The Object-Oriented Process

Week 8: Design and Implementation

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

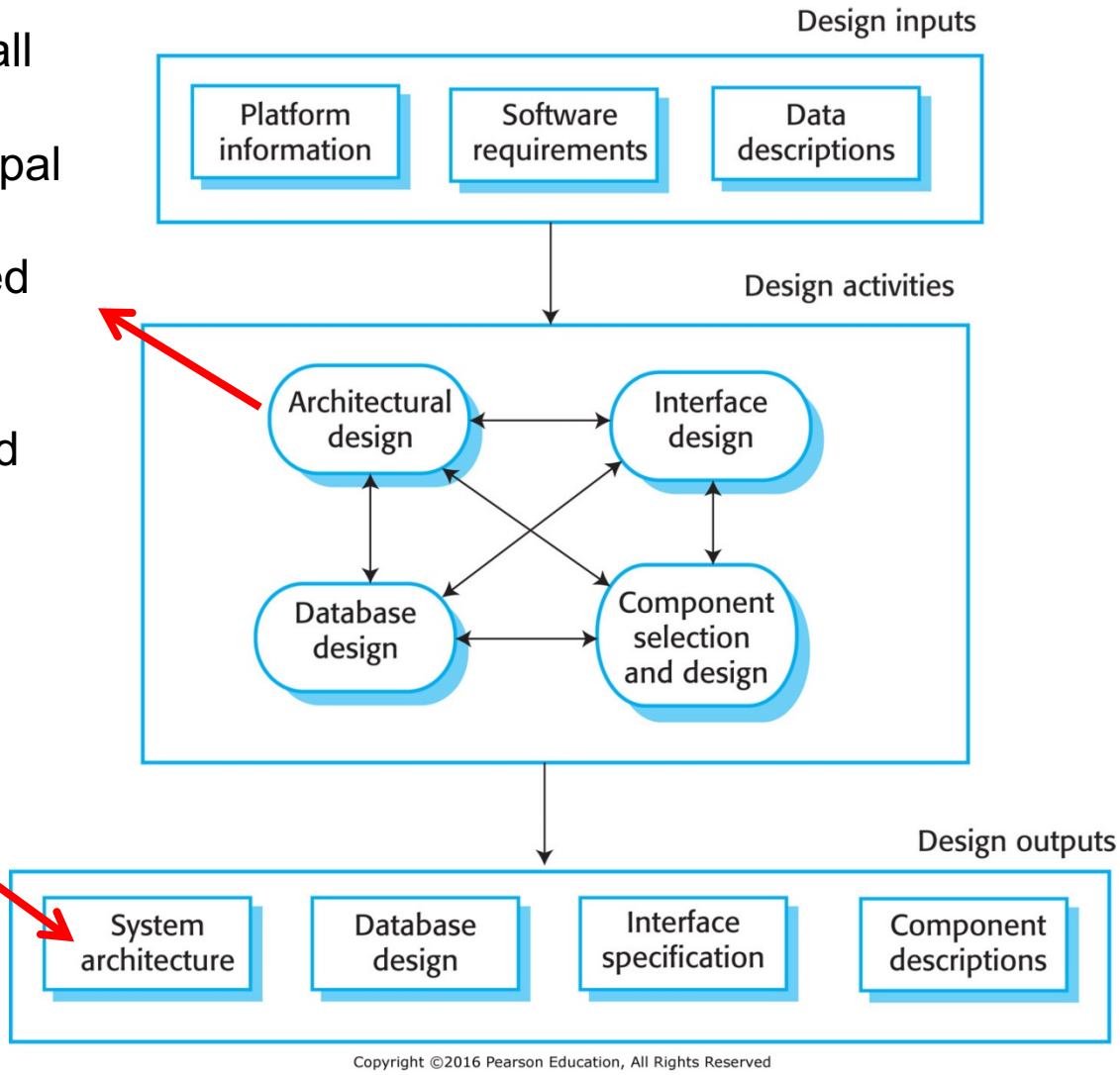
Design and Implementation

- Software design and implementation is the stage in the software engineering process at which an executable software system is developed.
- Software design and implementation activities are invariably interleaved.
  - Software design is a creative activity in which you identify software components and their relationships, based on a customer's requirements.
  - Implementation is the process of realizing the design as a program.

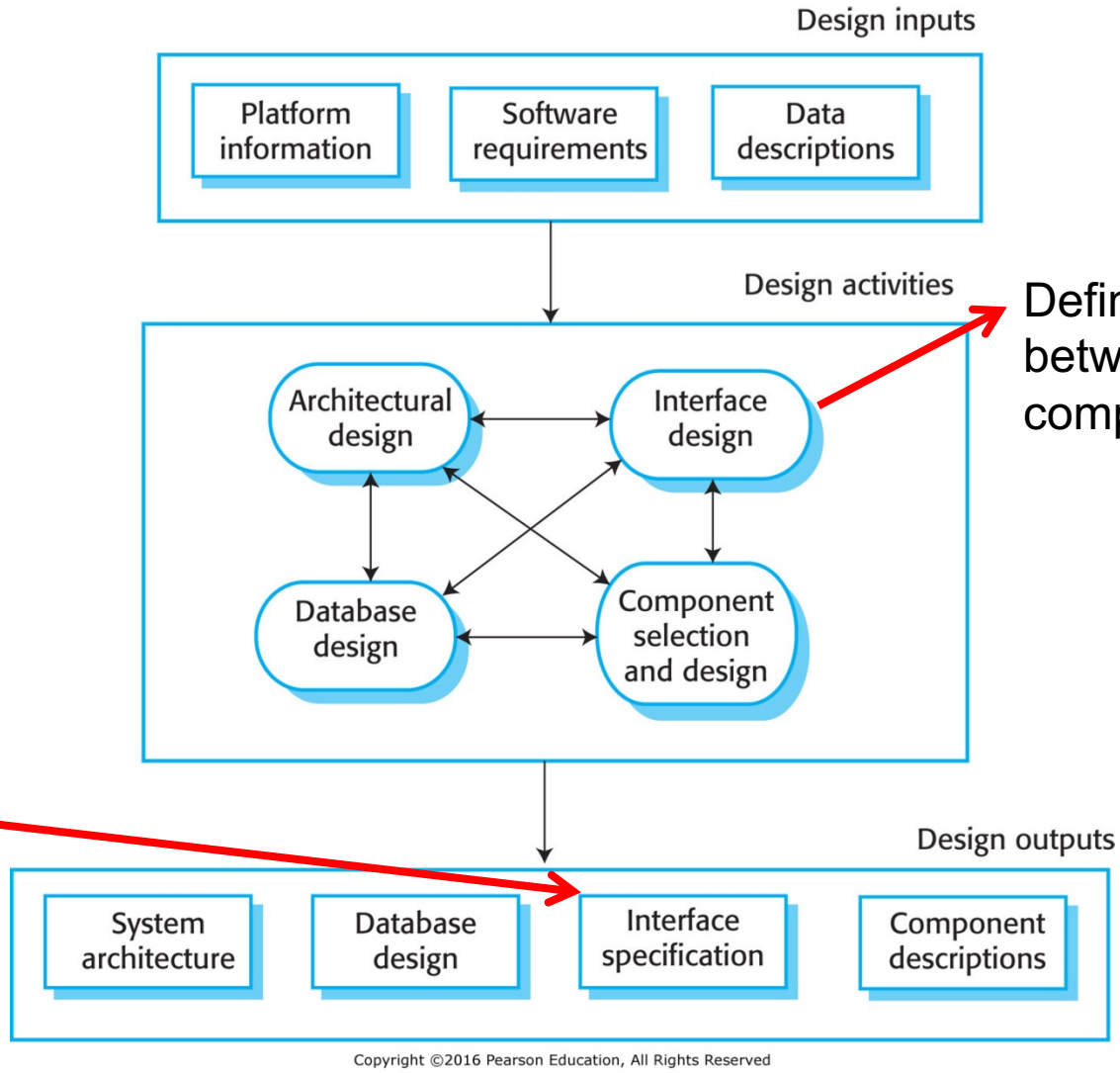
A General Model of the Design Process

Identify the overall structure of the system, its principal components (sometimes called subsystems or modules), their relationships, and how they are distributed.

Its output



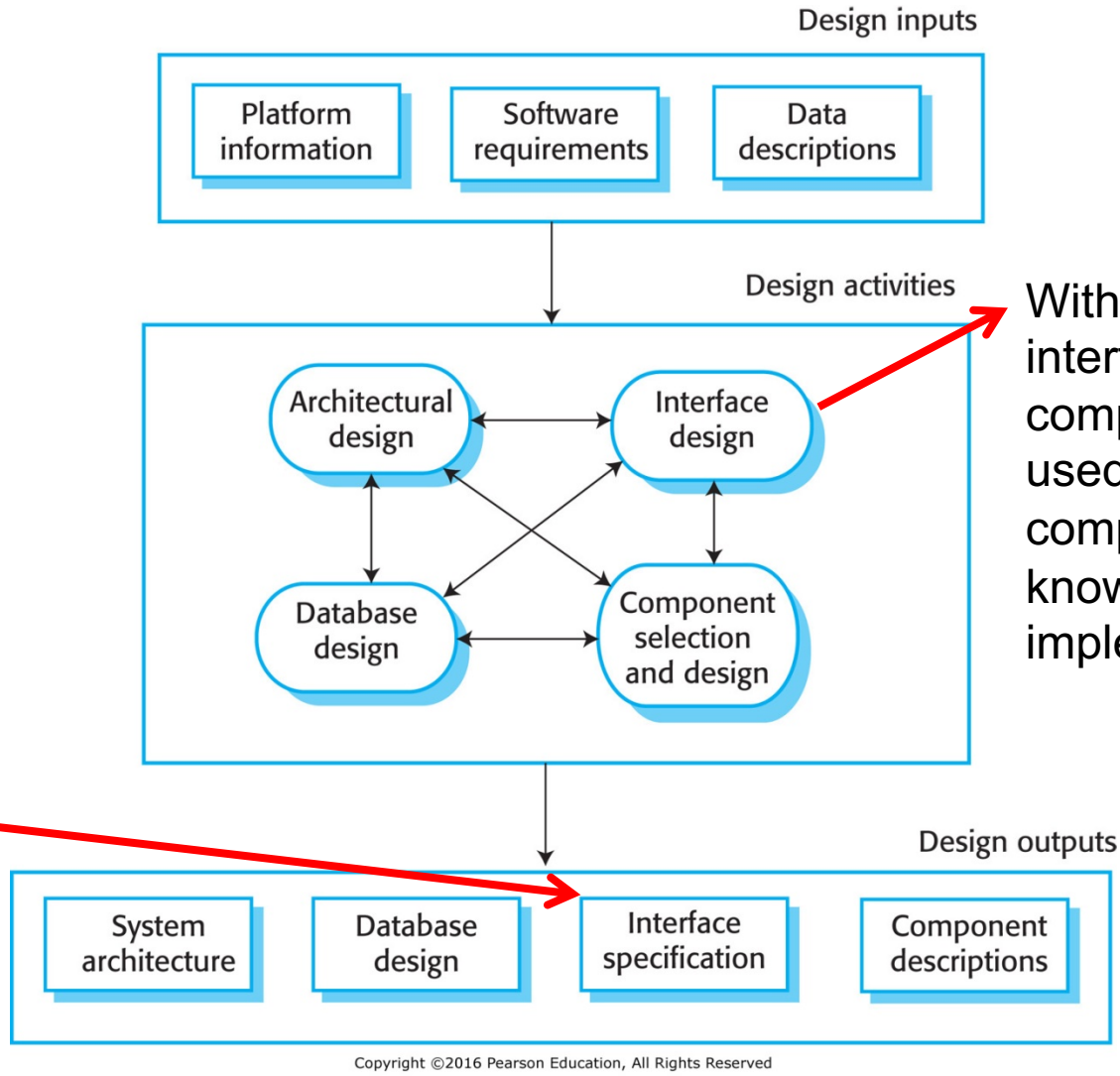
A General Model of the Design Process



Define the interfaces between system components.

Its output

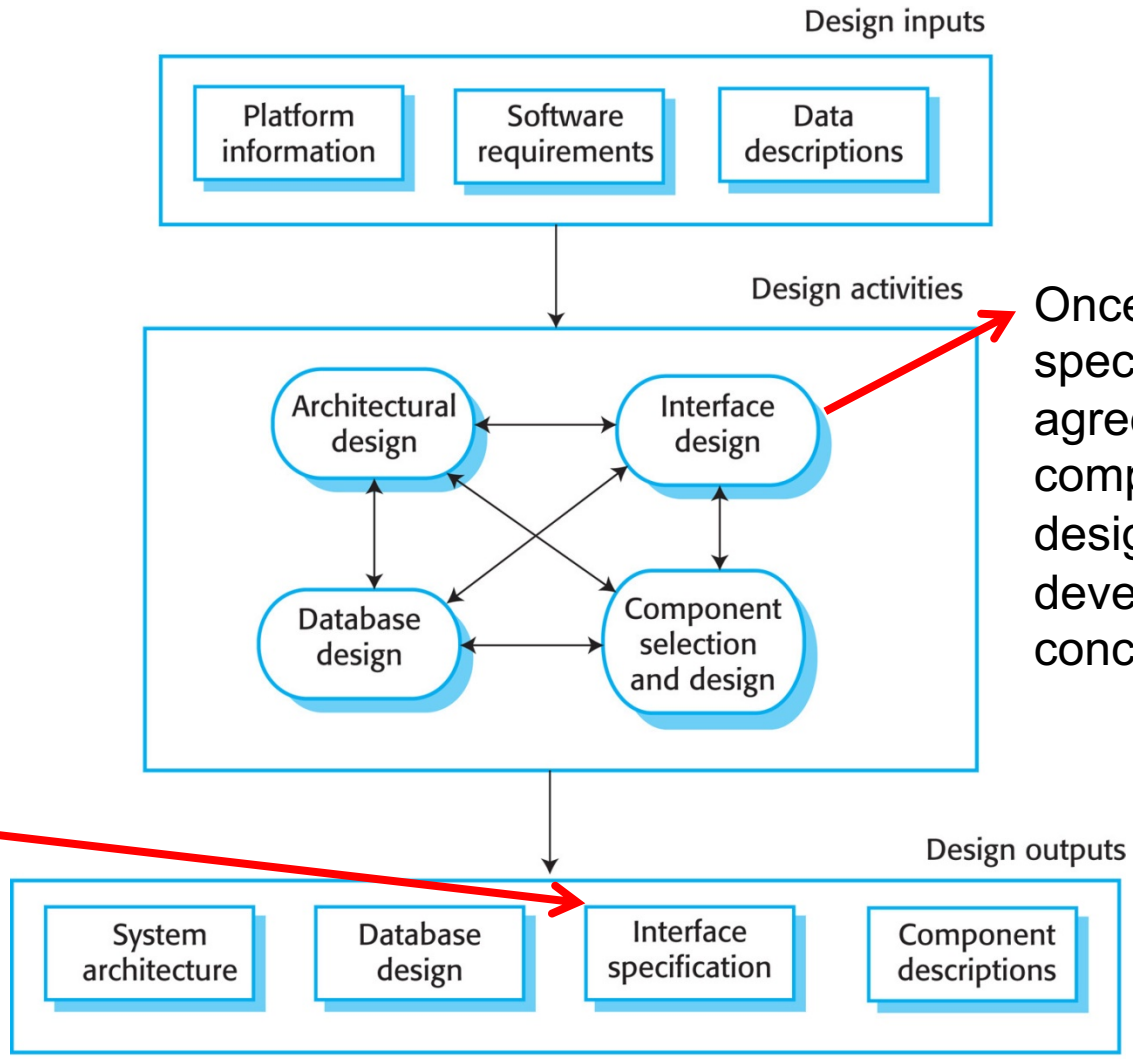
A General Model of the Design Process



With a precise interface, a component can be used without other components having to know how it is implemented.

