Object-Oriented and Classical Software Engineering

Fifth Edition, WCB/McGraw-Hill, 2002

Stephen R. Schach
srs@vuse.vanderbilt.edu

DESIGN PHASE

Overview

- Design and abstraction
- Action-oriented design
- Data flow analysis
- Transaction analysis
- Data-oriented design
- Object-oriented design
- Elevator problem: object-oriented design

Overview (contd)

- Formal techniques for detailed design
- Real-time design techniques
- Testing during the design phase
- CASE tools for the design phase
- Metrics for the design phase
- Air Gourmet Case Study: object-oriented design
- Challenges of the design phase

Data and Actions

- Two aspects of a product
  - Actions that operate on data
  - Data on which actions operate
- The two basic ways of designing a product
  - Action-oriented design
  - Data-oriented design
- Third way
  - Hybrid methods
  - For example, object-oriented design

Design Activities

- Architectural design
- Detailed design
- Design testing

Architectural Design

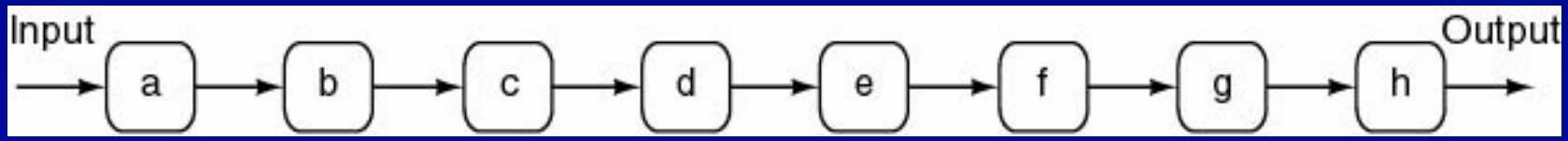
- Input: Specifications
- Output: Modular decomposition
- Abstraction

Detailed Design

- Each module is designed
  - Specific algorithms
  - Data structures

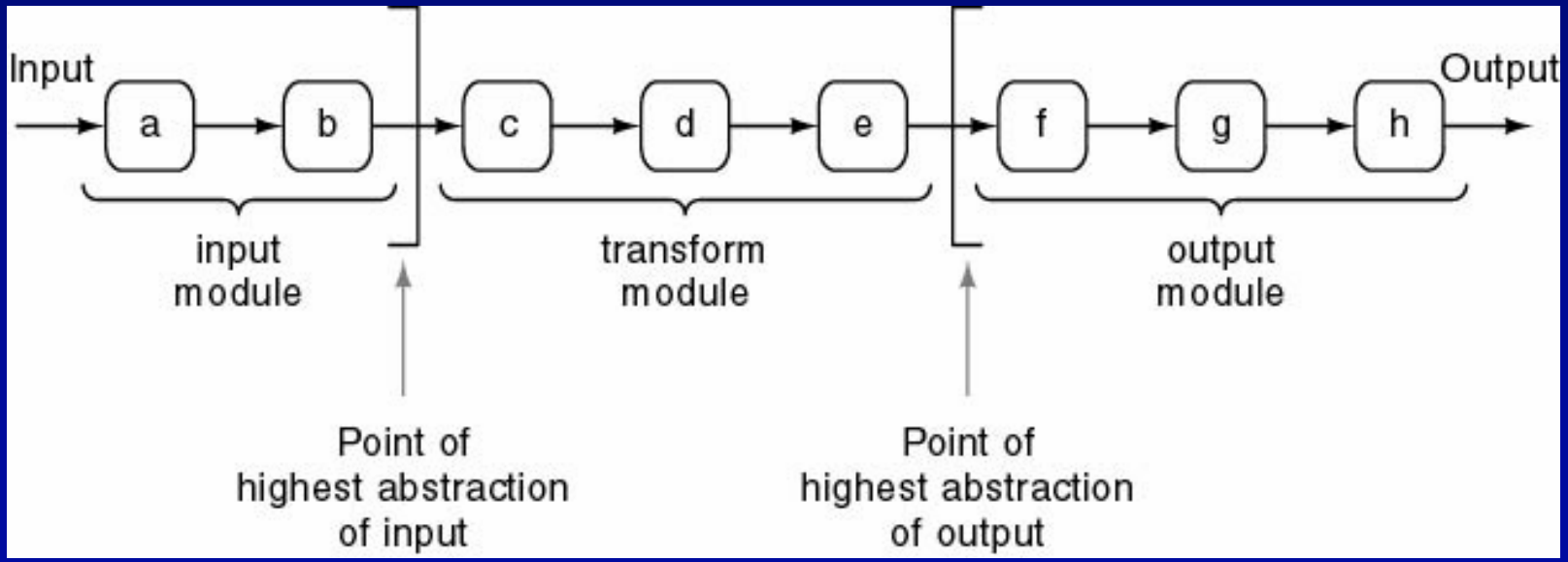
Action-Oriented Design Methods

- Data flow analysis
- When to use it
  - With most specification methods (Structured Systems Analysis here)
- Key point: We have detailed action information from the DFD



Data Flow Analysis

- Product transforms input into output
- Determine
  - "Point of highest abstraction of input"
  - "Point of highest abstract of output"



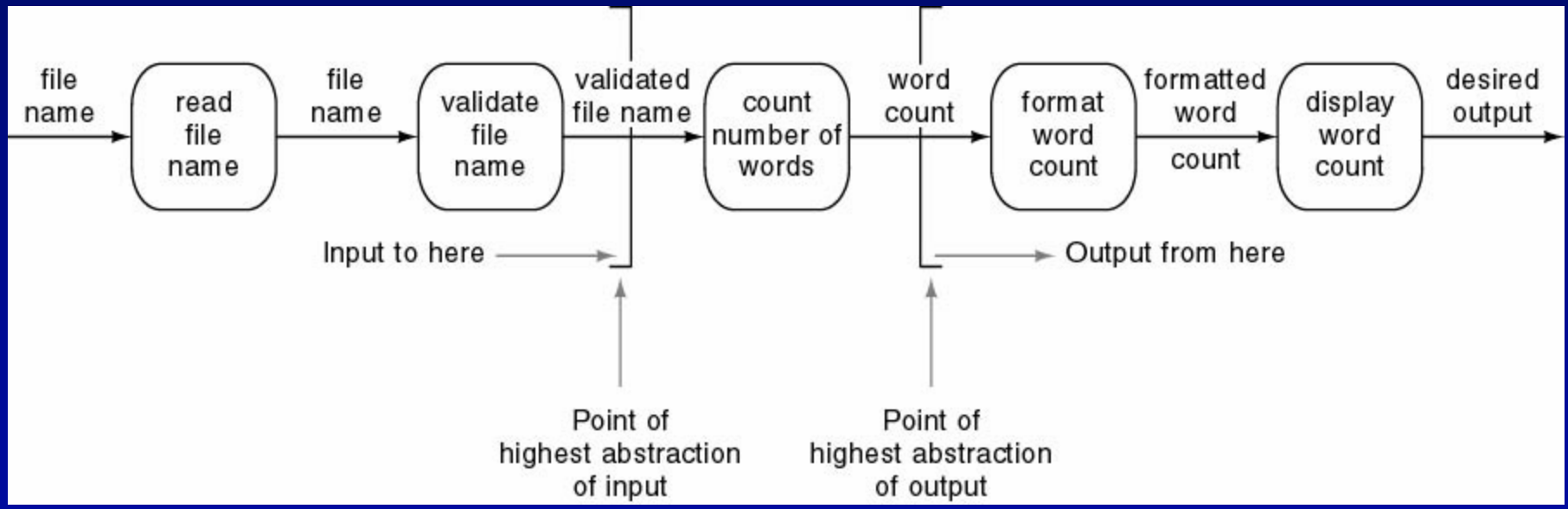
Data Flow Analysis (contd)

- Decompose into three modules
- Repeat stepwise until each module has high cohesion
  - Minor modifications may be needed to lower the coupling

Data Flow Analysis (contd)

