Object-Oriented Programming

Software Design

Michael L. Collard, Ph.D.

Department of Computer Science, The University of Akron

Scale of Software Systems

Layers of a System

- System
- Subsystem
- Namespace/Directory
- Class/File
- Method/Function
- Statements

System Layer

Example: A Weather Monitoring System

Collect, process, and analyze weather data from various sources and provide weather forecasts and alerts

- Very high-level
- Typically, the purpose of the real-world project
- The ultimate reason that you get paid

Subsystem Layer

***Data Collection Subsystem** Gather weather data from sensors and external APIs*

***Data Processing Subsystem** Data cleansing, conversion, and storage*

***Forecast Subsystem** Generate weather forecasts*

***Alert Subsystem** Send alerts for severe weather conditions*

***User Interface Subsystem** GUI to interact and receive notifications*

- Breaks the system into separate parts
- The subsystems are organized by high-level functionality
- Subsystems involve both separate and shared technologies

Individual Subsystem

***Data Collection Subsystem** Gather weather data from sensors and external APIs*

- Organizes the code
- A collection of *namespaces* (e.g., C++ namespaces) or directories of source code
- Each namespace or directory is composed of *classes* or free functions

Namespace/Directory Layer

```
WeatherMonitoringSystem/
└── DataCollection/
    ├── SensorDataReader.hpp
    ├── SensorDataReader.cpp
    ├── APIWeatherFetcher.hpp
    ├── APIWeatherFetcher.cpp
    └── Utilities/
        ├── DataConverter.hpp
        ├── DataConverter.cpp
        └── DataValidator.hpp
        └── DataValidator.cpp
```

- Classes related to the layer are stored in a separate directory
- May include even a separate *namespace*, i.e., a C++ `namespace`
- Further subdirectories may also be used

Class/File Layer

```
class SensorDataReader {};

class APIWeatherFetcher {};

class DataConverter {};

class DataValidator {};
```

- What classes make up the system
- Usually separated by `namespace`
- C libraries are organized by *file*, with each file a set of related functions

Method/Function Layer

```
namespace DataCollection {
    class SensorDataReader {
    public:
        SensorDataReader();
        std::string readData();

    private:
        void connectToSensor();
        std::string sensorData;
    };
}
```

- *What are the methods of each class?*

- For C-programs, this would be the functions in that directory or possibly namespace

Design Definition

Software design is the process of defining software methods, functions, objects, and the overall structure and interaction of your code so that the resulting functionality will satisfy user requirements[1]

- A *process*
- Defining *software methods, …*
- Defining *overall structure*
- Defining *interaction of code*
- Result *will satisfy user requirements*

Software Activities

OOP View of Software Development

Design Explanation

Software design is the process by which an agent creates a specification of a software artifact, intended to accomplish goals, using a set of primitive components and subject to constraints.[2]

- An *agent*
- Creates a *specification*
- *software artifact*
- Intent is to *accomplish goals*
- Uses *primitive components*
- Subject to *constraints*

Design Explanation

Software design may refer to either "all the activity involved in conceptualizing, framing, implementing, commissioning, and ultimately modifying complex systems" or "the activity following requirements specification and before programming, as ... [in] a stylized software engineering process."[3]

- Activities: conceptualizing, framing, implementing, commissioning, modifying
- Activity after *requirements specification*
- Activity before *programming*

Design Explanation

Software design usually involves problem-solving and planning a software solution. This includes both a low-level component and an algorithm design, as well as a high-level architecture design. [4]

- Involves *problem-solving*
- Planning a *software solution*
- Includes *low-level component design*
- Includes *algorithm design*
- Includes *high-level architecture design*

Design Levels

- **High-Level Design (HLD)**

  Close to *Analysis*

  Overall system design

  Includes *Software Architecture*

  Represents solutions to requirements

- **Low-Level Design (LLD)**

  Close to *Code*

  Detailed description of every *module*

  Expressed in the design of the classes and methods

Layers of Design

- System
- Subsystem
- Namespace/Directory
- **Class**/File
- **Method**/Function
- Statements

Types of Design

- *structured design*

  From structured programming

  What are the functions?