Its output

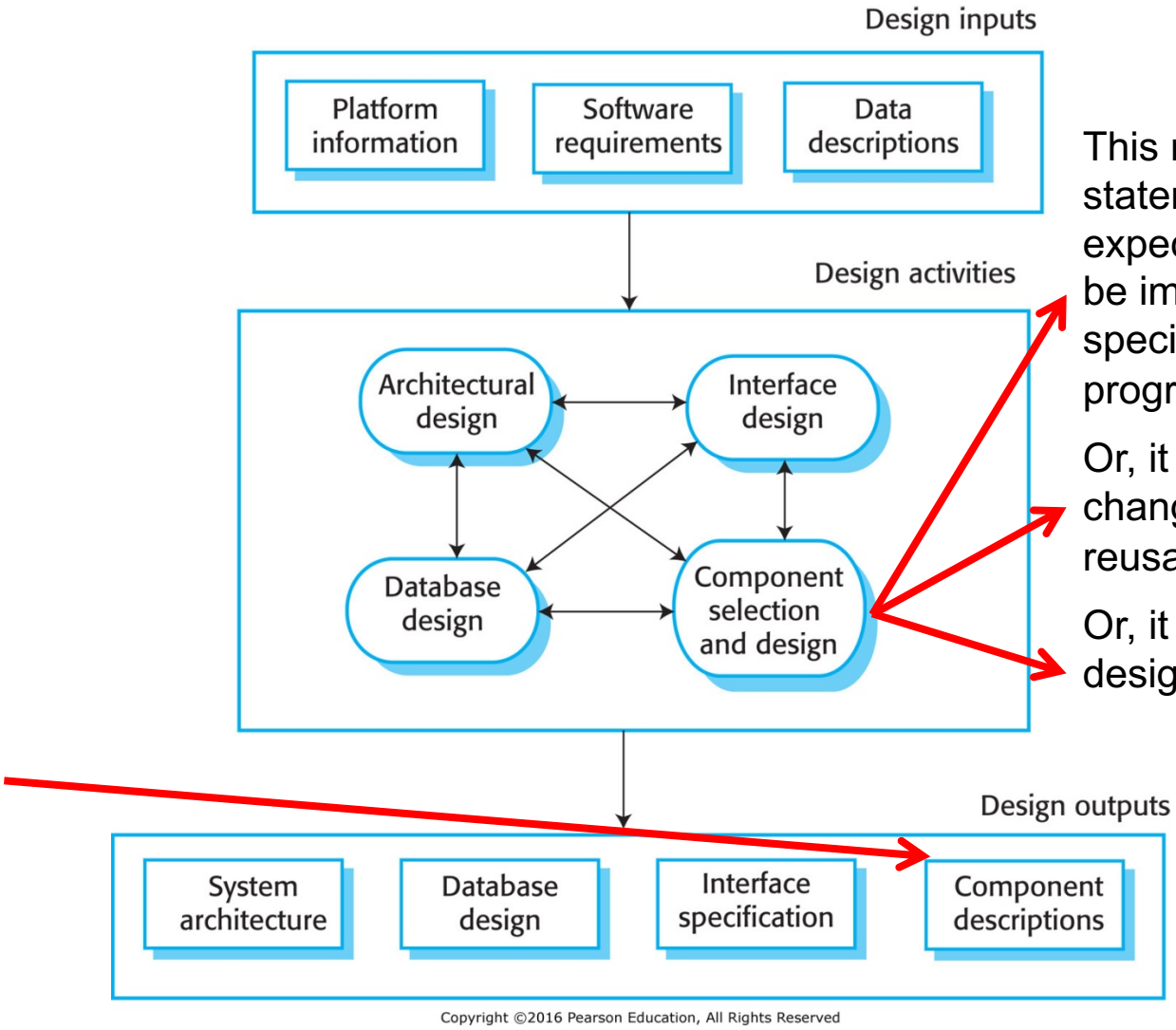
A General Model of the Design Process



Once interface specifications are agreed, the components can be designed and developed concurrently.

Its output

A General Model of the Design Process



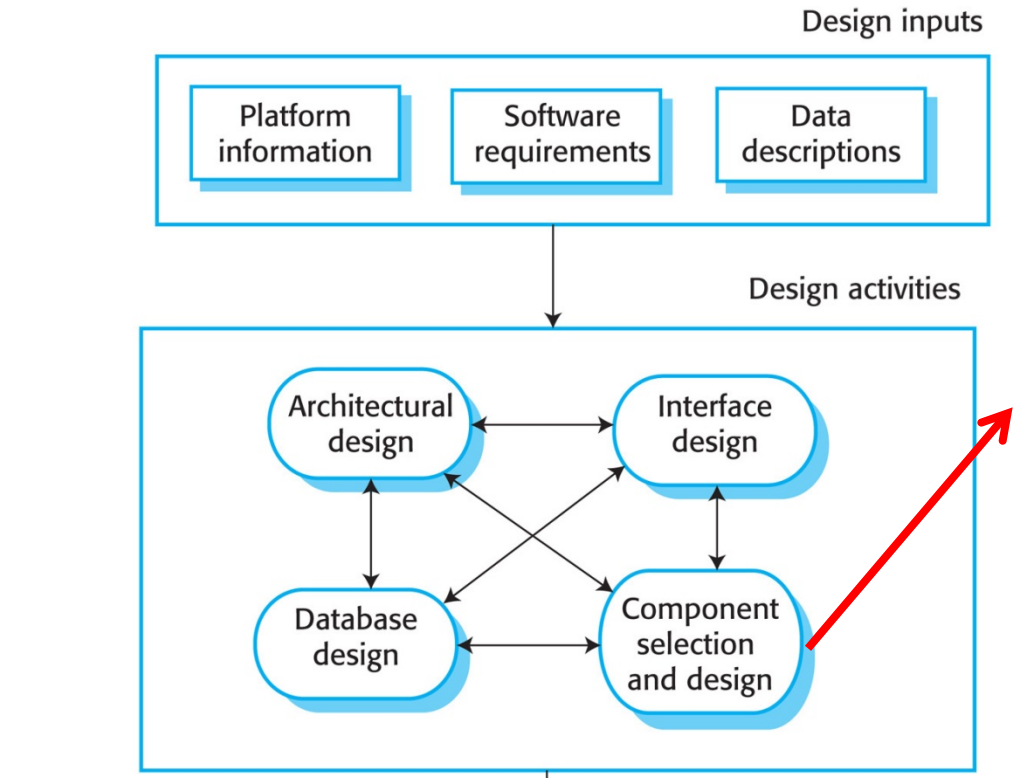
This may be a simple statement of the expected functionality to be implemented, with the specific design left to the programmer.

Or, it may be a list of changes to be made to a reusable component.

Or, it may be a detailed design model.

Its output

A General Model of the Design Process



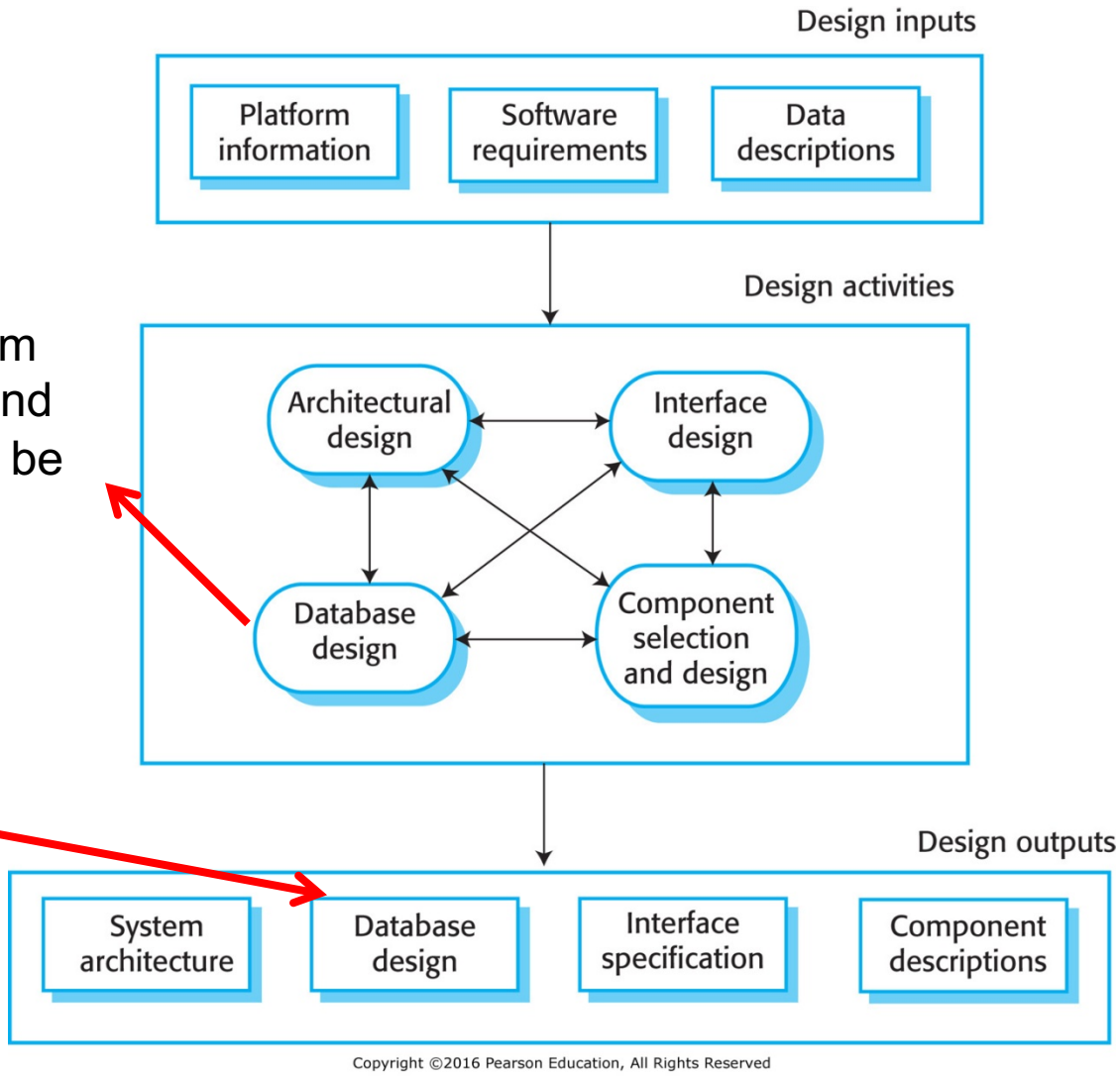
The design model may be used to automatically generate an implementation (model-driven engineering).

Its output

A General Model of the Design Process

Design the system data structures and how these are to be represented in a database.

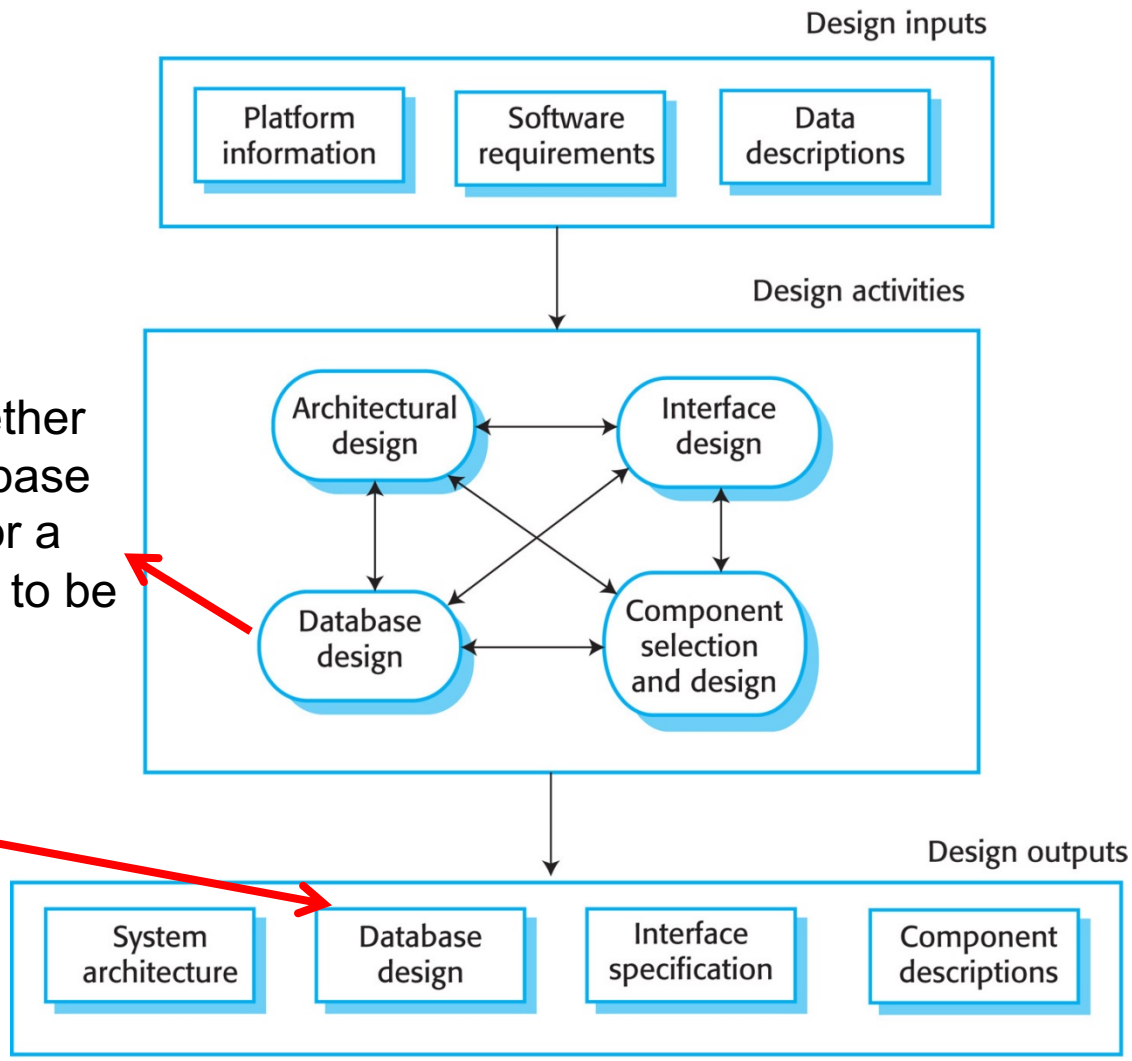
Its output



A General Model of the Design Process

The work here depends on whether an existing database is to be reused or a new database is to be created.

