STEVENS
INSTITUTE of TECHNOLOGY
THE INNOVATION UNIVERSITY®

SSW-540: Fundamentals of Software Engineering

Designing a Software System

Roberta (Robbie) Cohen, Ph.D.
Industry Professor
School of Systems and Enterprises

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 inter-leaved.
  - 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.

Build or buy?

- In a wide range of domains, it is now possible to buy off-the-shelf systems (COTS) that can be adapted and tailored to the users' requirements.
  - For example, if you want to implement a medical records system, you can buy a package that is already used in hospitals. It can be cheaper and faster to use this approach rather than developing a system in a conventional programming language.
- When you develop an application in this way, the design process becomes concerned with how to use the configuration features of that system to deliver the system requirements.
- Buying is a form of re-use which is not always desirable

Reuse advantages and disadvantages

- Advantages:
  - Reduced costs and risks as less software is developed from scratch
  - Faster delivery and deployment of system
- Disadvantages:
  - But requirements compromises are inevitable so system may not meet real needs of users
  - Loss of control over evolution of reused system elements

Rolling your own..

- Structured design processes involve developing a number of different system models.
- Once developed, they require a lot of effort for development and maintenance of these models
  - for small systems, this may not be cost-effective.
- However, for large systems developed by different groups, design models are an important communication mechanism.
- Design models specify the system's architecture at a more granular level than "architecting in the large".

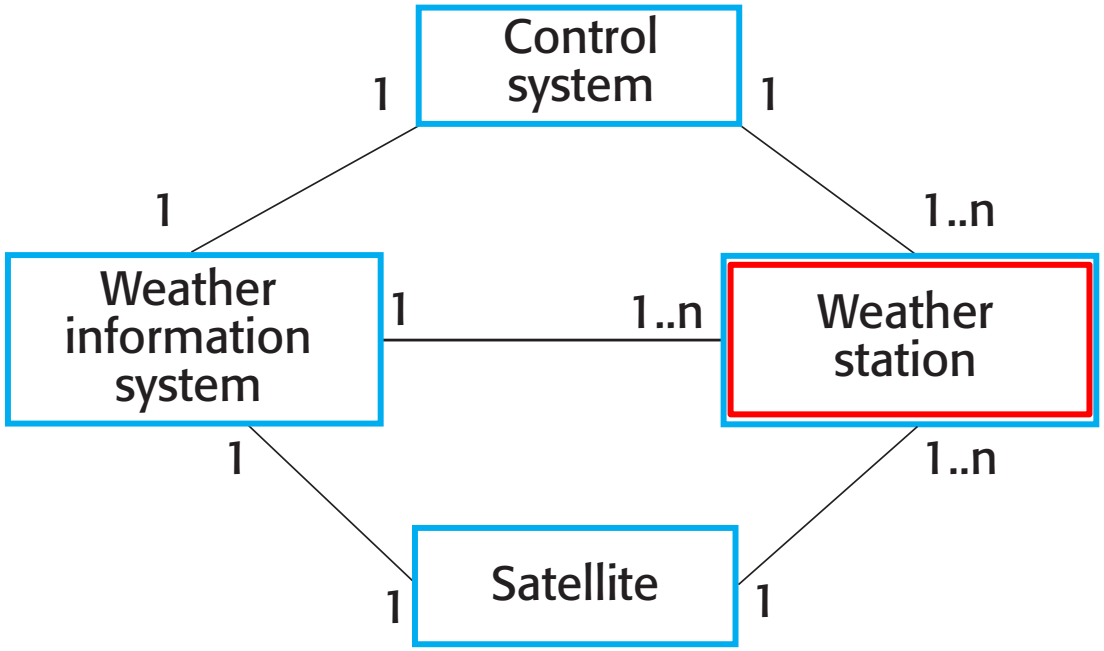
Object-oriented design process

- Object-oriented design is widely used for software systems.
- While there are a variety of different object-oriented design processes found in various organizations, common activities in these processes include:
  - Defining the context and modes of use of the system;
  - Designing the system architecture;
  - Identifying the principal system objects;
  - Developing design models;
  - Specifying object interfaces.
- Sommerville illustrates these process steps using a design for a wilderness weather station.
- Note those activities (above) that any design process would entail!

System context and interactions

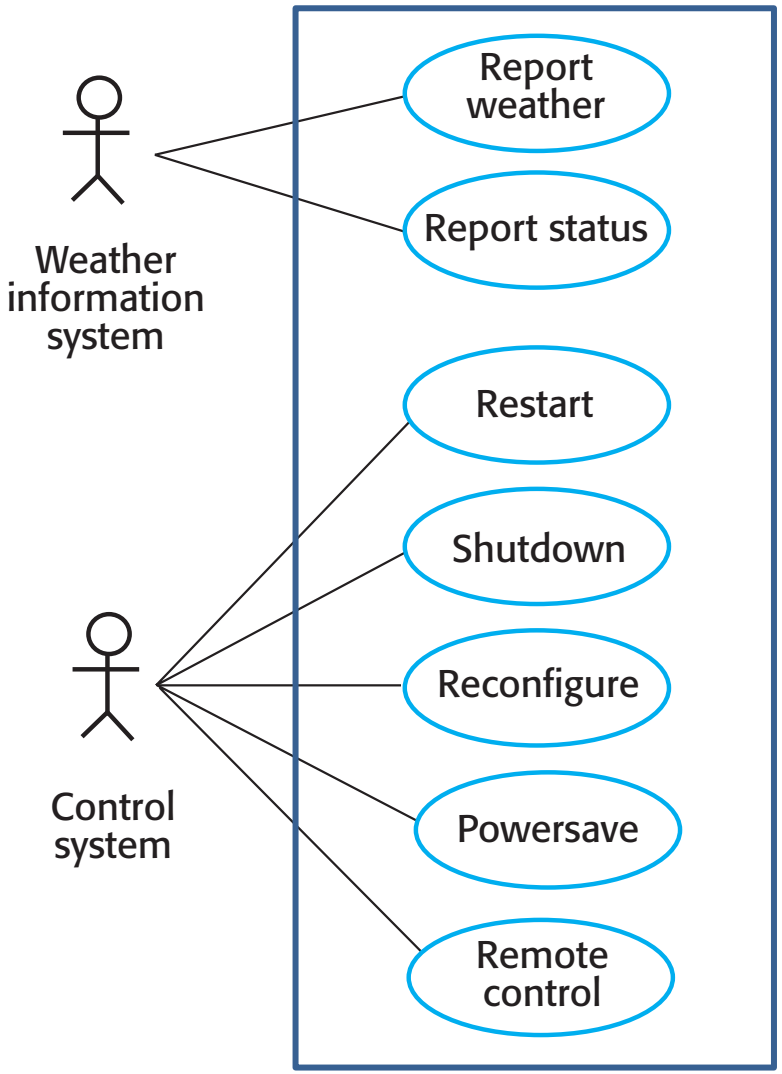
- Understanding the relationships between the software that is being designed and its external environment is essential for deciding how to provide the required system functionality and how to structure the system to communicate with its environment.
- Understanding of the context also lets you establish the boundaries of the system. Setting the system boundaries helps you decide what features are implemented in the system being designed and what features are in other associated systems.
- A system context model is a structural model that demonstrates the other systems in the environment of the system being developed.
- An interaction model is a dynamic model that shows how the system interacts with its environment as it is used.

