UML (contd.)

Terms and Concepts

- *Class operations (behaviors) are …*
  - named using a verb or verb phrases.
  - inferred from the problem statement.
  - Provide a requested service.
  - usually notated using the format
    - **[visibility] name [(parameter-list)] [: return-type] [{property-string}]**
    - example: +printFirstName( ) : void

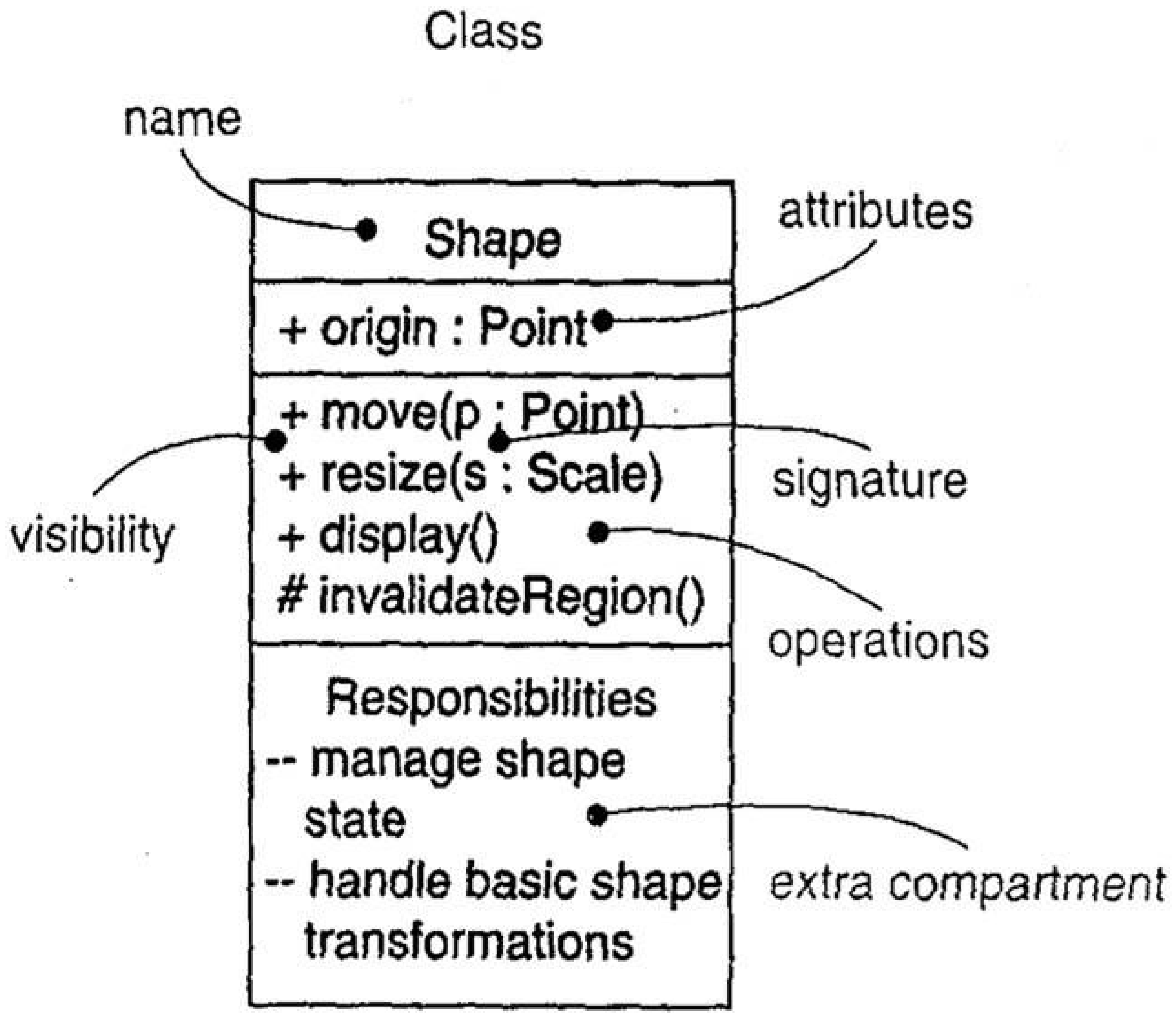
property-string

- leaf
- abstract (write name in italics)
- query
- sequential
- guarded
- concurrent
- static

