Dalhousie University
Faculty of Computer Science

CSCI 3132 – Object Orientation and Generic Programming

Week 2 – Domain Modelling

Contents

- Refining the use cases
- Domain Modelling concept
- Guidelines
- Extracting the domain model from high-level requirements
  - Identify domain classes
  - Remove ambiguities
  - Establish relationships
- Aggregation versus Generalization
- Examples and Exercises

Refining the Use Cases

Refining the Use Case

- When writing use cases, ask the following questions:
  - What happens
    - Starts your normal (sunny day) scenario
  - What happens next
    - Keep asking this until the sunny day scenario is complete
  - What else might happen
    - Identify all alternate (rainy-day) scenarios

Refining the Use Case – Example

- Write Reader Review use case
  - *"The Customer selects the book. The system displays a book detail page. The Customer clicks the Write Review button, and the system shows the Write Review screen. The user types in a review of the book, gives it a rating out of five stars, and clicks the Send button. The system ensures that the review isn't too long or short, and that the rating is out of five stars. The system then displays a confirmation screen, and the review is sent to a moderator, ready to be added."*[1]
- How can this use case be improved?

[1]Doug Rosenberg and Matt Stephens, "Use Case Driven Object Modeling with UML – Theory and Practice" Apress,

Refining the Use Case – Example

- Narrative should be in active voice
- Participating domain objects should be explicitly named
- Describes events leading up to the use case
  - These belong to a different use case and should be removed from here

Refining the Use Case – Example

- Write Reader Review use case
  - *"The Customer selects the book. The system displays a book detail page. The Customer clicks the Write Review button, and the system shows the Write Review screen. The user types in a review of the book, gives it a rating out of five stars, and clicks the Send button. The system ensures that the review isn't too long or short, and that the rating is out of five stars. The system then displays a confirmation screen, and the review is sent to a moderator, ready to be added."*[1]
- Some problems in the use case are highlighted.

[1]Doug Rosenberg and Matt Stephens, "Use Case Driven Object Modeling with UML – Theory and Practice" Apress,

Refining the Use Case – Example

- Modified Write **Customer** Review use case
  - *"The Customer **selects** Write Review for the book currently being viewed, and the system shows the **Write Review screen**. The **Customer** types in a **Book Review**, gives it a **Book Rating** out of five stars, and **sends** the Book Review. The system ensures that the Book Review is within 20 and 5000 words, and that the Book Rating is within one and five stars. The system then displays a confirmation message, and the review is sent to a **Moderator**, ready to be added."*[1]

[1]Adapted from Doug Rosenberg and Matt Stephens, "Use Case Driven Object Modeling with UML – Theory and Practice" Apress,

Refining the Use Case – Example

- Alternate courses
  - **User not logged in**: The user is first taken to the Login screen. Once successfully logged in, the user is taken to the Write Review screen.
  - **The user enters a review that is too long (text > 5000 words)**: The system rejects the review and responds with a message explaining why the review was rejected.
  - **The review is too short (< 20 words)**: The system rejects the review.

Refining the Use Case - Exercise

- Improve the following use cases, by identifying error/bad practices. Rewrite the use case text. Then compare your rewritten version with the improved version shown here.

Example Use Case – Edit Shopping Cart

- PRECONDITIONS:
  - The user has logged in.
  - The user has navigated to the Edit Shopping Cart page.
- BASIC FLOW:
  - The user adds or removes whatever items she wants to change, and then clicks the Update button. The system adds or removes the items, and then displays the page with the updated shopping cart.
- ALTERNATE FLOWS:
  - **Shopping cart is empty**: No items can be removed.

Problems Identified

- The basic text doesn't describe a specific scenario, but instead tries to cover all bases. As a result, an important behavioral aspect is missed: the user wouldn't necessarily want to add items from this page, just remove them (or change the quantity).

- The alternate course doesn't tie into any particular action in the use case text. There are also several relatively obvious alternate courses that are missing.

Improved Use Case

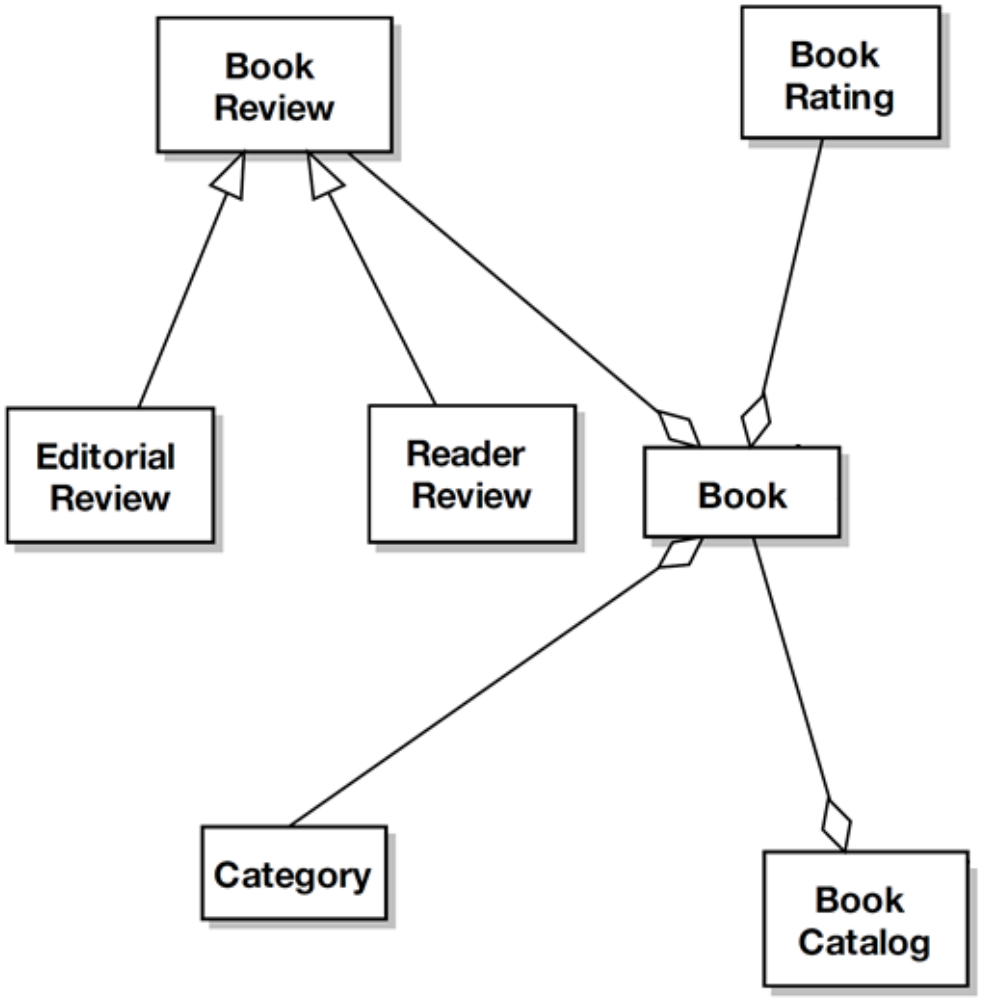
- **BASIC FLOW**:
  - The system displays the Shopping Cart page. The user selects Remove next to a Line Item. The system removes the item from the user's Shopping Cart, and then redisplays the page. The user then selects the Quantity text field for another Line Item, changes its value from 1 to 2, and selects Update. The system updates the Shopping Cart, recalculates the total amount, and redisplays the page.
- **ALTERNATE FLOWS**:
  - **Item not found**: The item that the user chose to remove wasn't found in the Shopping Cart. The system refreshes the Shopping Cart page, along with a warning message that the user's action failed because the page was out of date.
  - **Quantity changed to zero**: This counts as removing the item, so the item is removed from the Shopping Cart.
  - **Negative value or non-numeric "value" entered**: The page is redisplayed with the original Quantity value, and a message next to it informs the user that she entered an invalid value.

