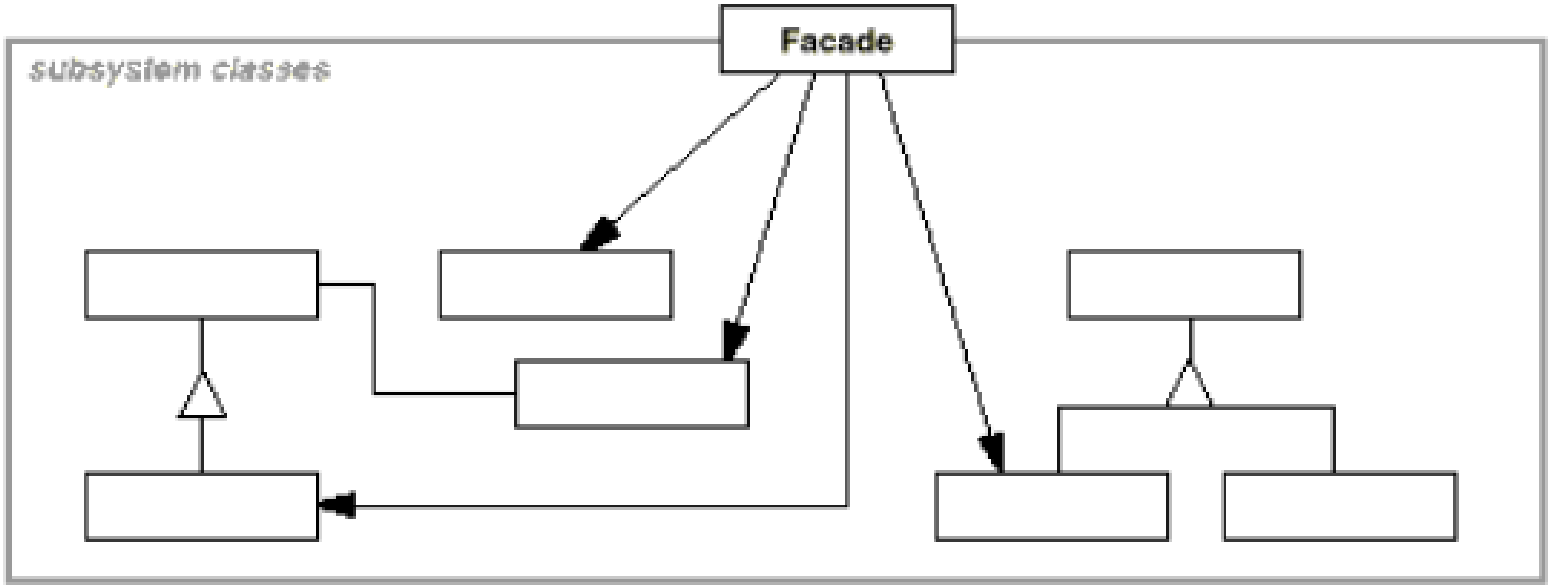




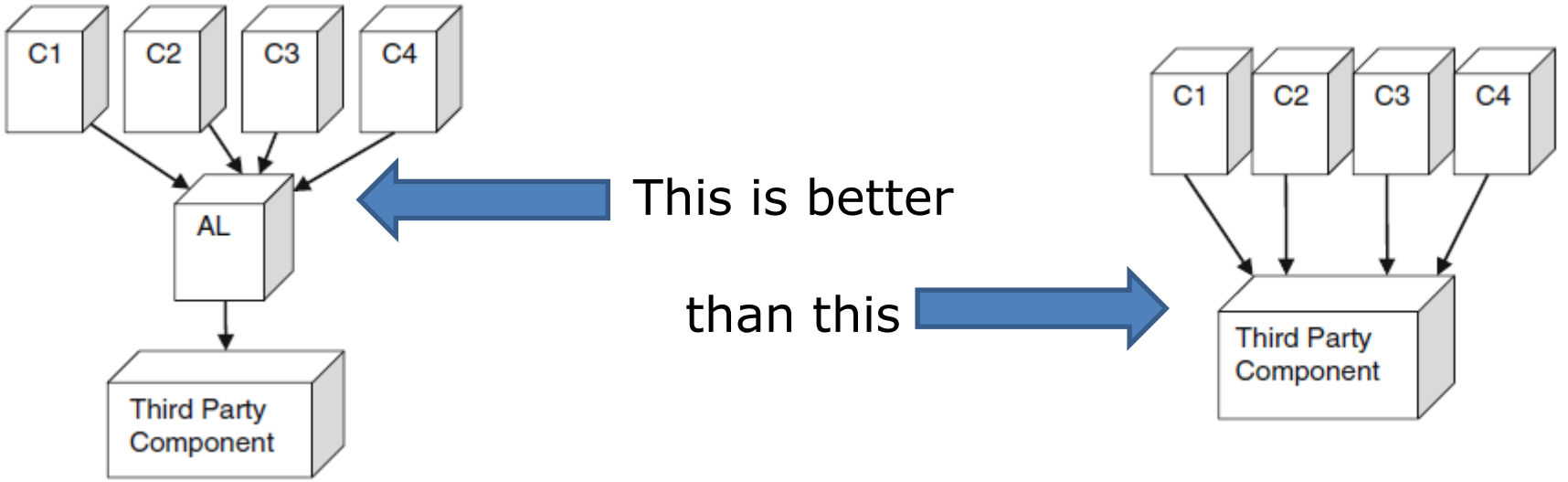
Only the AL (abstraction layer) component is directly dependent on the third party component. If the third party component is replaced, changes are restricted to the AL component only