Its output



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Define the Context and Modes of Use of the System

Week 8: Design and Implementation

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

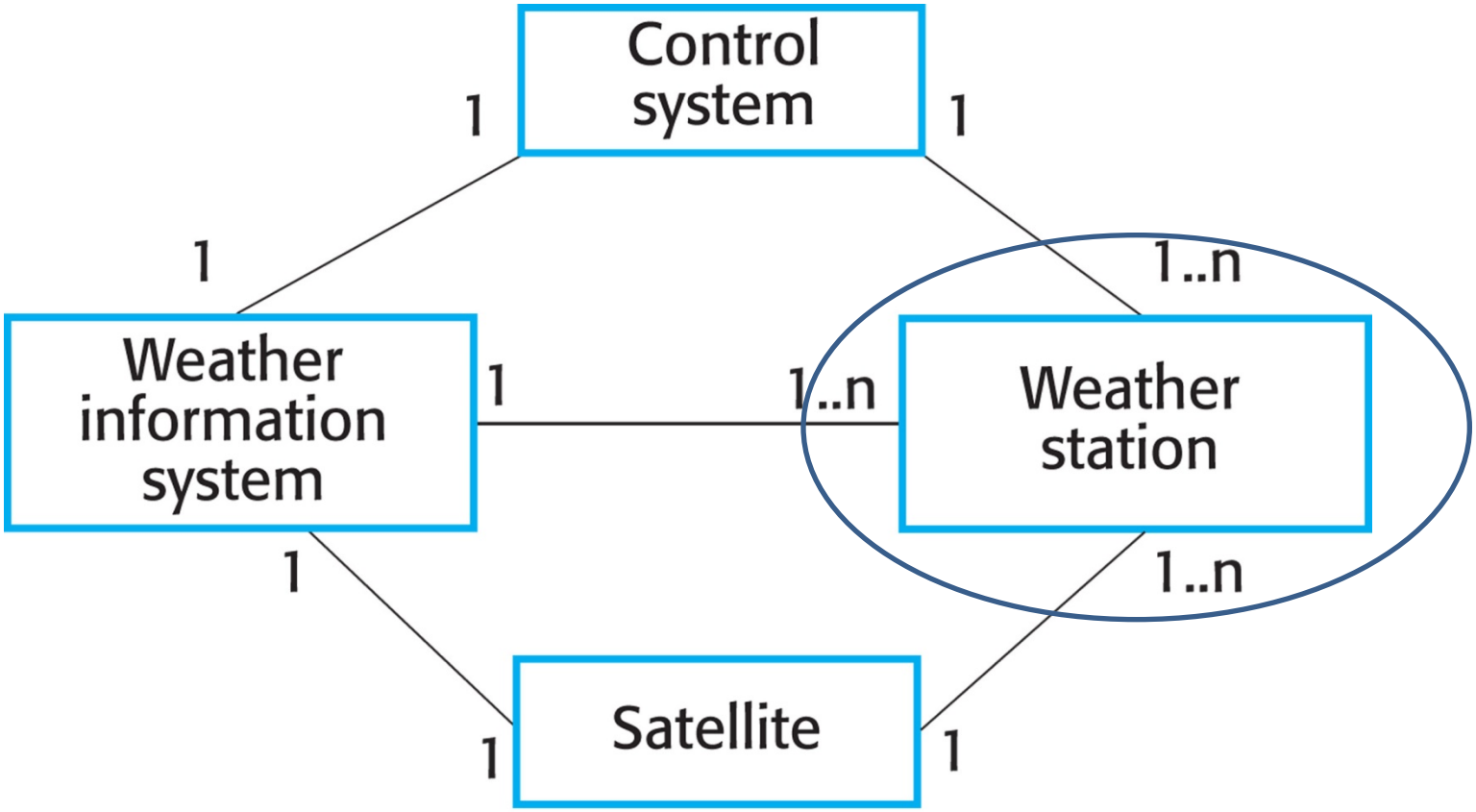
An Object-Oriented Design Process

- There are a variety of different object-oriented design processes that depend on the organization using the process.
- Common activities in these processes include:
  1. **Define the context and modes of use of the system.**
  2. Design the system architecture.
  3. Identify the principal system objects.
  4. Develop design models.
  5. Specify object interfaces.
- This process will be illustrated using a design for a weather station in wilderness, one of the four case studies described in our textbook.

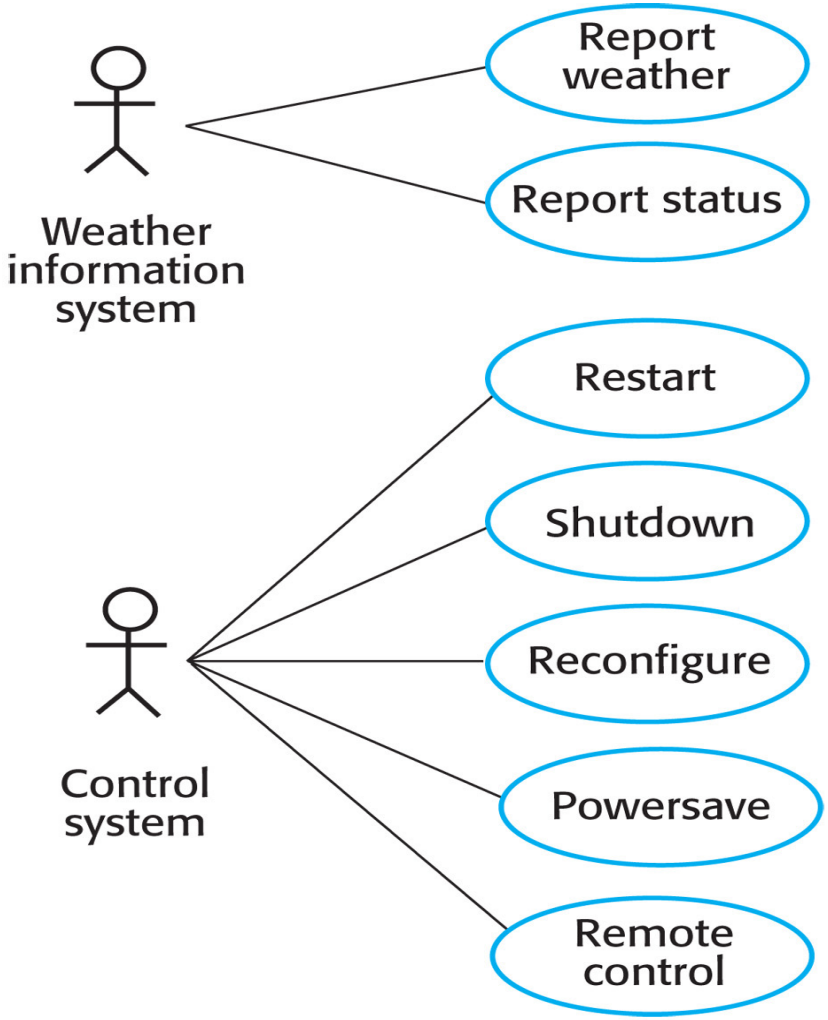
System Context and Interactions

- Understanding the relationships between the software that is being designed and its external environment is essential for deciding
  - How to provide the required system functionality, and
  - How to structure the system to communicate with its environment.
- Understanding of the context also lets you establish the **boundaries** of the system.
- Setting the system boundaries helps you decide what features are implemented in the system and what features are in other associated systems.

System Context and Interactions



Weather Station Use Cases



System	Weather station
Use case	Report weather
Actors	Weather information system, Weather station
Data	The weather station sends a summary of the weather data that has been collected from the instruments in the collection period to the weather information system. The data sent are the maximum, minimum, and average ground and air temperatures; the maximum, minimum, and average air pressures; the maximum, minimum and average wind speeds; the total rainfall; and the wind direction as sampled at 5-minute intervals.
Stimulus	The weather information system establishes a satellite communication link with the weather station and requests transmission of the data.
Response	The summarized data is sent to the weather information system.
Comments	Weather stations are usually asked to report once per hour, but this frequency may differ from one station to another and may be modified in future.

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Design the System Architecture

Week 8: Design and Implementation

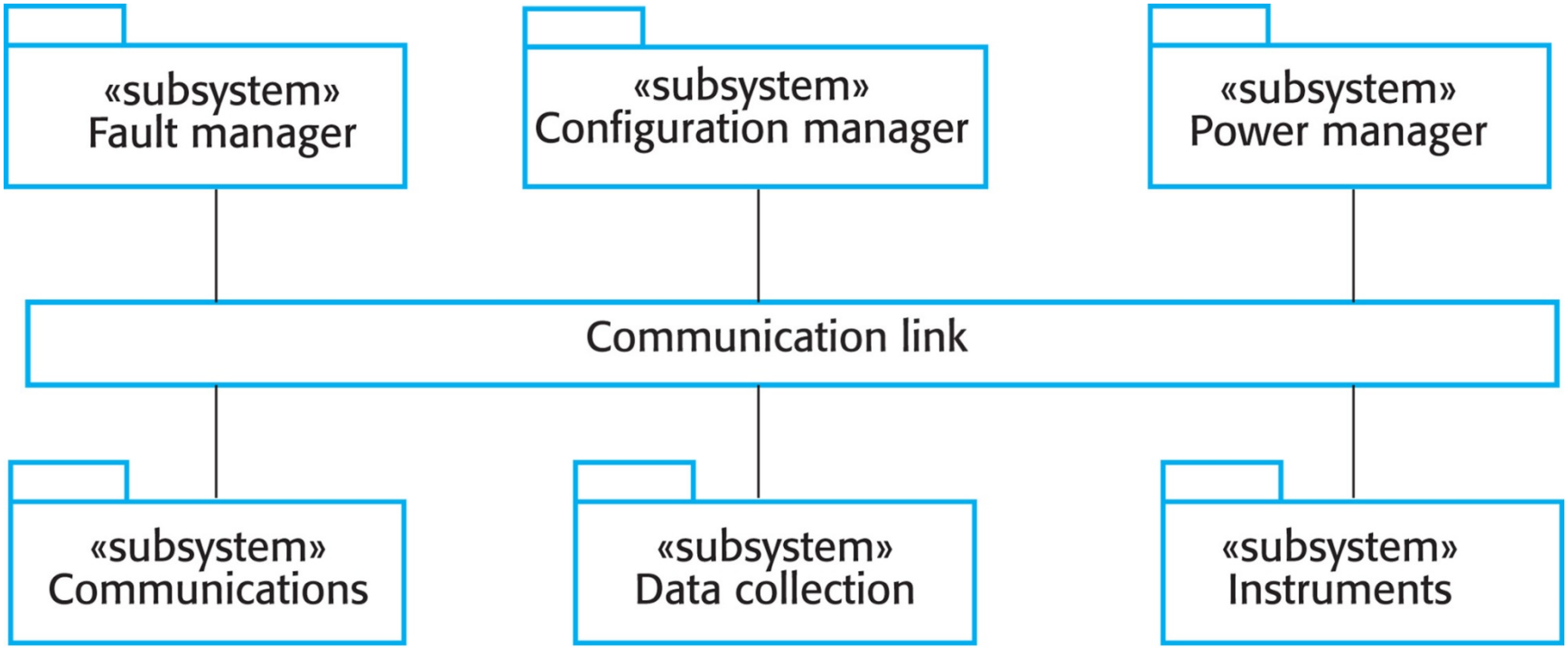
Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

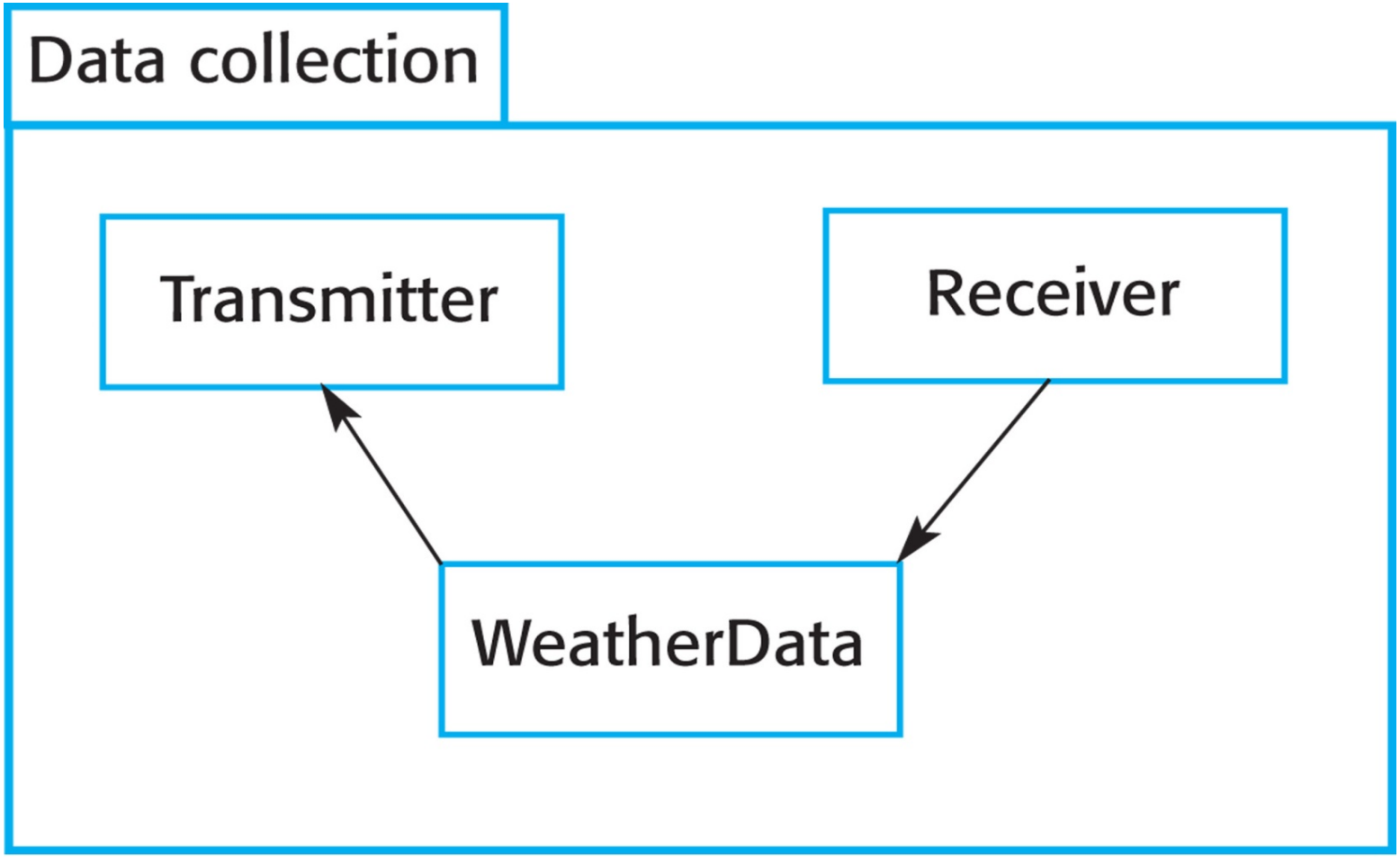
An Object-Oriented Design Process

- There are a variety of different object-oriented design processes that depend on the organization using the process.
- Common activities in these processes include:
  1. Define the context and modes of use of the system.
  2. **Design the system architecture.**
  3. Identify the principal system objects.
  4. Develop design models.
  5. Specify object interfaces.
- This process will be illustrated using a design for a weather station in wilderness, one of the four case studies described in our textbook.

High-Level Architecture of the Weather Station



Architecture of Data-Collection System



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Identify the Principal System Objects

Week 8: Design and Implementation

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

An Object-Oriented Design Process

- There are a variety of different object-oriented design processes that depend on the organization using the process.
- Common activities in these processes include:
  1. Define the context and modes of use of the system.
  2. Design the system architecture.
  3. **Identify the principal system objects.**
  4. Develop design models.
  5. Specify object interfaces.
- This process will be illustrated using a design for a weather station in wilderness, one of the four case studies described in our textbook.

Approaches to Identification

- Use a **grammatical approach** based on a natural language description of the system (used in the OOD method).
- Use **tangible entities** (things) in the application domain such as aircraft, roles such as manager or doctor, events such as requests, interactions such as meetings, locations such as offices, organizational units such as companies, and so on.
- Use a **scenario**-based analysis where various scenarios of system use are identified and analyzed in turn. As each scenario is analyzed, the team responsible for the analysis must identify the required objects, attributes, and operations.