- *object-oriented design*

  From object-oriented programming

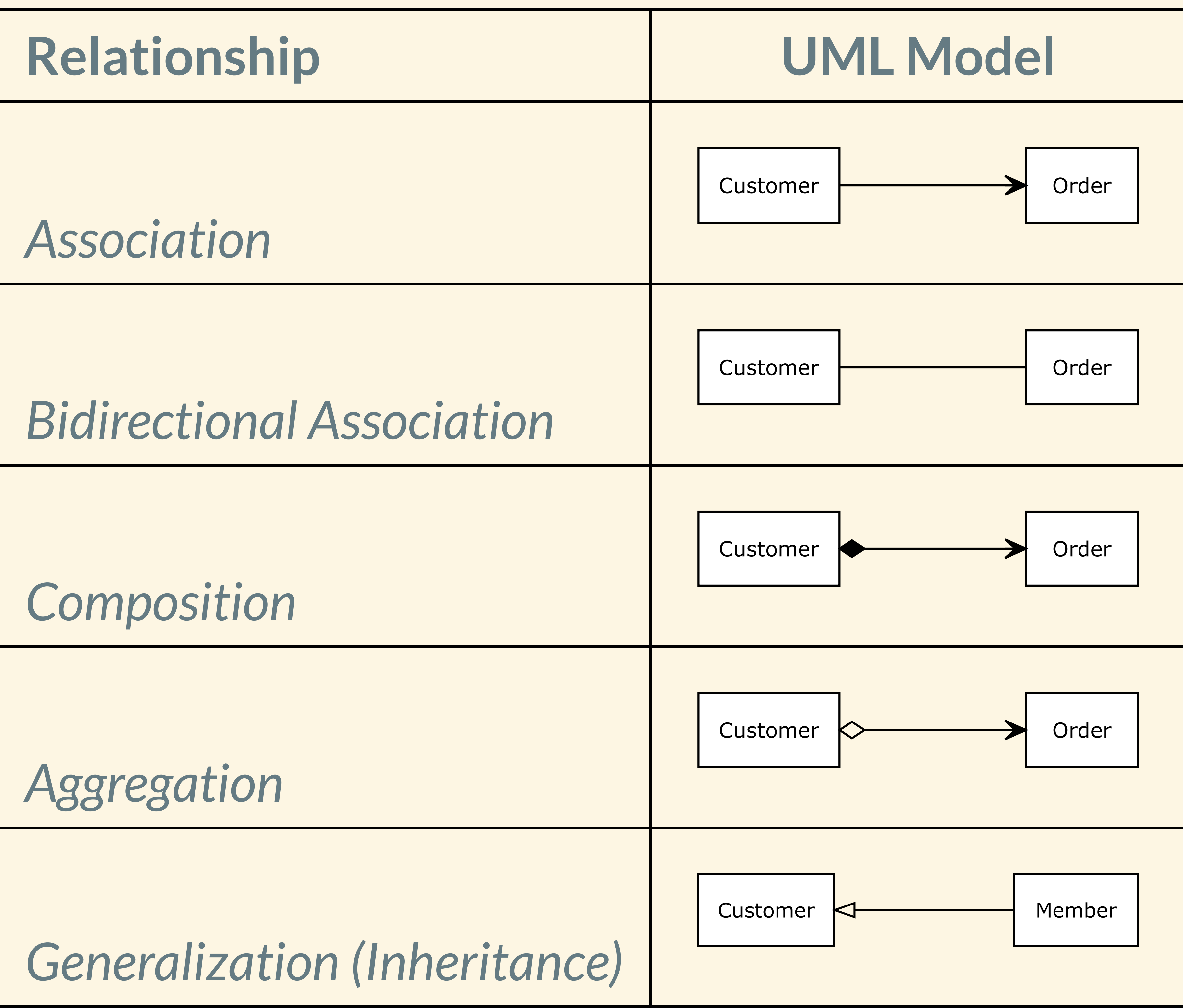
  What are the classes?

Class Design: Individual Class

MethodCount
+count(in methodName: String): Integer

- What classes exist
- Identifiers (names) of the classes
- Methods of each class:
  - Name
  - Parameters:
    - Name
    - Type
    - Direction
  - Return Type
  - Access: `public`, `private`, `protected`
  - (C++) Method Specifiers: `const`, `static`, `virtual`, `final`, `override`, and `friend`

Class Design: Class Relationships



Relationship	UML Model
Association	Customer → Order
Bidirectional Association	Customer — Order
Composition	Customer → Order
Aggregation	Customer → Order
Generalization (Inheritance)	Customer ← Member

Target Audience for Design Decisions

- You, as a developer
- Other developers
- You in a few months
- Other developers in a few months
- Future developers

Informally, what indicates a good design?