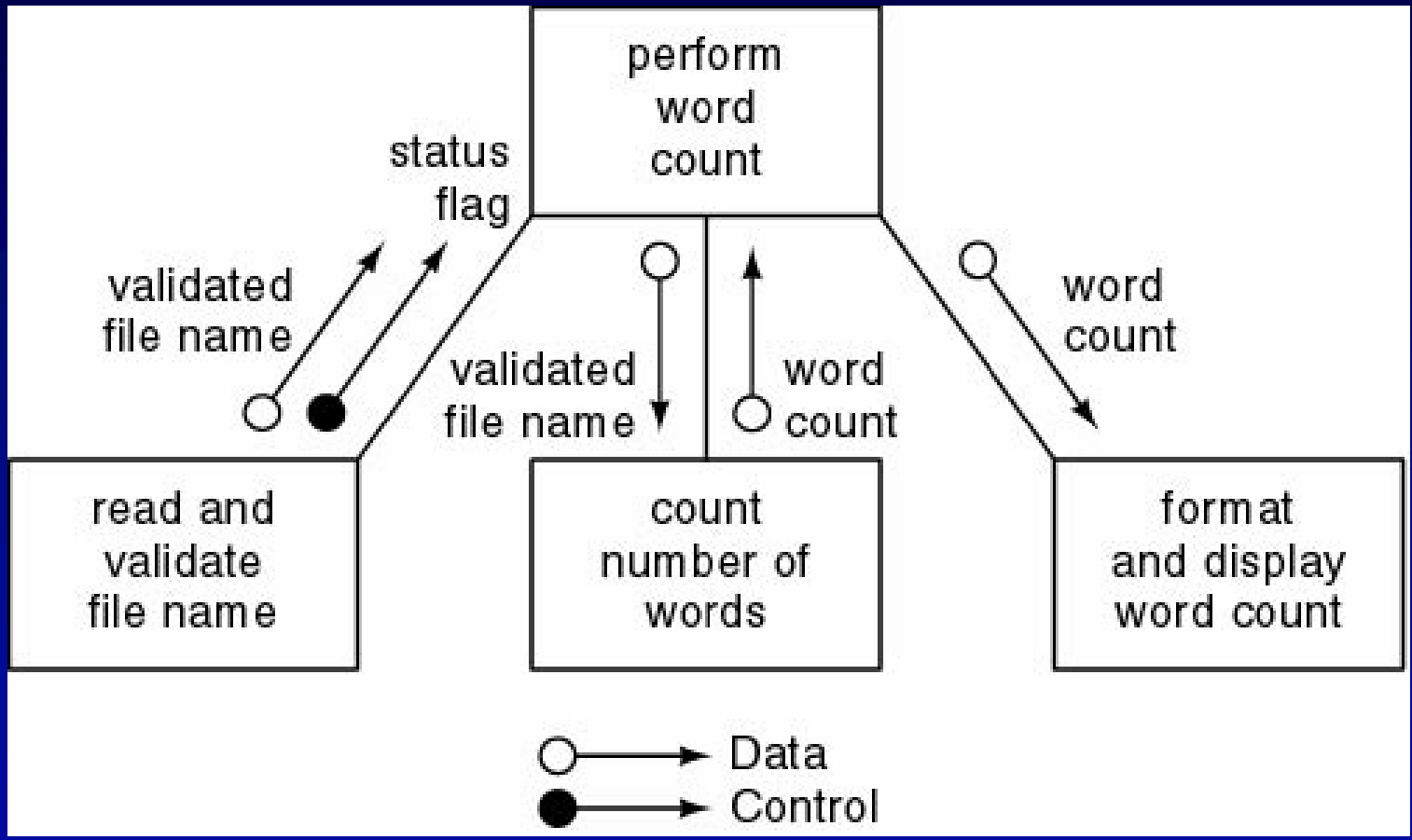
- Example

  Design a product which takes as input a file name, and returns the number of words in that file (like UNIX *wc*)



Data Flow Analysis Example (contd)

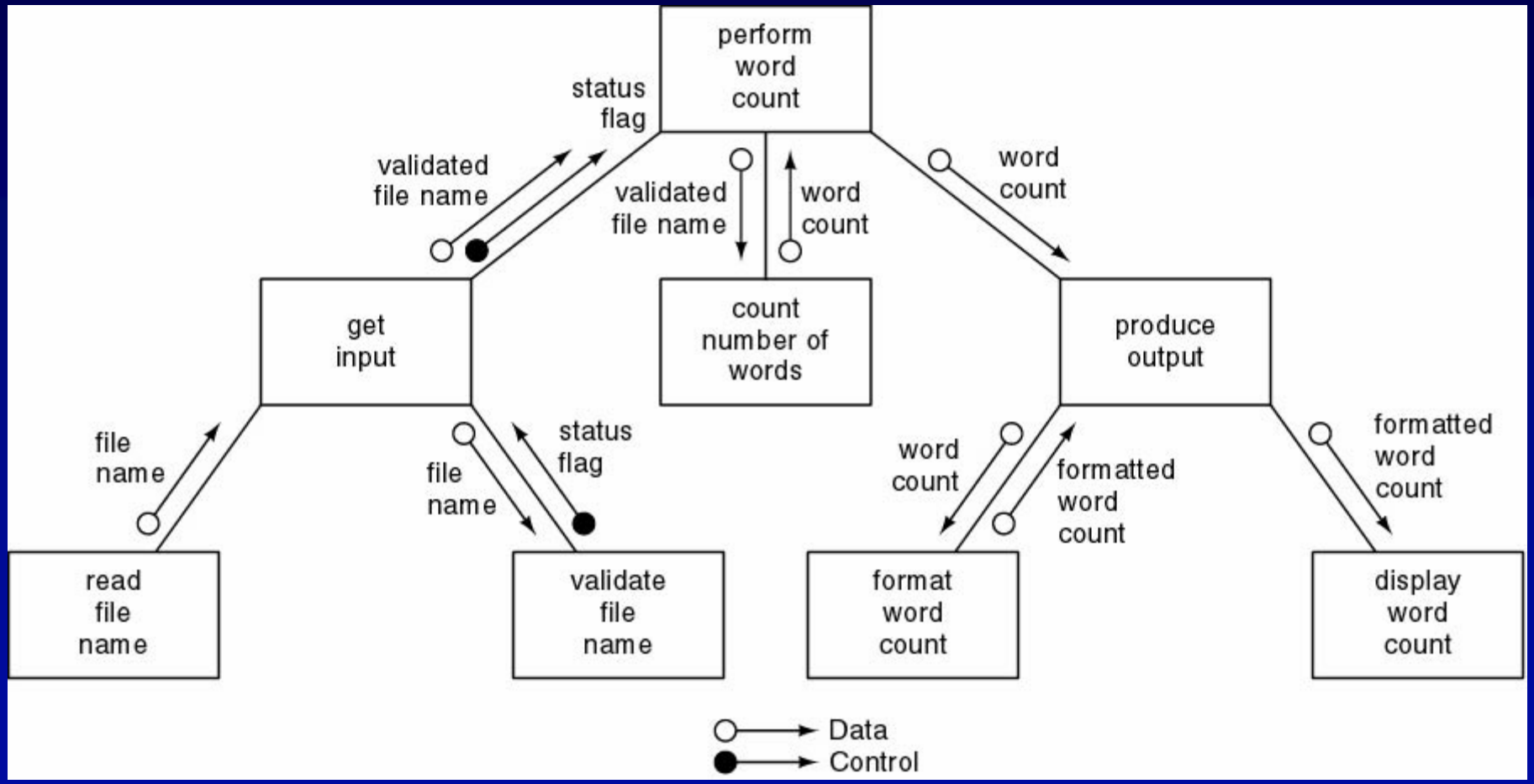
- First refinement



- Now refine the two modules of communicational cohesion

Data Flow Analysis Example (contd)