Domain Modelling

Domain Modelling

- Domain model
  - Is a live dictionary of terms used in your project
  - Defines the scope for the project
  - Forms the foundation on which use cases are built
- Provides a common vocabulary for clear communication
- Can be considered as a simplified class diagram

Example of a Domain Model



Domain Model versus Use Case

- Useful to make a domain model, then write use cases
- Domain model forms static while use cases form the dynamic part of the system model
  - Static part describes the structure
  - Dynamic part describes the behaviour

Top 10 Domain Modelling Guidelines[1]

1. Focus on real-world (problem domain) objects.
2. Use generalization (is-a) and aggregation (has-a) relationships to show how the objects relate to each other.
3. Limit your initial domain modeling efforts to a couple of hours.
4. Organize your classes around key abstractions in the problem domain.
5. Don’t mistake your domain model for a data model.
6. Don’t confuse an object (which represents a single instance) with a database table (which contains a collection of things).
7. Use the domain model as a project glossary.

[1] Doug Rosenberg and Matt Stephens, “Use Case Driven Object Modeling with UML – Theory and Practice” Apress, pp. 26

Top 10 Domain Modelling Guidelines[1]

8. Do your initial domain model before you write your use cases, to avoid name ambiguity.
9. Don't expect your final class diagrams to precisely match your domain model, but there should be some resemblance between them.
10. Don't put screens and other GUI-specific classes on your domain model.

[1] Doug Rosenberg and Matt Stephens, "Use Case Driven Object Modeling with UML – Theory and Practice" Apress, pp. 26

Extracting the Domain Model from High-Level Requirements

Internet Bookstore Example[2]

The bookstore must be able to sell books from sellers, with orders accepted over the Internet. The user must be able to add books into an online shopping cart, prior to checkout. Similarly, the user must be able to remove items from the shopping cart. The user must be able to maintain wish lists of books that he or she wants to purchase later. The user must be able to cancel orders before they've shipped. The user must be able to pay by credit card or purchase order. The customer must be able to create a customer account, so that the system remembers the user's details (name, address, credit card details) at login. The system shall maintain a list of user accounts in its central database. When a user logs in, his or her password must always be matched against the passwords in the master account list. The user must be able to search for books by various search methods—title, author, keyword, or category – and then view the books' details.

[2] Adapted from Rosenberg et. Al., "Use Case Driven Object Modeling with UML – Theory and Practice" example

Step 1 – Identify Domain Classes

The **bookstore** must be able to sell **books** from sellers, with **orders** accepted over the **Internet**. The user must be able to add books into an online **shopping cart**, prior to **checkout**. Similarly, the user must be able to remove **items** from the shopping cart. The user must be able to maintain **wish lists** of books that he or she wants to purchase later. The user must be able to cancel orders before they've shipped. The user must be able to pay by **credit card** or **purchase order**. The **customer** must be able to create a **customer account**, so that the system remembers the user's details (name, address, credit card details) at login. The system shall maintain a **list of user accounts** in its central database. When a user logs in, his or her **password** must always be matched against the passwords in the **master account list**. The user must be able to search for books by various **search methods—title**, **author**, **keyword**, or **category –** and then view the **books' details**.

Adapted from Rosenberg et. Al., "Use Case Driven Object Modeling with UML – Theory and Practice" example

Step 1 – Identify Domain Classes

Separate nouns and noun phrases

- bookstore
- book
- seller
- order
- Internet
- shopping cart
- checkout
- item
- wish list
- credit card
- purchase order
- customer
- user
- customer account
- user account
- list of accounts
- password
- master account list
- search method
- title
- author
- keyword
- category
- books' details

Step 2 – Remove Ambiguities

- Remove duplicates
- Remove actors
- Remove too small, too broad or generic objects

Step 2 – Remove Ambiguities

Separate nouns and noun phrases

- ~~***bookstore***~~
- **book**
- ~~seller~~
- order
- ~~*Internet*~~
- shopping cart
- checkout
- *line* item
- wish list
- credit card
- purchase order
- ~~customer~~
- ~~user~~
- **customer account**
- ~~**user account**~~
- ~~**list of accounts**~~
- ~~*password*~~
- **master account list**
- search method
- ~~*title*~~
- author
- ~~*keyword*~~
- category
- ~~**books' details**~~

Adapted from Rosenberg et. Al., "Use Case Driven Object Modeling with UML – Theory and Practice" example

Step 2 – Remove Ambiguities

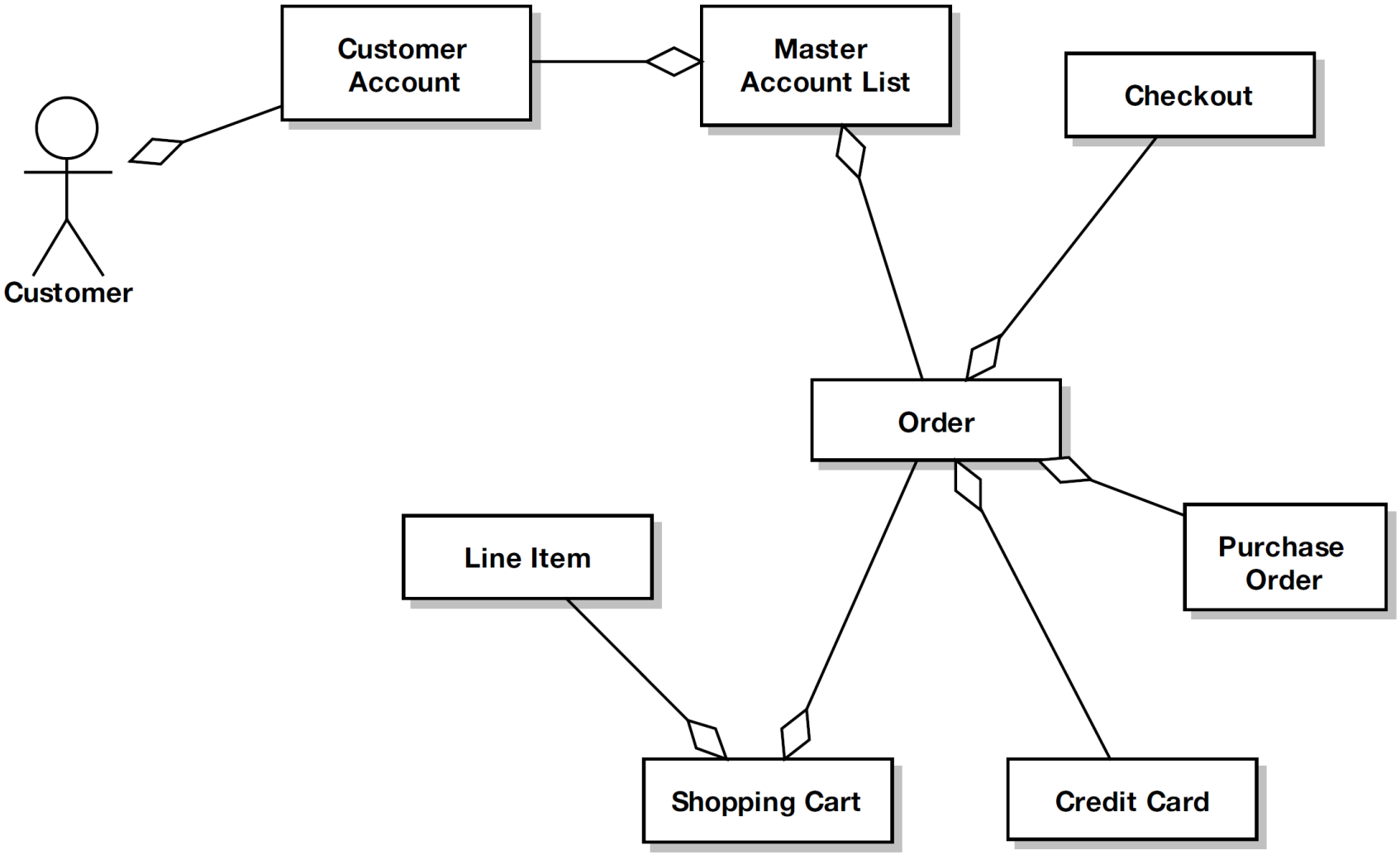
Separate nouns and noun phrases

- book
- order
- shopping cart
- checkout
- line item
- wish list
- credit card
- purchase order
- customer account
- master account list
- search method
- author
- category