Weather Station Description

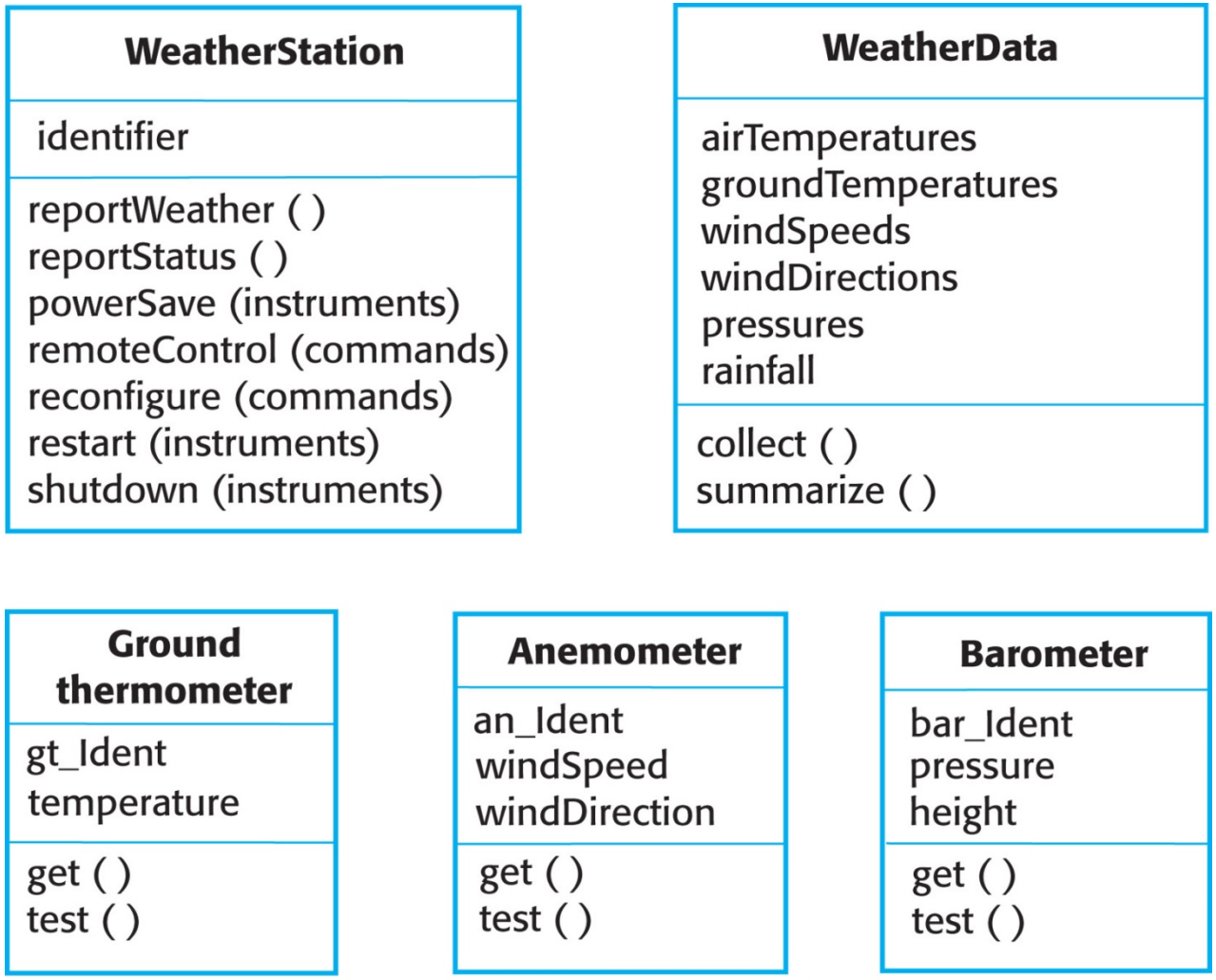
A weather station is a package of software-controlled instruments that collects data, performs some data processing, and transmits this data for further processing. The instruments include air and ground thermometers, an anemometer, a wind vane, a barometer, and a rain gauge. Data are collected periodically.

When a command is issued to transmit the weather data, the weather station processes and summarizes the collected data. The summarized data are transmitted to the mapping computer when a request is received.

Weather Station Object Classes

- Object class identification in the weather station system may also be based on the tangible hardware and data in the system.
  - Ground thermometer, anemometer, barometer
    - Application domain objects that are “hardware” objects related to the instruments in the system
  - Weather station
    - The basic interface of the weather station to its environment. It therefore reflects the interactions identified in the use-case model.
  - Weather data
    - Encapsulates the summarized data from the instruments

Weather Station Object Classes



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Develop Design Models

Week 8: Design and Implementation

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING
& COMPUTER
SCIENCE

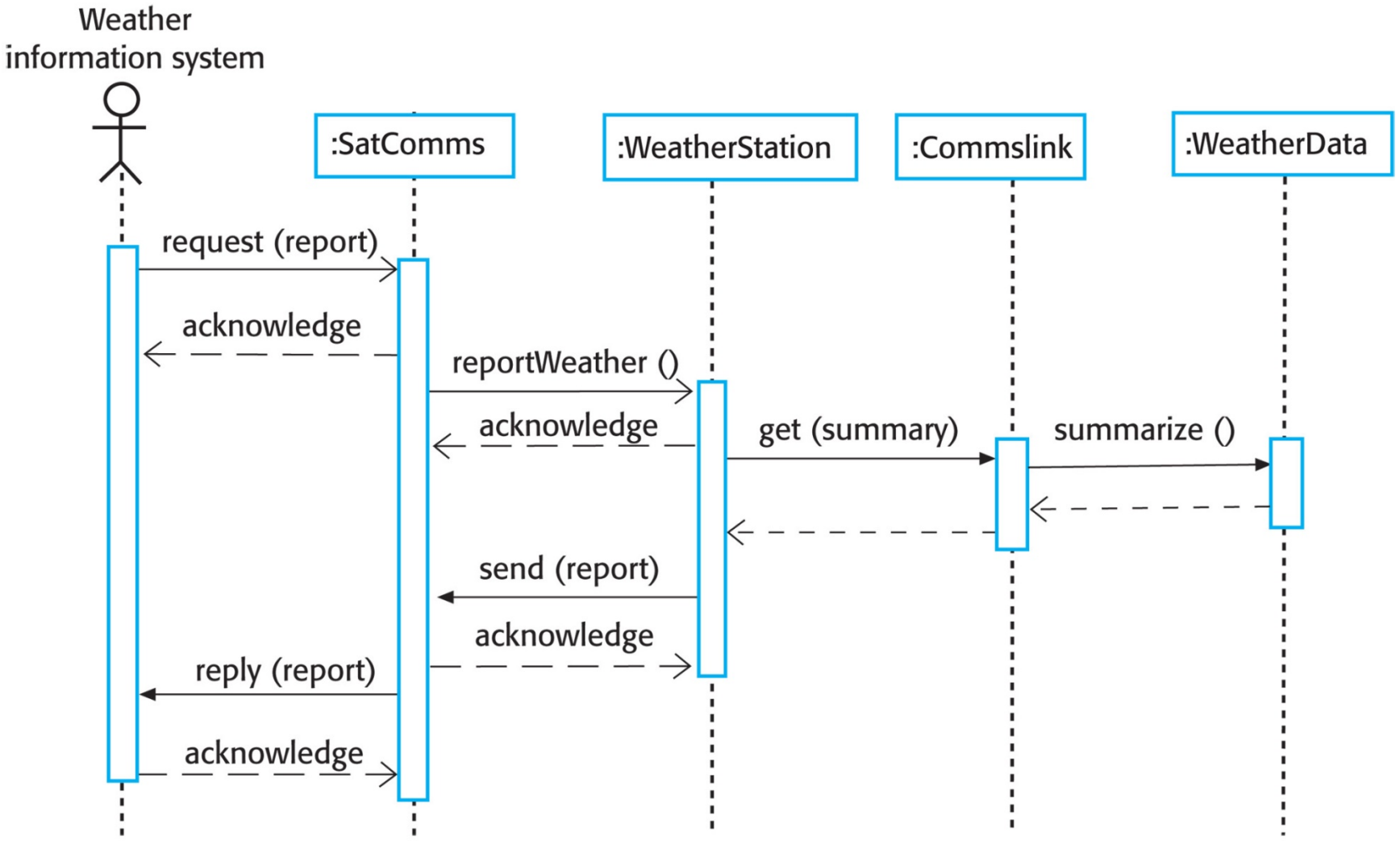
An Object-Oriented Design Process

- There are a variety of different object-oriented design processes that depend on the organization using the process.
- Common activities in these processes include:
  1. Define the context and modes of use of the system.
  2. Design the system architecture.
  3. Identify the principal system objects.
  4. **Develop design models.**
  5. Specify object interfaces.
- This process will be illustrated using a design for a weather station in wilderness, one of the four case studies described in our textbook.

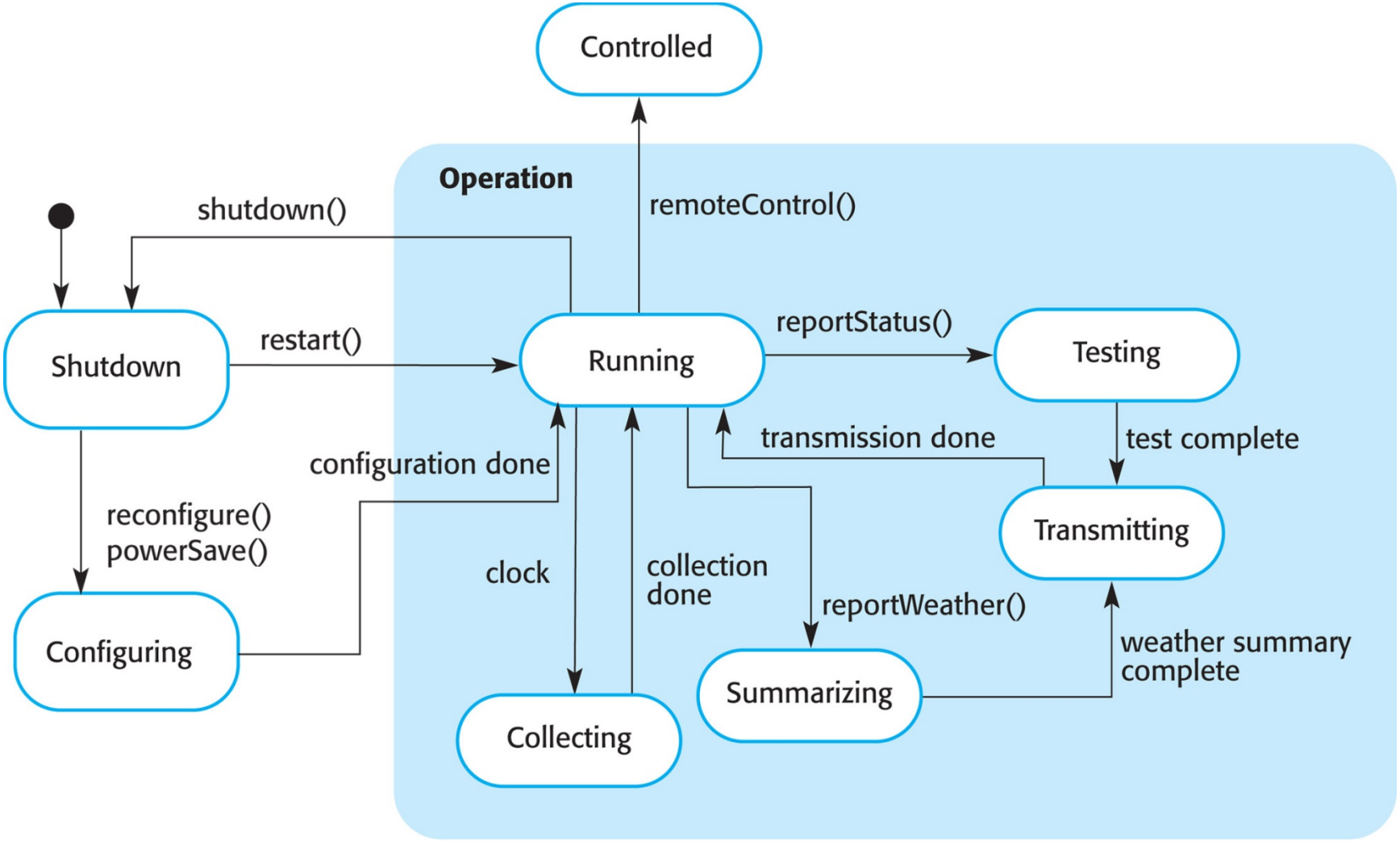
Design Models

- Design models show the objects and object classes and relationships between these entities.
  - **Static models** describe the static structure of the system in terms of object classes and relationships.
  - **Dynamic models** describe the dynamic interactions between objects.

Sequence Diagram Describing Data Collection



Weather Station State Diagram



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Specify Object Interfaces

Week 8: Design and Implementation

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

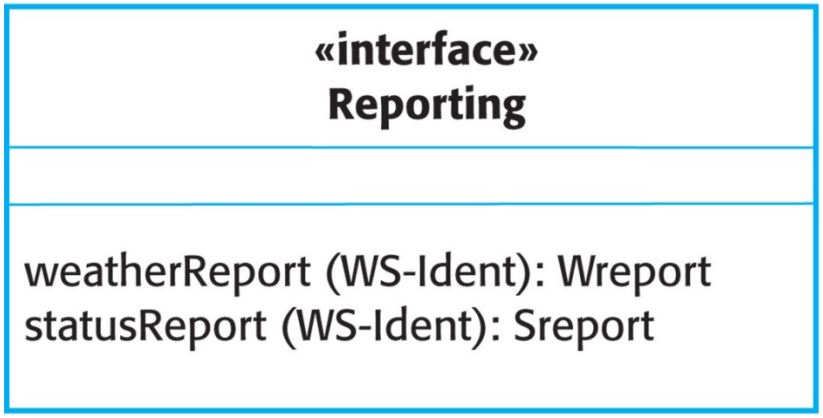
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

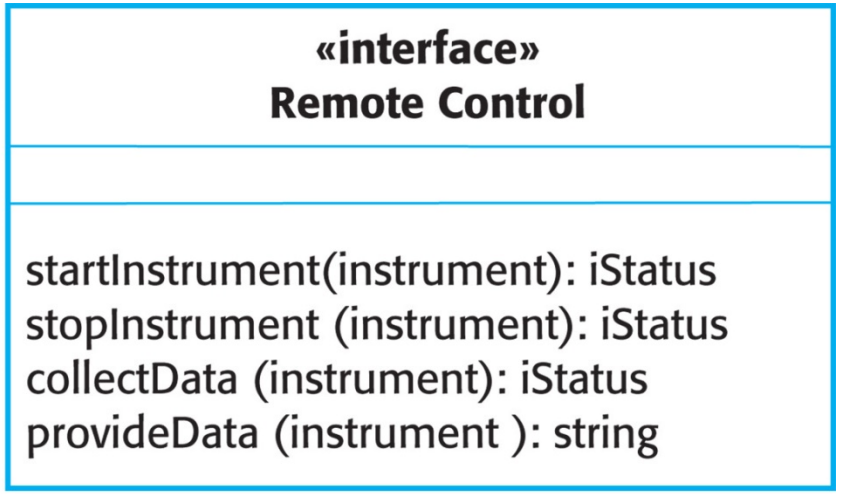
An Object-Oriented Design Process

- There are a variety of different object-oriented design processes that depend on the organization using the process.
- Common activities in these processes include:
  1. Define the context and modes of use of the system.
  2. Design the system architecture.
  3. Identify the principal system objects.
  4. Develop design models.
  5. **Specify object interfaces.**
- This process will be illustrated using a design for a weather station in wilderness, one of the four case studies described in our textbook.

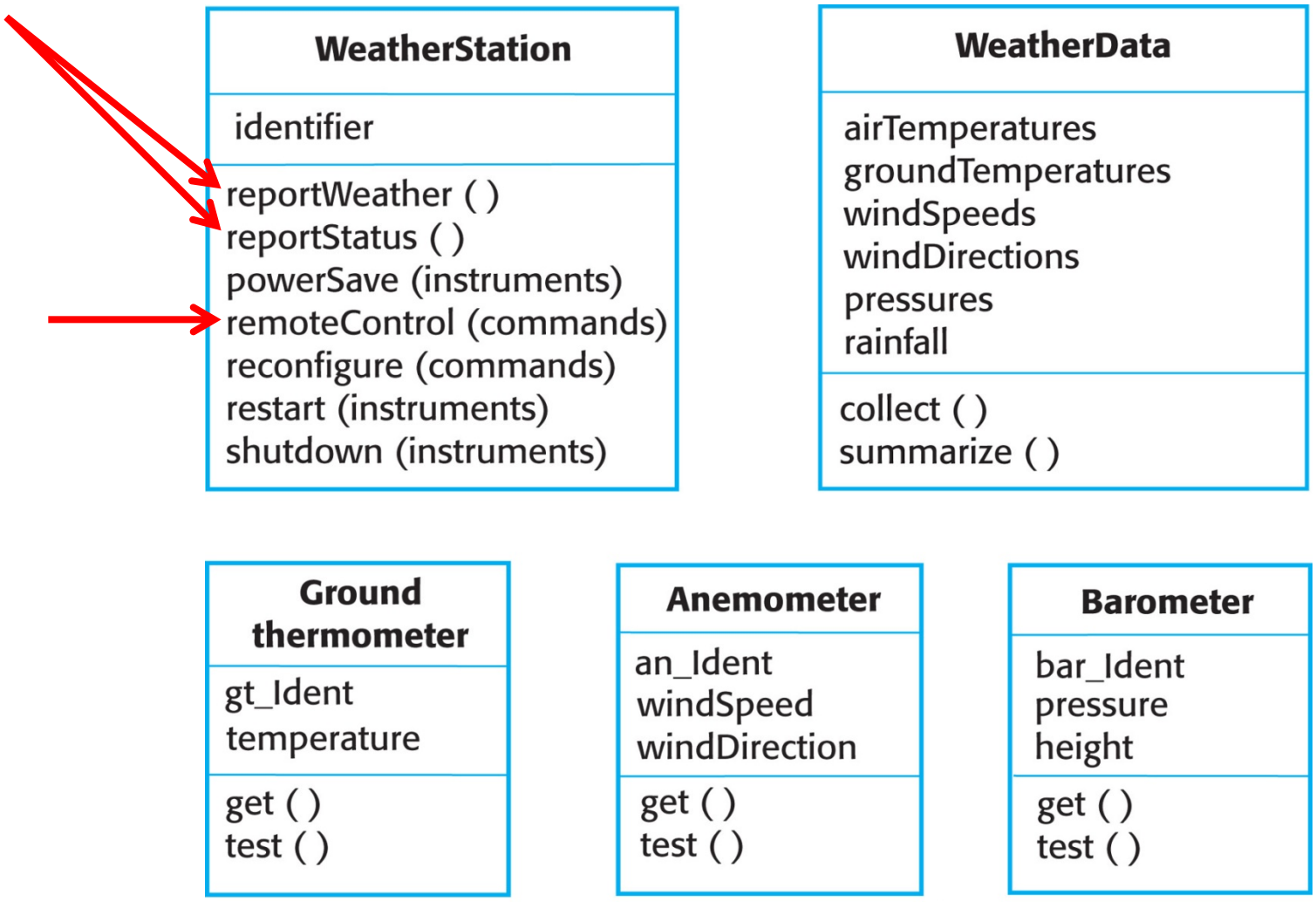
Interface Specification

- Object interfaces have to be specified so that the objects and other components can be designed in parallel.
- The UML uses class diagrams for interface specification, but Java may also be used.
- Designers should not include details of data representation, since attributes are not defined in an interface specification.
- Objects may have several interfaces.





