UML for e-Commerce

Doug Rosenberg
ICONIX Software Engineering, Inc.

http://www.iconixsw.com

About this webinar

- Based on ICONIX UML for e-Commerce class
- http://www.iconixsw.com/UMLecommerce.html
- Based on Internet Bookstore Example from workbook (Applying Use Case Driven Object Modeling)
- http://www.iconixsw.com/UMLworkbook.html
- Preview of public classes
- http://www.iconixsw.com/public_courses.htm

UML for e-Commerce

- 2 or 3 day course targeted at internet/intranet development
- Students walk through the internet bookstore example.
- Hands on lab where the example is extended.
- This webinar will walk through 2 use cases
- Complete example available in the workbook
- http://www.iconixsw.com/UMLecommerce.html

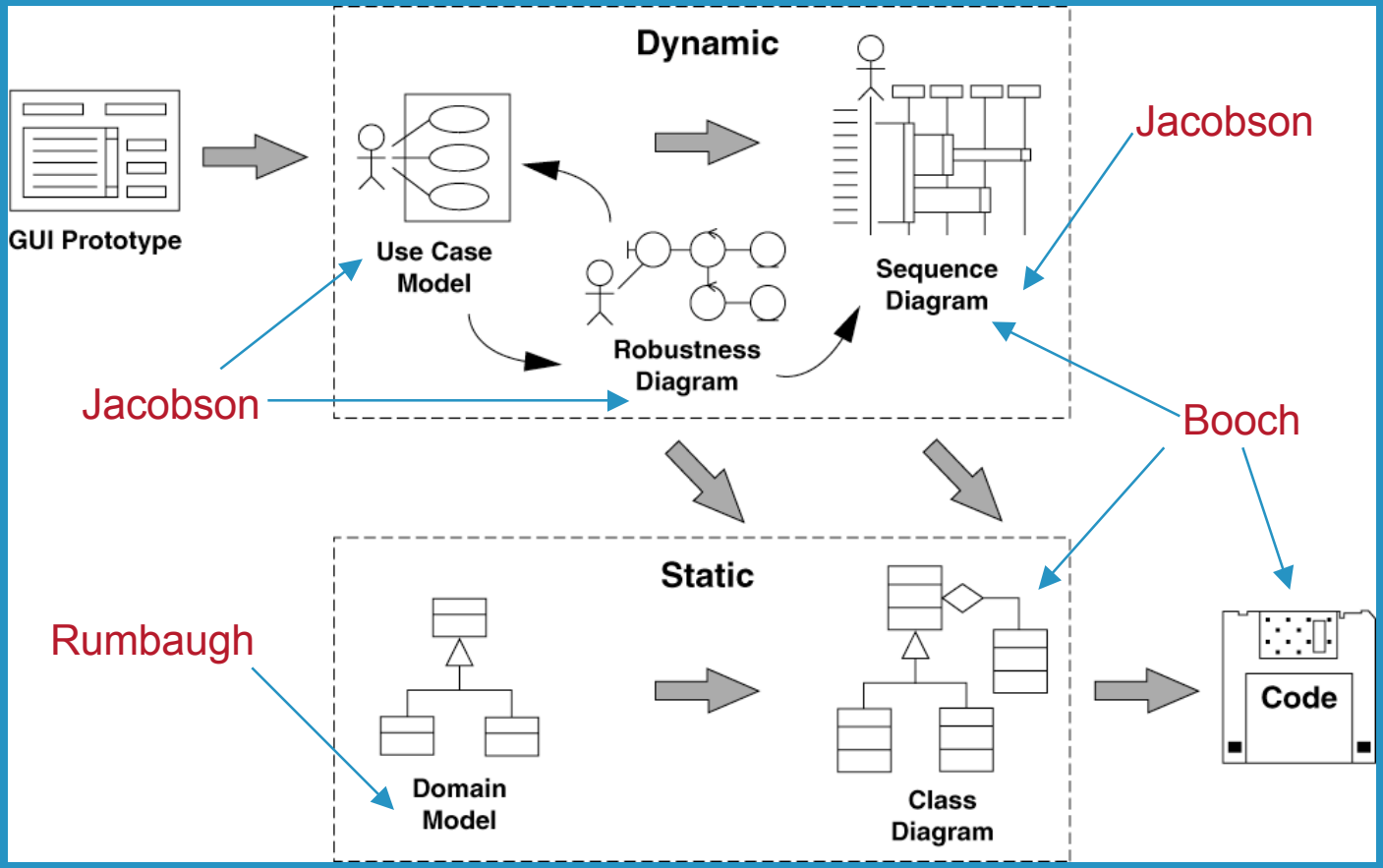
The Workbook

- Companion work to Use Case Driven Object Modeling
- First book provides theory
- Workbook is focused on practice
- Internet Bookstore example is dissected in great detail, starting from requirements through detailed design