System context for the weather station in a structural model



Weather station use cases

- The Weather Station is inside the box!
- Note that the weather station interacts with the Weather Information System and the Control System.
- This diagram shows use cases, but also provides a dynamic context view.



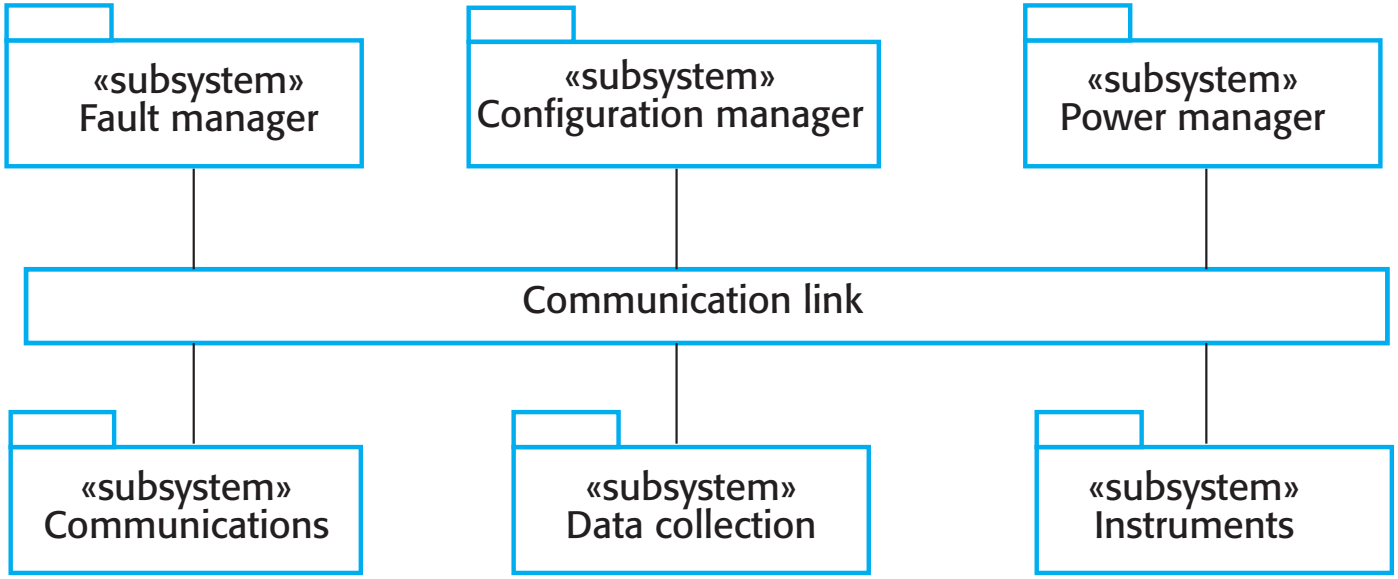
Use case description—Report weather

System	Weather station
Use case	Report weather
Actors	Weather information system
Description	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 five-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 the future.