Weather Station Object Classes



SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Design Patterns

Week 8: Design and Implementation

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

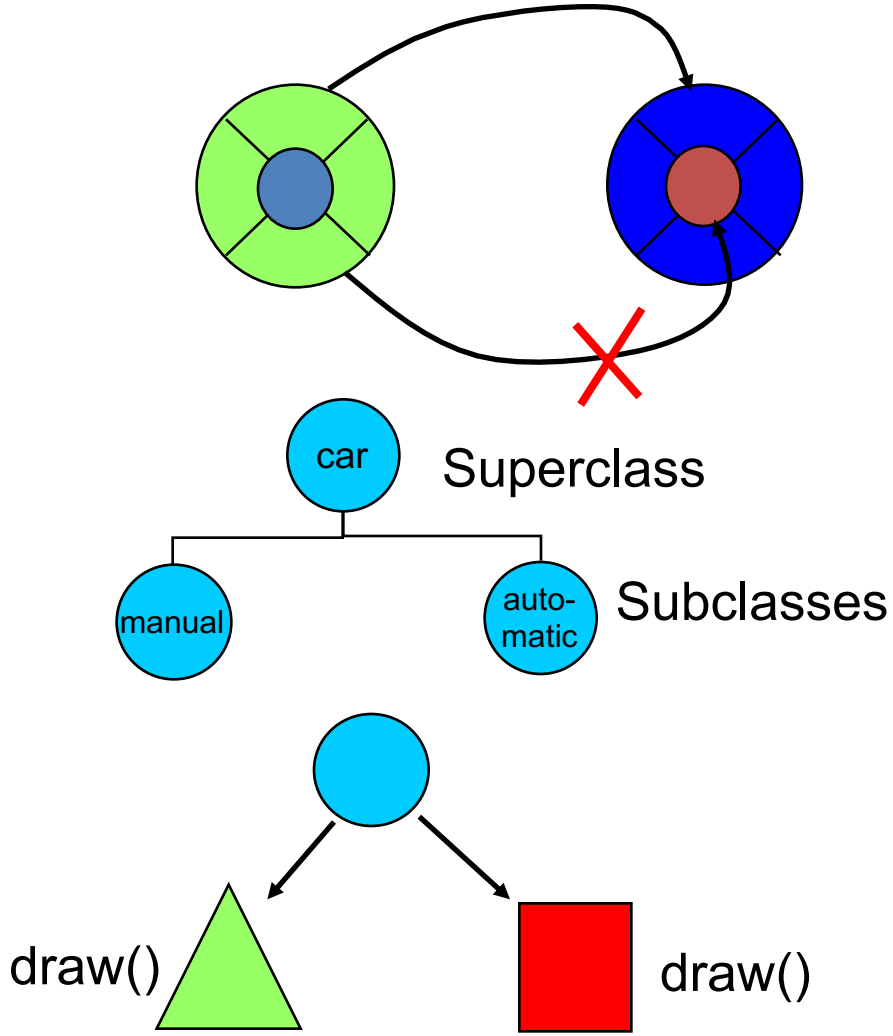
SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Design Patterns

- A design pattern
  - Is a way of reusing abstract knowledge about a problem and its solution
  - Is a description of the problem and the essence of its solution
  - Should be sufficiently abstract to be reused in different settings
  - Usually makes use of object-oriented characteristics such as inheritance and polymorphism

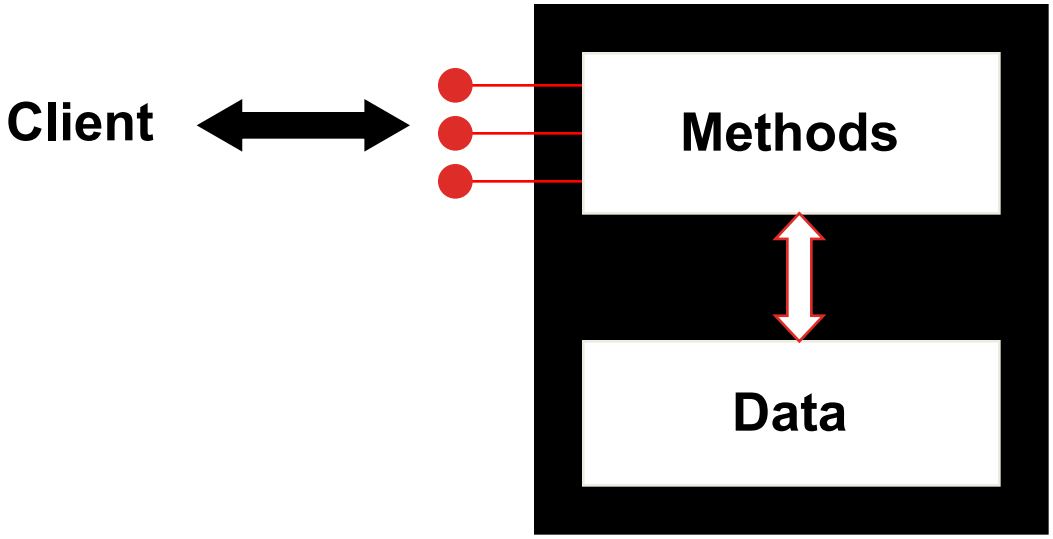
The Three Pillars of OOP

- **Encapsulation**
  - Objects hide their functions (**methods**) and data (**instance variables**).
- **Inheritance**
  - Each **subclass** inherits all variables of its **superclass**.
- **Polymorphism**
  - Same interface despite different data types.



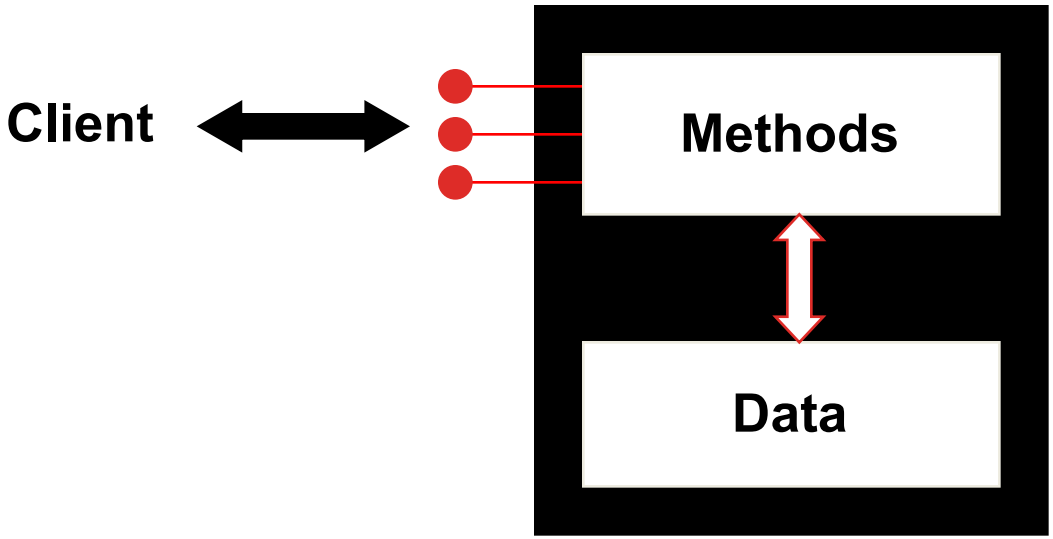
Encapsulation

- One object (called the **client**) may use another object for the services it provides.
- The client of an object may request its services (i.e., call its methods), but it should not have to be aware of how those services are accomplished.



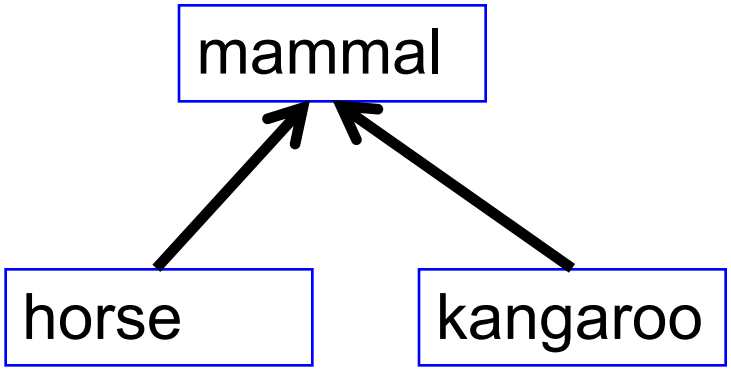
Encapsulation

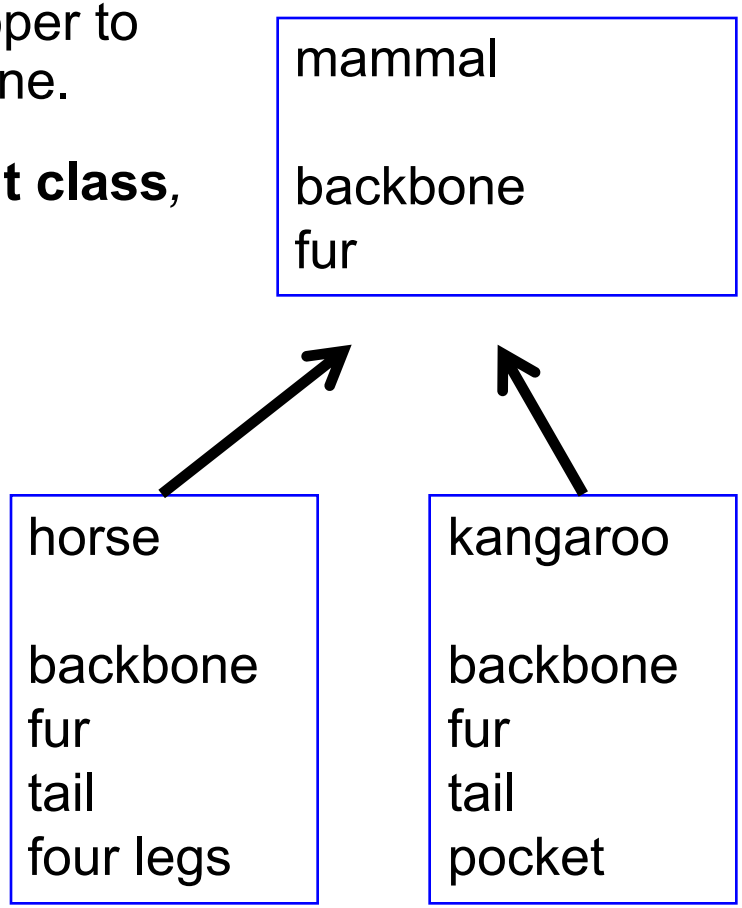
- Any changes to the object's state (its variables) should be made by that object's methods.
  - In other words, an object should be **self-governing**.
  - We should make it difficult, if not impossible, for a client to access an object's variables directly.



Inheritance

- **Inheritance** allows a software developer to derive a new class from an existing one.
- The existing class is called the **parent class**, or **superclass**, or **base class**.





- The derived class is called the **child class** or **subclass**.
- As the name implies, the child inherits characteristics of the parent.

Polymorphism

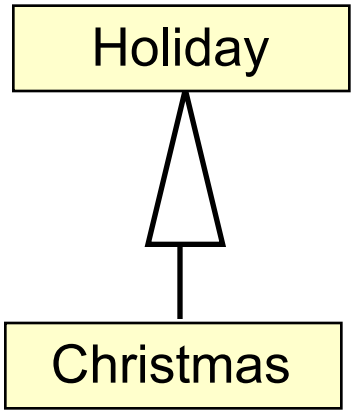
- The term **polymorphism** literally means “having many forms.”
- A **polymorphic reference** is a variable that can refer to different types of objects at different points in time.
- Suppose we create the following reference variable:

  Holiday day;

- This reference can point to an Occupation object, or to any object of any compatible type.
- This compatibility can be established using inheritance or using interfaces.

Polymorphism

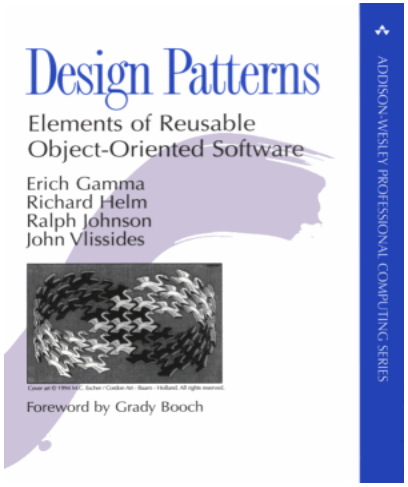
- For example, if Holiday is the superclass of Christmas, then a Holiday reference could be used to refer to a Christmas object.
- These type compatibility rules are just an extension of the **is-a** relationship established by inheritance.




```
Holiday day;
day = new Christmas();
```

Gang of Four

- **Erich Gamma, Richard Helm, Ralph Johnson, and John Vlisides,** in their ***Design Patterns*** book, define 23 design patterns divided into three types:
  - **Creational patterns**—create objects for you
  - **Structural patterns**—help you compose groups of objects into larger structures
  - **Behavioral patterns**—help you define and control the communication between objects in your system

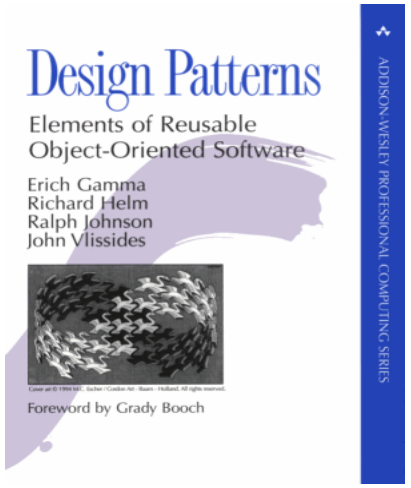


- The original design patterns were intended for lower-level, detailed design, but the same concept can be and have been applied to architectural design, hence architectural patterns.