Terms and Concepts

- We may provide zero or more parameters, each of which follows the syntax
  - **[direction] name : type [= default-value]**
- Direction of the parameters may be any of the following values:
  - **in, out, inout**
- ***Class responsibility…***
  - Is a contract or obligation of a class to the users of the class.
  - Every well structured class has at least one responsibility and at most just a handful

Class Artifact

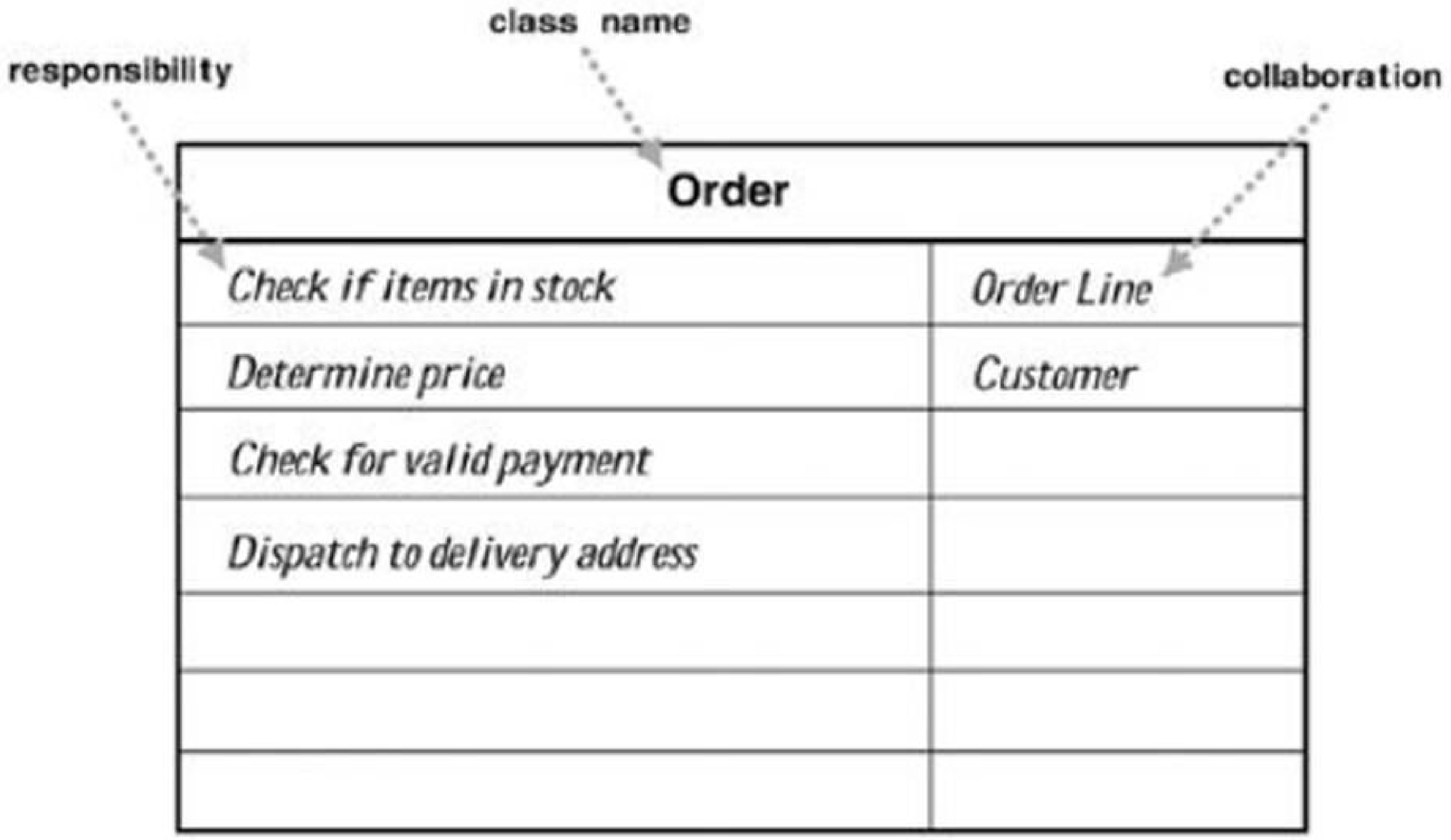


Common Modeling Techniques

- **Identify the classes**
  - Formulate a problem statement. The problem statement may be scenarios associated with use cases.
  - Identify the nouns in the problem statement.
  - Use CRC (Class, Responsibility, Collaboration) cards or use-cases to isolate each class.

    - Invented **by Ward Cunningham** in late 1980s

CRC Cards


class name
responsibility
collaboration
Order
Check if items in stock
Order Line
Determine price
Customer
Check for valid payment
Dispatch to delivery address

Common Modeling Techniques

- Determine the responsibilities of each class.
  - Even out the work load between classes.
    - A class with too much responsibility should be broken up into multiple classes
    - A class with too little responsibility should be absorbed into another class
  - Classes should exhibit high cohesion and low coupling

Common Modeling Techniques

- A class should have a single well-defined purpose.
  - This promotes software reusability
- A class should interact with a limited number of other classes.
  - This simplifies modifications to the program.
- Model abstract (non-software) things (such as people) that are outside of the system boundaries.

Automated Teller Machine (ATM) Case Study (from O-O Modeling and Design with UML by Blaha and Rumbaugh)

- System Concept
  - Develop software so that customers can access a bank's computer and carry out their own financial transactions without the mediation of a bank employee

Design the software to support a computerized banking network including both human cashiers and ATMs to be shared by a consortium of banks. Each bank provides its own computer to maintain its own accounts and process transactions against them. Cashier stations are owned by individual banks and communicate directly with their own bank's computers. Human cashiers enter account and transaction data.

ATMs communicate with a central computer that clears transactions with the appropriate banks. An ATM accepts a cash card, interacts with the user, communicates with the central systems to carry out the transaction, dispenses cash, and prints receipts.

The system requires appropriate recordkeeping and security provisions. The system must handle concurrent accesses to the same account correctly.

The banks will provide their own software for their own computers; you are to design the software for the ATMs and the network. The cost of the shared system will be apportioned to the banks according to the number of customers with cash cards.