- Has the required efficiency
- Has the required security
- Handles errors safely
- Easy to fix bugs
- Easy to determine the source of bugs
- Easy to add features

Why does bad design occur?

- Design primarily involves making choices between tradeoffs
- Design decisions are often made before the problem is fully understood
- Incomplete knowledge by current and previous software engineers
- Requirements have changed since the design was made
- Security requirements have changed since the design was made

Features of Good Design

- Consistent with a shared vocabulary
- Simplicity
- Clear roles
- High *cohesion*
- Low *coupling*

Design Example: MethodCount

Count the number of methods per class for a software system written in C++

- Input the software system (files, directories, fragments)

- Parse the source code into some form so we can process it

- Count the methods in the parsed form

Design Choices

MethodCount

count()

MethodCount

input()
count()

MethodCount

input()
parse()
count()

MethodCount

count()

Parser

input()
parse()

Design Choice: Internal Input & Parsing

MethodCount
count()

- Method `count()` must do all the input, parsing, and counting
- Difficult to generalize as there is limited space for parameters
- For Users, if flexibility is not needed, then this is the easiest-to-use

Design Choice: Internal Parsing

MethodCount
input() count()

- Separate `input()`
- Method `count()` does all the parsing and counting
- Users do not have to concern themselves with parsing

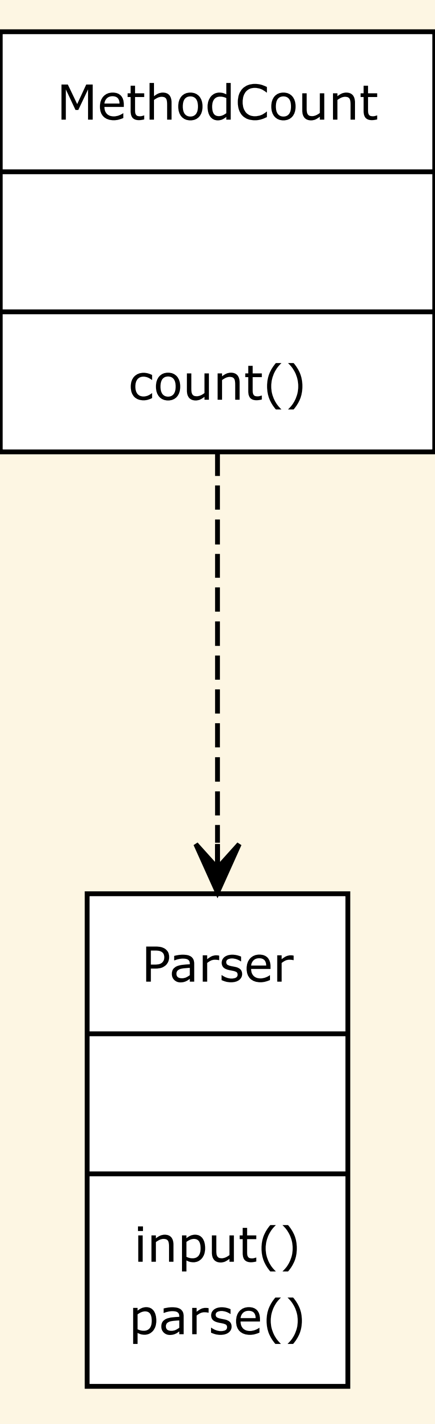
Design Choice: Separate Interfaces

MethodCount

input()
parse()
count()

- Each part has its own method, `input()`, `parse()`, and `output()`
- Lots of flexibility for users
- But it is more complex to use

Design Choice: Separate Parser



- Ultimate flexibility for users
- But again, introduce complexity
- But the complexity may be needed

Design Choices

MethodCount
count()

MethodCount
input()
count()

MethodCount
input()
parse()
count()

MethodCount
count()

Parser
input()
parse()

Layers of Design

- System
- Subsystem
- Namespace/Directory
- **Class**/File
- **Method**/Function
- Statements

Software Design

- Software design is a skill
- Requires deciding between tradeoffs, e.g., complexity vs. efficiency
- Requires knowing alternatives at a high level and language features to support the design at a low level
- Needs to clearly and cleanly implement the design, so code consistency, standards, and documentation are essential
- It is an iterative process that requires exploring alternatives as requirements and the environment changes