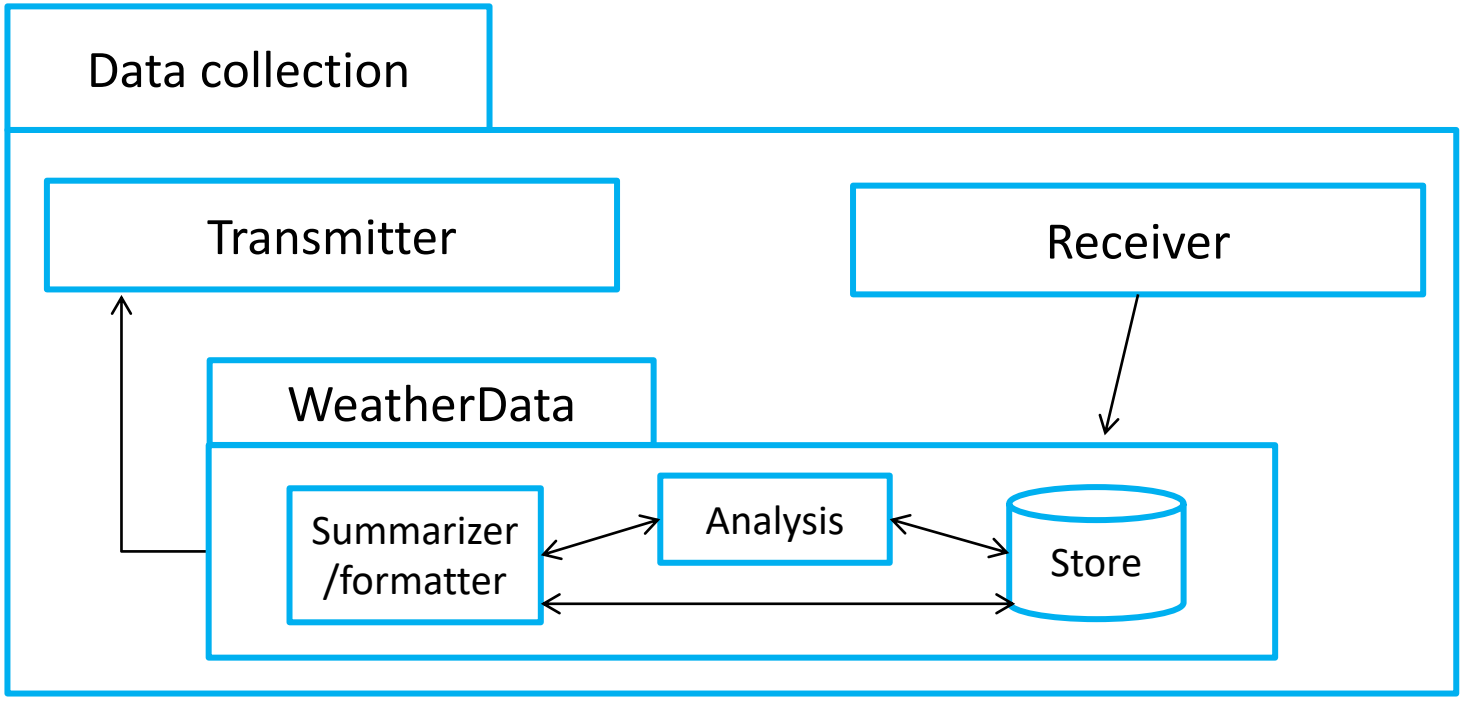
Architectural design

- Once interactions between the system and its environment have been understood, you use this information for designing the system architecture.
- You identify the major components that make up the system and their interactions, and then you may organize the components using an architectural pattern such as a layered or client-server model.

Weather Station: hi-level view



Architecture of data collection sub-system

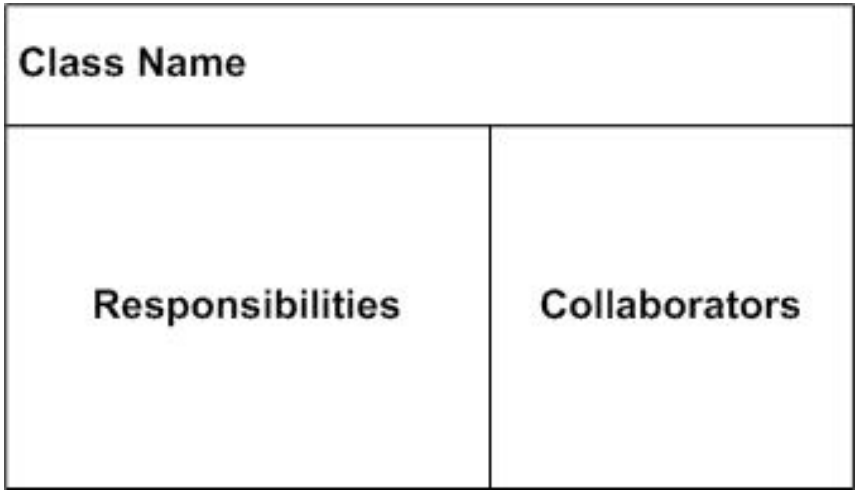


Object class identification

- In object-oriented design, object classes know things and know how to do things!
- Identifying object classes is often the most difficult part of object-oriented design.
  - There is no 'magic formula' for object identification. It relies on the skill, experience and domain knowledge of system designers.
  - Object identification is an iterative process. Try, and iterate!
- Approaches to identifying objects
  - Use a grammatical approach based on a natural language description of the system.
  - Base the identification on tangible things in the application domain.
  - Use a behavioural approach and identify objects based on what participates in what behaviour.
  - Use a scenario-based analysis. The objects, attributes and methods in each scenario are identified.