		Purpose ← Reflects what a pattern does		
		Creational	**Structural**	**Behavioral**
Scope ↑ Domain over which a pattern applies	**Class**	Factory Method (107)	Adapter (139)	Interpreter (243) Template Method (325)
	Object	Abstract Factory (87) Builder (97) Prototype (117) Singleton (127)	Adapter (139) Bridge (151) Composite (163) Decorator (175) Facade (185) Proxy (207)	Chain of Responsibility (223) Command (233) Iterator (257) Mediator (273) Memento (283) Flyweight (195) Observer (293) State (305) Strategy (315) Visitor (331)

Design Pattern Elements

- Name
  - A meaningful pattern identifier
- Description
- Problem description
- Solution description
  - Not a concrete design but a template for a design solution that can be instantiated in different ways
- Consequences
  - The results and trade-offs of applying the pattern



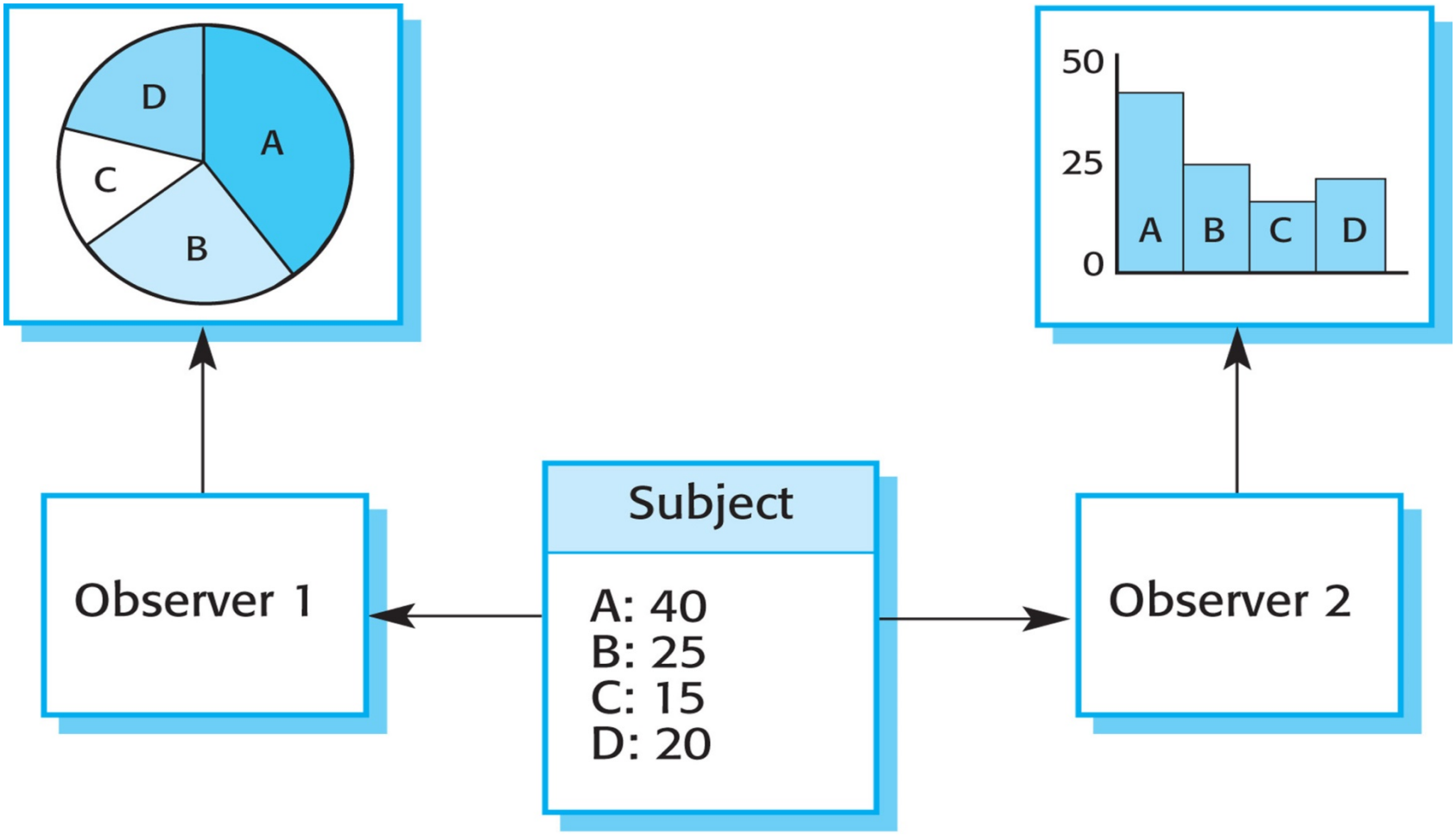
The Observer Pattern

- Name
  - Observer
- Description
  - Separates the display of object state from the object itself
- Problem description
  - Used when multiple displays of state are needed
- Solution description
  - See slide with UML description
- Consequences
  - Optimizations to enhance display performance are impractical.

The Observer Pattern

Pattern name	Observer
Description	Separates the display of the state of an object from the object itself and allows alternative displays to be provided. When the object state changes, all displays are automatically notified and updated to reflect the change.
Problem description	In many situations, you have to provide multiple displays of state information, such as a graphical display and a tabular display. Not all of these may be known when the information is specified. All alternative presentations should support interaction and, when the state is changed, all displays must be updated. This pattern may be used in all situations where more than one display format for state information is required and where it is not necessary for the object that maintains the state information to know about the specific display formats used.

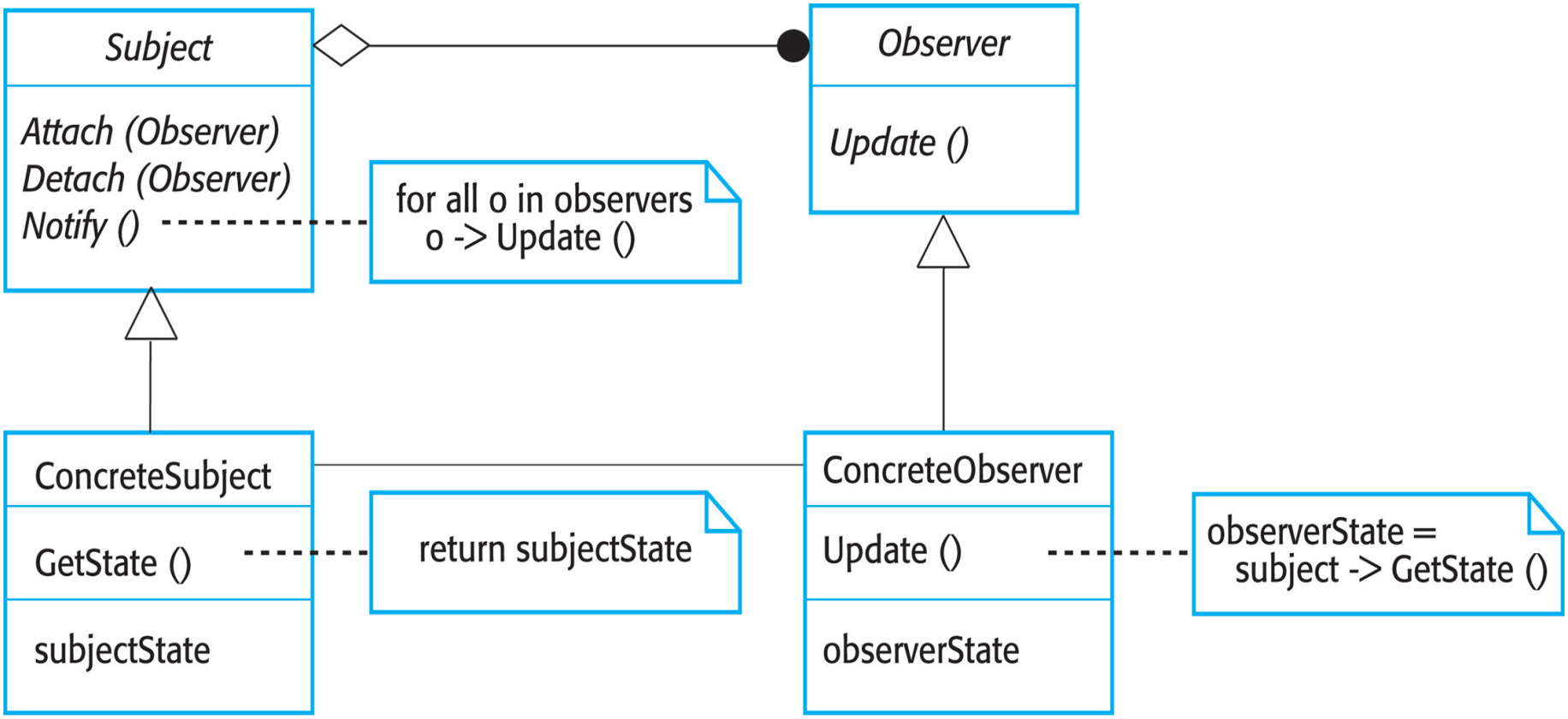
Multiple Displays



The Observer Pattern

Pattern name	**Observer**
Solution description	This involves two abstract objects, Subject and Observer, and two concrete objects, ConcreteSubject and ConcreteObject, which inherit the attributes of the related abstract objects. The abstract objects include general operations that are applicable in all situations. The state to be displayed is maintained in ConcreteSubject, which inherits operations from Subject, allowing it to add and remove Observers (each observer corresponds to a display) and to issue a notification when the state has changed. The ConcreteObserver maintains a copy of the state of ConcreteSubject and implements the Update() interface of Observer that allows these copies to be kept in step. The ConcreteObserver automatically displays the state and reflects changes whenever the state is updated.
Consequences	The subject knows only the abstract Observer and does not know details of the concrete class. Therefore, there is minimal coupling between these objects. Because of this lack of knowledge, optimizations that enhance display performance are impractical. Changes to the subject may cause a set of linked updates to observers to be generated, some of which may not be necessary.

A UML Model of the Observer Pattern



Design Problems

- To use patterns in your design, you need to recognize that any design problem you are facing may have an associated pattern that can be applied.
  - Tell several objects that the state of some other object has changed (**Observer pattern**).
  - Tidy up the interfaces to a number of related objects that have often been developed incrementally (**Façade pattern**).
  - Provide a standard way of accessing the elements in a collection, irrespective of how that collection is implemented (**Iterator pattern**).
  - Allow for the possibility of extending the functionality of an existing class at run-time (**Decorator pattern**).

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Configuration Management

Week 8: Design and Implementation

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

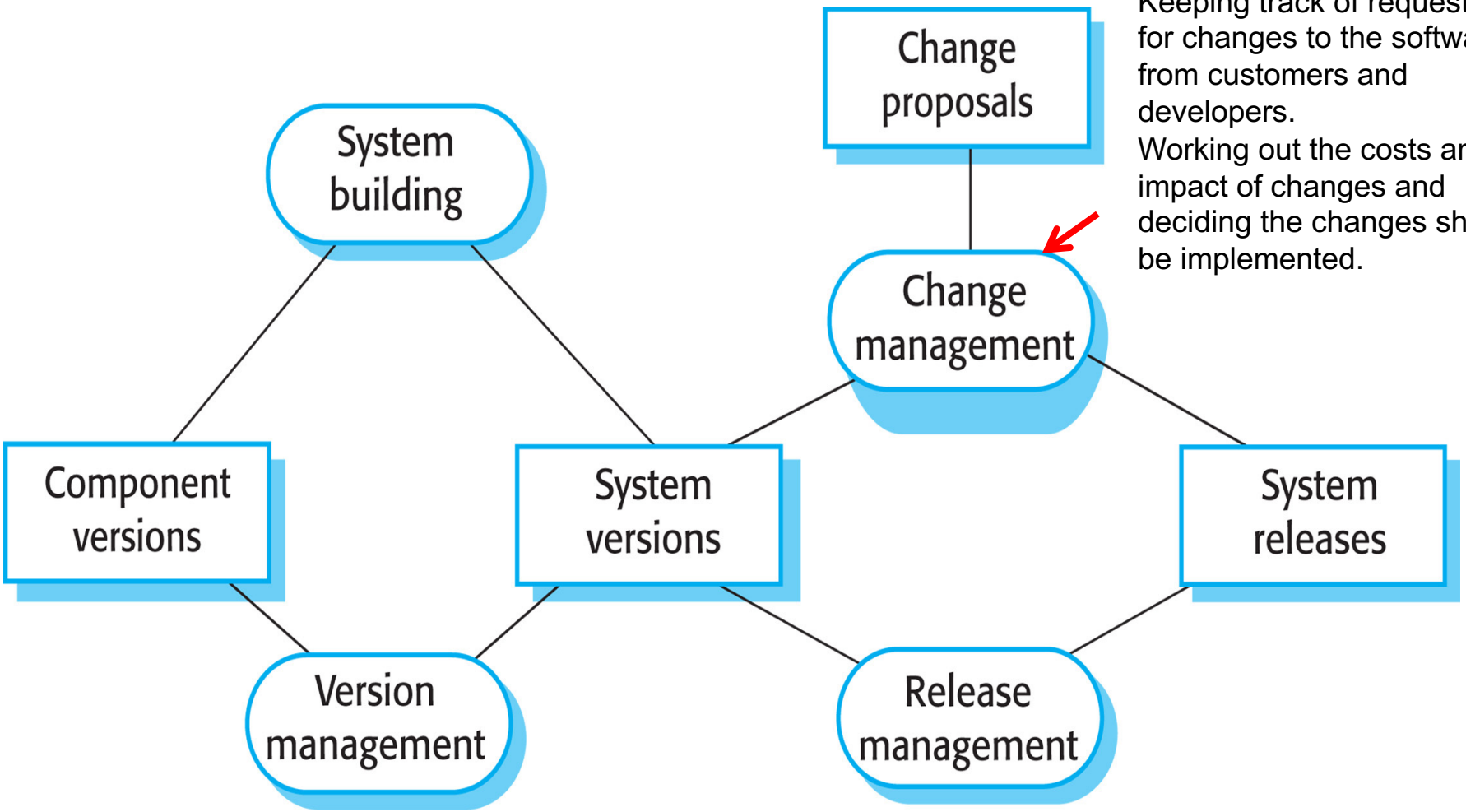
Implementation Issues

- The focus this week is not on programming—although this is obviously important and will be discussed next week—but on other implementation issues that are often not covered in programming texts, such as
  - Configuration management
  - Host-target development
  - Open source development
  - Software reuse (to be discussed next week)

Configuration Management

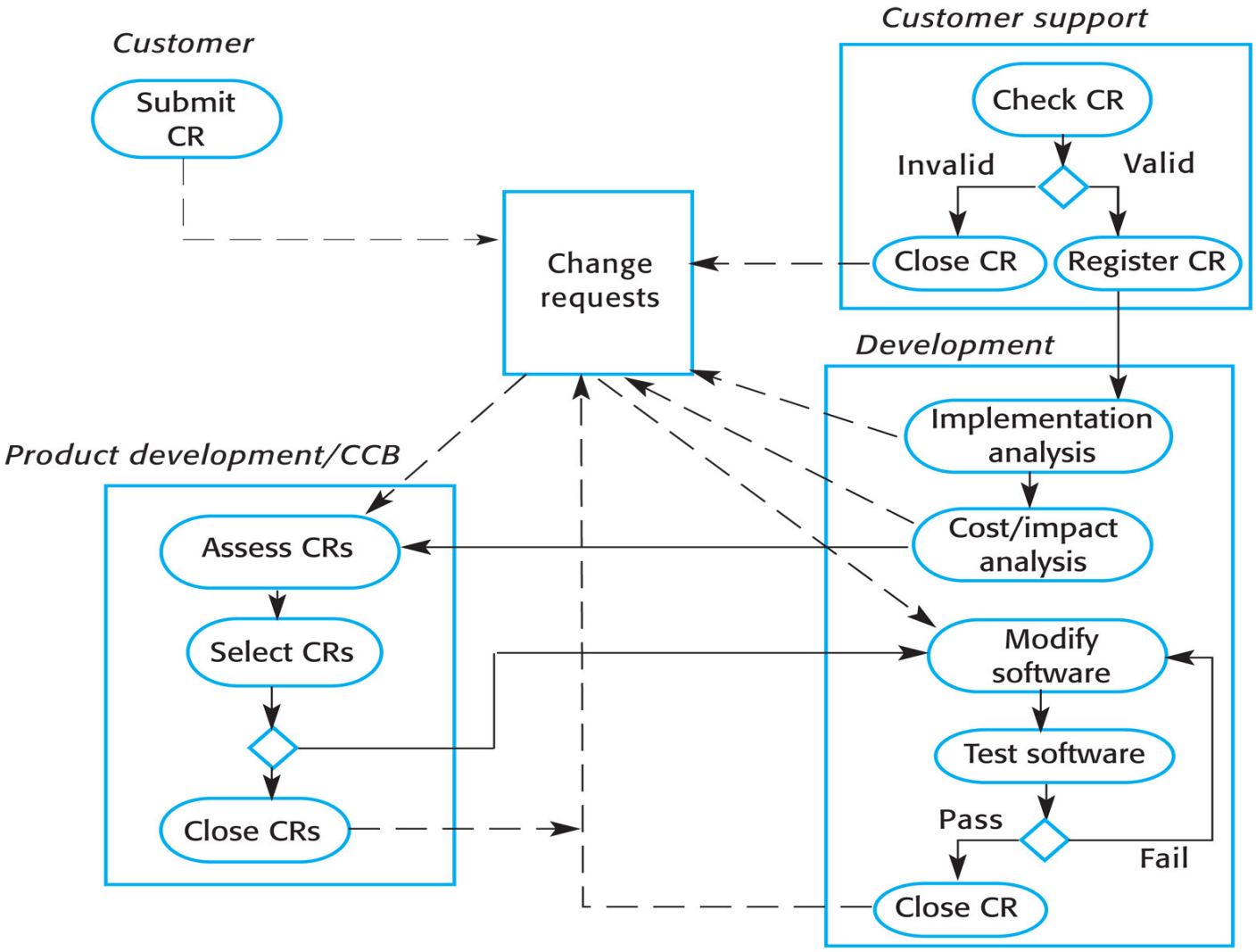
- Software systems are constantly changing during development and use. Configuration management (CM) is the name given to the general process of managing changing software systems.
  - You need CM because it is easy to lose track of what changes have been incorporated into each system version.
  - CM is essential for team projects to control changes made by different developers.
- The aim of configuration management is to support the system integration process so that all developers can access the project code and documents in a controlled way, find out what changes have been made, and compile and link components to create a system.

