



ILLINOIS INSTITUTE  
OF TECHNOLOGY

*Transforming Lives. Inventing the Future.*  
[www.iit.edu](http://www.iit.edu)

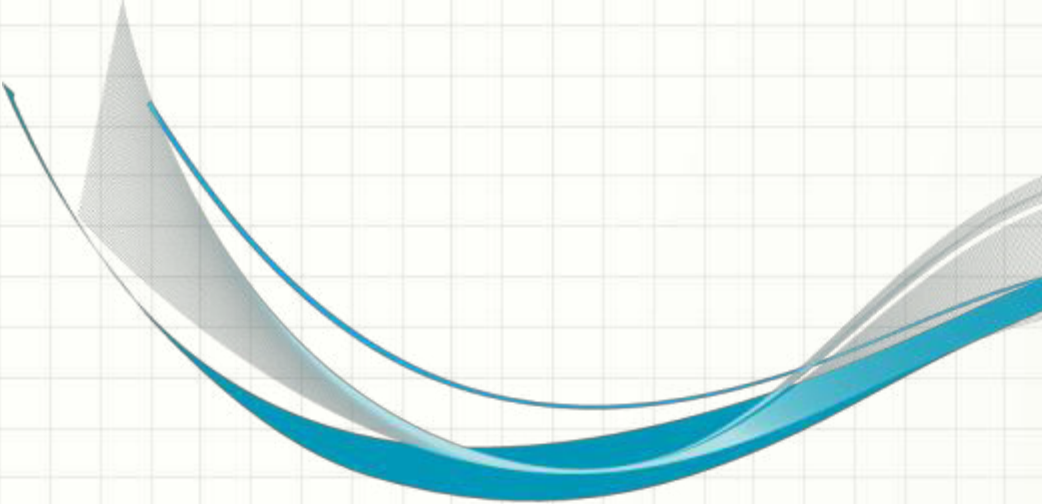
# SOFTWARE ENGINEERING

## CS 487

Dennis Hood  
Computer Science  
Summer '19

# Lesson Overview

- Specification: Automation and Reuse
- Reading:
  - Case Study – [Ariane 5 Failure](#)
  - Ch. 15 – Software Reuse
  - Ch. 16 – Component-based Software Engineering
  - Ch. 17 – Distributed Software Engineering
- Objectives
  - Revisit agility/flexibility in developing software as well as the benefits and challenges associated with such approaches
  - Examine reuse in the software development process in more detail using the Ariane 5 case to better understand associated risks and ways to mitigate them