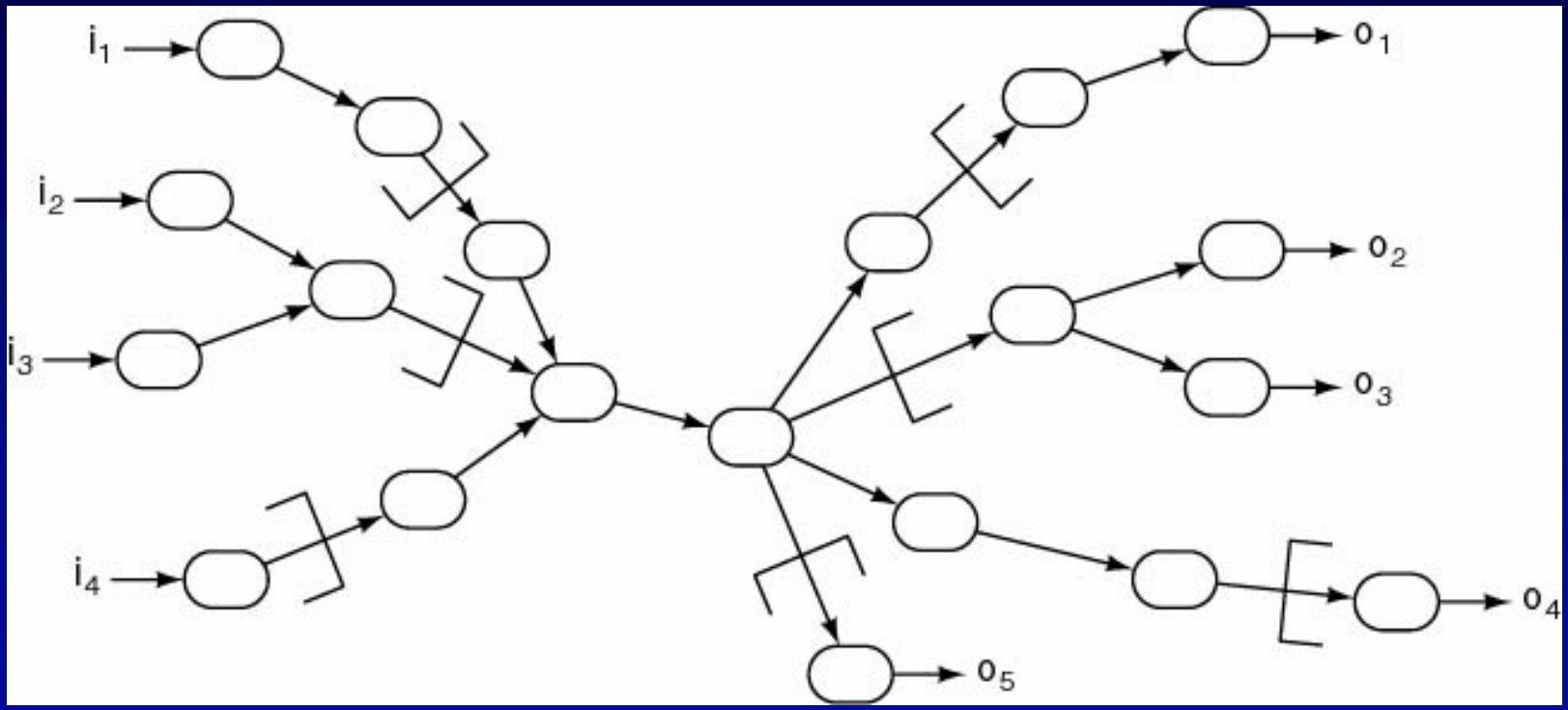
- Second refinement



- All eight modules now have functional cohesion

Multiple Input and Output Streams

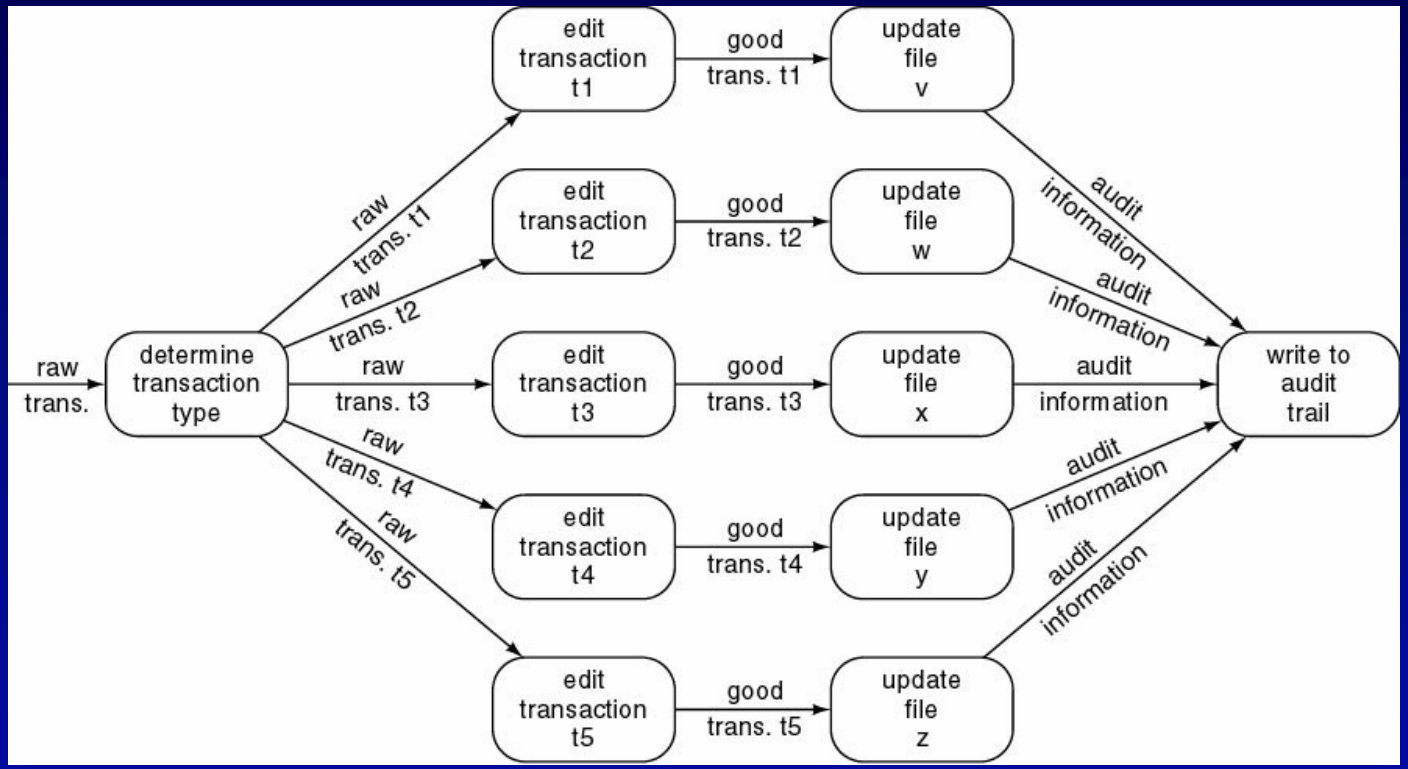
- Point of highest abstraction for each stream



- Continue until each module has high cohesion
  - Adjust the coupling if needed

Transaction Analysis

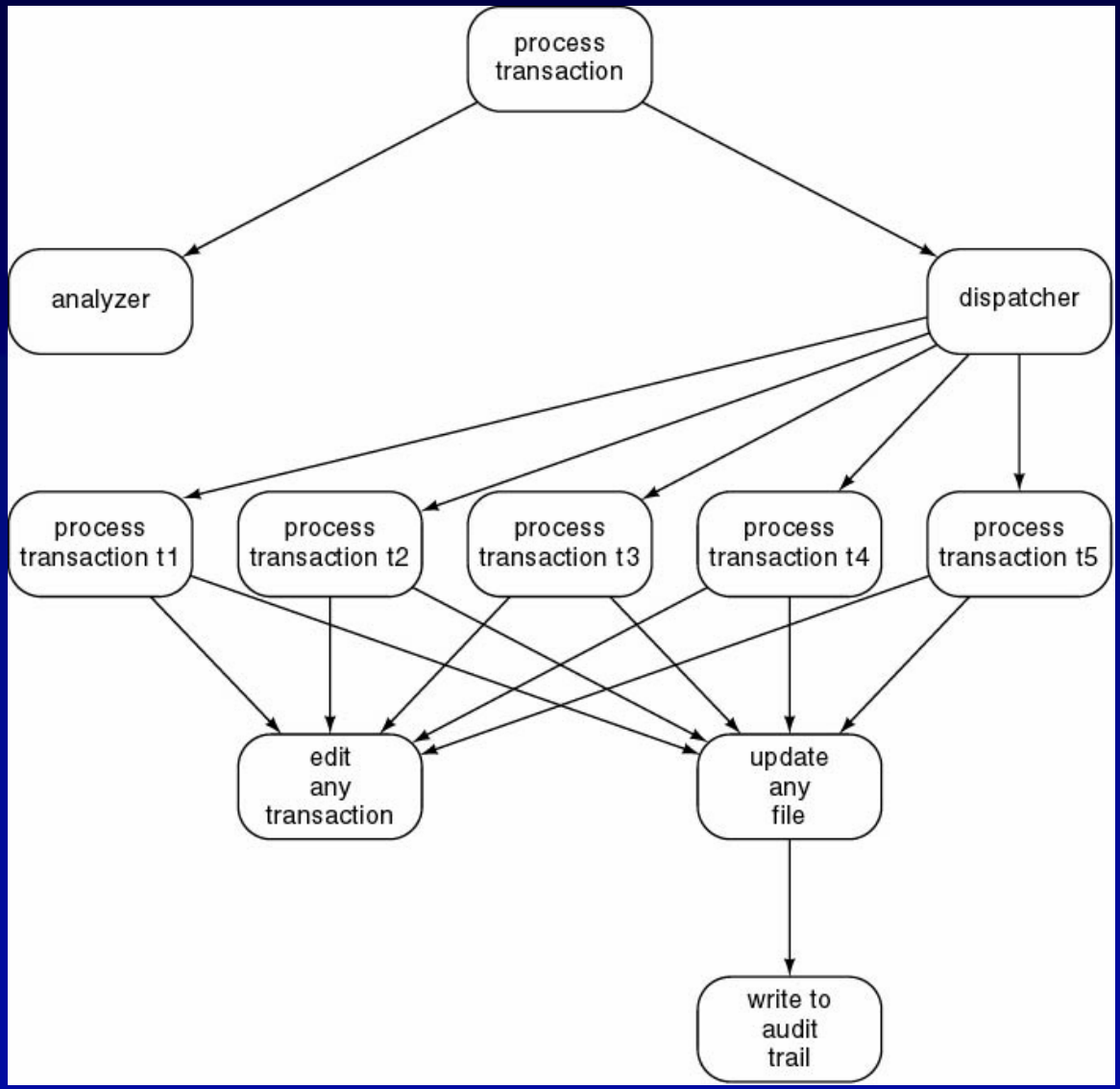
- DFA poor for transaction processing products
  - Example: ATM (automated teller machine)