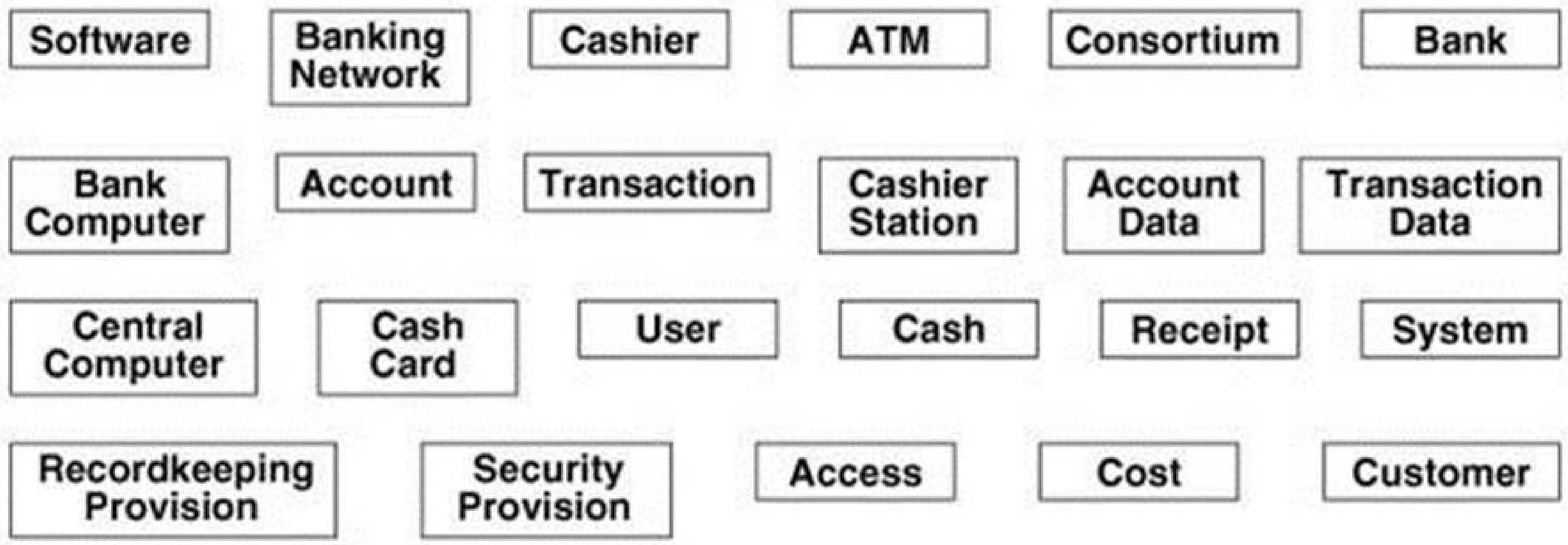


Figure 12.3 ATM classes extracted from problem statement nouns.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh.

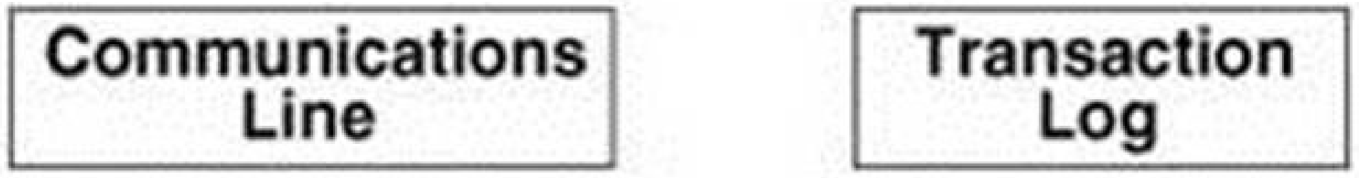


Figure 12.4 ATM classes identified from knowledge of problem domain.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh.

Eliminate Wrong Classes

- Redundant classes
- Irrelevant classes
- Vague classes
- Attributes
- Operations
- Roles
- Implementation constructs
- Derived classes