- ICONIX Process is explained in detail: domain models, use cases, robustness diagrams, sequence diagrams, detailed static models
- 3 chapters on reviews
- http://www.iconixsw.com/UMLworkbook.html

ICONIX Process

- Synthesized from original Booch/Rumbaugh/Jacobson methods before UML existed (1992)
- Refined over 10 years, hundreds of training workshops and products
- Minimal yet sufficient subset of UML that is almost universally needed
- Book: Use Case Driven Object modeling
- http://www.iconixsw.com/UMLBook.html
- CDROM: Inside the ICONIX Process
- http://www.iconixsw.com/ICONIXProcess.html

Main elements of the ICONIX Process, and where they came from



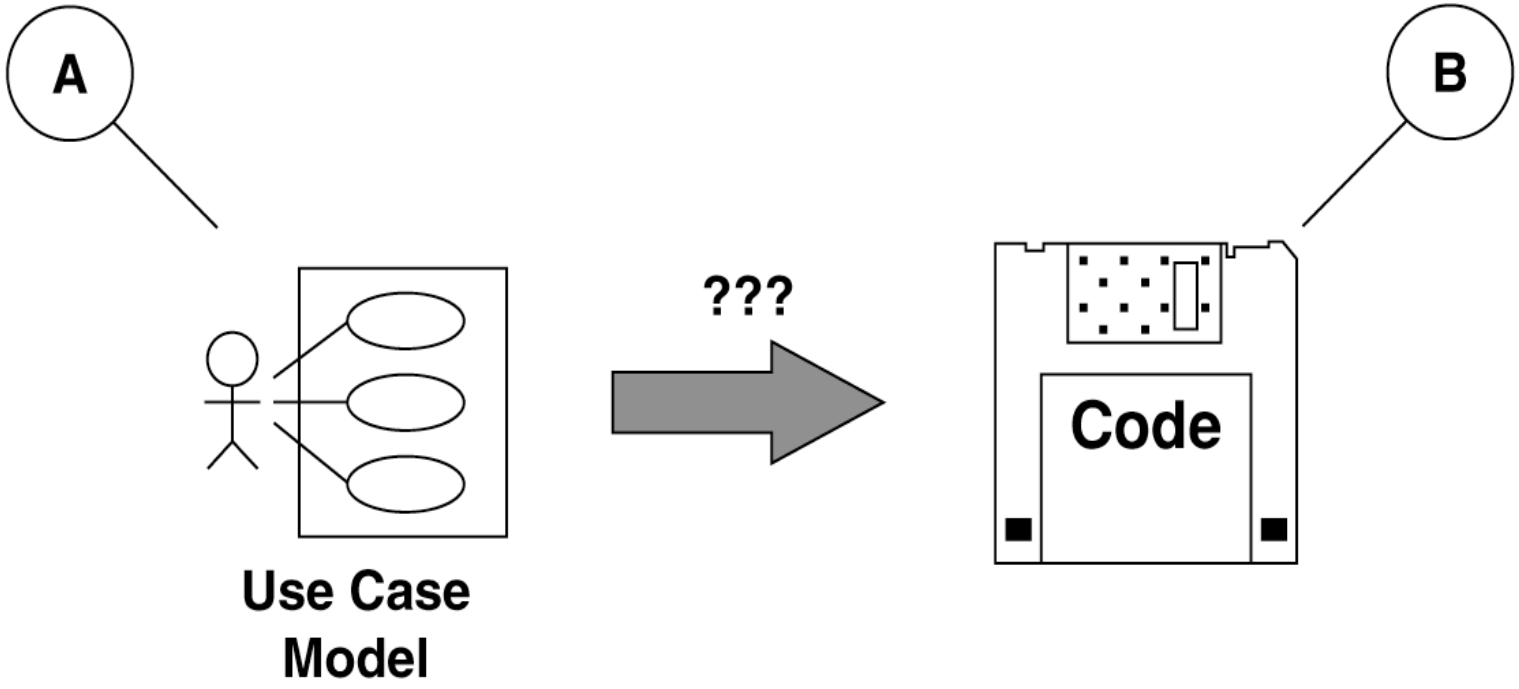
Theory vs. practice

- In theory, there is no difference between theory and practice, but in practice there is.
- In practice, there's never enough time for modeling.
- The ICONIX Process is a STREAMLINED approach to software development that helps you get from use cases to code quickly and efficiently, using a concentrated subset of the UML and related tools and techniques.