- Poor design
  - Logical cohesion, control coupling

Corrected Design Using Transaction Analysis

- Software reuse
- Have one generic edit module, one generic update module
- Instantiate them 5 times



Data-Oriented Design

- Basic principle
  - The structure of a product must conform to the structure of its data
- Three very similar methods
  - Warnier
  - Orr
  - Jackson
- Data-oriented design
  - Has never been as popular as action-oriented design
  - With the rise of OOD, data-oriented design has largely fallen out of fashion

Object-Oriented Design (OOD)

- Aim
  - Design the product in terms of the classes extracted during OOA
- If we are using a language without inheritance (C, Ada 83)
  - Use abstract data type design
- If we are using a language without a type statement (FORTRAN, COBOL)
  - Use data encapsulation

Object-Oriented Design Steps

- OOD consists of four steps:
  - 1. Construct interaction diagrams for each scenario
  - 2. Construct the detailed class diagram
  - 3. Design the product in terms of clients of objects
  - 4. Proceed to the detailed design

Elevator Problem: OOD

- Step 1. Construct interaction diagrams for each scenario
- Sequence diagrams
- Collaboration diagrams
  - Both show the same thing
  - Objects and messages passed between them
  - But in a different way

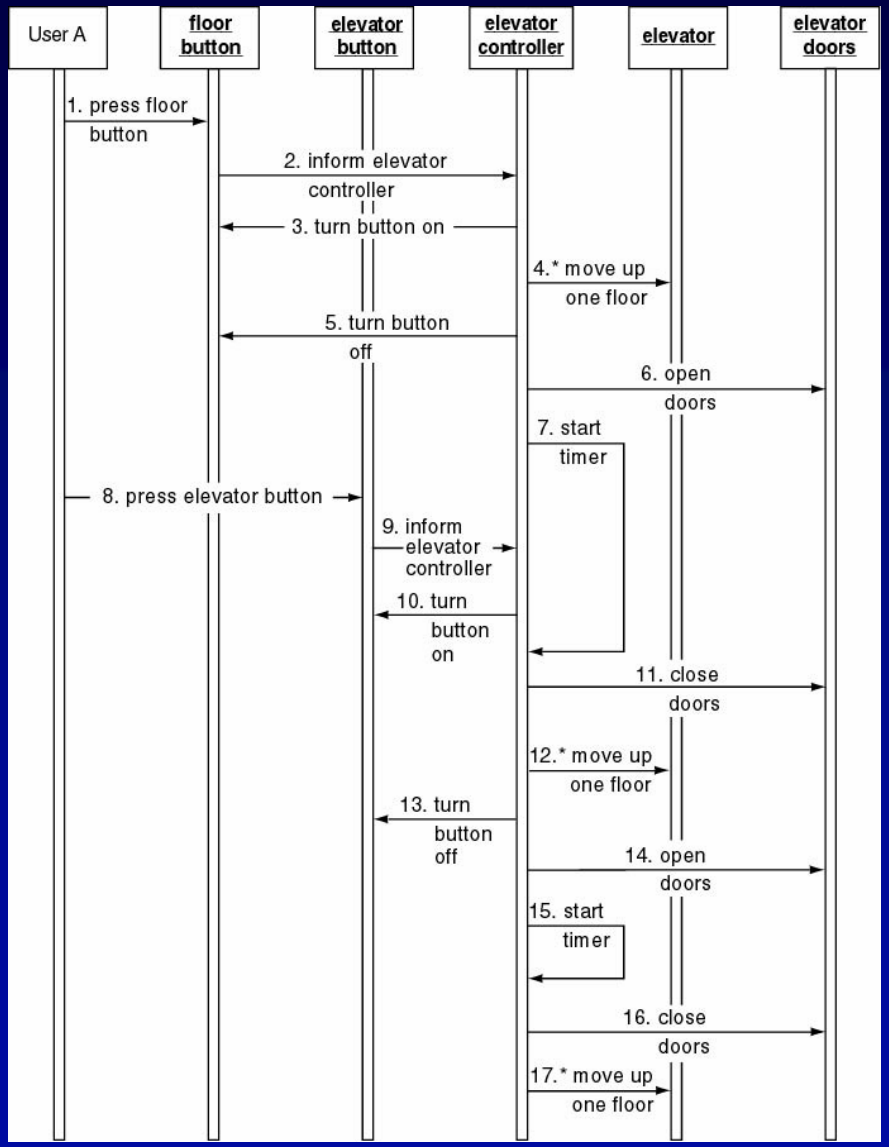
Elevator Problem: OOD (contd)

- Normal scenario

1. User A presses the Up floor button at floor 3 to request an elevator. User A wishes to go to floor 7.
2. The floor button informs the elevator controller that the floor button has been pushed.
3. The elevator controller sends a message to the Up floor button to turn itself on.
4. The elevator controller sends a series of messages to the elevator to move itself up to floor 3. The elevator contains User B, who has entered the elevator at floor 1 and pressed the elevator button for floor 9.
5. The elevator controller sends a message to the Up floor button to turn itself off.
6. The elevator controller sends a message to the elevator doors to open themselves.
7. The elevator control starts the timer.
   User A enters the elevator.
8. User A presses elevator button for floor 7.
9. The elevator button informs the elevator controller that the elevator button has been pushed.
10. The elevator controller sends a message to the elevator button for floor 7 to turn itself on.
11. The elevator controller sends a message to the elevator doors to close themselves after a timeout.
12. The elevator controller sends a series of messages to the elevator to move itself up to floor 7.
13. The elevator controller sends a message to the elevator button for floor 7 to turn itself off.
14. The elevator controller sends a message to the elevator doors to open themselves to allow User A to exit from the elevator.
15. The elevator controller starts the timer.
    User A exits from the elevator.
16. The elevator controller sends a message to the elevator doors to close themselves after a timeout.
17. The elevator controller sends a series of messages to the elevator to move itself up to floor 9 with User B.