# Lecture 8

## Automation and Reuse

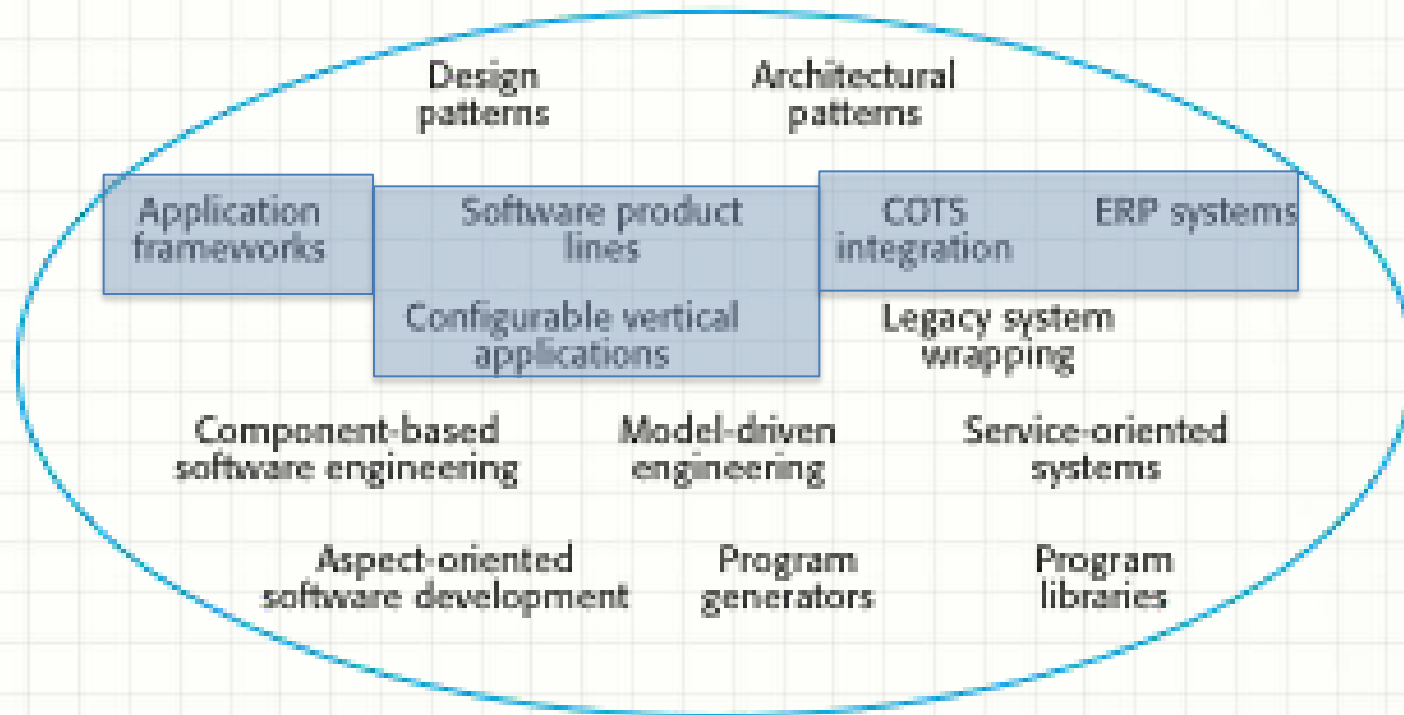
# Case Study: Ariane 5 Launcher

- Reuse plan
  - Inertial reference software performed successfully in the Ariane 4, so it was reused in the Ariane 5
  - The code contained “extras”
  - The Ariane 5 had more powerful engines
- Failure
  - Code (*that was not required*) attempted to convert a fixed-point number to an integer
  - Ariane 4 had never generated such a large number
  - The code generated an exception and shut down
  - The code was never tested since it wasn't required for Ariane 5

# Reuse

- Benefits
  - Speed
  - Peace of mind
  - Reliability
  - Reduced risk
  - Ease of maintenance
- Challenges
  - Modules must be somewhat generic
  - Anticipation (of future need) is key
- Drawbacks
  - “One size fits all” may not be best
  - Requires infrastructure (library, etc.) and process (check-in/check-out, etc.)

# Reuse Opportunities





# Other Reuse Approaches

- Generator-based reuse
  - CASE tool support to recognize opportunities for reuse
- COTS products
- Application frameworks
  - Objects may be too specific to be an effective abstraction
  - A collection of classes and the interfaces between them
  - Example frameworks
    - System infrastructure
    - Middleware integration
    - Enterprise application

# Benefits of Rapid Iteration

- Reduce the opportunity for change
  - Customer needs evolve over time
  - Business/competitive demands
  - Technology evolves rapidly
- Focus on what is known
  - Complete understanding is difficult to achieve until we make significant progress
  - Implement what is understood while learning about what should come next
- Everyone wants the system built quickly
  - Better opportunity to involve the user
  - Less business risk for everyone



# Challenges of Rapid Iteration

- Difficult to maintain discipline
  - The perception is that formal processes take longer
  - Pressure to cut corners
  - Supported by the feeling that we can skip it now and catch it on a future iteration
- Overly narrow focus
  - Each iteration necessarily focuses on the small, but planning is required to have the collection of iterations result in a “big picture”
  - New tools, approaches, skills, etc. may be called for, but who has time for that?!?