Step 3 – Establish Relationships

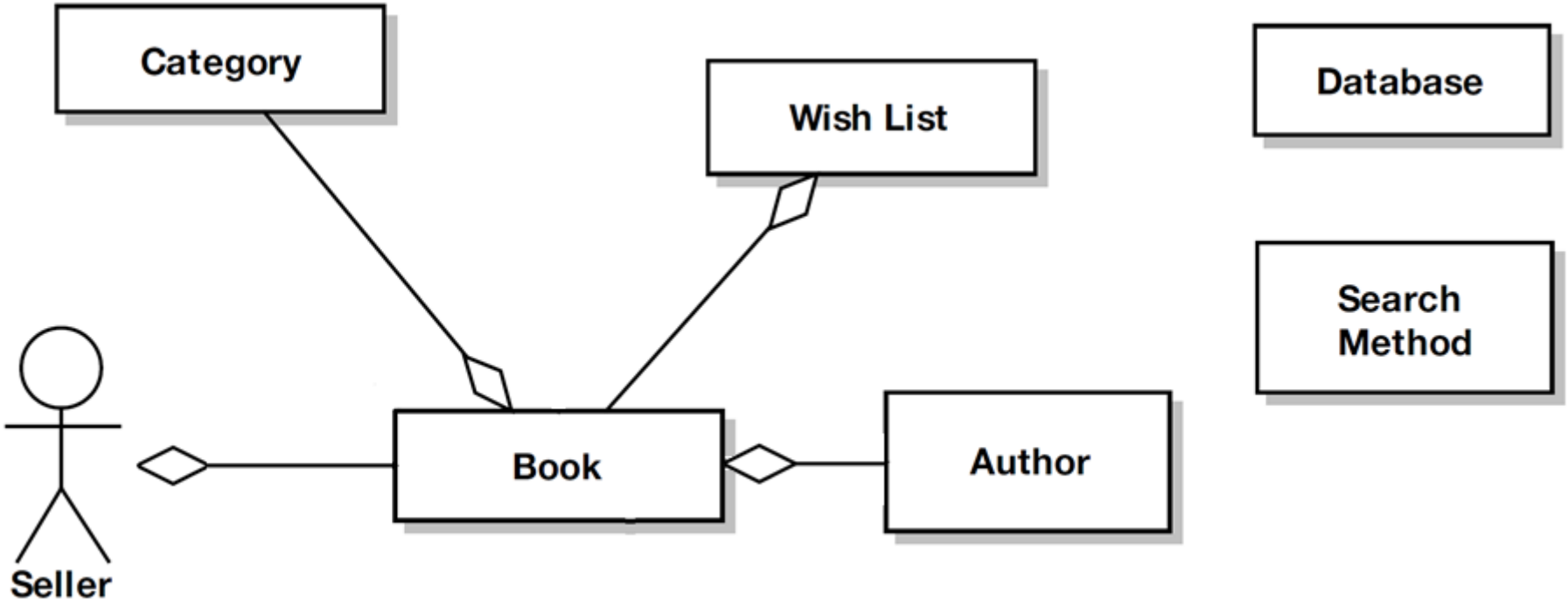
- Aggregation or *has-a* relationship
  - E.g. Book has an Author, Shopping cart has line items
- Generalization or *is-a* (kind-of) relationship
  - E.g. Cat is a kind-of animal, Related books is a kind-of book list

Step 3 – Establish Relationships



Adapted from Rosenberg et. Al., "Use Case Driven Object Modeling with UML – Theory and Practice" example

Step 3 – Establish Relationships



Q: Can you spot an incorrect relationship?

Which one is correct? Author has a Book, or **Book has an Author**

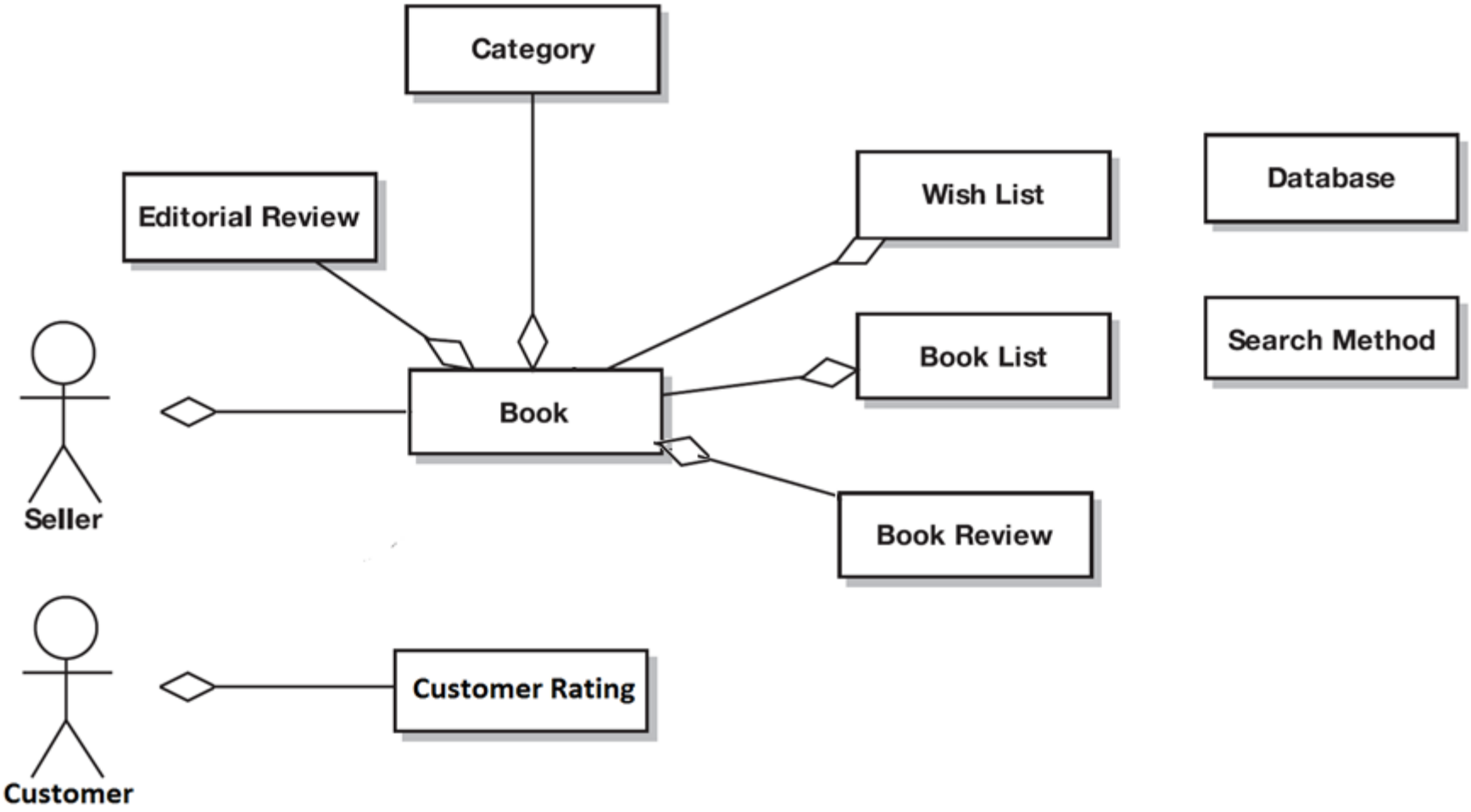
Adapted from Rosenberg et. Al., "Use Case Driven Object Modeling with UML – Theory and Practice" example

Exercise

- It must be possible for the user to post reviews of favorite books; the **review comments** should appear on the book details screen. The review should include a **customer rating** (1–5), which is usually shown along with the book title in **book lists**. It must be possible for staff to post **editorial reviews** of books. These should also appear on the book details screen.