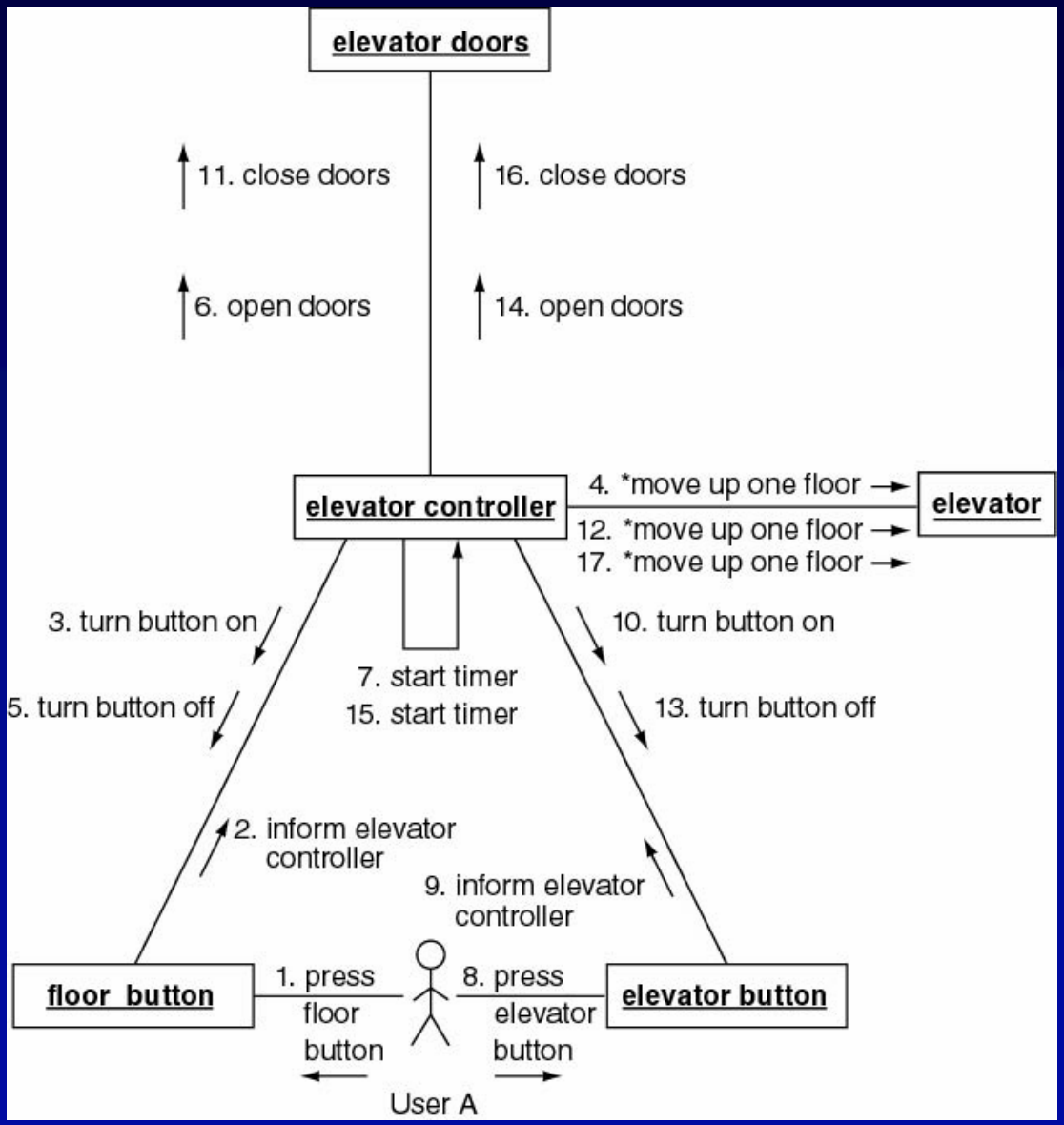
Elevator Problem: OOD (contd)

- Sequence diagram



Elevator Problem: OOD (contd)

- Collaboration diagram



Elevator Problem: OOD (contd)

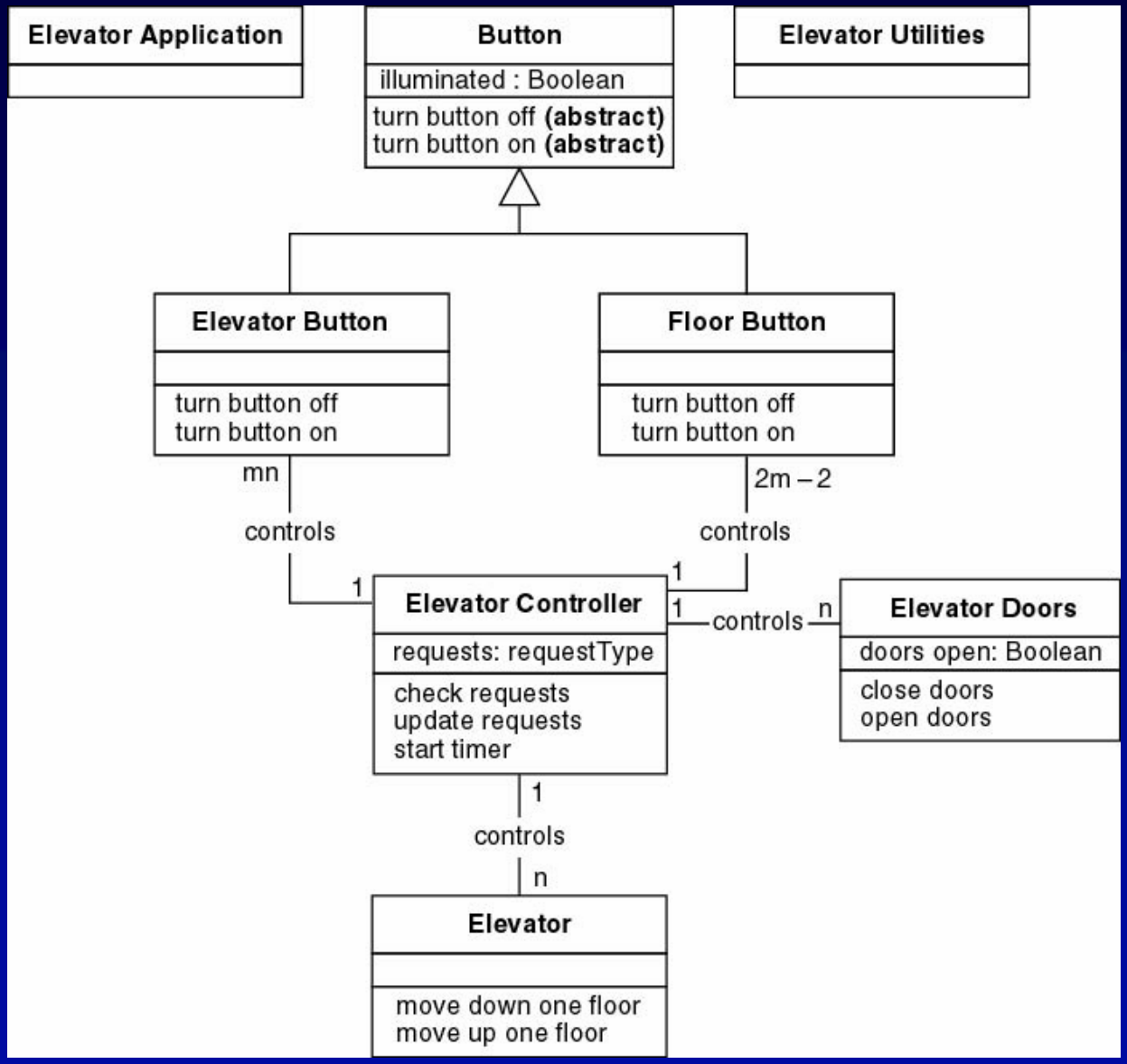
- Step 2. Construct the detailed class diagram
- Do we assign an action to a class or to a client of that class?
- Criteria
  - Information hiding
  - Reducing number of copies of each action
  - Responsibility-driven design
- Examples

  close doors is assigned to **Elevator Doors**

  move one floor down is assigned to **Elevator**

Elevator Problem: OOD (contd)

