



ILLINOIS INSTITUTE
OF TECHNOLOGY

Transforming Lives. Inventing the Future.

www.iit.edu

SOFTWARE ENGINEERING

CS 487

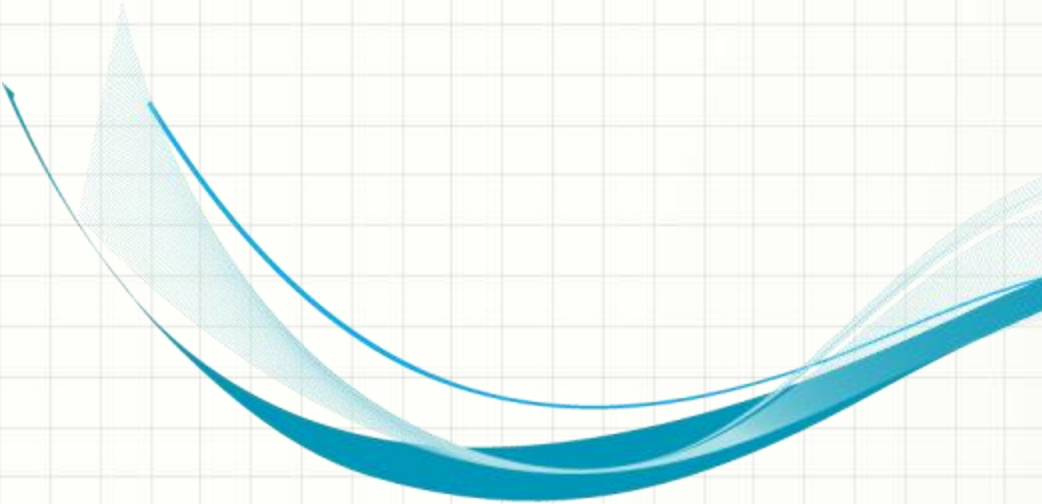
Prof. Dennis Hood
Computer Science

Lesson Overview

- Automation and Reuse
- Reading
 - Ch. 13 – Security Engineering
 - Ch. 14 – Resilience Engineering
 - Ch. 15 – Software Reuse
 - Ch. 16 – Component-based Software Engineering
 - [Ariane 5 test flight explosion](#) (case study)
- Objectives
 - Examine automation – both as the goal of software development and a means of improving the software development process
 - Highlight the need for flexibility in the development approach to support a “learn-as-you-grow” process of software development
 - Present challenges associated with an agile methodology as well as approaches for mitigating the risks
 - Examine reuse both in general as well as in the context of supporting agile methods

Participation – P6

- W.r.t the Ariane 5 case:
 - Use pseudocode to show the flow of the automated navigation process at issue in the case.
 - Use risk assessment to justify the decision to reuse the Ariane 4 navigation system.
 - Use pseudocode to describe an exception manager which would have helped avoid the explosive result.
 - Describe a set of test cases which would have identified the design flaw during reactive testing.
 - Describe a specific proactive activity which would have prevented the explosive result.



Lecture 7

Automation and Reuse

Automation

- Computers doing what humans used to do
- The benefits of automation are clear
 - Speed: computers can outperform humans
 - Reliability: computers always follow the rules
 - Robustness: computers never get tired, bored, distracted, frustrated, etc.
 - Cost: computers don't get paid – per hour, per transaction
- Software engineers create a system which follows that steps of the given process
 - Easier if those steps are well-defined
 - And simple to understand
 - And not likely to change (significantly / frequently)

Reuse

- Taking a previously produced artifact of the software development process and using it in future development efforts
- Better: creating artifacts which are measurably reusable and can therefore be used in future development efforts with minimal modification
- The benefits of reuse are clear:
- Software engineers create artifacts such that they can be reused in the future
 - Easier if the artifact addresses a well-defined problem
 - And is self-contained (i.e., modular)
 - Most effective if the artifact is likely to be needed again
 - And is easy to “find” and has a well-defined interface

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 generate 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