# Incremental vs. Prototyping

- Incremental development consists of a series of planned (relatively small) efforts designed to result in a complete system to user specification
- Prototyping can be used to facilitate incremental development by incrementally improving the prototype into the finished system
- Throw-away prototyping on the other hand is used to produce communication vehicles
  - Akin to R&D
  - Can be “quick and dirty”

# Agile Methods

- Principles
  - User involvement
  - Incremental delivery
  - Exploit developers' skills (over process)
  - Embrace change
  - Keep it simple
- Challenges
  - Users can be difficult to involve effectively
  - The atmosphere can be pretty intense
  - Teamwork (cooperation) is key
  - Change is hard
  - Inherently difficult to manage

# Agile Approaches

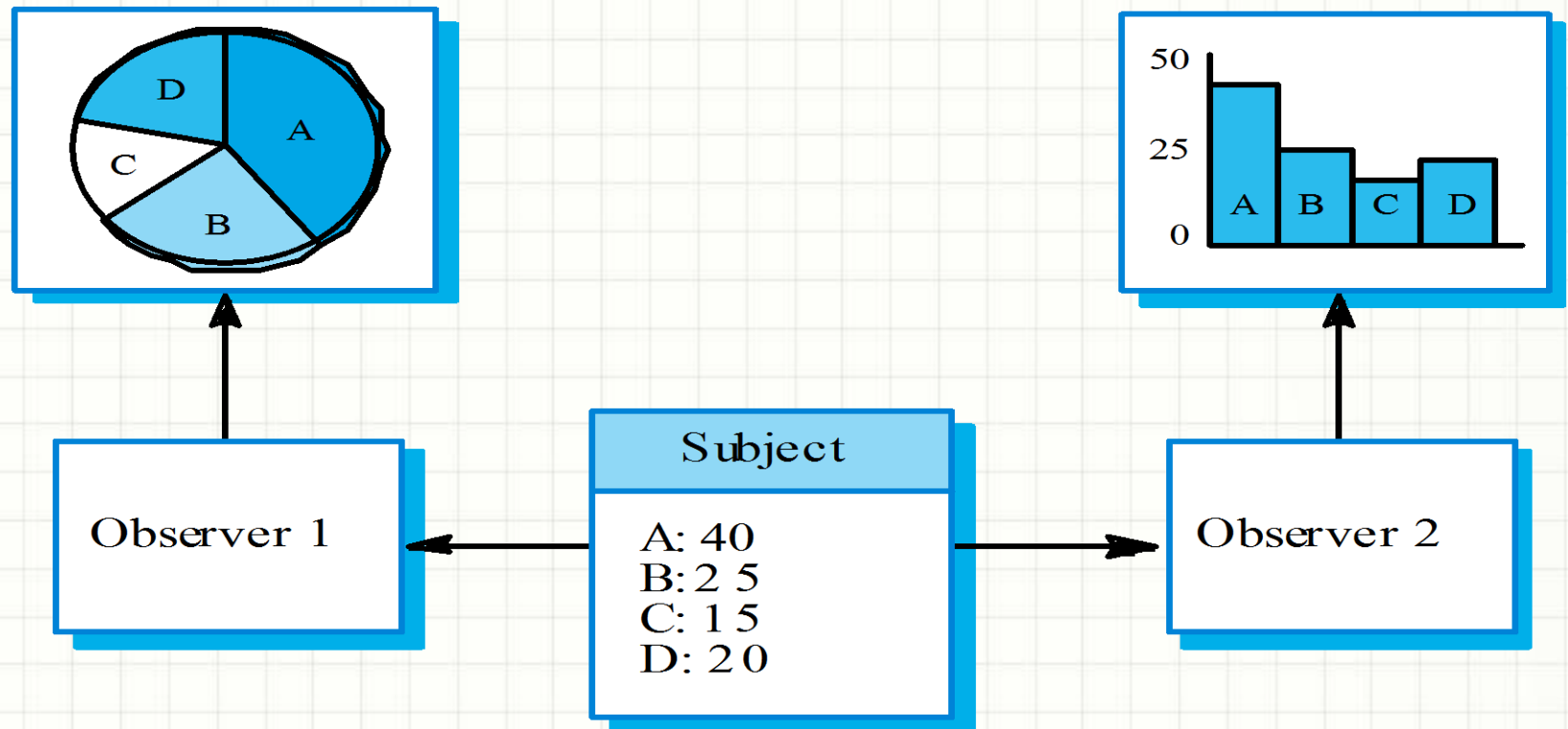
- Extreme Programming (XP)
  - Pair development
- Rapid Application Development (RAD)
- Prototyping