- Detailed class diagram

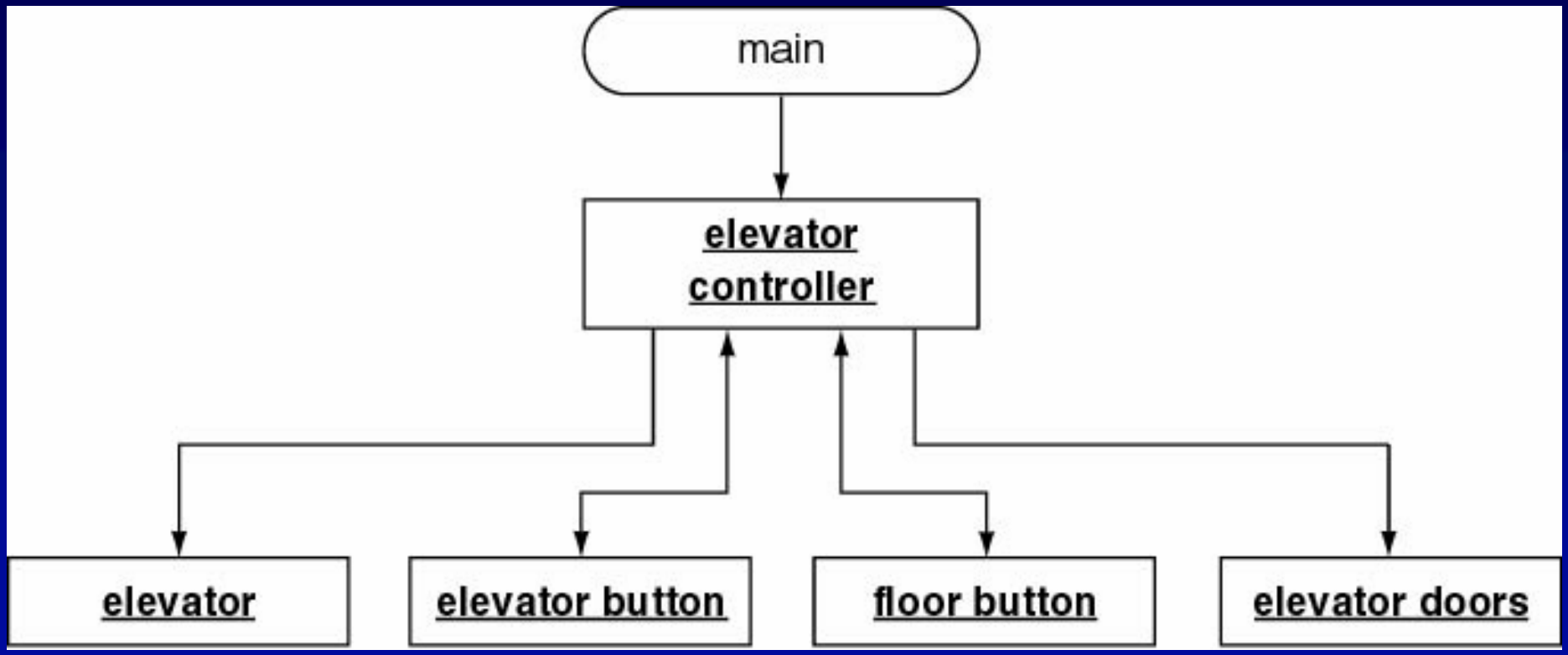


Elevator Problem: OOD (contd)

- Step 3. Design the product in terms of clients of objects
- Draw an arrow from an object to a client of that object
- Objects that are not clients of any object have to be initiated, probably by the `main` method
  - Additional methods may be needed

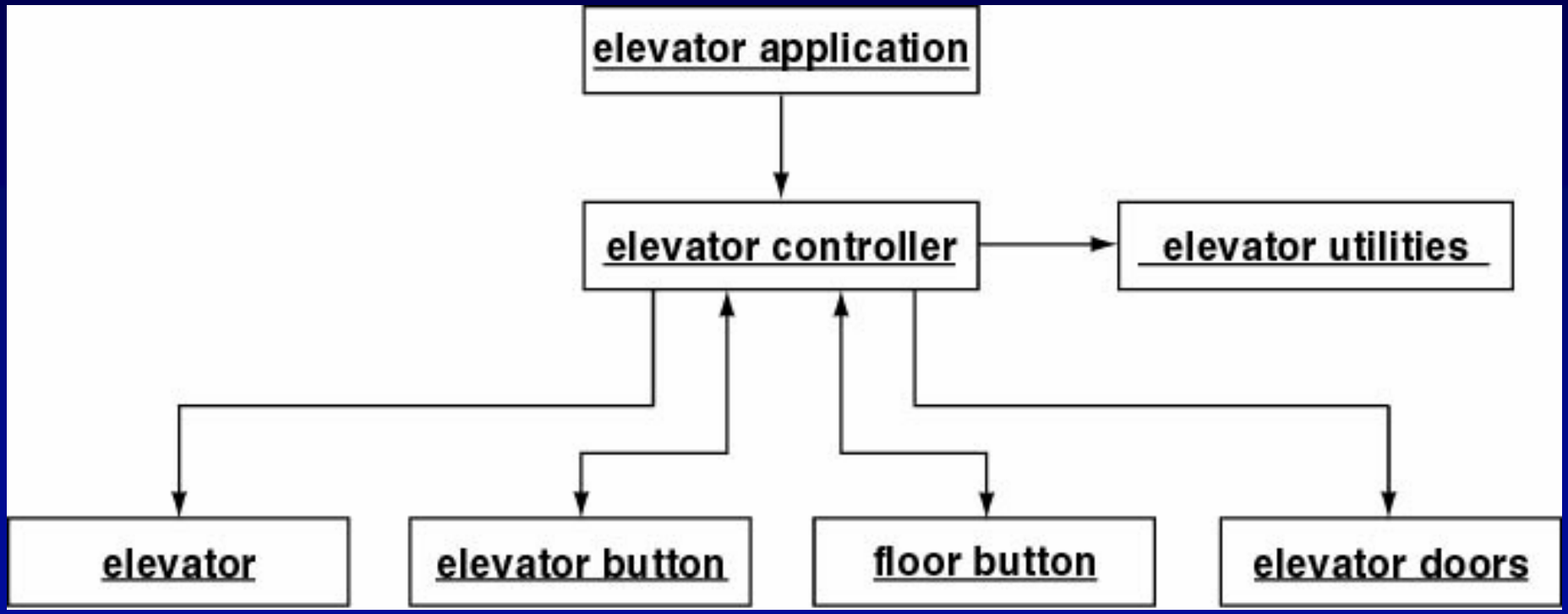
Elevator Problem: OOD (contd)

- C++ client-object relations



Elevator Problem: OOD (contd)

- Java client-object relations



Elevator Problem: OOD (contd)

- elevator controller needs method elevator control loop so that *main* (or elevator application) can call it

Elevator Problem: OOD (contd)

- Step 4. Perform the detailed design
- Detailed design of method elevator controller loop

```
void elevator event loop (void)
{
  while (TRUE)
  {
    if (a button has been pressed)
      if (button is not on)
      {
        update requests;
        button::turn button on;
      }
    else if (elevator is moving up)
    {
      if (there is no request to stop at floor f)
        elevator::move one floor up;
      else
      {
        stop elevator by not sending a message to move;
        elevator doors::open doors;
        start timer;
        if (elevator button is on)
          elevator button::turn button off;
        update requests;
      }
    }
    else if (elevator is moving down)
      [similar to up case]
    else if (elevator is stopped and request is pending)
    {
      elevator doors::close doors;
      if (floor button is on)
        floor button::turn button off;
      determine direction of next request;
      elevator::move one floor up/down;
    }
    else if (elevator is at rest and not (request is pending))
      elevator doors::close doors;
    else
      there are no requests, elevator is stopped with elevator doors closed, so do nothing;
  }
}
```

Formal Techniques for Detailed Design

- Implementing a complete product and then proving it correct is hard
- However, use of formal techniques during detailed design can help
  - Correctness proving can be applied to module-sized pieces
  - The design should have fewer faults if it is developed in parallel with a correctness proof
  - If the same programmer does the detailed design and implementation
    - The programmer will have a positive attitude to the detailed design
    - This should lead to fewer faults

Design of Real-Time Systems

- Difficulties associated with real-time systems
  - Inputs come from the real world
    - Software has no control over the timing of the inputs
  - Frequently implemented on distributed software
    - Communications implications
    - Timing issues
  - Problems of synchronization
    - Race conditions
    - Deadlock (deadly embrace)
- Major difficulty in the design of real-time systems
  - Determining whether the timing constraints are met by the design

Real-Time Design Methods

- Usually, extensions of nonreal-time methods to real-time
- We have limited experience in use of any real-time methods
- The state-of-the-art is not where we would like it to be

Testing during the Design Phase

- Design reviews
  - The design must correctly reflect the specifications
  - The design itself must be correct
  - Transaction-driven inspections

CASE Tools for the Design Phase

- UpperCASE tools
  - Built around a data dictionary
  - Consistency checker
  - Screen, report generators
  - Modern tools represent OOD using UML
  - Examples:
    - Software through Pictures
    - Rose

Metrics for the Design Phase

- The five basic metrics
- Cyclomatic complexity is problematic
  - Data complexity is ignored
  - Not used much with the object-oriented paradigm

Air Gourmet Case Study: Object-Oriented Design

- OOD consists of four steps:
  - 1. Construct interaction diagrams for each scenario
  - 2. Construct the detailed class diagram
  - 3. Design the product in terms of clients of objects
  - 4. Proceed to the detailed design