Minimal, yet sufficient

- UML User Guide, Ch. 32, page 431 says...
- 80% of modeling can be done with 20% of the UML.
  Which 20% was that again?
- We're supposed to be "Use Case Driven" but...
- "How do we get from Use Cases to Code???"
- Smaller than RUP, bigger than XP
- Add additional UML diagrams as needed

ICONIX Process: Do OOAD but Keep It Simple

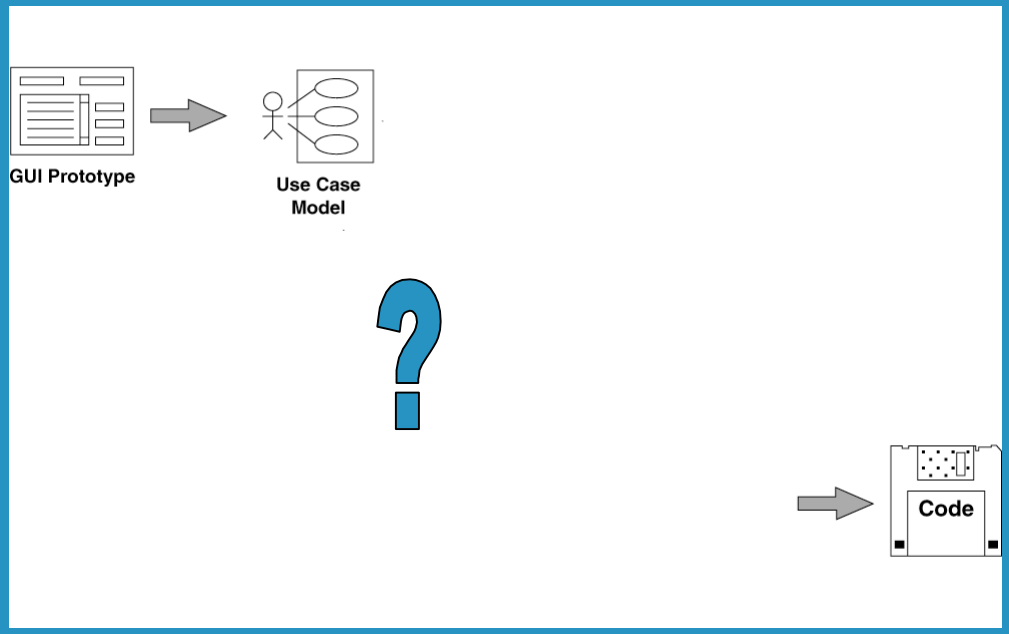


How do we get from use cases to code?

Let's work backwards from code

Let's assume that we've done a little prototyping, and started to write some use cases.

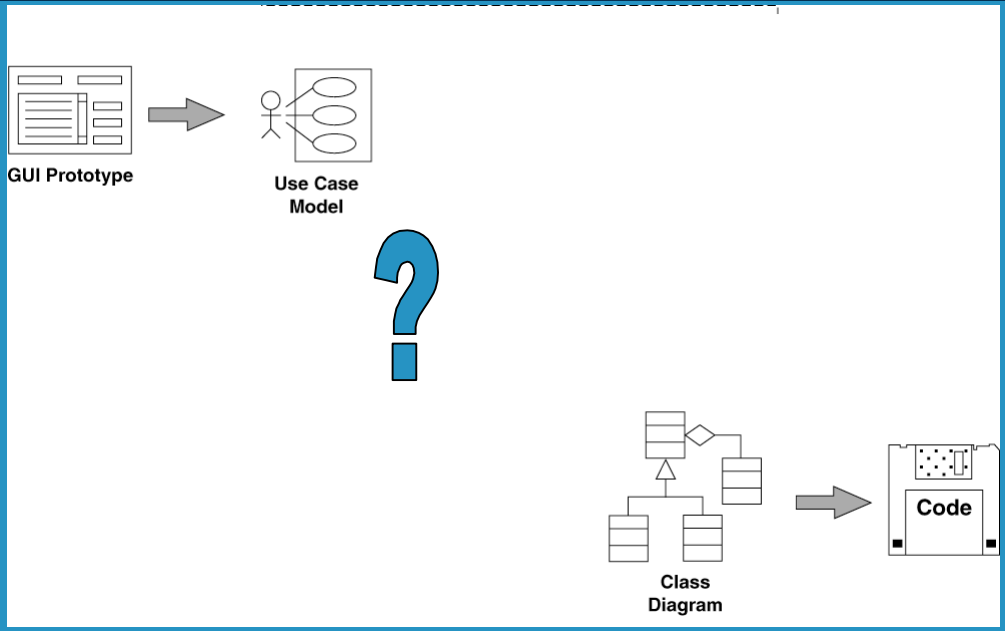
But code is our desired destination.