# Design Patterns

- Borrowed from the world of architecture
- Common solutions for common problems
- Making it happen
  - Meaningful names
  - Describe the common problem clearly
  - Describe the common solution and how to apply it
  - State the consequences

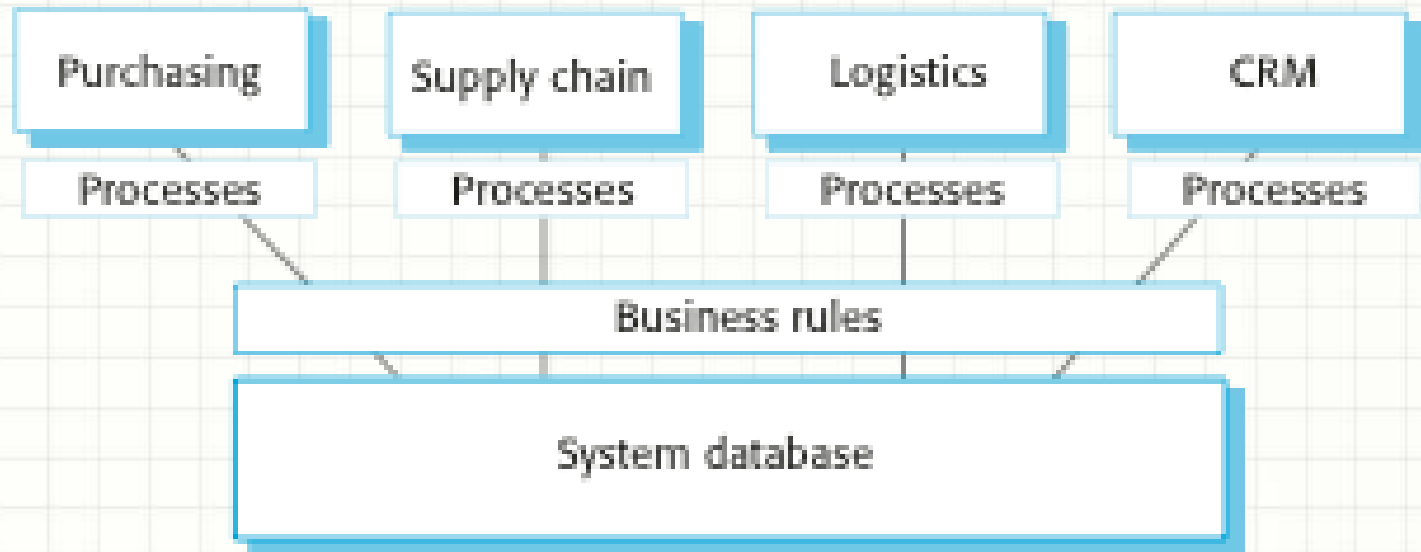
# Ex.: Model-View-Controller

- Supports multiple presentations of an object and separate means of interaction with each presentation