Step 1. Interaction Diagrams

- Extended scenario for making a reservation

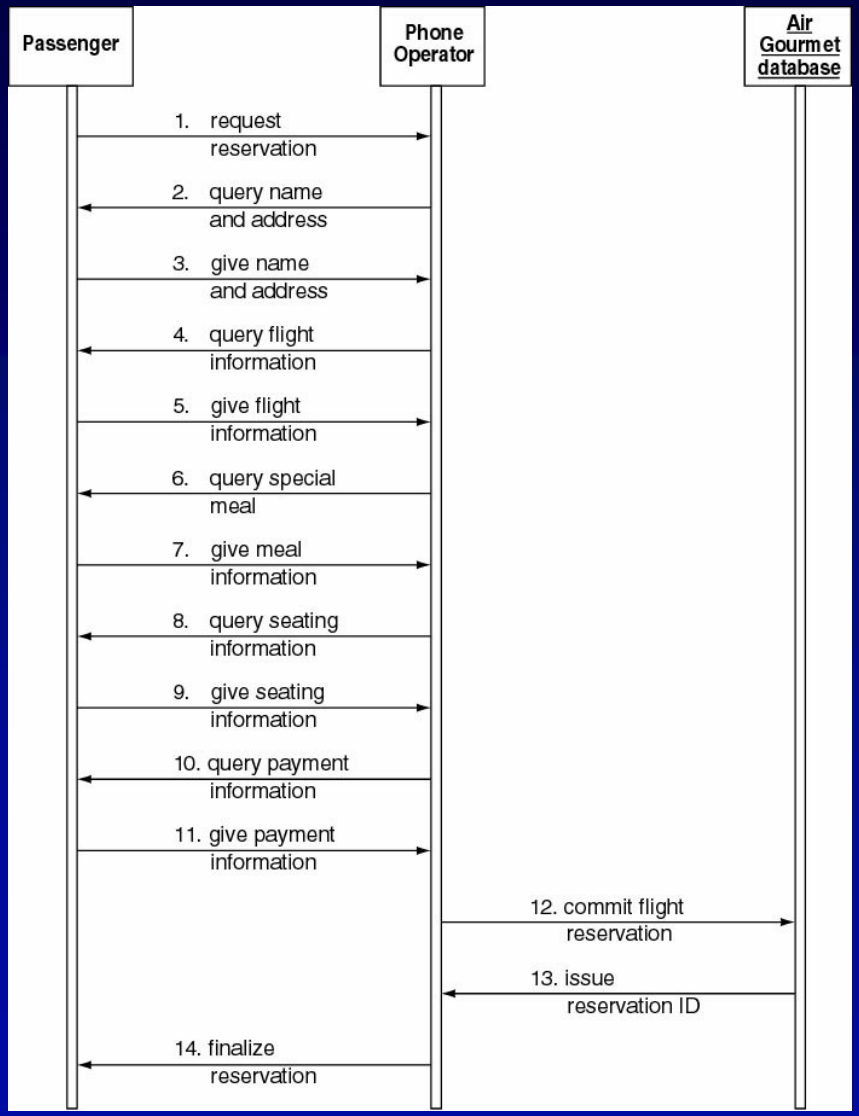
1. The Passenger contacts the Air Gourmet call center Phone Operator and expresses the desire to reserve a flight.
2. The Phone Operator requests the name and address of the Passenger.
3. The Passenger provides the requested information.
4. The Phone Operator asks the Passenger the date of the flight and the flight number.
5. The Passenger provides the requested information.
6. The Phone Operator asks the Passenger if a special meal is desired.
7. The Passenger states what kind of special meal, if any, is desired.
8. The Phone Operator asks the Passenger for his or her seating preference.
9. The Passenger provides his or her seating preference.
10. The Phone Operator requests payment information.
11. The Passenger provides a valid credit card number.
12. The Phone Operator commits the reservation to the Air Gourmet database.
13. The Air Gourmet database issues the reservation ID to the Phone Operator.
14. The Phone Operator gives the reservation ID to the Passenger and informs the Passenger that the ticket will be mailed soon.

Possible Alternatives

A. The flight number requested does not exist on the particular day that the Passenger wants to fly.
B. The flight that the Passenger requested already is full.
C. The Passenger has an unusual meal request that Air Gourmet cannot fulfill.
D. There are problems with the credit card number provided by the Passenger.

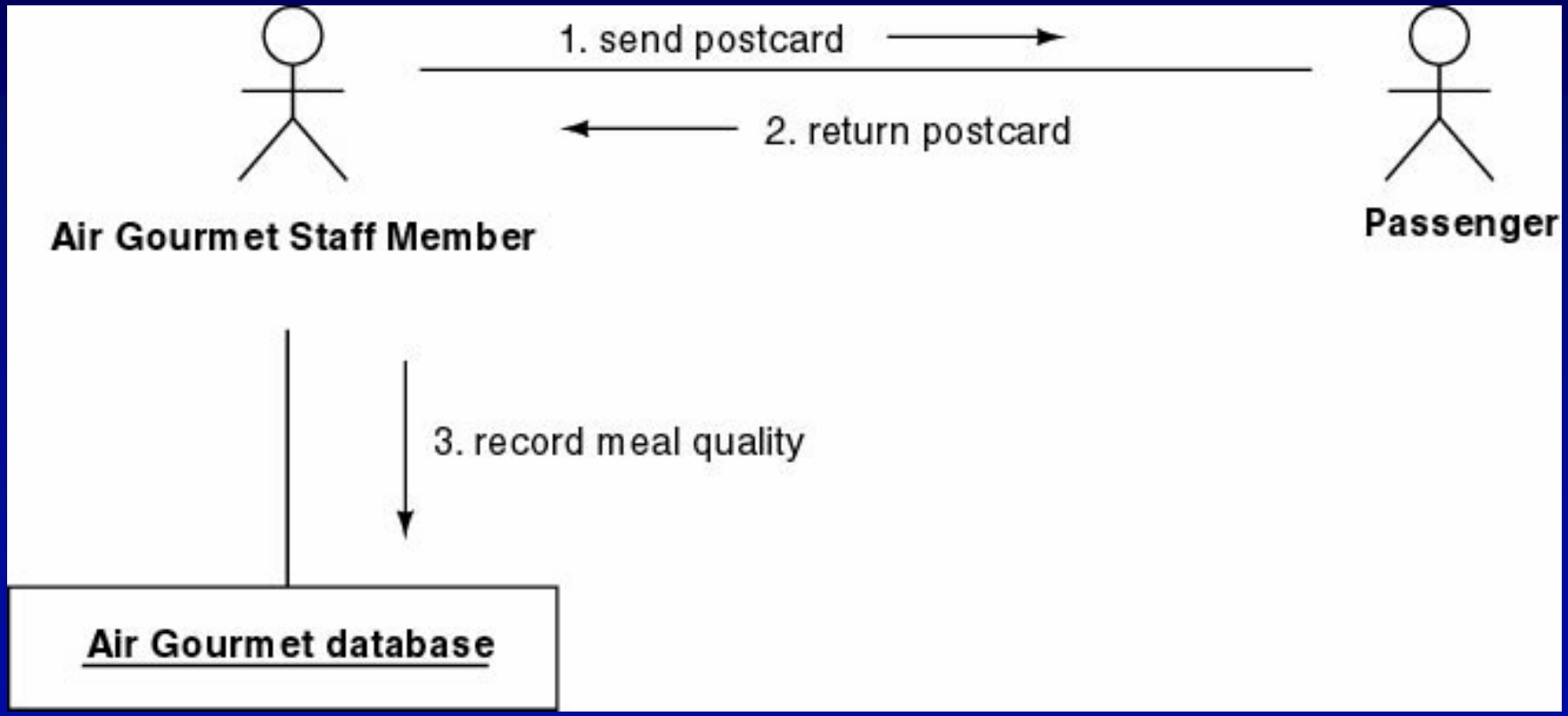
Step 1. Interaction Diagrams (contd)

- Sequence diagram for making a reservation

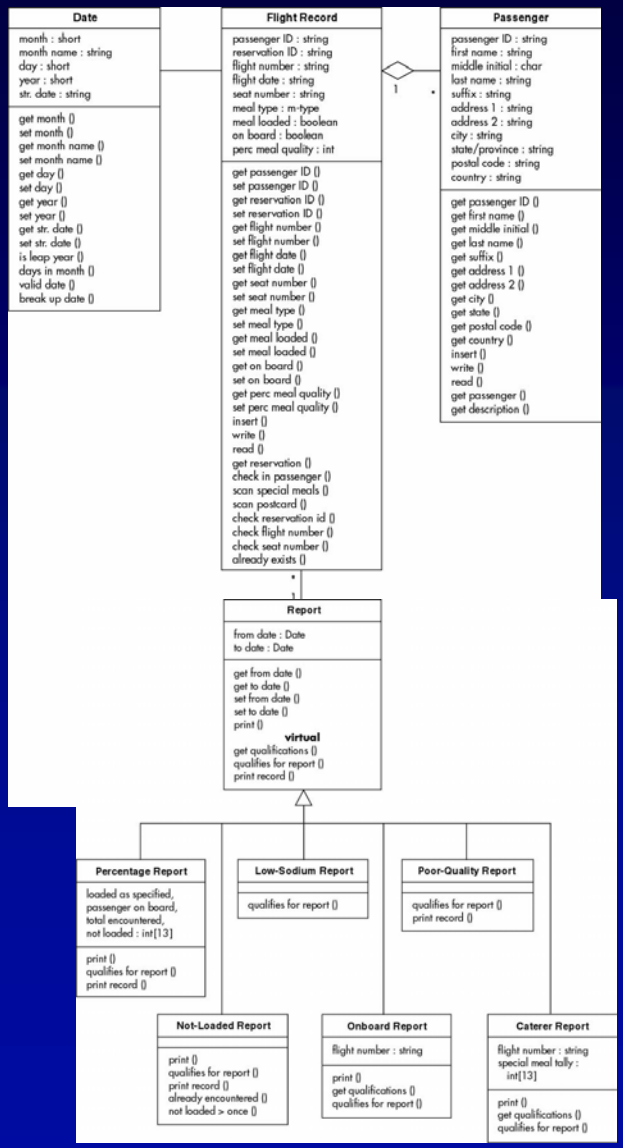


Step 1. Interaction Diagrams (contd)