CRC card for identifying objects and classes

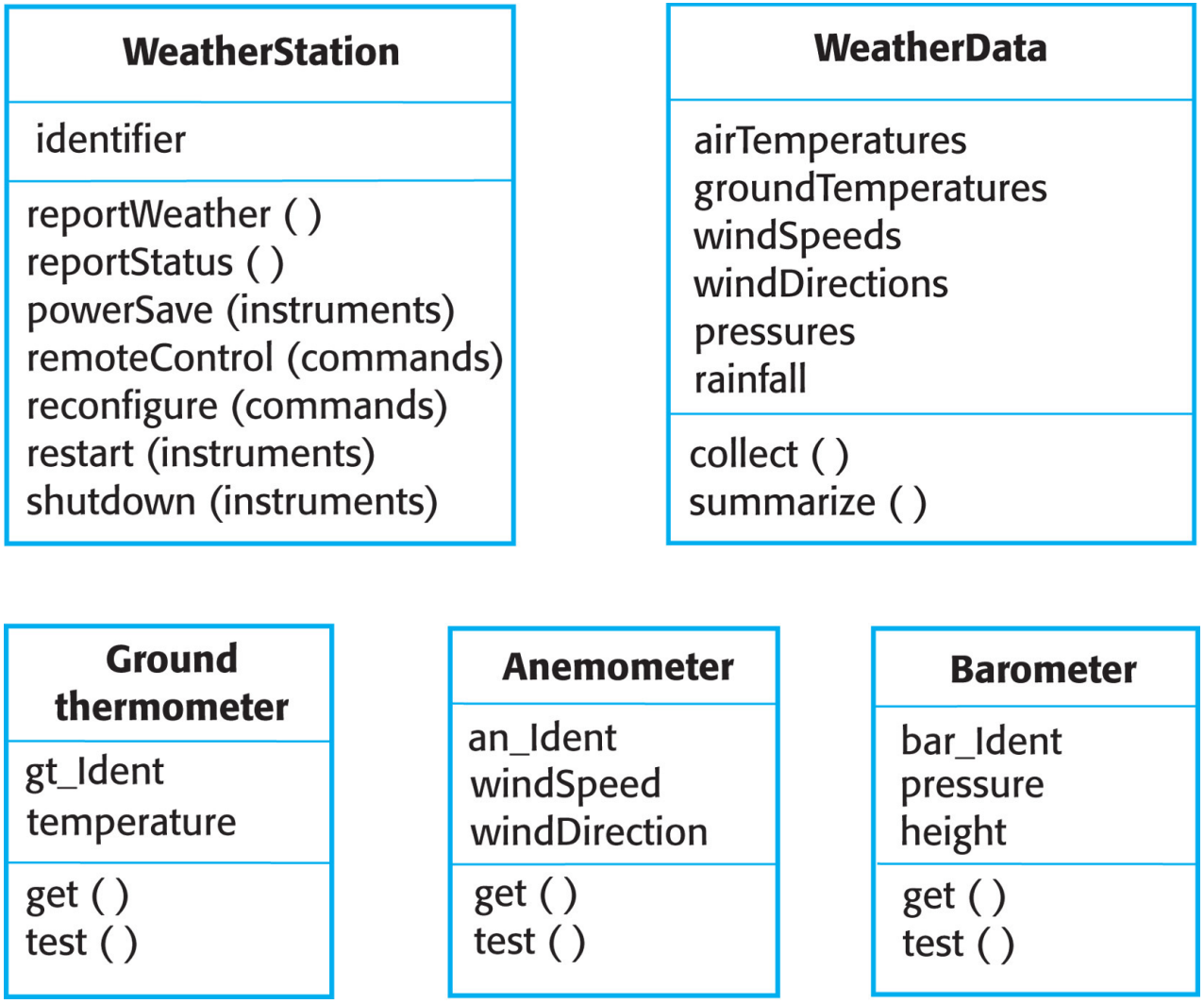
- Class-responsibility-collaboration (**CRC**) **cards** are a brainstorming tool used in the design of object-oriented software.
- The Class – the name of the object class
- The Responsibilities – what the class must know and do
- The Collaborations – the other classes that help the class accomplish its responsibilities



Weather station object classes

- Object class identification in the weather station system may 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



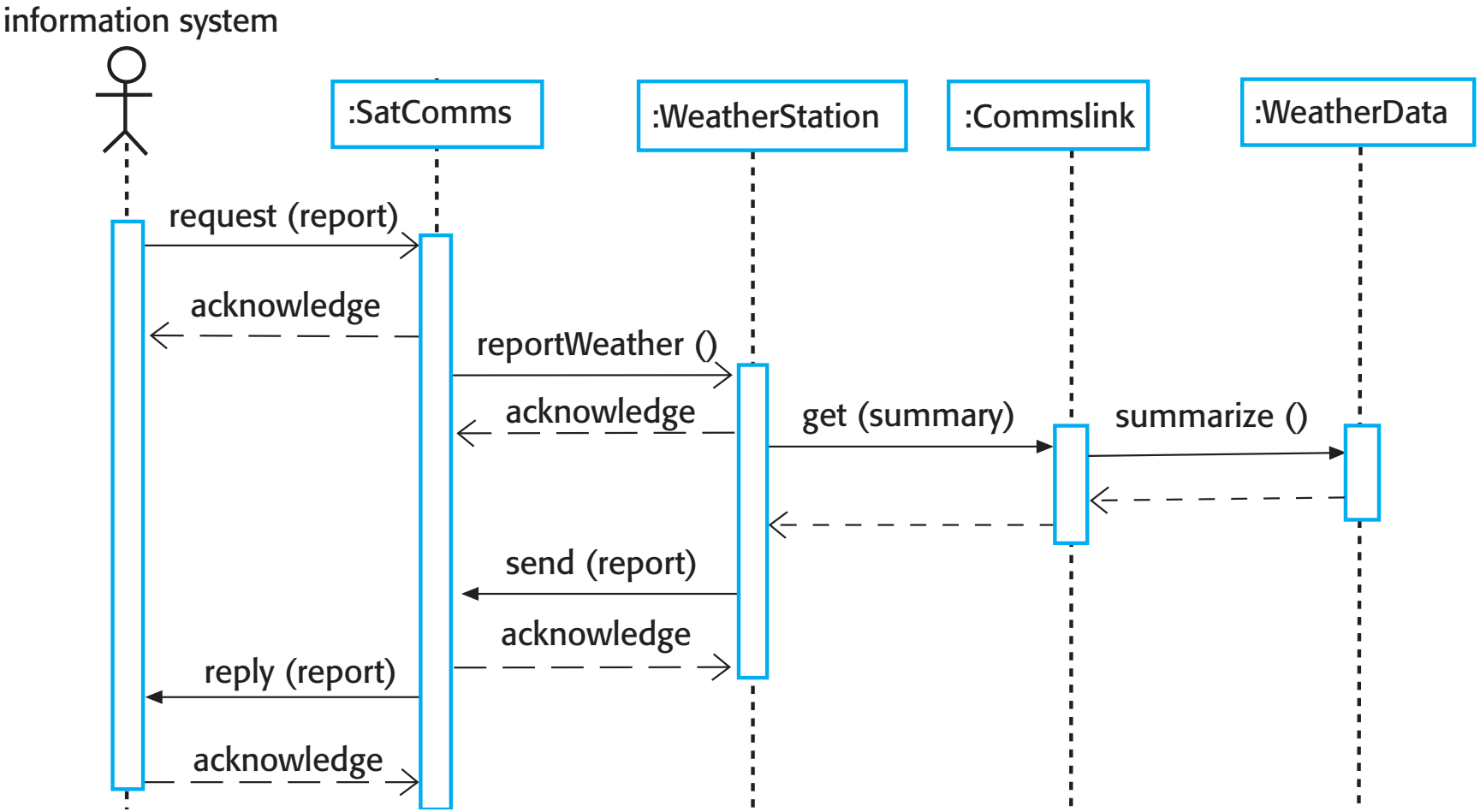
Design models

- Design models show the objects and object classes and relationships between these entities.
- There are two kinds of design model:
  - Structural models describe the static structure of the system in terms of object classes and relationships.
  - Dynamic models describe the dynamic interactions between objects.