Adapted from Rosenberg et. Al., "Use Case Driven Object Modeling with UML – Theory and Practice" example

Solution



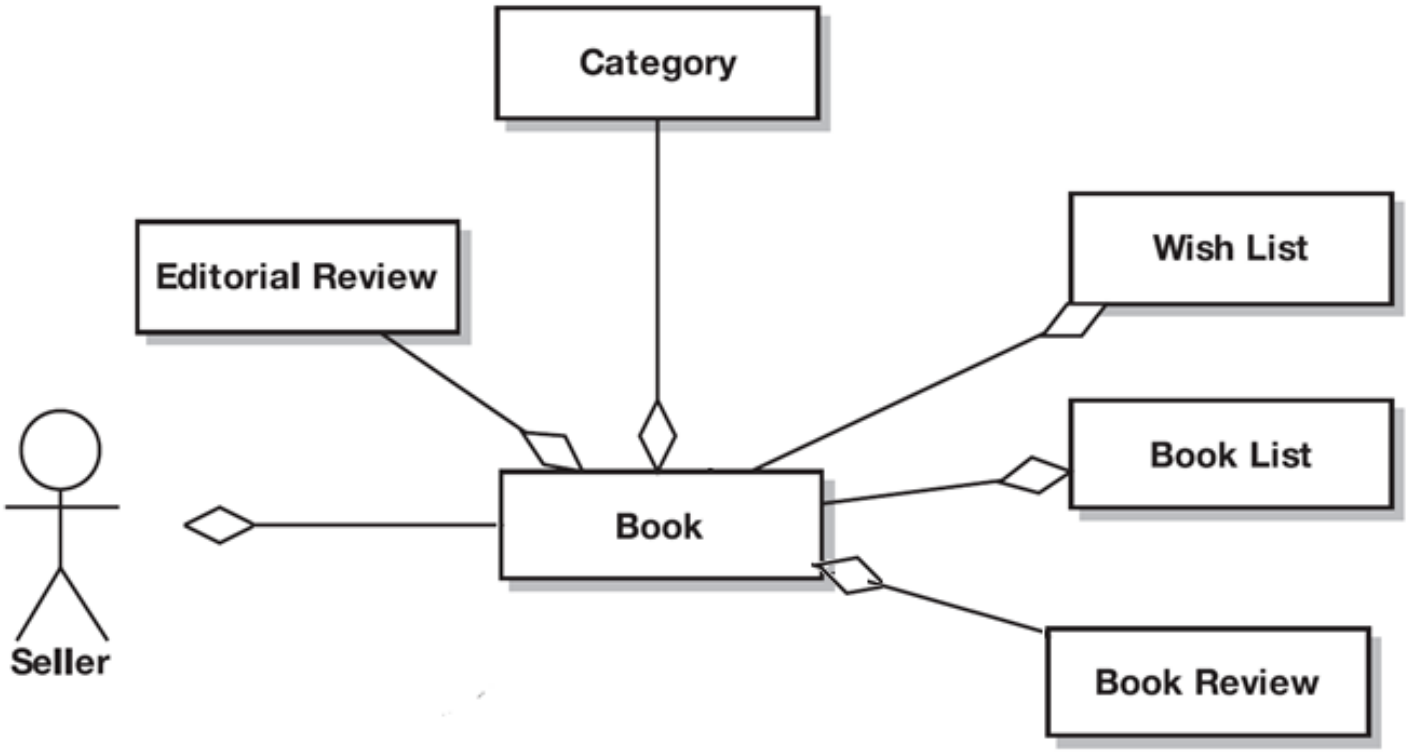
Adapted from Rosenberg et. Al., "Use Case Driven Object Modeling with UML – Theory and Practice" example

Generalization Relationship

- Often known as the "is-a" relationship
- The more general class is called a superclass while the more specific class is called a subclass
  - Creating subclasses from a superclass is known as **subtyping**
  - The subclasses **inherit** the attributes and operations (or methods) of their superclass
- Generalization relationships are implemented through **inheritance**

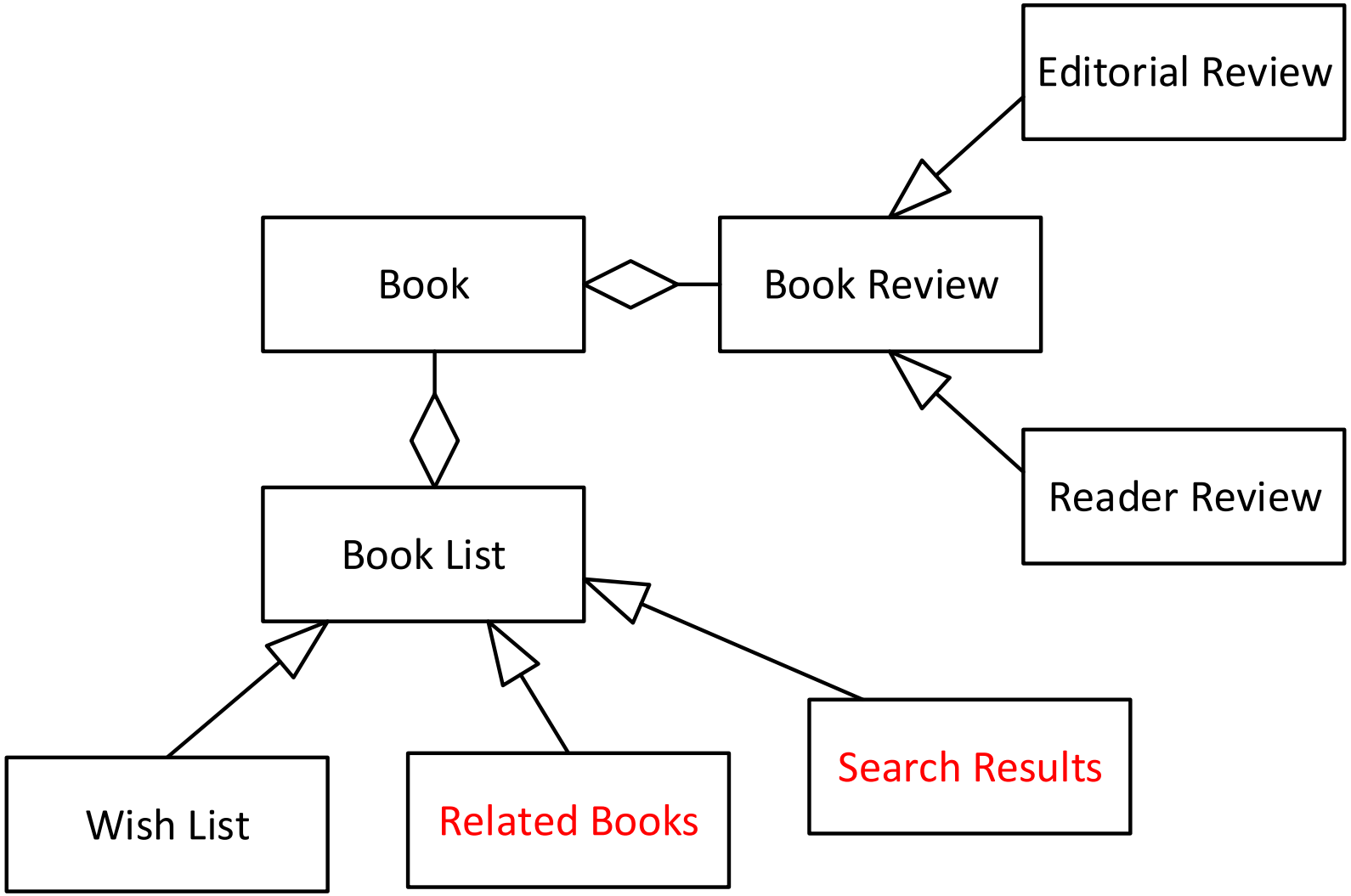
Generalization Example

- Can you identify any generalization relationships among the classes shown below



Adapted from Rosenberg et. Al., "Use Case Driven Object Modeling with UML – Theory and Practice" example