# ERP System Architecture



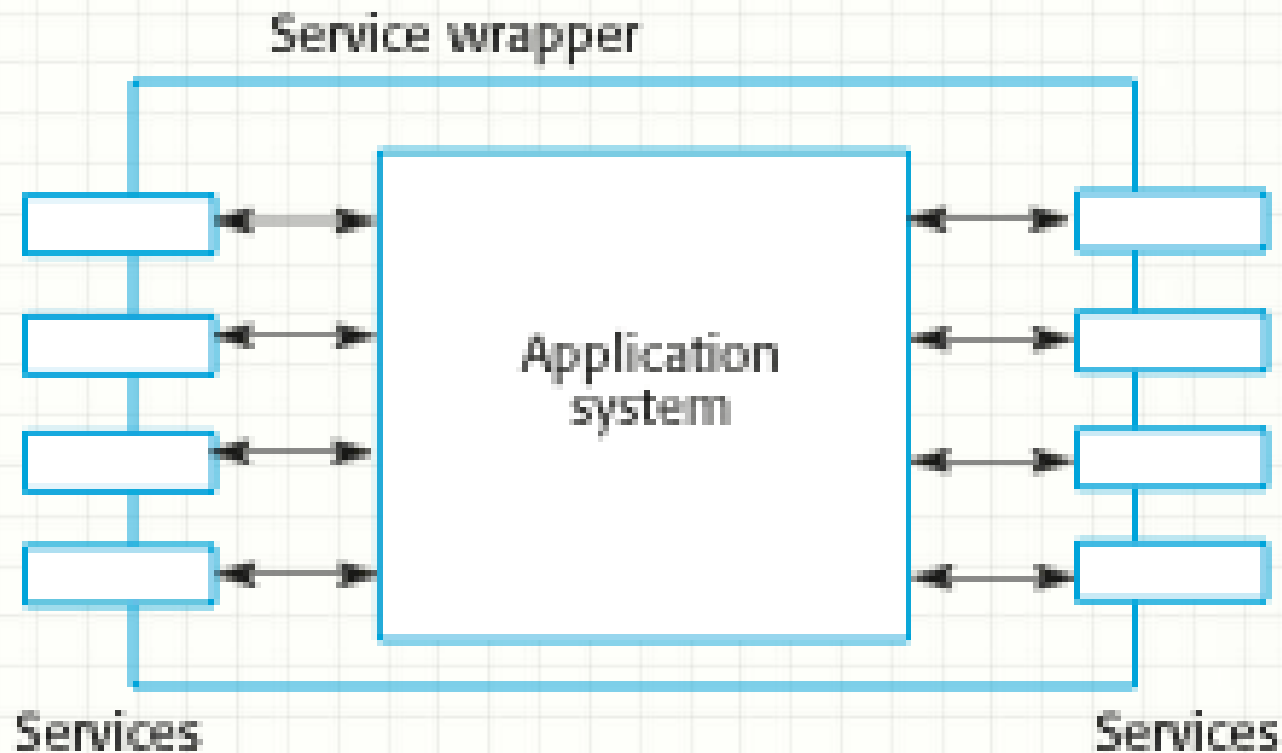
# ERP Architecture

- A number of modules to support different business functions.
- A defined set of business processes, associated with each module, which relate to activities in that module.
- A common database that maintains information about all related business functions.
- A set of business rules that apply to all data in the database.

# ERP Configuration

- Selecting the required functionality from the system.
- Establishing a data model that defines how the organization's data will be structured in the system database.
- Defining business rules that apply to that data.
- Defining the expected interactions with external systems.
- Designing the input forms and the output reports generated by the system.
- Designing new business processes that conform to the underlying process model supported by the system.
- Setting parameters that define how the system is deployed on its underlying platform.

# Application Wrapper



# Component-based SW Engineering

- By components we are referring to entities that are larger than objects but smaller than applications
  - Note that reuse is possible at all 3 of these levels and in truth at any level of granularity
  - Being this open-minded should help us realize better opportunities for reuse
  - In other words, sometimes objects are too small and specific, sometimes applications are too large and broad, and maybe sometimes components are just right (for marketing as reusable entities)

# Reusable Components

- Just as with the other levels, discipline is required to achieve reusability with components
  - The component should conform to a standardized model which enforces interfaces, documentation, deployment, etc.
  - A reusable component should be able to exist independent of other components
  - Public access (methods, knowledge of attributes, etc.) must be available but also strictly controlled (defined interfaces)
  - Deployable as a standalone entity
  - Fully documented including syntax and semantics
- Components are defined by their interfaces
  - Provides services to its clients
  - Requires services from its environment



# Component Models

- Standards for implementation, documentation, and deployment to ensure interoperability
  - CORBA, Java Beans, COM+
- Elements of the model
  - Interfaces
    - Operation names, parameters, exceptions, etc.
    - The appropriate interface definition language (IDL)
  - Information
    - Naming convention
    - Metadata (data about the component itself)
    - How to customize (configure) for a given deployment
  - Deployment
    - How to package the component for deployment

# Component Interfaces



# Component-based Reuse Challenges

- Complexity
- Trust
- Tight coupling with specific applications
  - As opposed to more stable business objects
- Maintainability
- Customization costs
- Inconsistency