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

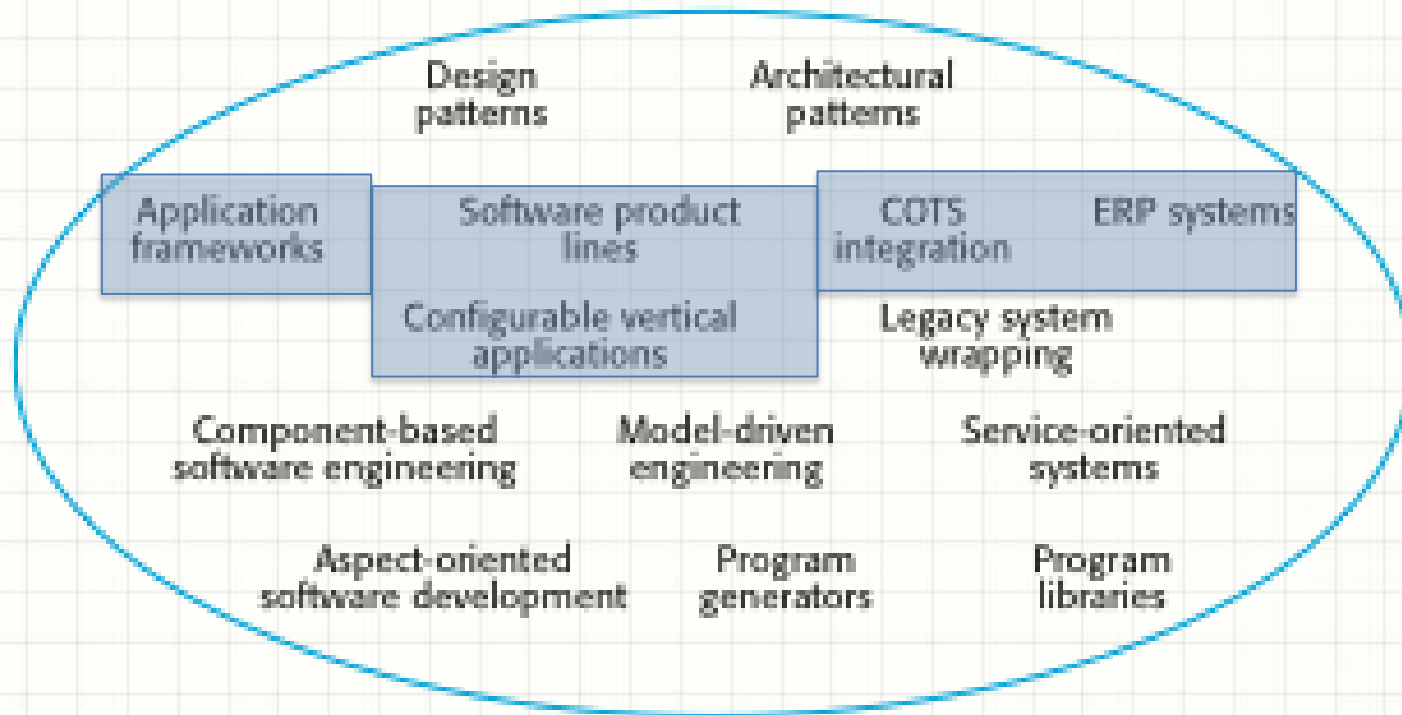
Agile Approaches

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

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.)