This can be accomplished by use of a consistent approach called the *Facade design pattern* (more on design patterns in a few weeks)

- Facade implements an abstraction layer



Another key point: Architectural decisions are not just personal taste; some are better than others



- It depends on the context (the requirements, existing code, system considerations, etc)
- Some architectures are better at times, worse at other times
- Some are always terrible

SW Non-functional requirements are those that don't appear in test cases; not *what* it does, but conditions on *how* it does it

Three major kinds of non-functional requirements:

- Technical constraints:
  - "We will be developing in Java"
  - "The system has 8GB of RAM"
  - these are usually non-negotiable
- Business constraints
  - "We must interface with MATLAB"
  - "We must use open-source code for the backend"
  - also usually non-negotiable
- Quality attributes
  - Portability
  - Scalability
  - Performance
  - …

The Architecture largely determines our compliance with Non-functional requirements

Three major kinds of non-functional requirements:

- Technical constraints:
  - "We will be developing in Java"
  - "The system has 8GB of RAM"
  - these are usually non-negotiable

Component Selection

- Business constraints
  - "We must interface with MATLAB"
  - "We must use open-source code for the backend"
  - also usually non-negotiable

External Interface

- Quality attributes
  - Portability
  - Scalability
  - Performance
  - …

Component Selection

Interrelationships between components

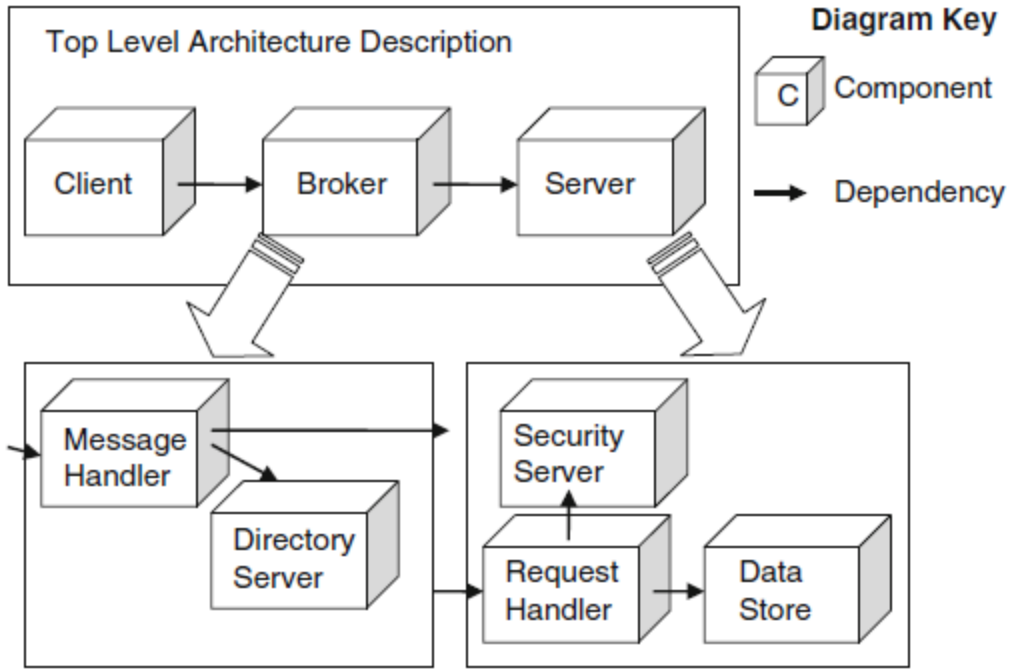
Architecture should start with a careful look at the data to be used and the key objects in the system

- Identify the objects in play
  - Sometimes they are real-world objects (players, books)
  - Sometimes they are data objects (packets, etc.)
  - Sometimes more esoteric

- How is data communicated and stored?
- What initiates actions?
- What existing code should (can) we reuse?
- What external components might we use?