Examples of design models

- Subsystem models that show logical groupings of objects into coherent subsystems.
  - Show how the design is organized into logically related groups of objects.
  - In the UML, these are shown using packages - an encapsulation construct. This is a logical model, not an actual organization of objects.
- Sequence models that show the sequence of object interactions.
  - Objects are arranged horizontally across the top;
  - Time is represented vertically so models are read top to bottom;
  - Interactions are represented by labelled arrows, Different styles of arrow represent different types of interaction;
  - A thin rectangle in an object lifeline represents the time when the object is the interacting object in the system.
- State machine models and other models including use-case models, aggregation models, generalization models, etc.

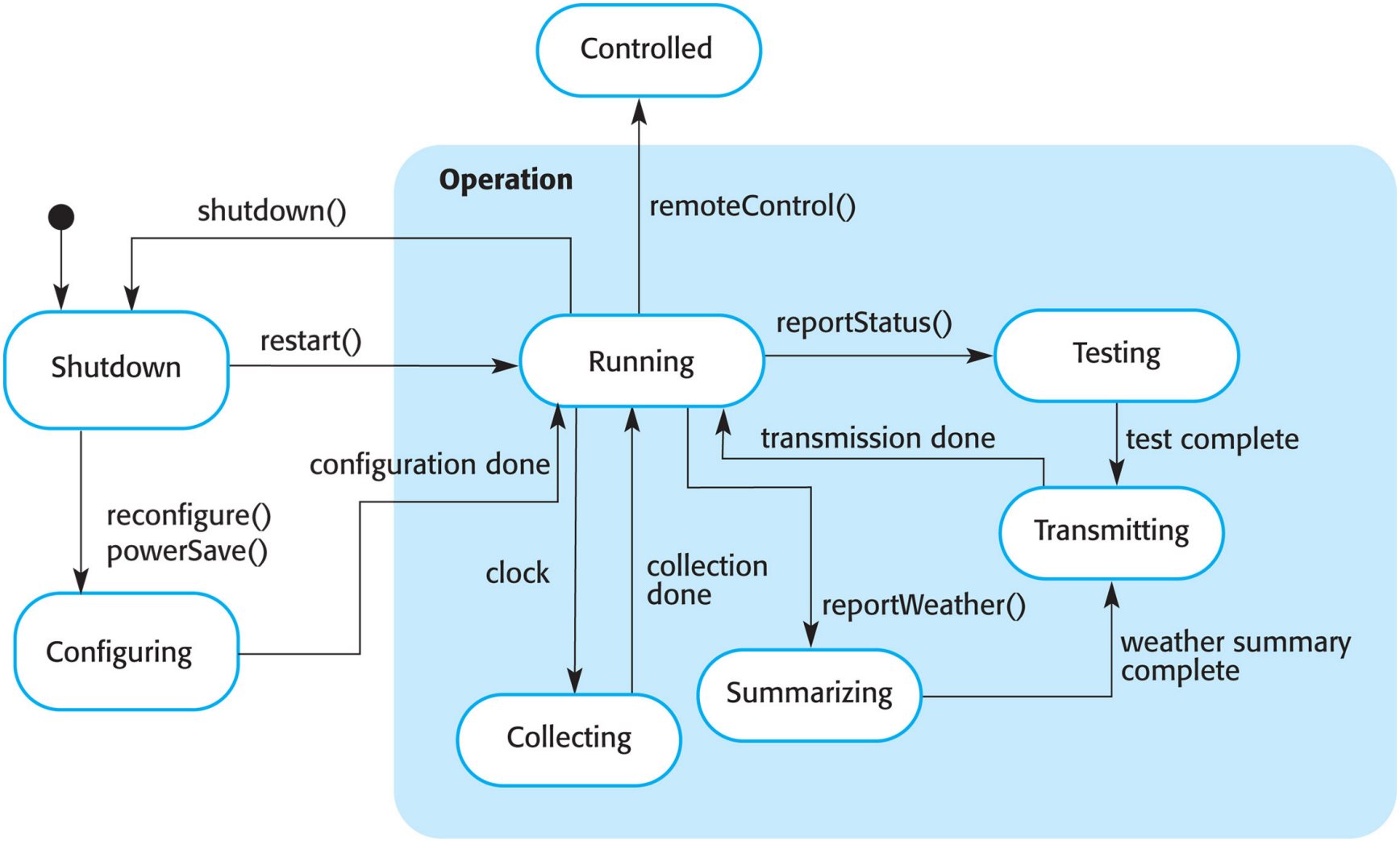
Sequence diagram describing data collection



State diagrams

- State diagrams are used to show how objects respond to different service requests and the state transitions triggered by these requests.
- State diagrams are useful high-level models of a system or an object's run-time behavior.
- You usually don't need a state diagram for all of the objects in the system. Many of the objects in a system are relatively simple and a state model adds unnecessary detail to the design.
- State models are very useful in modeling systems that monitor and respond to events.