CM Activities

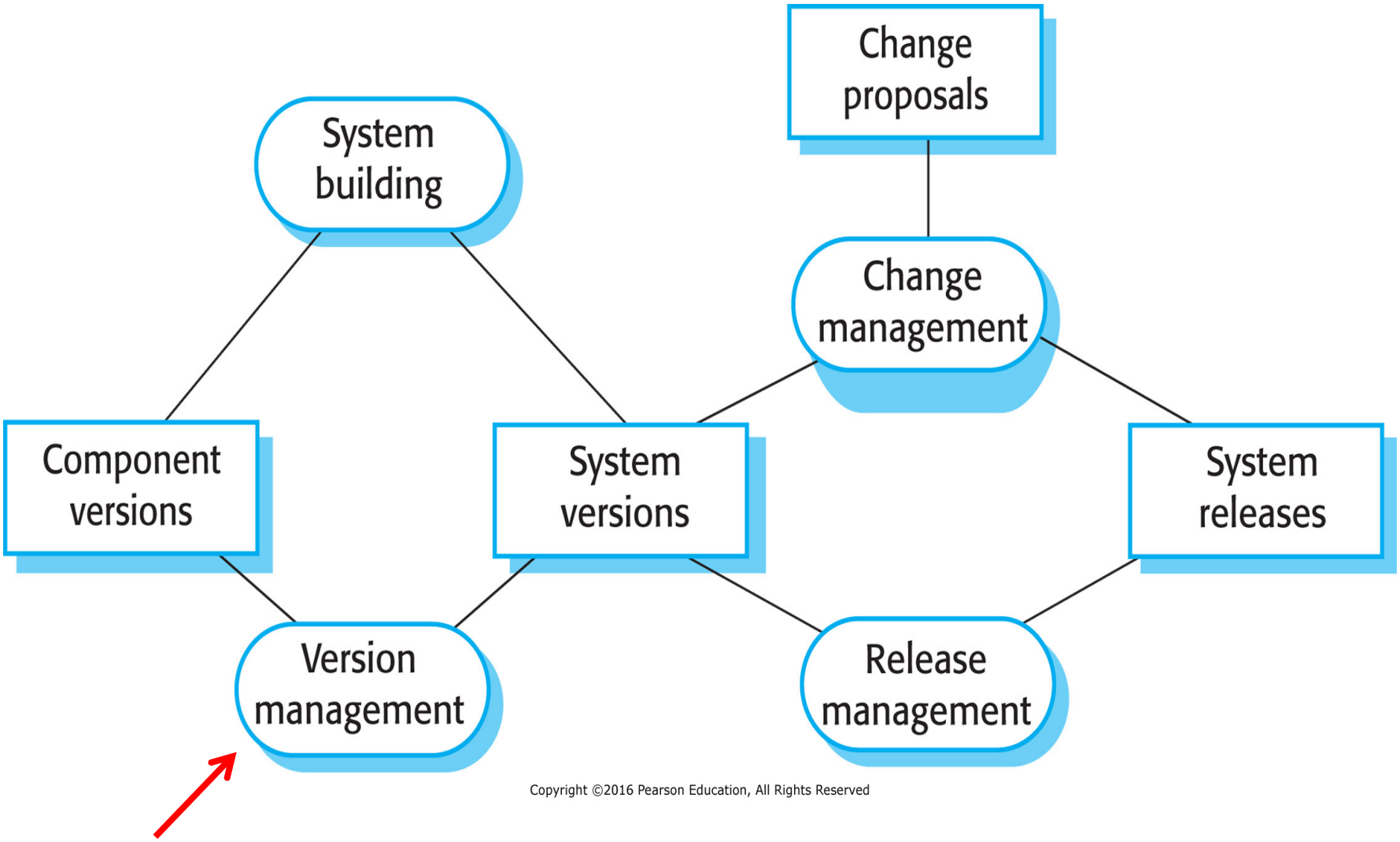


Keeping track of requests for changes to the software from customers and developers.
Working out the costs and impact of changes and deciding the changes should be implemented.

The Change Management Process

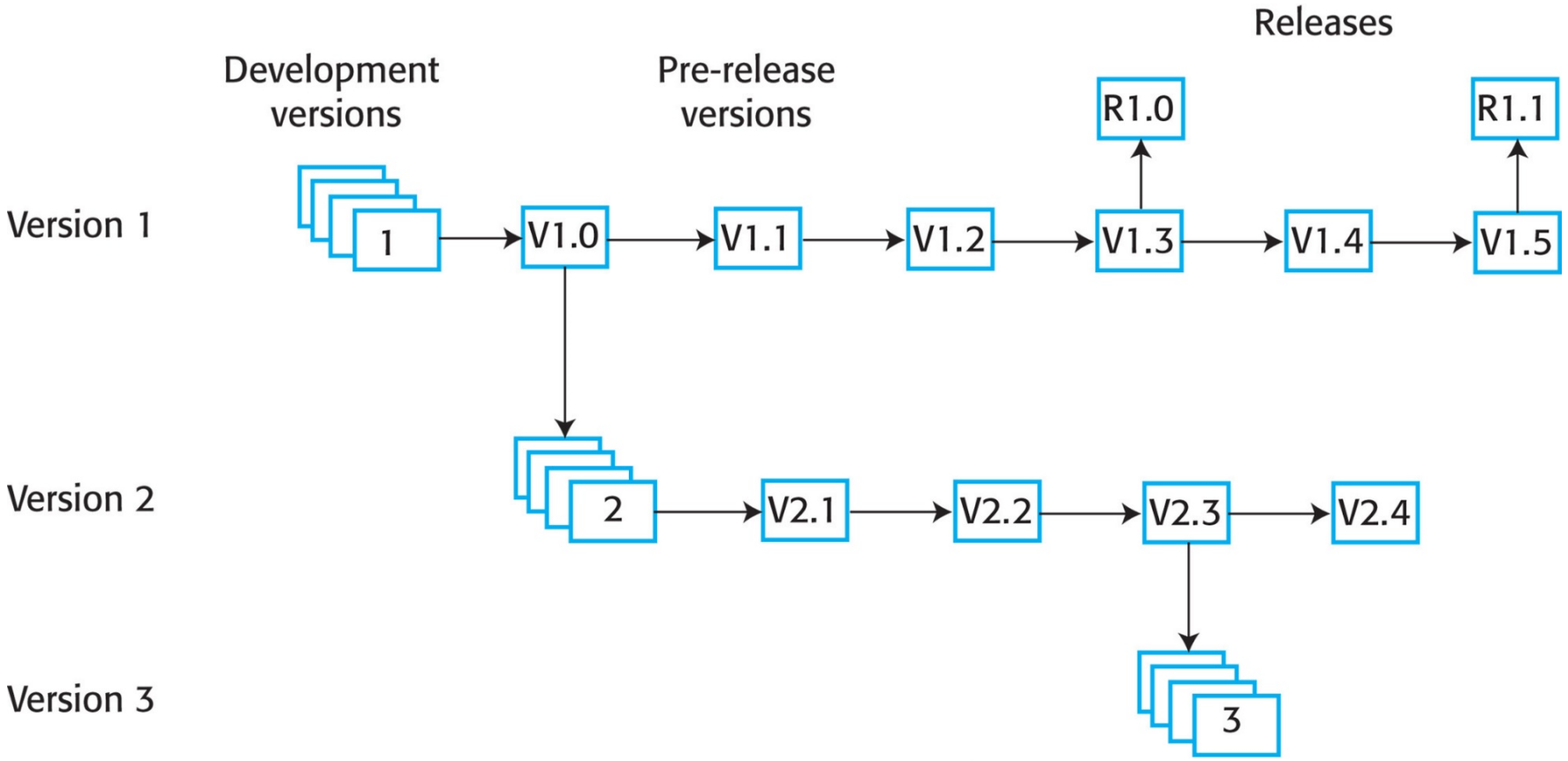


CM Activities

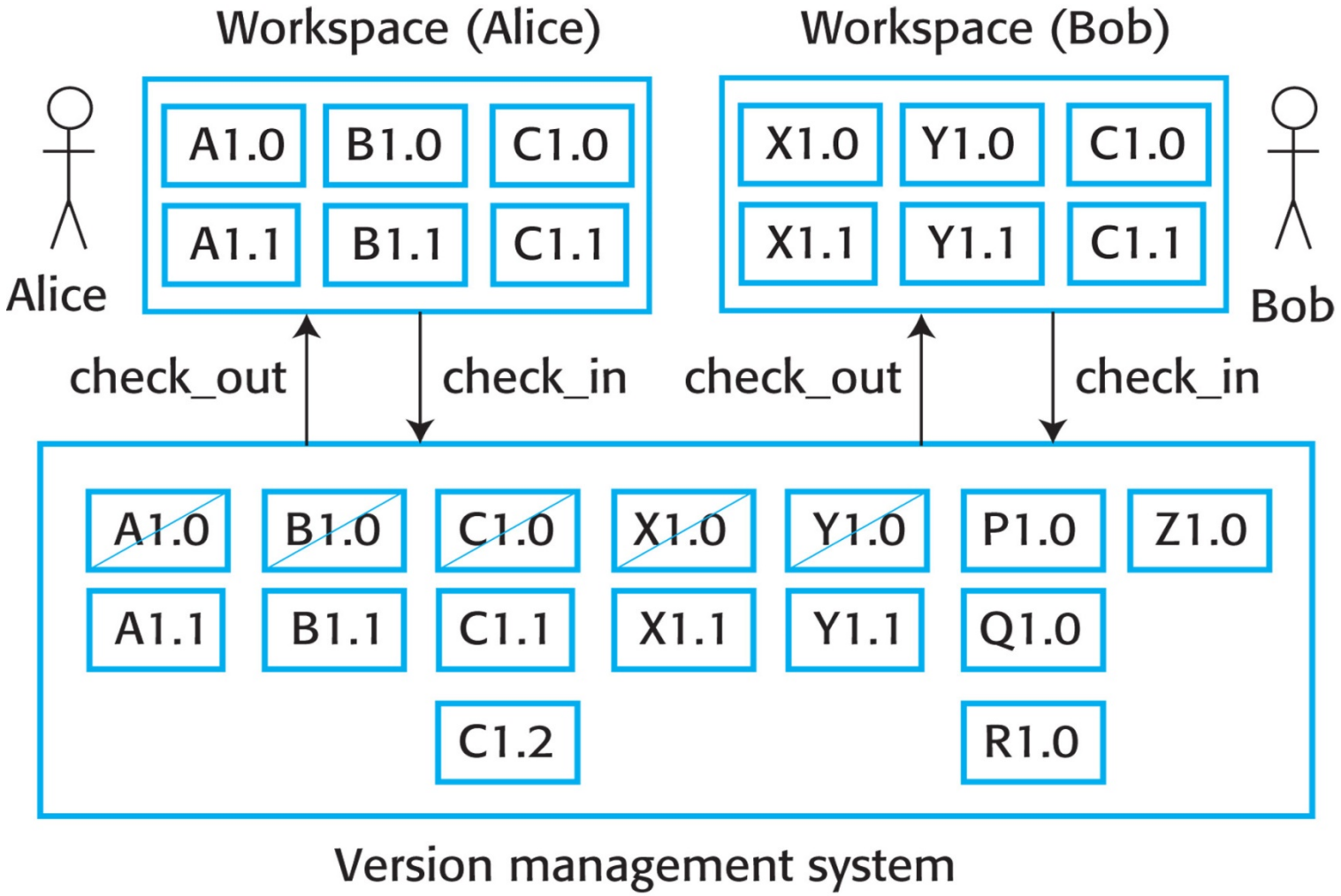


Version Management

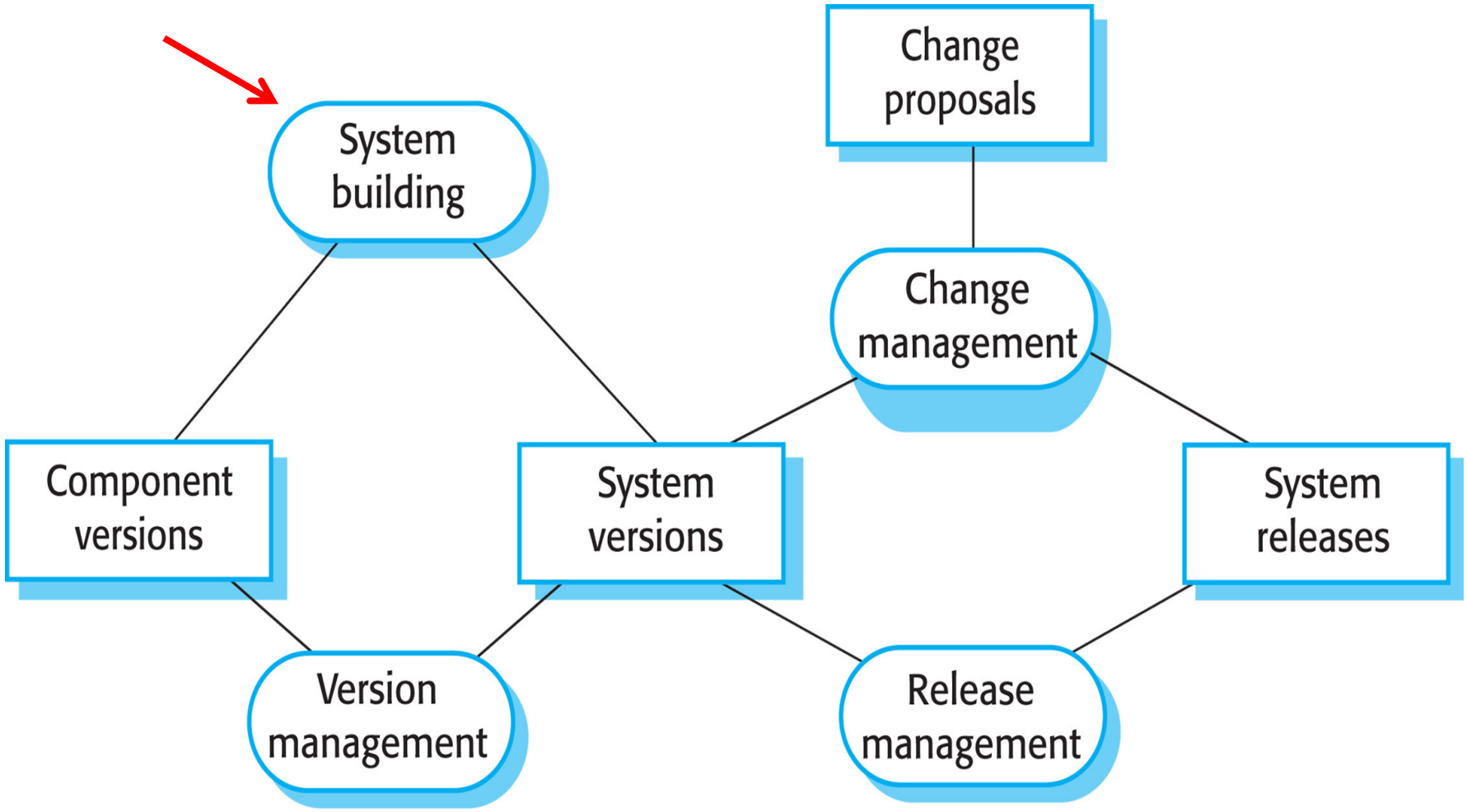
- Keeping track of the multiple versions of system components and ensuring that changes made to components by different developers do not interfere with each other