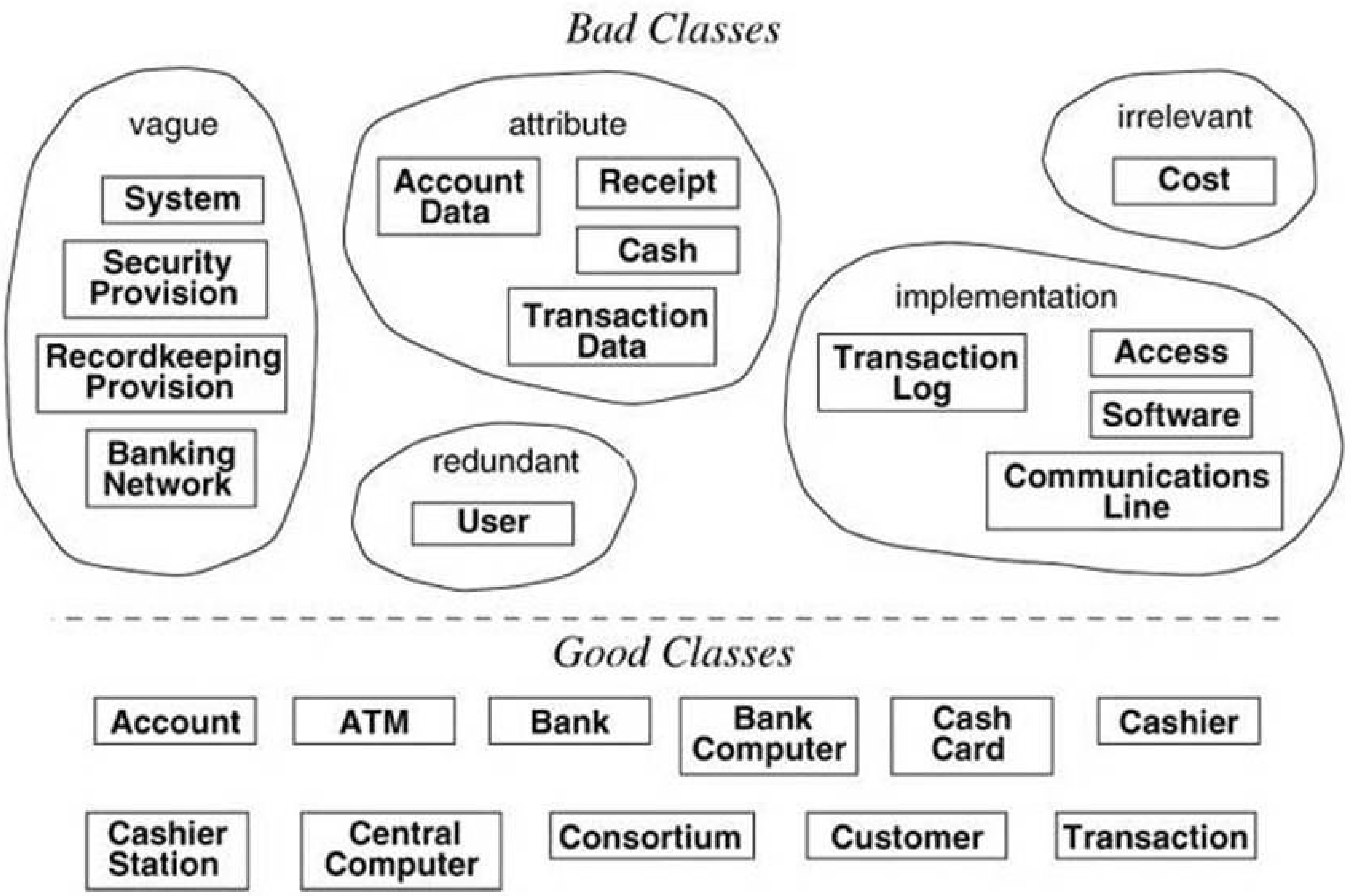


Figure 12.5 Eliminating unnecessary classes from ATM problem.

Object-Oriented Modeling and Design with UML, Second Edition by Michael Blaha and James Rumbaugh. ISBN 0-13-1-015920-4. © 2005 Pearson Education, Inc., Upper Saddle River, NJ. All rights reserved.

Relationships

- An object by itself is uninteresting.
- Objects contribute to the behavior of a system by collaborating with one another
- Consider the airplane - a collection of parts
- Only the collaborative efforts of all the component objects of an airplane enable it to fly
- Two of the important kinds of object relationships:

  1. Links
  2. Aggregation

Relationships

- A link means that messages may pass along this path
- Links denote peer-to-peer or client/supplier relationships
- Aggregation denotes a whole/part hierarchy

- An object is an instance of a class

Example

- A daisy **is a** kind of flower.
- A rose **is a** (different) kind of flower
- Red roses and yellow roses are both kinds of roses
- A petal is a **part of** both kinds of flowers
- Ladybugs eat certain pests, which may be infesting certain kinds of flowers

- From this simple example we conclude that classes, like objects, do not exist in isolation.

Three basic Relationships among classes

- The first of these is generalization/specialization, denoting an "is a" relationship.
- The second is whole/part, which denotes a "part of" relationship (aggregation)
- The third is association, which denotes some semantic dependency among otherwise unrelated classes