Before we get to code...

- We need a complete set of classes, with accompanying attributes and methods.
- We show this information on design-level class diagrams.

Design-Level Class Diagrams

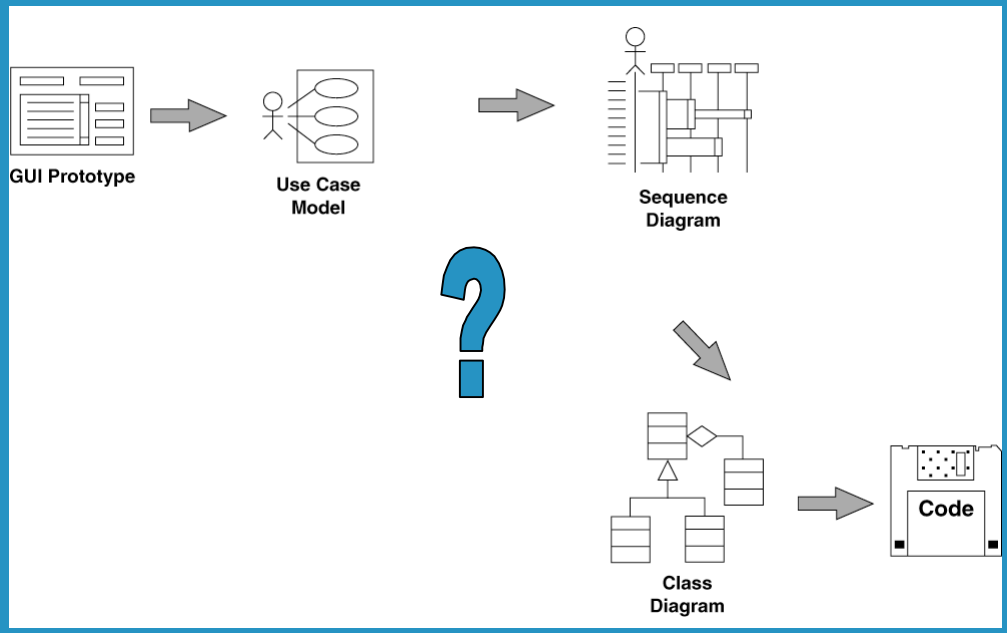


Our design-level class diagrams serve as the structure for our code.

Before we have classes with attributes and methods, though...

- We need to allocate behavior into our classes
- We have only enough information to make good decisions about which classes are responsible for which methods while we are drawing sequence diagrams.
- So, we need to draw a sequence diagram for each use case.