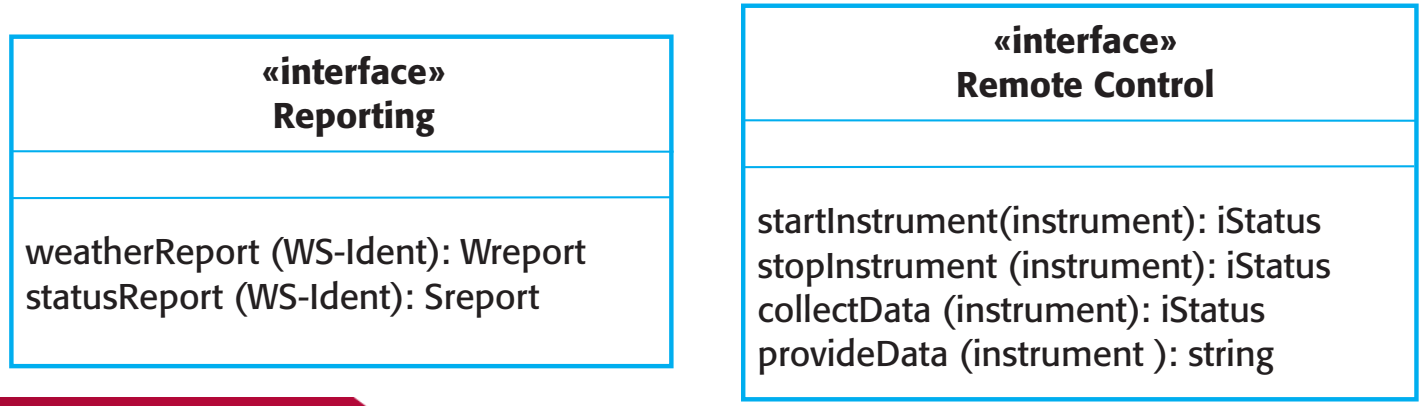
Weather station state diagram



Interface specification

- Object interfaces have to be specified so that the objects and other components can be designed in parallel.
- Designers hide the interface data representation in the object (interfaces are defined to have no “state”/attributes, only methods)
- Objects may have several interfaces which are viewpoints for the object’s methods
- The UML class diagrams are used for interface specification (and sequence diagrams are often used to specify interface detail).

Class diagrams



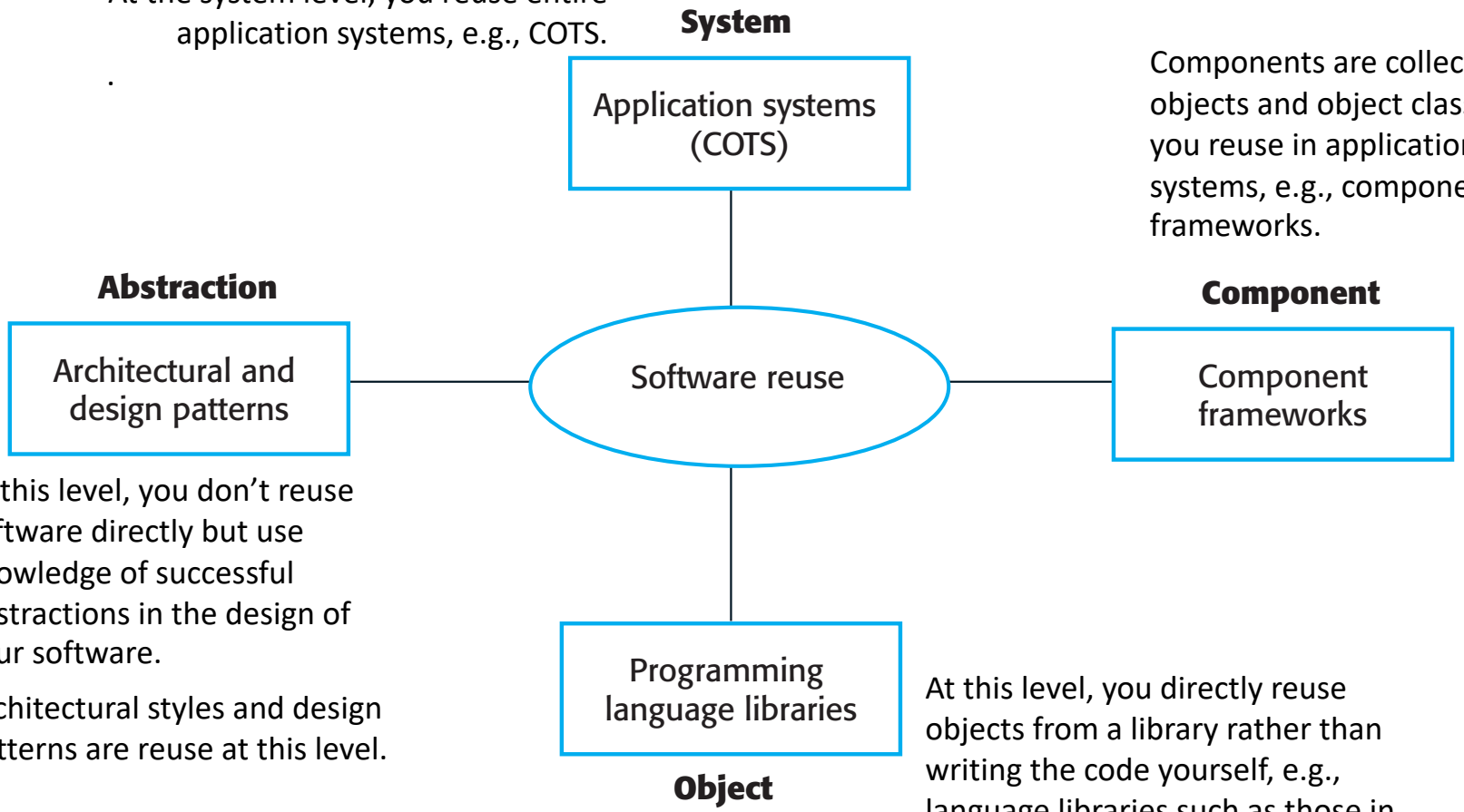
Design reuse

- Architectures and designs, especially interfaces are ripe for reuse
- From the 1960s to the 1990s, most new software was developed from scratch, by writing all code in a high-level programming language.
  - The only significant reuse or software was the reuse of functions and objects in programming language libraries.
- Costs and schedule pressure made this approach increasingly unviable, especially for commercial and Internet-based systems.
- Software architectures and designs today are widely reused—we refer to this as reusable patterns.

Levels of software reuse

At the system level, you reuse entire application systems, e.g., COTS.

.



Components are collections of objects and object classes that you reuse in application systems, e.g., component frameworks.

At this level, you don't reuse software directly but use knowledge of successful abstractions in the design of your software.

Architectural styles and design patterns are reuse at this level.