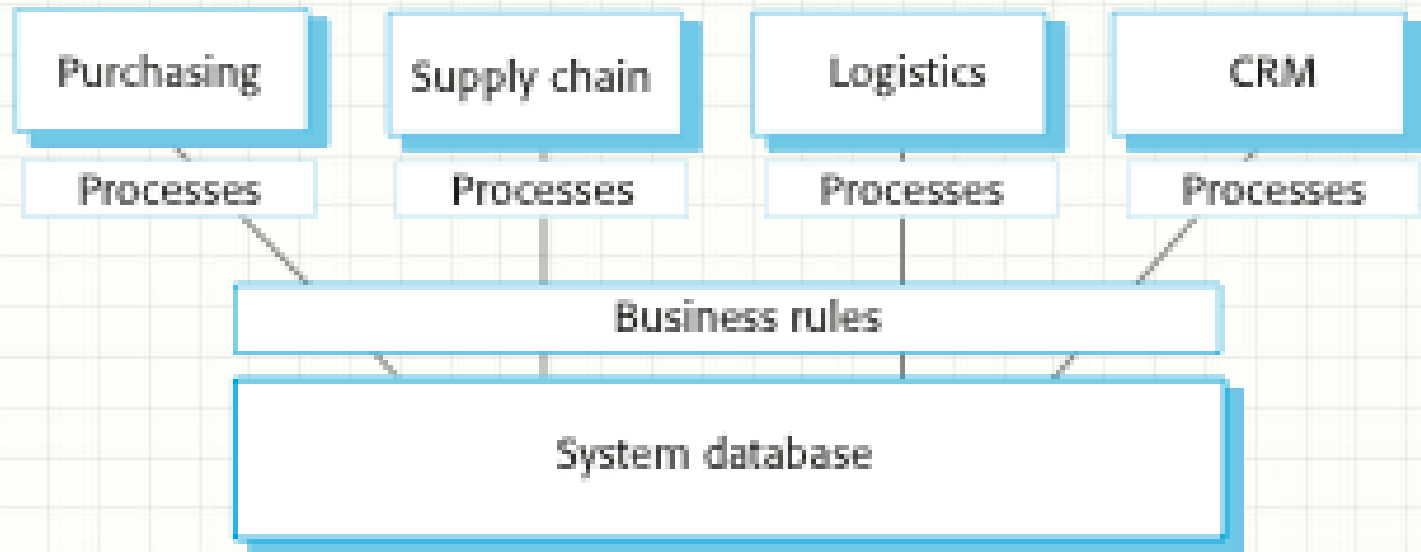
Reuse Opportunities



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

ERP System Architecture



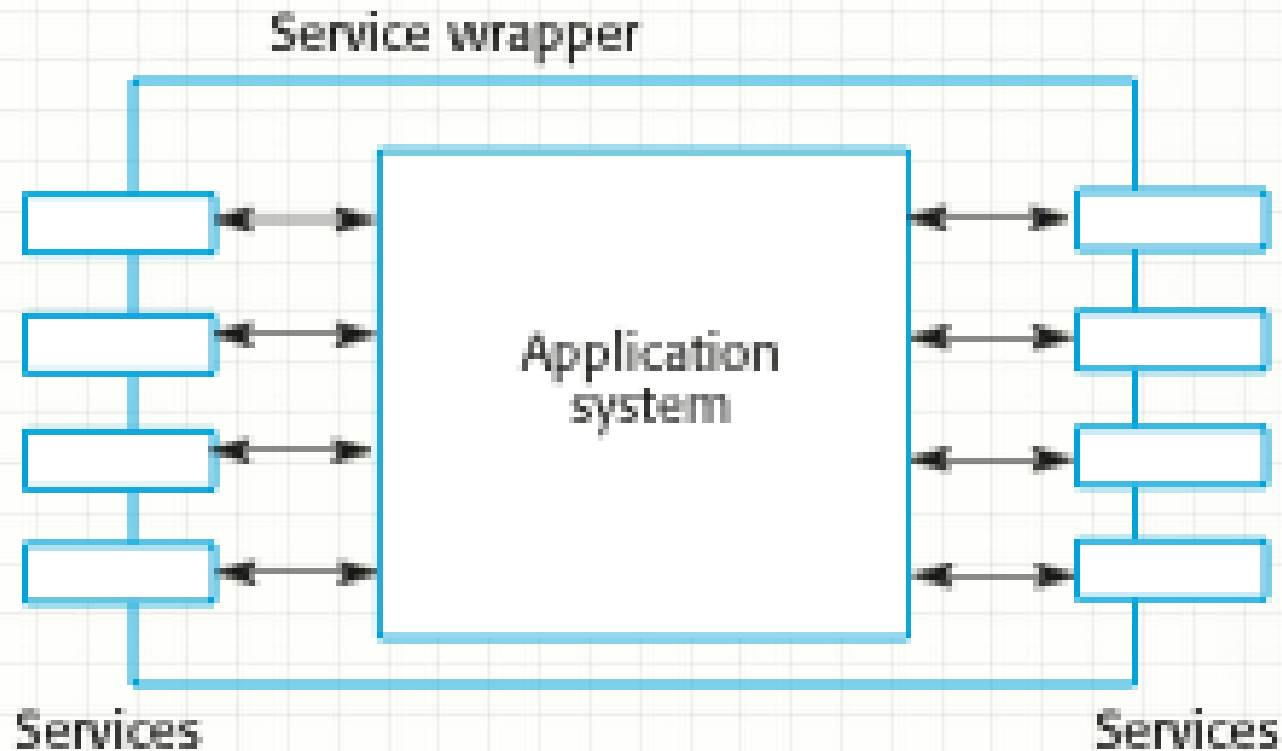
ERP Architecture

- 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)

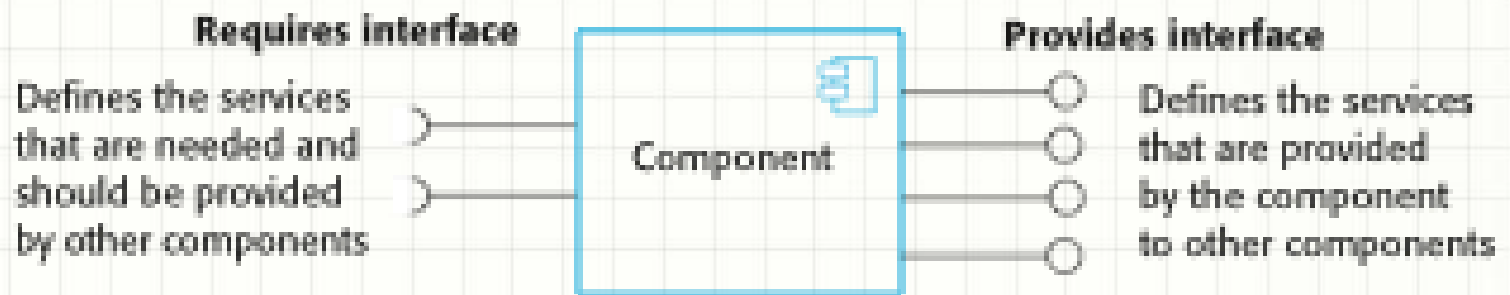
Reusable Components

- 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