Sequence Diagrams

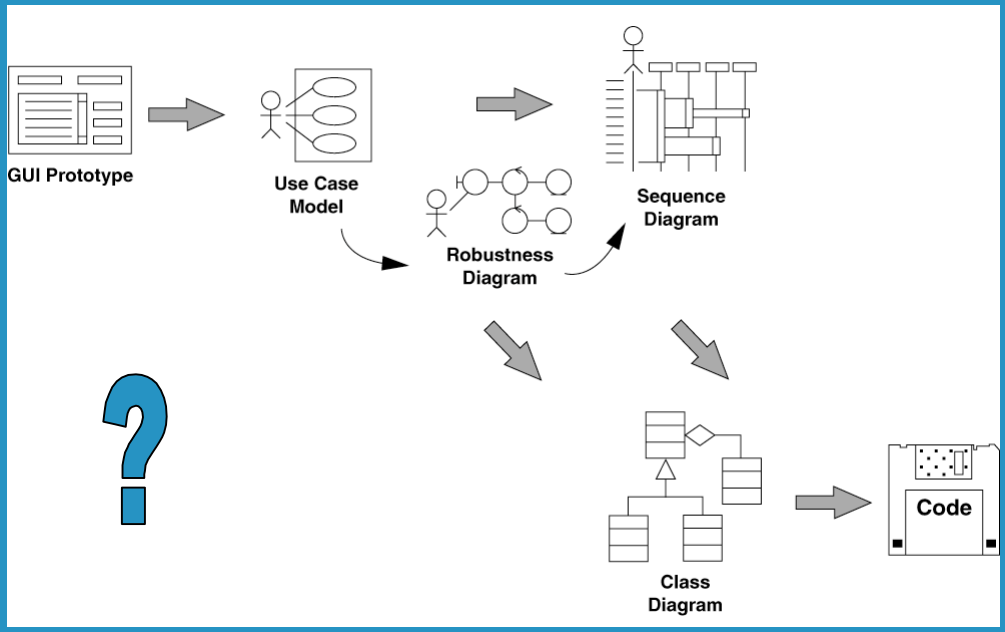


We allocate methods to classes as we draw sequence diagrams.

Before we do sequence diagrams, though...

- We need to have a good idea about what objects will be performing in which use case, and what functions the system will perform as a result of user actions.
- We get this information from robustness diagrams, the result of robustness analysis.

Robustness Diagrams -- the missing link!



We discover new objects, and add attributes to classes, as we draw robustness diagrams.

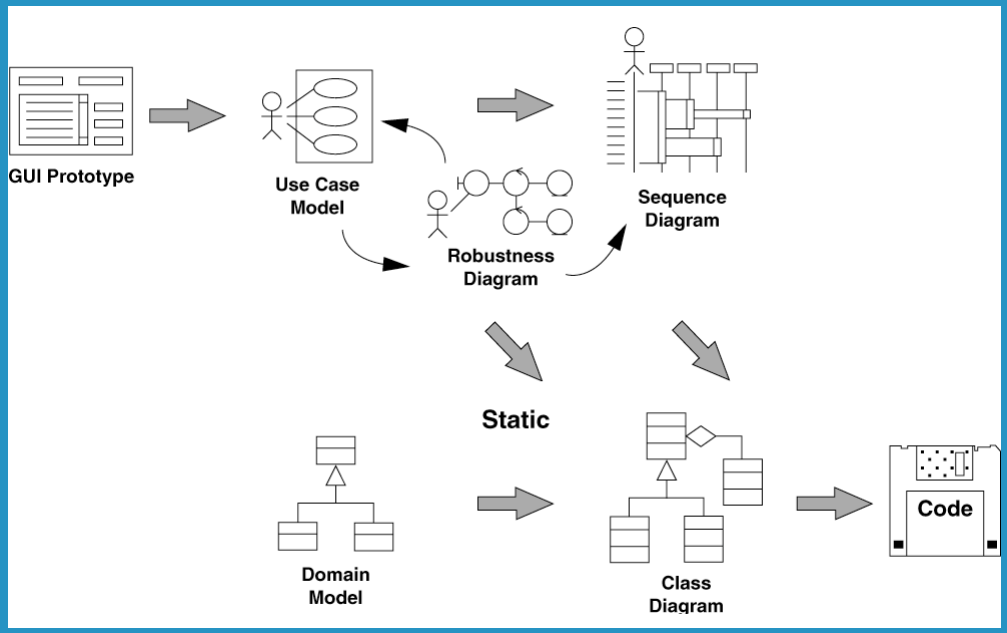
But we can't draw robustness diagrams before...

- We describe
  system usage *in the context of the object model*.
- This means that we don't write abstract, vague use cases that we can't design from.
- Instead, we need to **write use case text that references the names of objects in the problem domain.**
- We also reference the names of "boundary objects" in the use case text.

First, though...

- We need to identify the main abstractions that are present in the problem domain.
- In other words, we need a domain model.
- We show our domain model on class diagrams.