At this level, you directly reuse objects from a library rather than writing the code yourself, e.g., language libraries such as those in Python

Design patterns

- Patterns are a means of representing, sharing and reusing knowledge.
- An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.
- Patterns appear “obvious” because they so often appear!
- Patterns include information about when they are and when they are not useful.
- Patterns may be represented using tabular and graphical descriptions.

How patterns play out

- We have already seen how different kinds of software applications lend themselves to different styles of architectures.
- Design patterns, popularized by the book "*Design Patterns: Elements of Reusable Object-Oriented Software*" (1994) by the "Gang of Four", provide solutions to commonly recurring problems in software design.

Design patterns

- Analogous to architecture styles, a design pattern is a way of reusing abstract knowledge about a specific problem and its solution.
- A pattern is a description of the problem and the essence of its solution.
- It should be sufficiently abstract to be reused in different settings.
- Pattern descriptions often make use of object-oriented characteristics such as inheritance.
  - Inheritance means that any sub-class can inherit the attributes and behaviors of the more general class (or classes). What is good for the parent is good for the child!

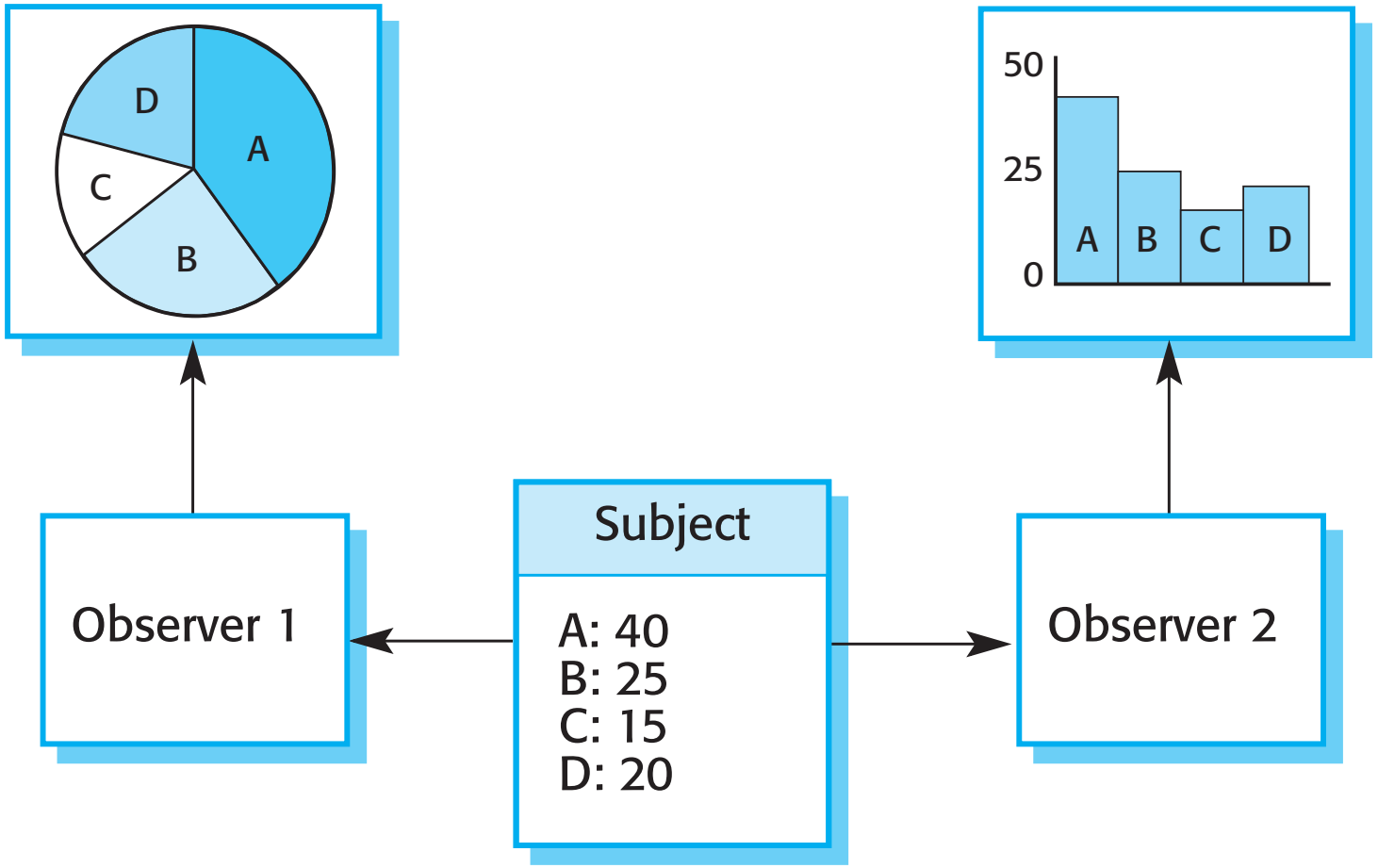
Patterns and pattern elements

- *Patterns and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience.*
- Pattern Elements include:
  - Name
    - A meaningful pattern identifier.
  - 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.