Generalization Example



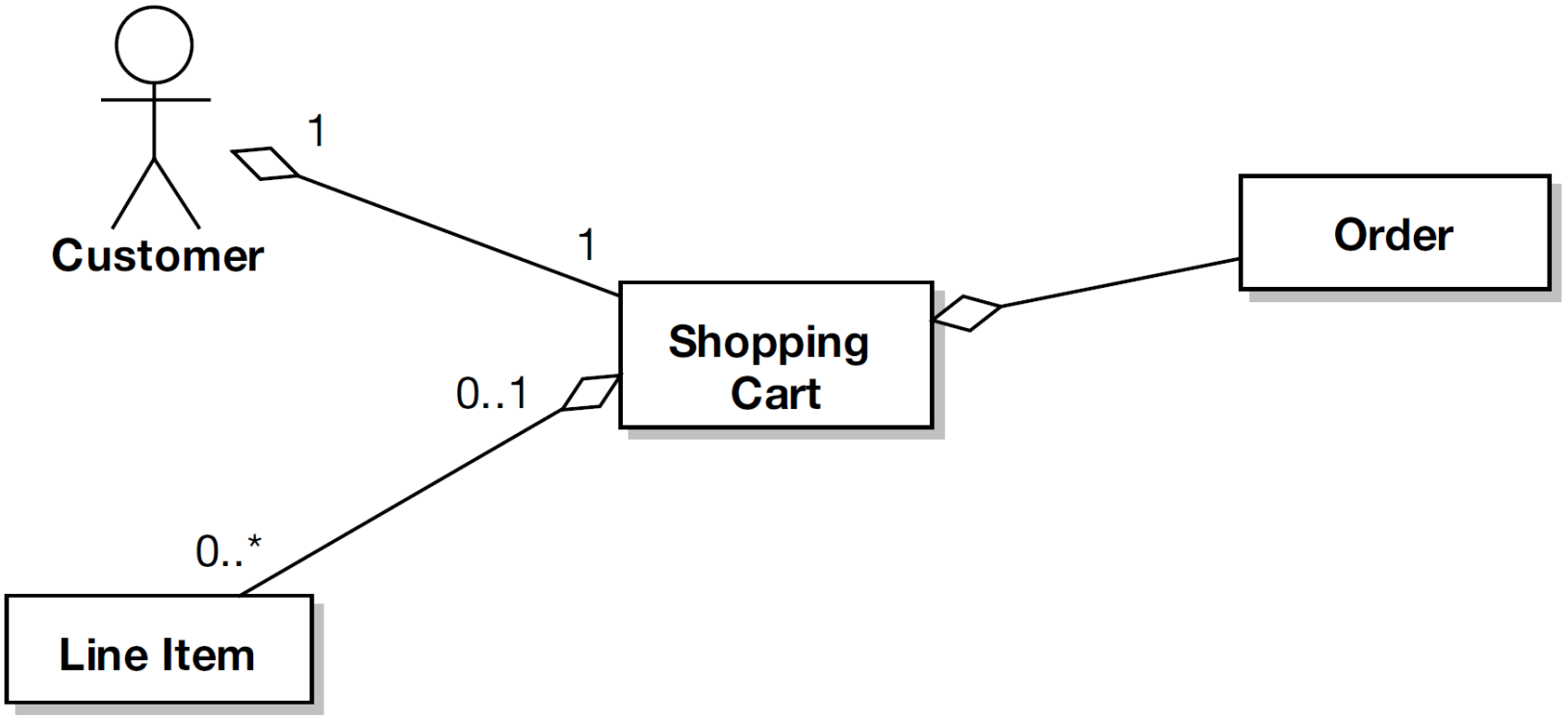
Adapted from Rosenberg et. Al., "Use Case Driven Object Modeling with UML – Theory and Practice" example

Domain Modeling Examples

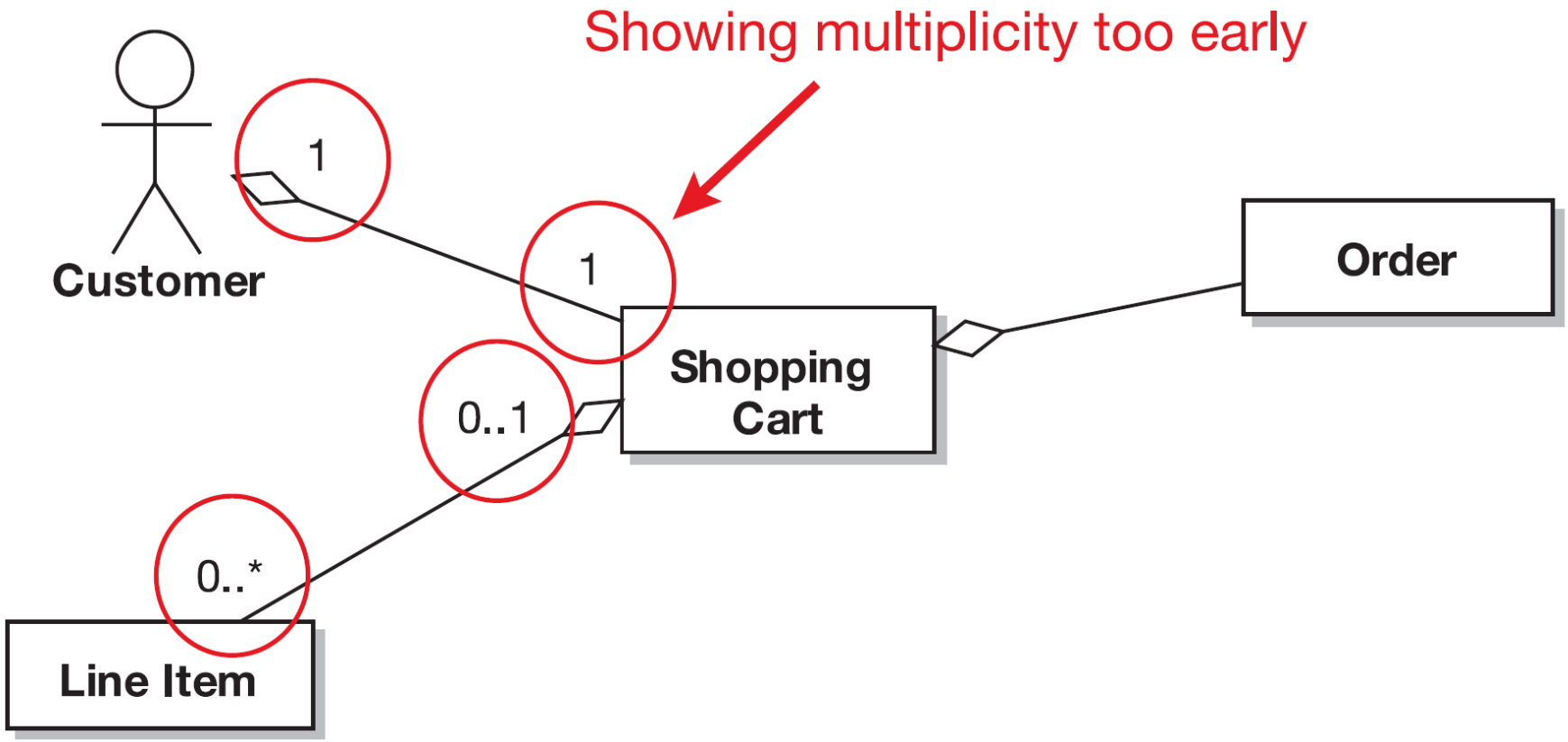
- Let us view some first pass domain models containing process-related or conceptual errors.
  - We need to find and fix these errors