An example of a two-layer hierarchical decomposition of a SW architecture

- Usually we design the top level first
- Then each block becomes a design at the next level



There are four common "views" in which to describe an architecture: Logical, Process, Physical and Development

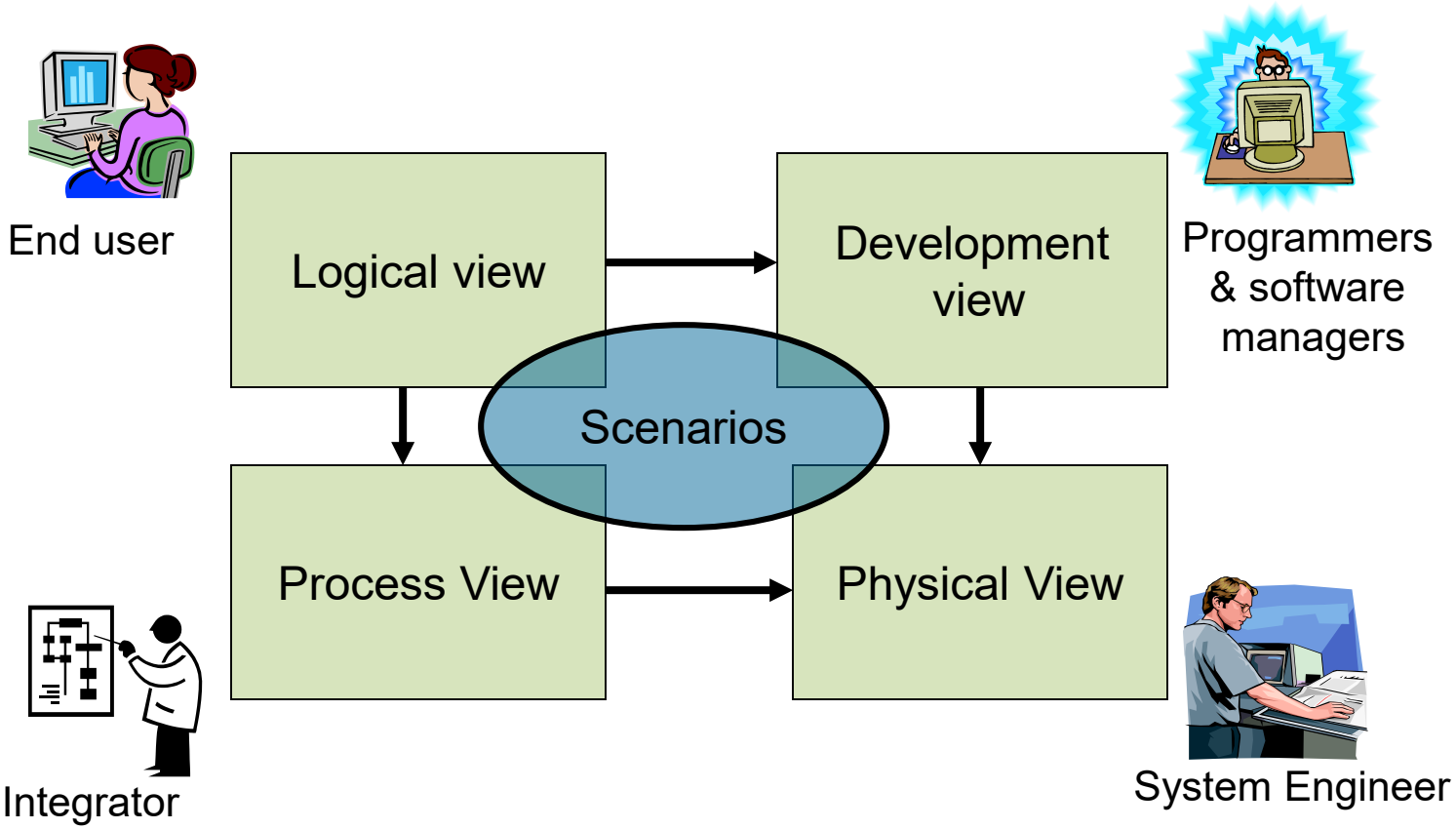
- Logical View is like a block diagram of the components
- Process View shows the order of processing
- Physical View is concerned with how the components execute on the hardware
- Development View is the description of project and solutions, or packages, that the system SW is developed and maintained in
- This is called the 4+1 Architectural View model