Domain Model

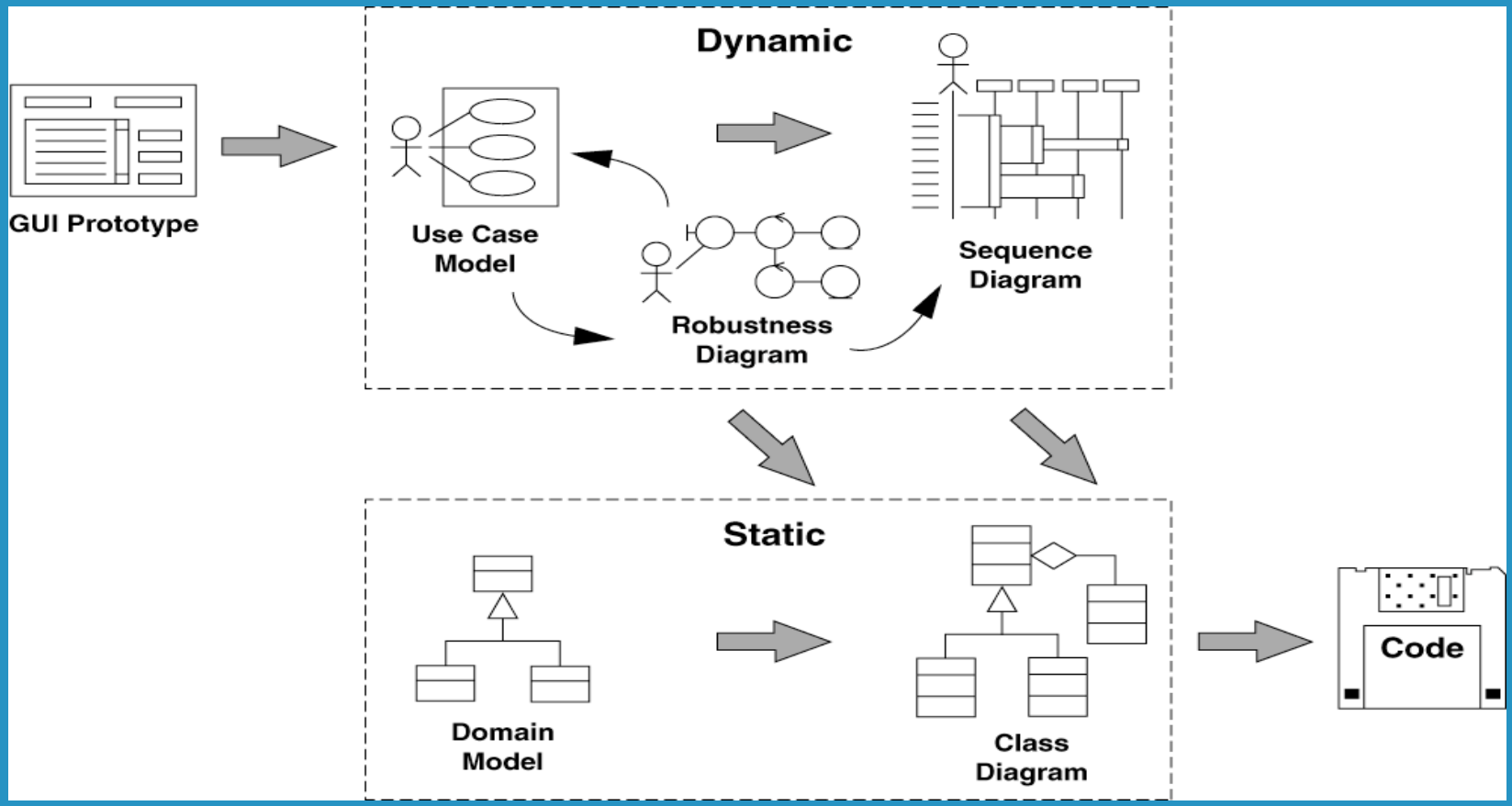


The domain model completes the picture.
Note the domain model is done before we write use case text

Refining our class diagrams

- We'll refine our (static) analysis level class diagrams (our domain model) continuously as we explore the dynamic behavior of the system in more and more detail during analysis and design.
- This will ultimately result in our design-level class diagrams, which we can code from.