Example 1

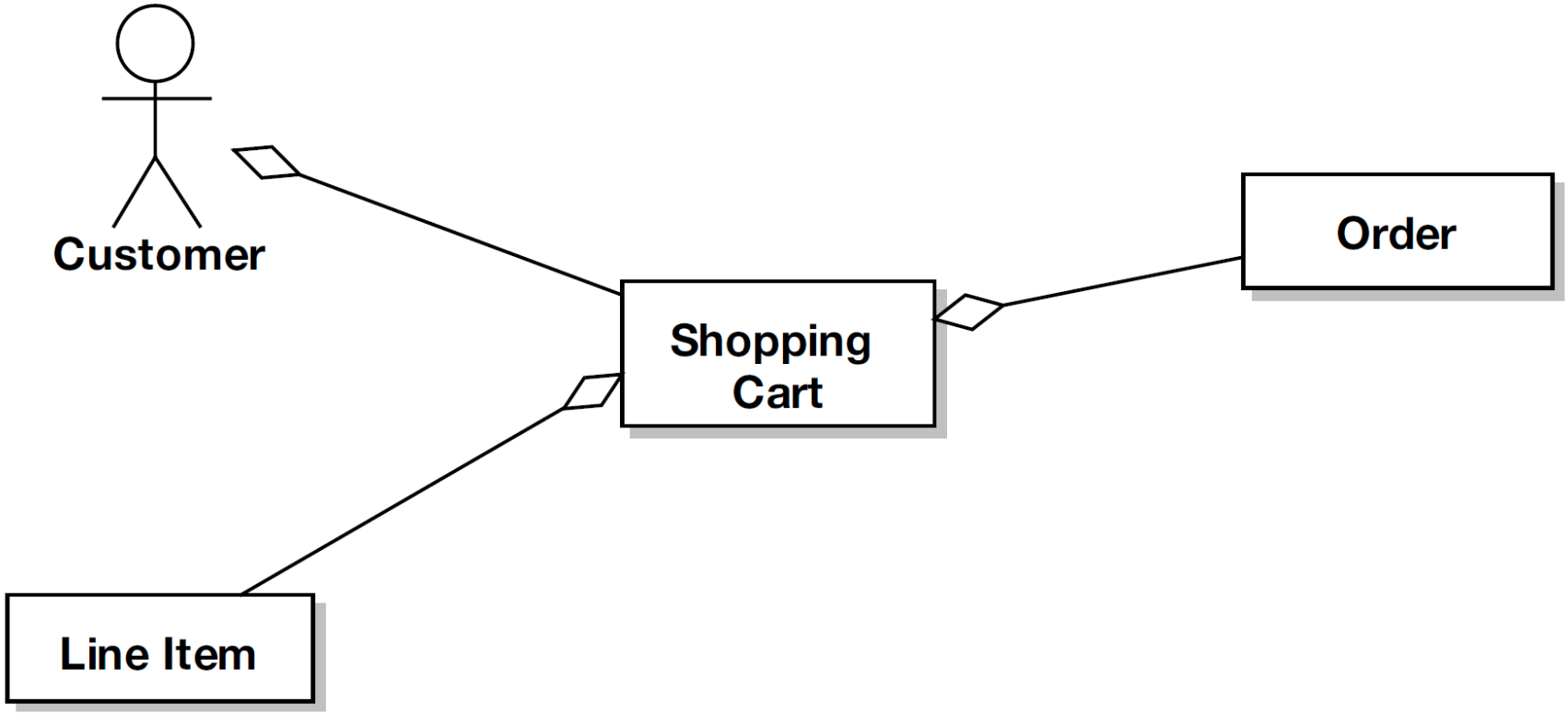
- UML syntax is correct, however, diagram has a process-related error. Identify it.



Solution – Example 1

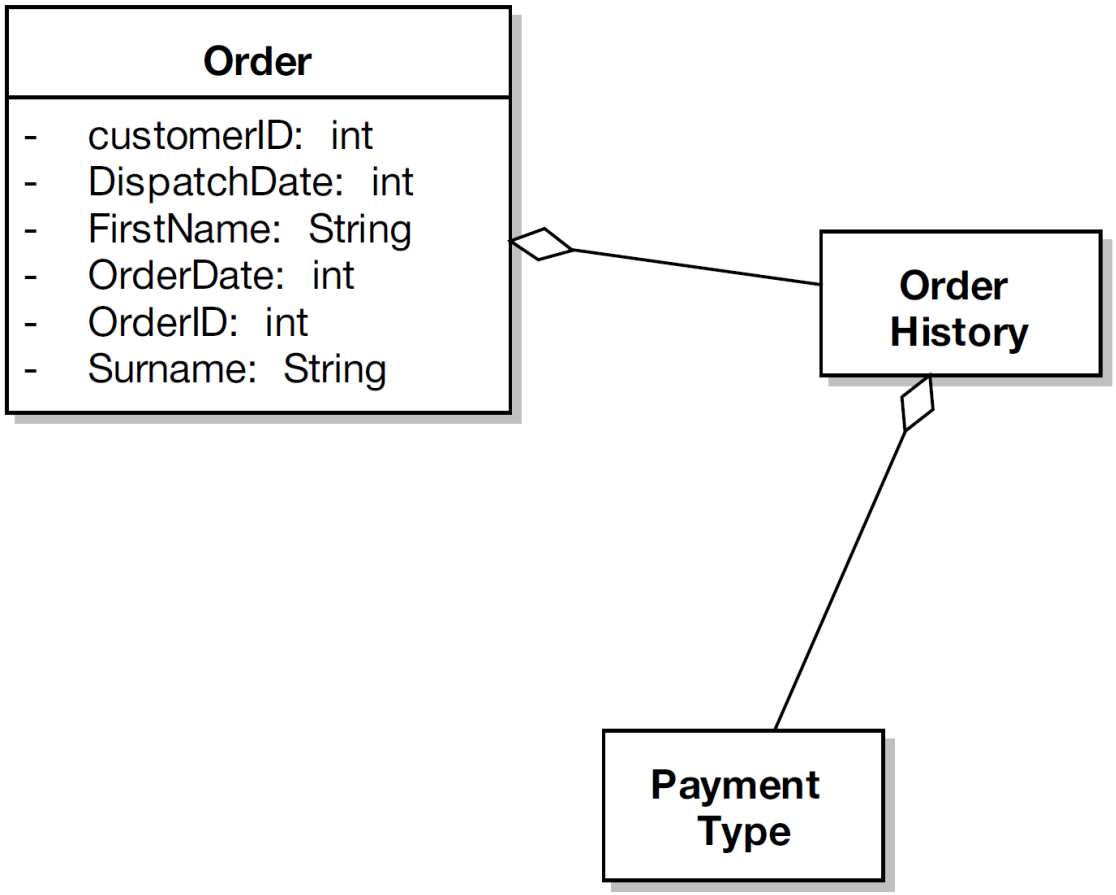


Solution – Example 1



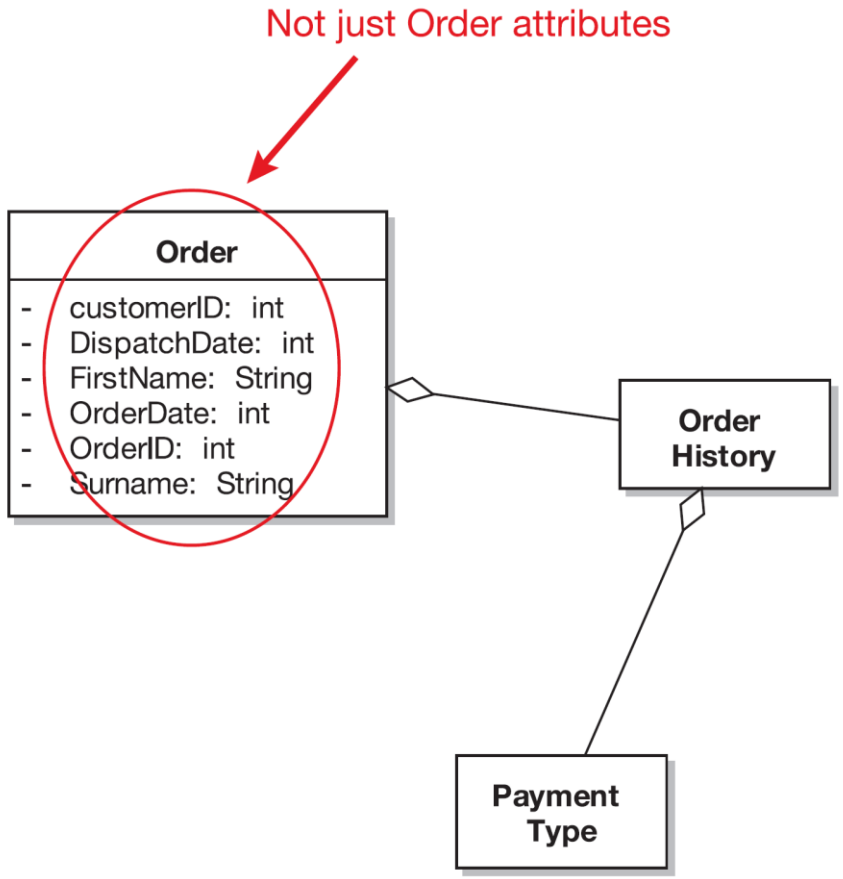
Example 2

- Figure shows a domain model diagram with attributes on the Order class. What database related problem does the diagram suggest?