Version Management

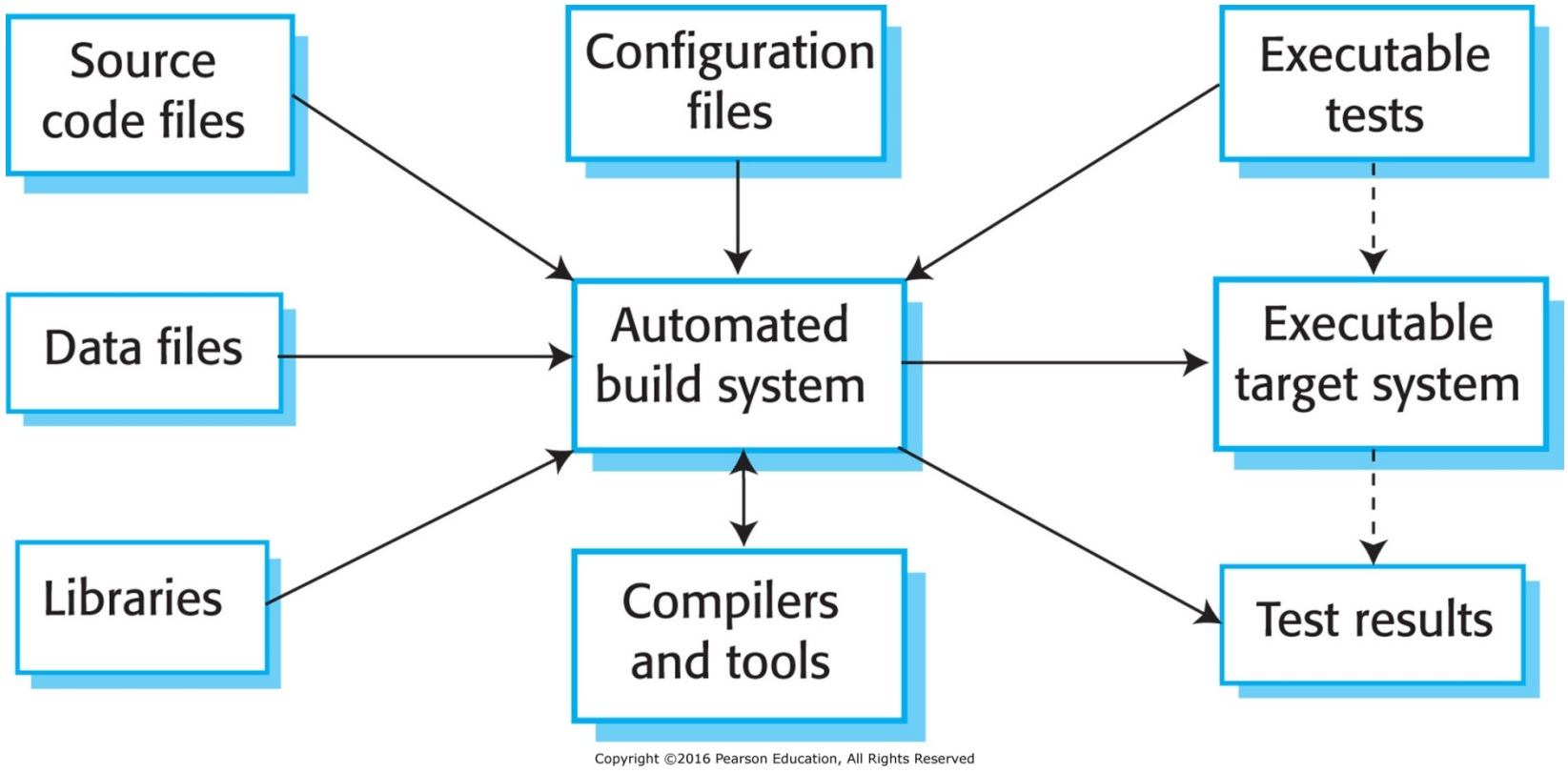


CM Activities



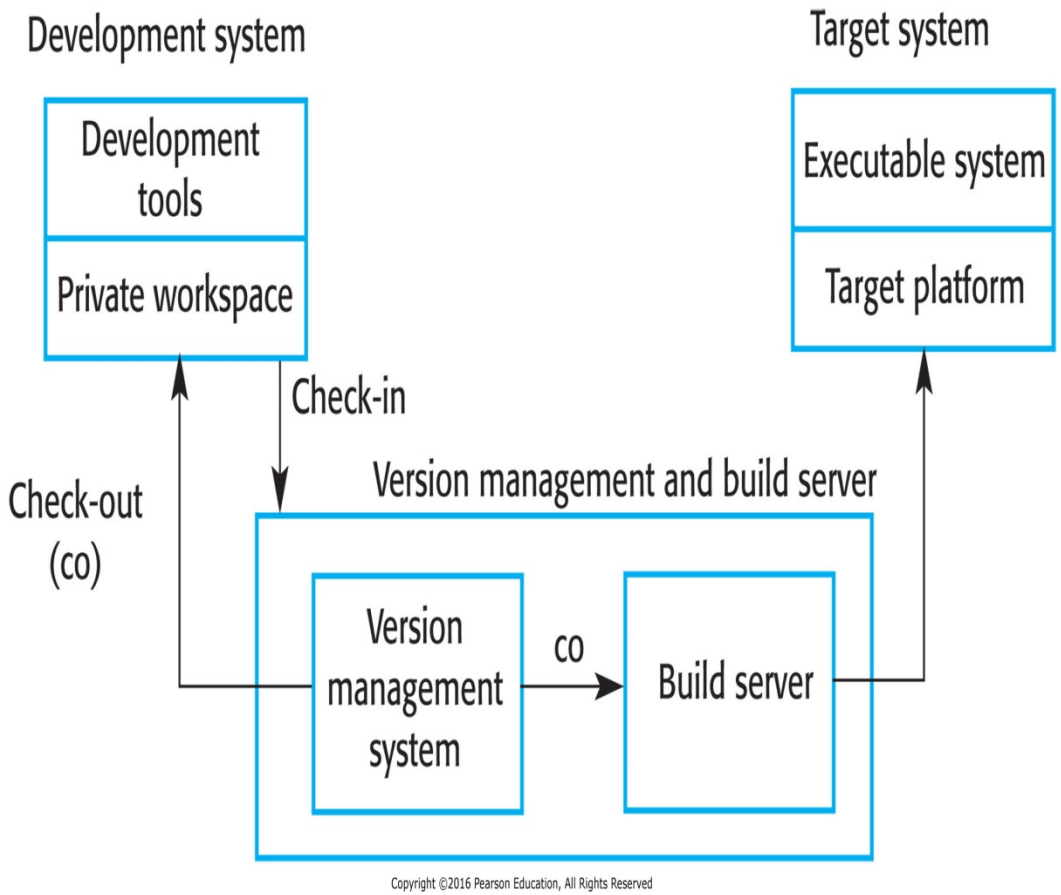
System Building

- System building is the process of creating a complete, executable system by compiling and linking the system components, external libraries, configuration files, and so on.

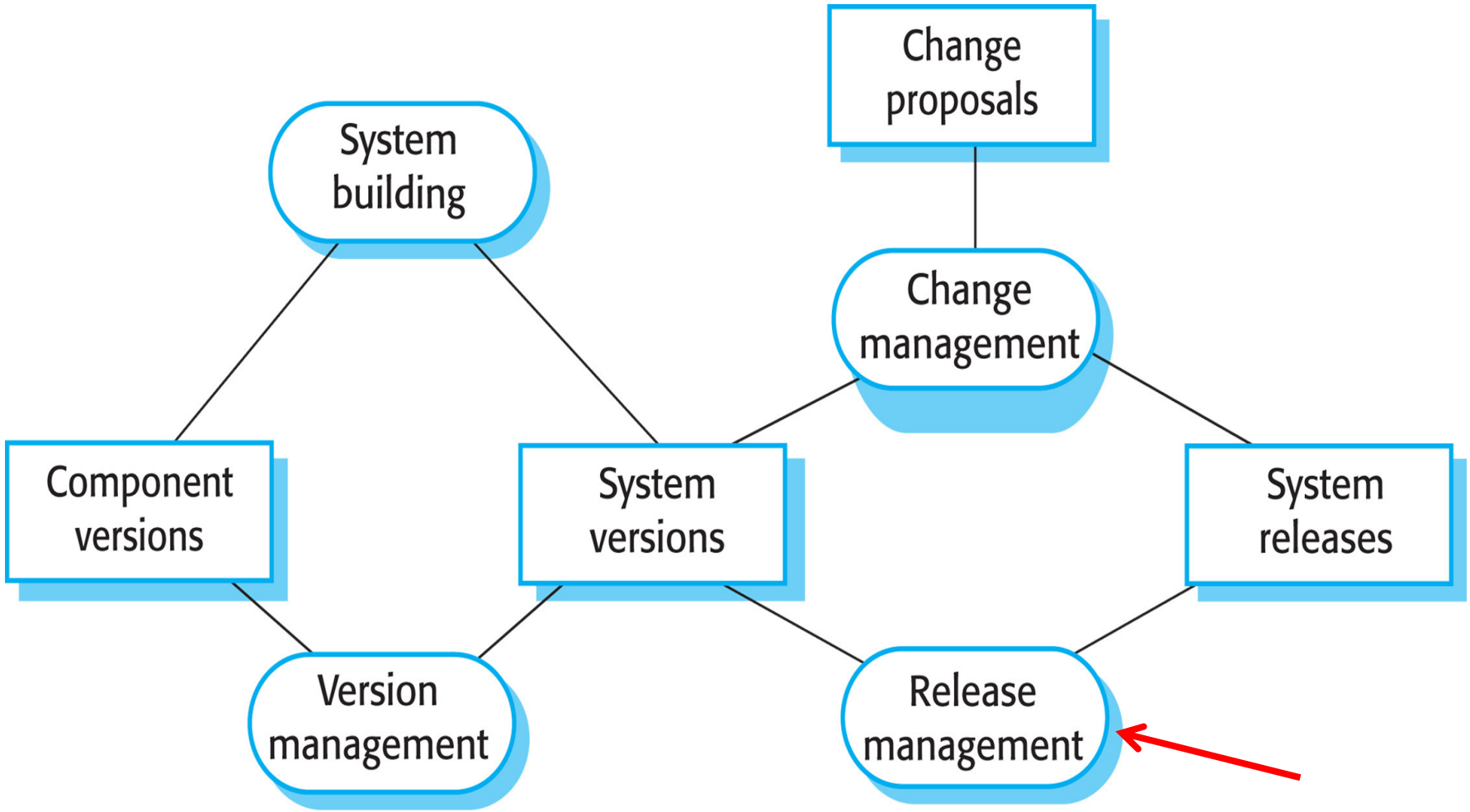


Build System Functionality

- Build script generation
- Version management system integration
- Minimal recompilation
- Executable system creation
- Test automation
- Reporting
- Documentation generation



CM Activities



Release Management

- Release management is the process of preparing software for external release and keeping track of the system versions that have been released for customer use.
- A system release is a version of a software system that is distributed to customers.
  - For mass market software, it is usually possible to identify two types of release: **major releases**, which deliver significant new functionality, and **minor releases**, which repair bugs and fix customer problems that have been reported.
  - For custom software or software product lines, releases of the system may have to be produced for each customer, and individual customers may be running several different releases of the system at the same time.

Release Components

- As well as the executable code of the system, a release may also include:
  - **Configuration files** defining how the release should be configured for particular installations
  - **Data files**, such as files of error messages, that are needed for successful system operation
  - **An installation program** that is used to help install the system on target hardware
  - Electronic and paper **documentation** describing the system
  - Packaging and associated **publicity** that have been designed for that release

Factors Influencing System Release Planning

Factor	Description
Competition	For mass-market software, a new system release may be necessary because a competing product has introduced new features and market share may be lost if these are not provided to existing customers.
Marketing requirements	The marketing department of an organization may have made a commitment for releases to be available at a particular date. For marketing reasons, it may be necessary to include new features in a system so that users can be persuaded to upgrade from a previous release.
Platform changes	You may have to create a new release of a software application when a new version of the operating system platform is released.
Technical quality of the system	If serious system faults are reported that affect the way in which many customers use the system, it may be necessary to correct them in a new system release. Minor system faults may be repaired by issuing patches, distributed over the Internet, which can be applied to the current release of the system.

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Host-Target Development and Open Source Development

Week 8: Design and Implementation

Edmund Yu, PhD
Associate Professor
esyu@syr.edu

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE

Host-Target Development

- Most software is developed on one computer (the host) but runs on a separate machine (the target).
- More generally, we can talk about a **development platform** and an **execution platform**.
  - A platform is more than just hardware.
  - It includes the installed operating system plus other supporting software such as a database management system or, for development platforms, an interactive development environment.
- Development platform usually has different installed software than execution platform.
  - They may even have different architectures.

Development Platform Tools

- A syntax-directed **editor** and an integrated **compiler** that allows you to create, edit and compile code
- A **debugger**
- **Graphical editing tools**, such as tools to edit UML models
- **Testing tools**, such as **JUnit** that can automatically run a set of tests on a new version of a program
- **Project support tools** that help you organize the code for different development projects

IDEs

- Software development tools are often grouped to create an integrated development environment (IDE).
- An IDE is a set of software tools that supports different aspects of software development, within some common framework and user interface.
- An IDE may be created to support development in a specific programming language such as Java.
- Or, it may be an instantiation of a general-purpose IDE, with specific language-support tools.

Open Source Development

- Open source development is an approach to software development in which the source code of a software system is published and volunteers are invited to participate in the development process.
- Its roots are in the Free Software Foundation (www.fsf.org), which advocates that source code should not be proprietary but rather should always be available for users to examine and modify as they wish.
- Open source software extended this idea by using the Internet to recruit a much larger population of volunteer developers (users).

Open Source Systems

- The best-known open source product is, of course, the Linux operating system, which is widely used as a server system and, increasingly, as a desktop environment.

- Other important open source products are Java, the Apache web server, and the mySQL database management system.

Open Source Issues

- Should the product that is being developed make use of open source components?
- Should an open source approach be used for the software's development?

Open Source Business

- More and more product companies are using an open source approach to development.
- Their business model is not reliant on selling a software product but on selling support for that product.
- They believe that involving the open source community will allow software to be developed more cheaply and more quickly and will create a community of users for the software.

Open Source Licensing

- A fundamental principle of open source development is that source code should be freely available, but this does not mean that anyone can do as they wish with that code.
  - Legally, the developer of the code (either a company or an individual) still owns the code. They can place restrictions on how it is used by including legally binding conditions in an open source software license.
  - Some open source developers believe that if an open source component is used to develop a new system, then that system should also be open source.
  - Others are willing to allow their code to be used without this restriction. The developed systems may be proprietary and sold as closed source systems.

License Models

- The GNU General Public License (GPL) is a so-called "reciprocal" license which means that if you use open source software that is licensed under the GPL license, then you must make that software open source.
- The GNU Lesser General Public License (LGPL) is a variant of the GPL license where you can write components that link to open source code without having to publish the source of these components.
- The Berkley Standard Distribution (BSD) License is a nonreciprocal license, which means you are not obliged to republish any changes or modifications made to open source code. You can include the code in proprietary systems that are sold.

SYRACUSE UNIVERSITY
ENGINEERING & COMPUTER SCIENCE