The ICONIX Process



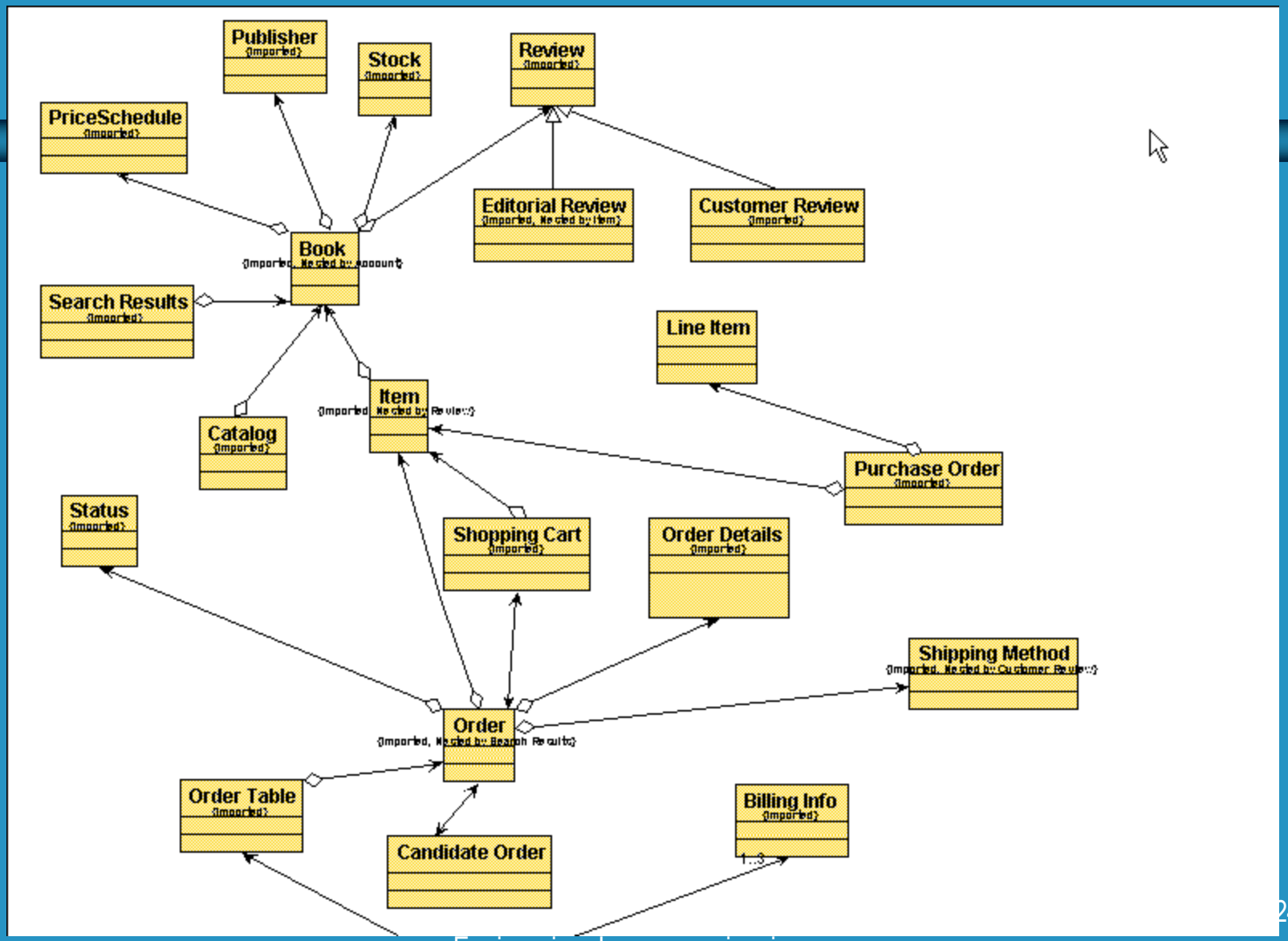
The Internet Bookstore Example

- Domain Model
- Use Case Model
- 2 use cases: Login, Edit Shopping Cart
- Robustness and Sequence Diagrams for each use case
- Show common errors (Wrong way / Right way)