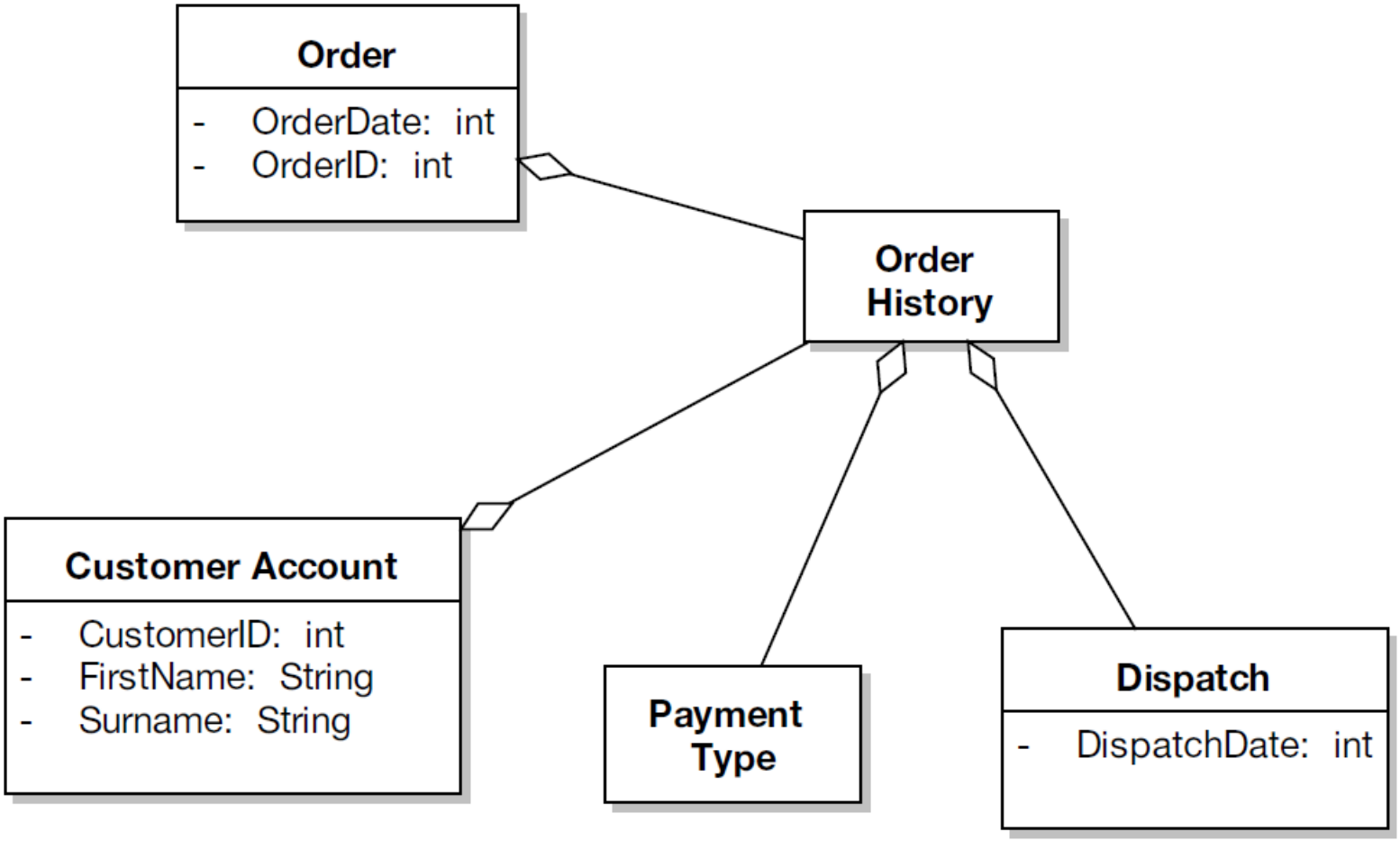
Solution – Example 2

- Order domain class includes attributes that do not belong in an order class. Domain classes seem to be mapped directly from a relational database schema.



Solution – Example 2

- Extra attributes separated out into their own class



UML

Unified Modelling Language

What is UML?

- A visual modelling language for Object Oriented modelling
- Three common approaches* to UML modelling:
  - UML as a sketch
  - UML as a blueprint
  - UML as executable

*Stephan J. Mellor, "Agile MDA", MDA Journal, June 2004

UML Standard Diagrams

- Structural diagrams
  - Class diagram
  - Object diagram
  - Component diagram
  - Deployment diagram
- Behavioral diagrams
  - Use case diagram
  - Sequence diagram
  - Collaboration diagram
  - Statechart diagram
  - Activity diagram

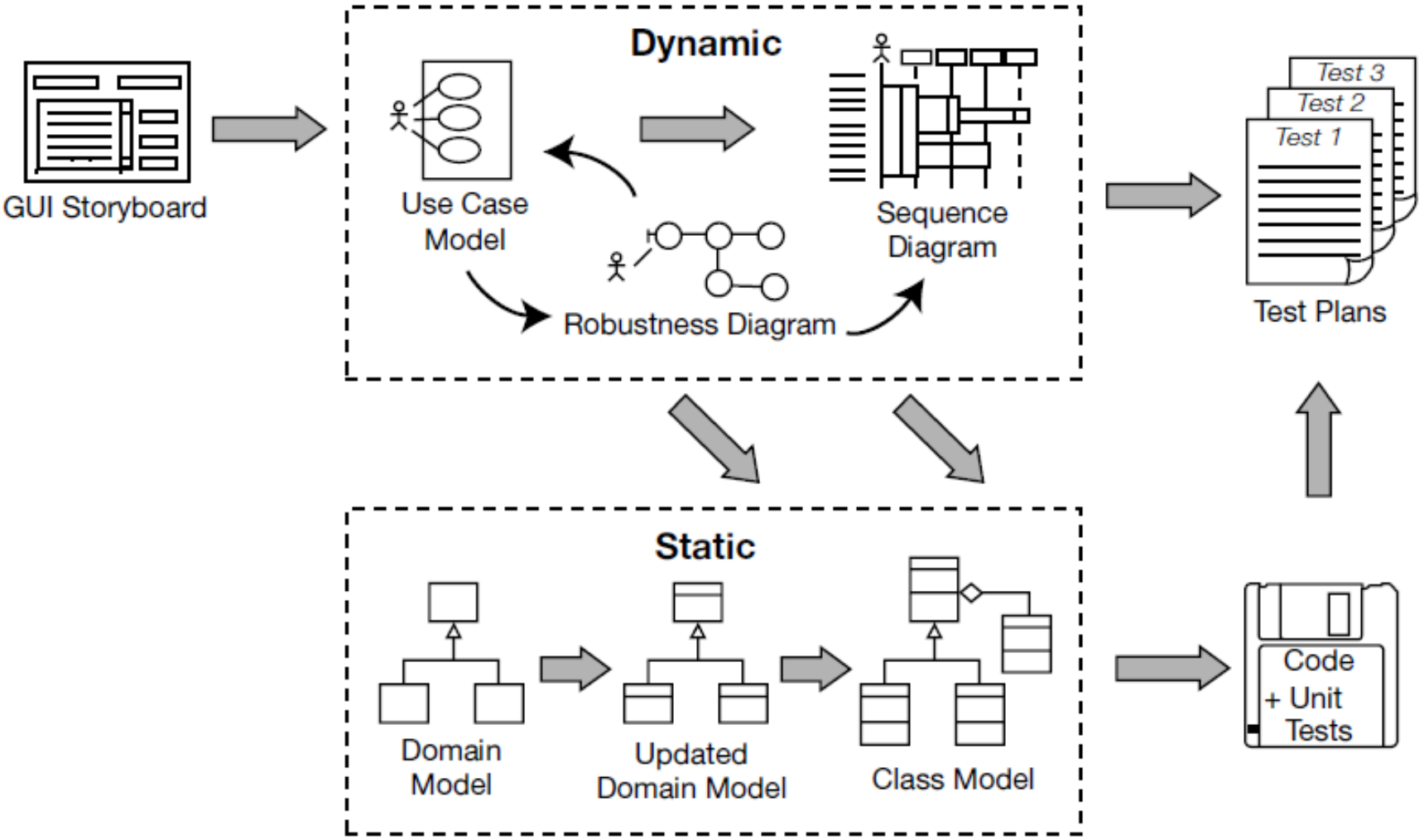
UML

- All aspects of UML are useful
  - In practice, not enough time available to do modelling, analysis and design
  - Pressure from management to jump to code
- Difficult to codify OOAD approaches in simple repeatable set of steps (a cookbook!)