There are four common "views" in which to describe an architecture: Logical, Process, Physical and Development

- Logical View is like a block diagram of the components
  class, object, component, package and use case diagrams
- Process View shows the order of processing
  interaction, sequence, activity and state diagrams
- Physical View is concerned with how the components execute on the hardware
  deployment and timing diagrams
- Development View is the description of project and solutions, or packages, that the system SW is developed and maintained in
  somewhat in package and deployment diagrams

- This is called the 4+1 Architectural View model

The "4+1" model portrays how different roles can see the same SW in very different ways



Another useful set of ways to look at architecture is called the "Views and Beyond" approach

Three Views:

- Module: the structural layout of the components – but with a little more detail (classes, inheritance, etc)
- Component and Connector: identifies components that need to talk and the mechanisms for doing so (CORBA, sockets, etc.)
- Allocation: Communications, hardware and development responsibilities.

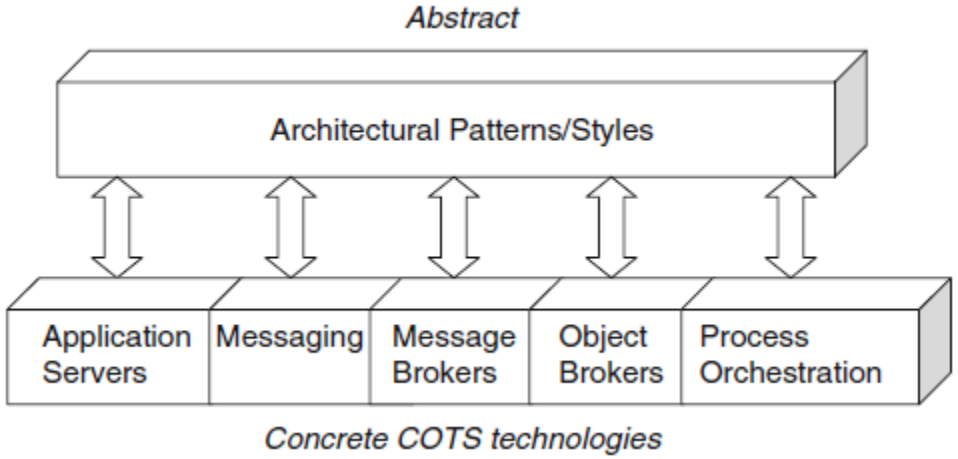
Another useful set of ways to look at architecture is called the "Views and Beyond" approach

Three Views:

- Module: the structural layout of the components – but with a little more detail (classes, inheritance, etc)
  We will call this the "Structural View"
- Component and Connector: identifies components that need to talk and the mechanisms for doing so (CORBA, sockets, etc.)
  We will call this the "Behavioral View"
- Allocation: Communications, hardware and development responsibilities.

There are many tested SW architectures, design patterns and SW packages that implement or facilitate architectures, available to us

- Many are Commercial Off-the-Shelf (COTS) components
  - These days, this term is also used to include open-source



- Our challenge is to understand these and make key decisions early on

In some organizations, Architects are specific people – in others, developers do a little or a lot of the architecture work

- **Chief Architect:** Typically a senior position who manages a team of architects within an organization. Operates at a broad, often organizational level, and coordinates efforts across system, applications, and product lines. Very experienced, with a rare combination of deep technical and business knowledge.

- **Product/Technical/Solution Architect:** Typically someone who has progressed through the technical ranks and oversees the architectural design for a specific system or application. They have a deep knowledge of how some important piece of software really works.

- **Enterprise Architect:** Typically a much less technical, more business-focus role. Enterprise architects use various business methods and tools to understand, document, and plan the structure of the major systems in an enterprise.

Topics for Today

- Software Design
  - Design Challenges
  - Key Design Concepts
  - Levels in a Software Design
  - Design Practices
  - Top Down and Bottom Up
  - Prototyping
- Software Architecture
  - Structure
  - Relationship to Non-functional requirements
  - Views of SW Architecture
  - Software Architect roles