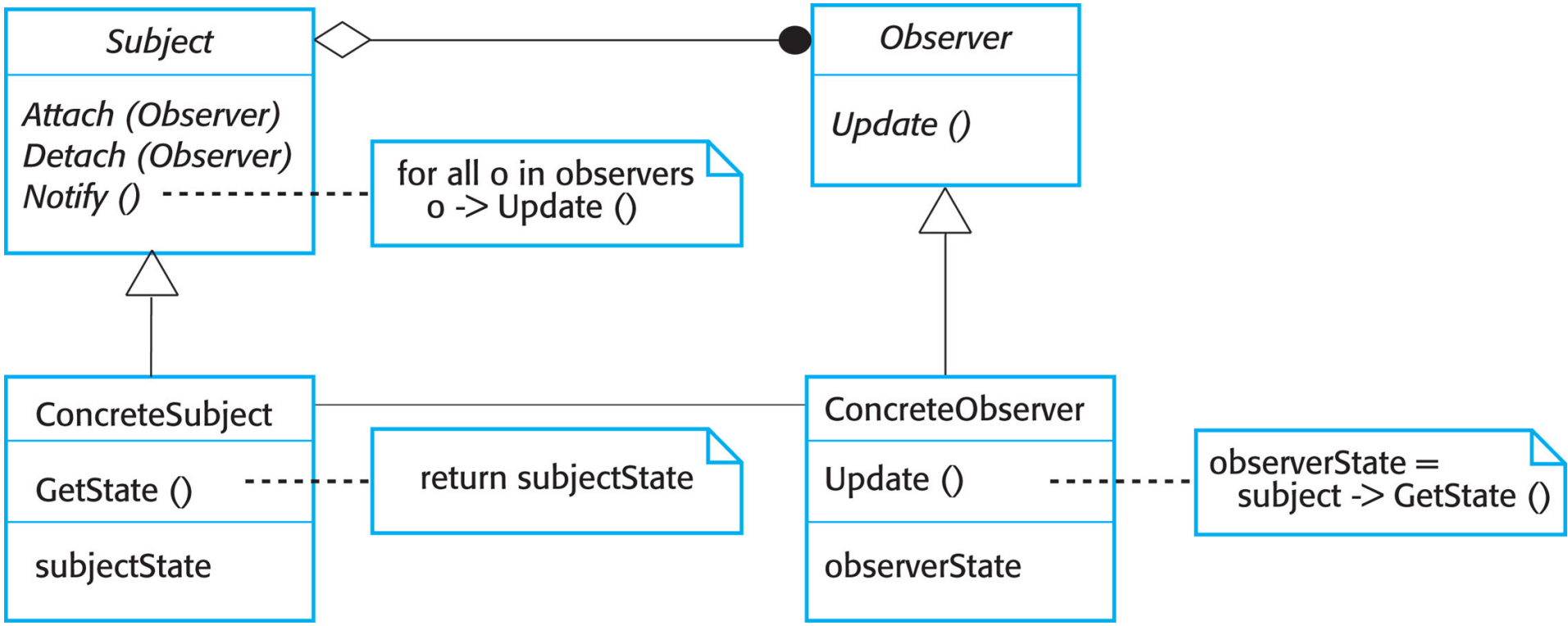
Observer

- Pattern Name - observer, aka dependents, publish-subscribe
- Problem - Problem of maintaining consistency between/among related objects.
- Solution - a subject (object being observed) and one or more observers are implemented and all observers are notified when subject changes state, and then each observer synchronizes with subject. Subject is publisher, observers subscribe to the notifications.
- Consequences - can vary subjects and observers independently, loosely coupled, supports broadcast. As more observers subscribe to a subject, it can be costly changing the subject. Also if it is a simple broadcast (stating simply that a change has occurred but not specifying the change), observers may have significant work finding what has changed.

Multiple displays using the Observer pattern



A UML model of the Observer pattern



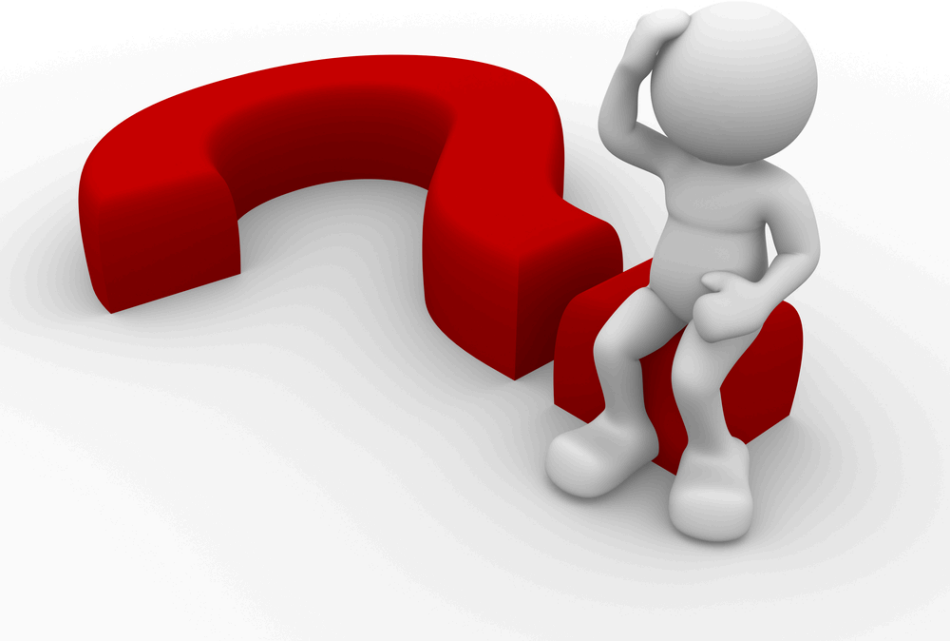
Design problems

- 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. E.g.,
  - 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).
- Many design patterns exist. See catalog at http://c2.com/cgi/wiki?SoftwareDesignPatternsIndex.

Summary

- The process of object-oriented design includes activities to design the system architecture, identify objects in the system, describe the design using different object models and document the component interfaces.
- When developing software, you should always consider the possibility of reusing existing software, either as components, services or complete systems.
- Design patterns are commonly reused.

Questions?



Python Pearl: Unqualified except

Avoid unqualified except, e.g.:

```
1  inp = input ("Enter an integer between 1 and 20: ")
2  try:
3      n = int(inp)
4  except:  #DON'T USE UNQUALIFIED except STATEMENTS !!!
5      print (f"{inp} is not an integer")
6  else:
7      print (f"{n} + 1 == {n + 1}")
```

This block is invoked for ANY exception, not just the ones you expect

Unlike other languages with exceptions, Python uses exceptions for many situations, e.g.:

- ImportError
- IndexError
- NameError
- TypeError
- ValueError
- ZeroDivisionError
- IOError
- …

What was the name of that exception?

When in doubt, try it out…

What's the name of the exception for divide by 0?

```
3/0

ZeroDivisionError: division by zero
```


```
try:
        quotient = inp / 0
except ZeroDivisionError:
    print ("Oops! Divide by 0 doesn't work")
```