UML Cookbook – ICONIX

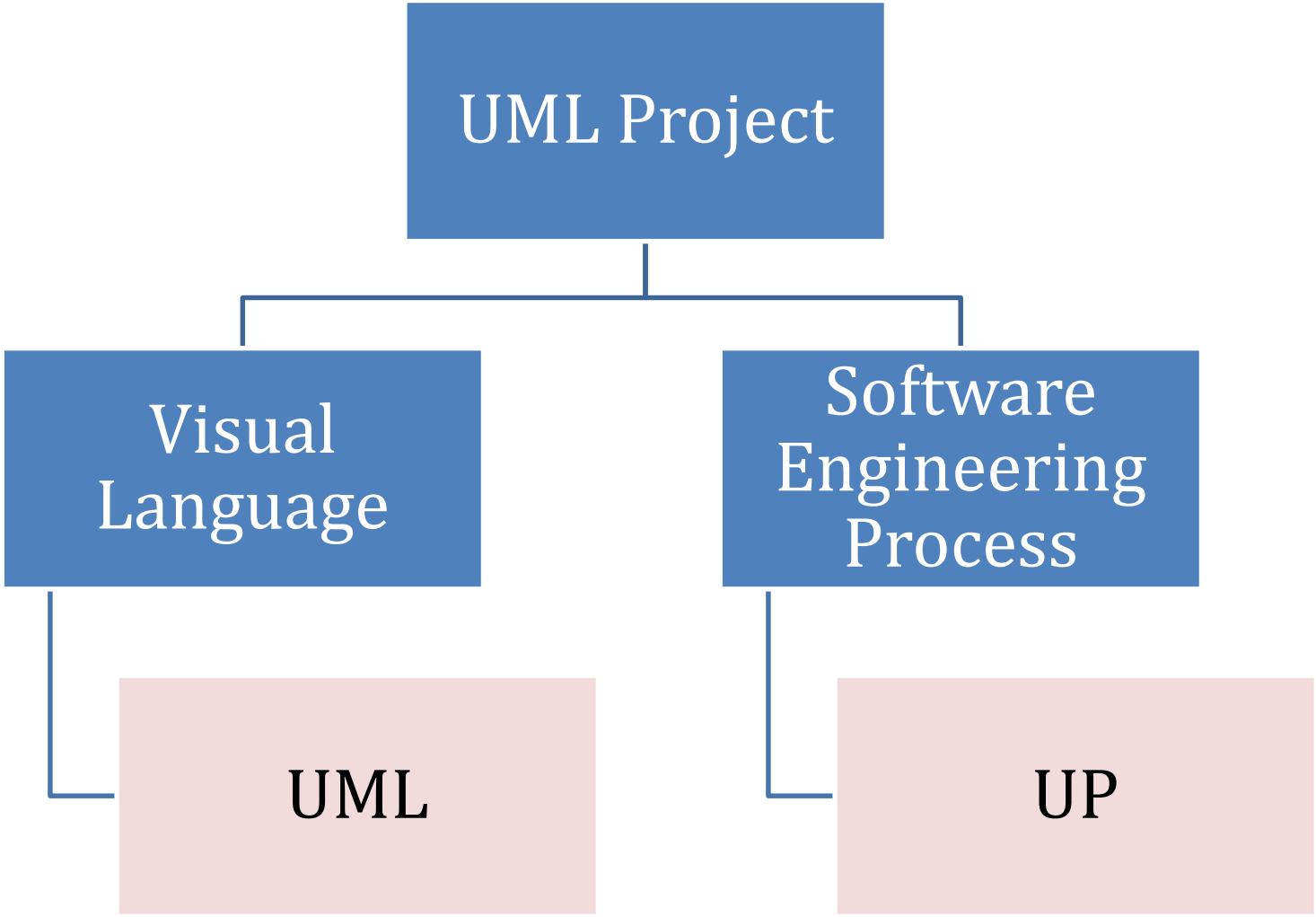
- ICONIX process lays down a simple, minimal set of steps
  - Pros
    - Generally leads to consistently good results
    - As close as anything to a simple cookbook approach
  - Cons
    - Assumes sophisticated design, not needed by every project

The ICONIX Process

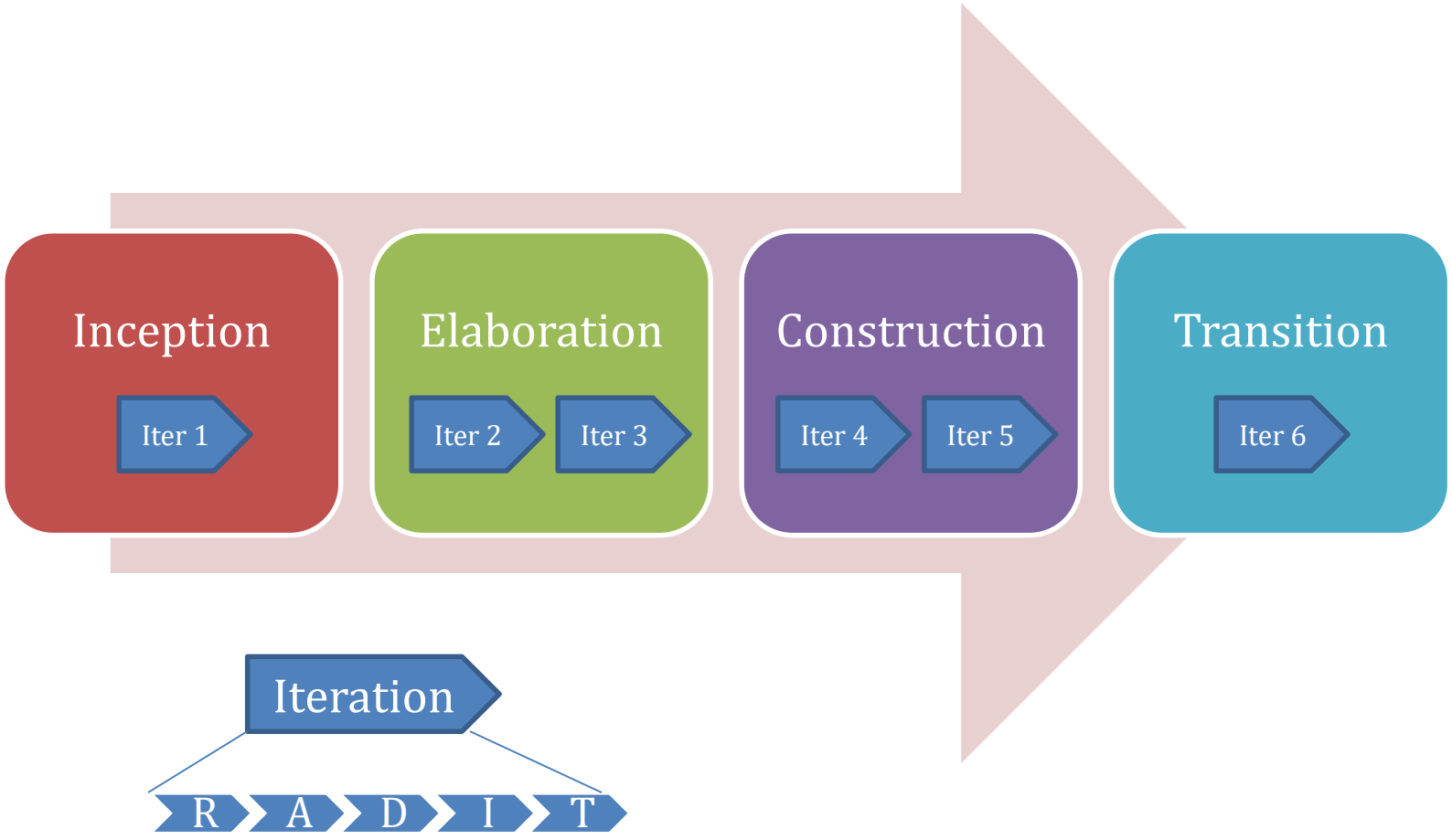


Doug Rosenberg and Matt Stephens, "Use Case Driven Object Modeling with UML – Theory and Practice" Apress,

The Unified Process (UP or USEP)



The Unified Process (UP)



The Unified Process (UP)