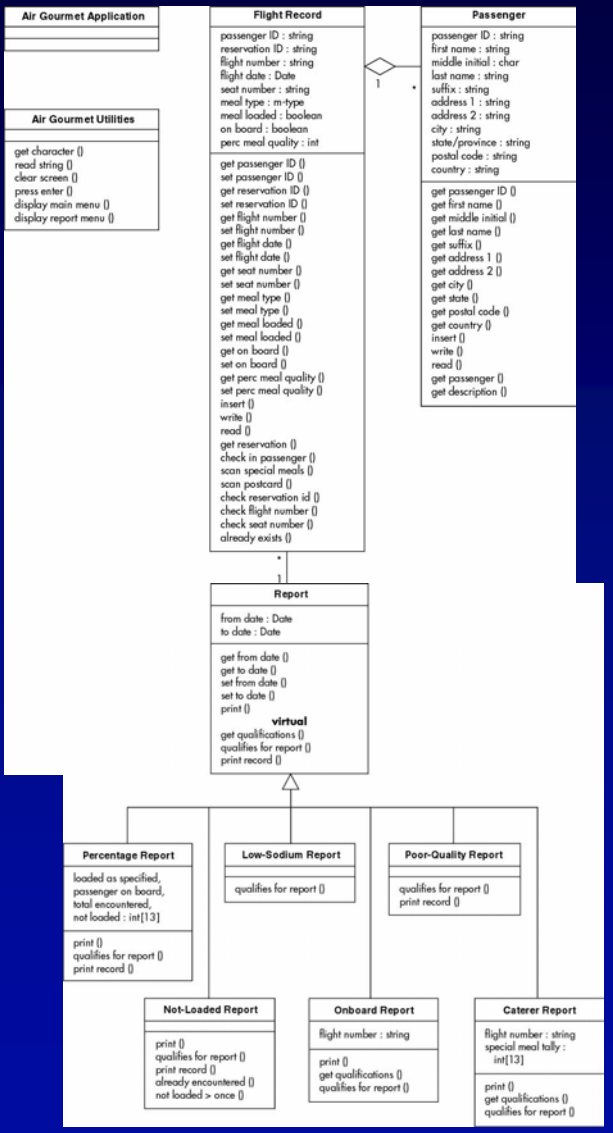
- Collaboration diagram for sending and returning a postcard



Step 2. Detailed Class Diagram

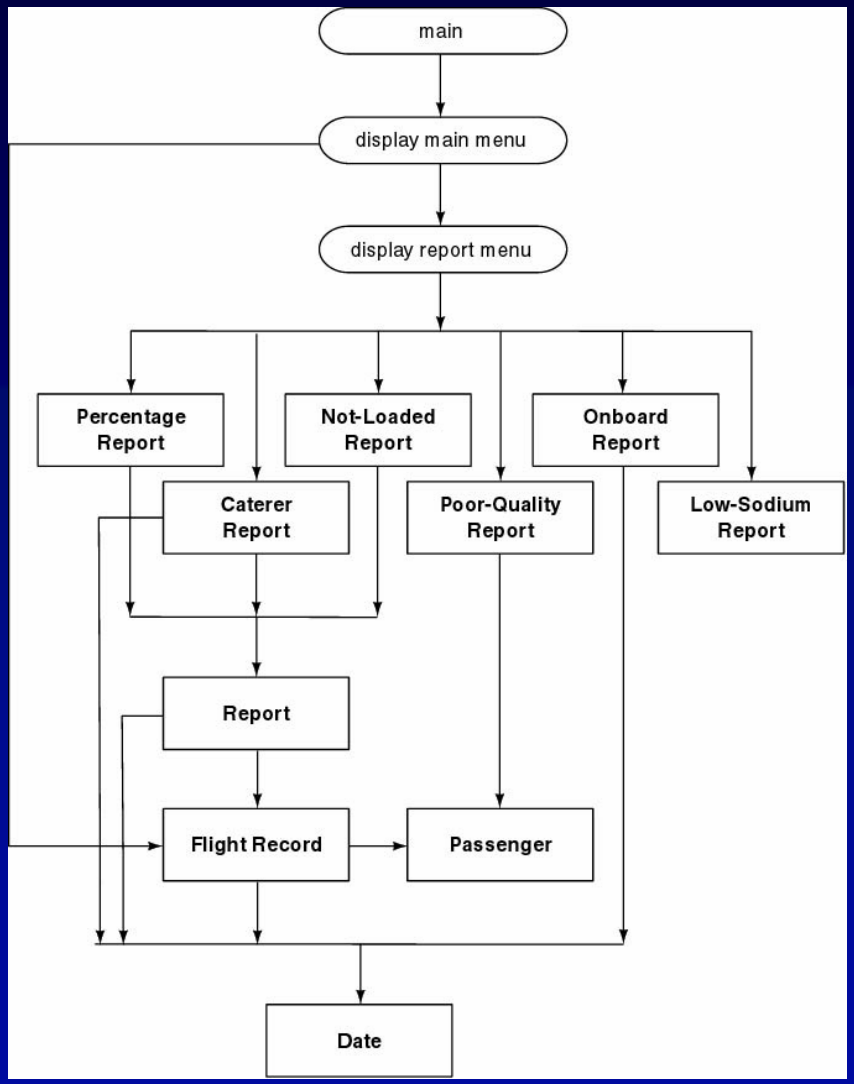


C++ version

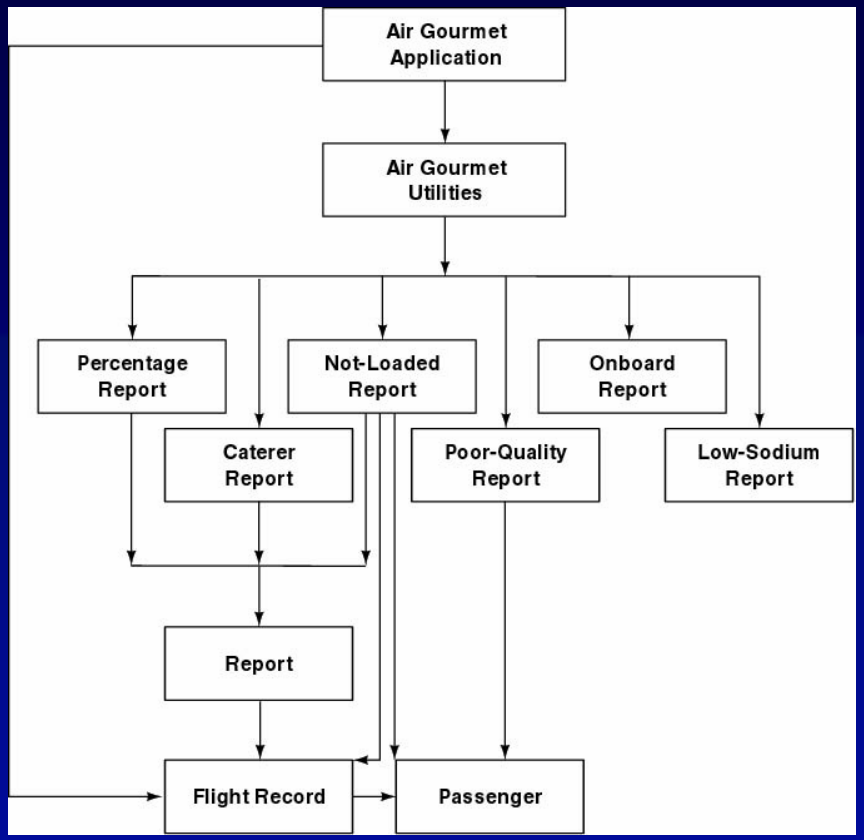


Java version

Step 3. Client–Object Relations



C++ version



Java version

Step 4. Detailed Design

- The detailed design can be found in
  - Appendix H (for implementation in C++)
  - Appendix I (for implementation in Java)

Challenges of the Design Phase

- The design team should not do too much
  - The detailed design should not become code
- The design team should not do too little
  - It is essential for the design team to produce a complete detailed design
- We need to grow great designers