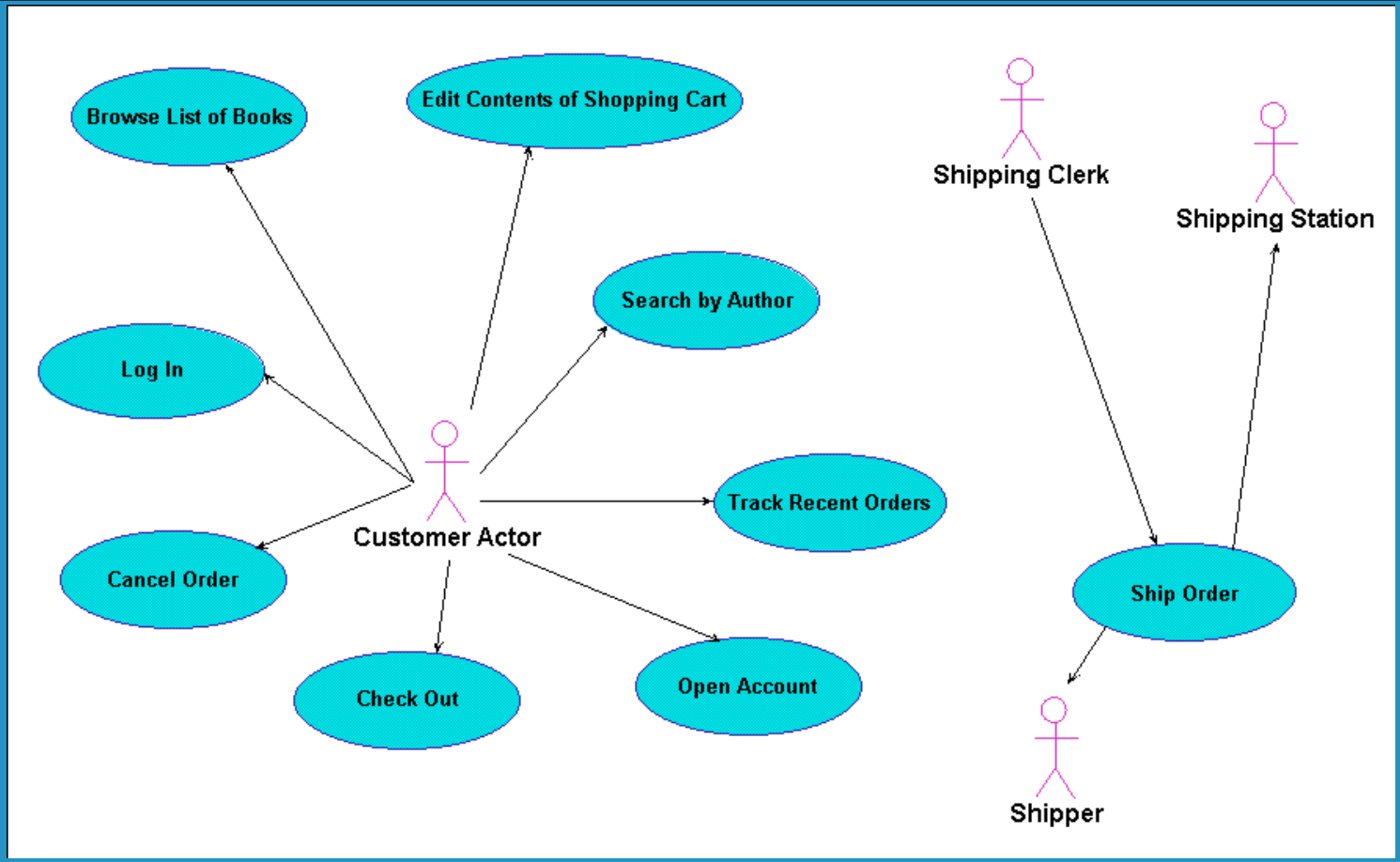
Requirements

- The bookstore shall accept orders over the Internet.
- The bookstore shall maintain a list of accounts for up to 1,000,000 customers.
- The bookstore shall provide password protection for all accounts.
- The bookstore shall provide the ability to search the master book catalog.
- The bookstore shall provide a number of search methods on that catalog, including search by author, search by title, search by ISBN number, and search by keyword.
- The bookstore shall provide a secure means of allowing customers to pay by credit card.
- The bookstore shall provide a secure means of allowing customers to pay via purchase order.
- The bookstore shall provide a special kind of account that is preauthorized to pay via purchase order.
- The bookstore shall provide electronic links between the Web and database and the shipping fulfillment system.
- The bookstore shall provide electronic links between the Web and database and the inventory management system.
- The bookstore shall maintain reviews of books, and allow anyone to upload review comments.
- The bookstore shall maintain ratings on books, based on customer inputs.

Domain Model



Use Case Model



Login: bad use case text

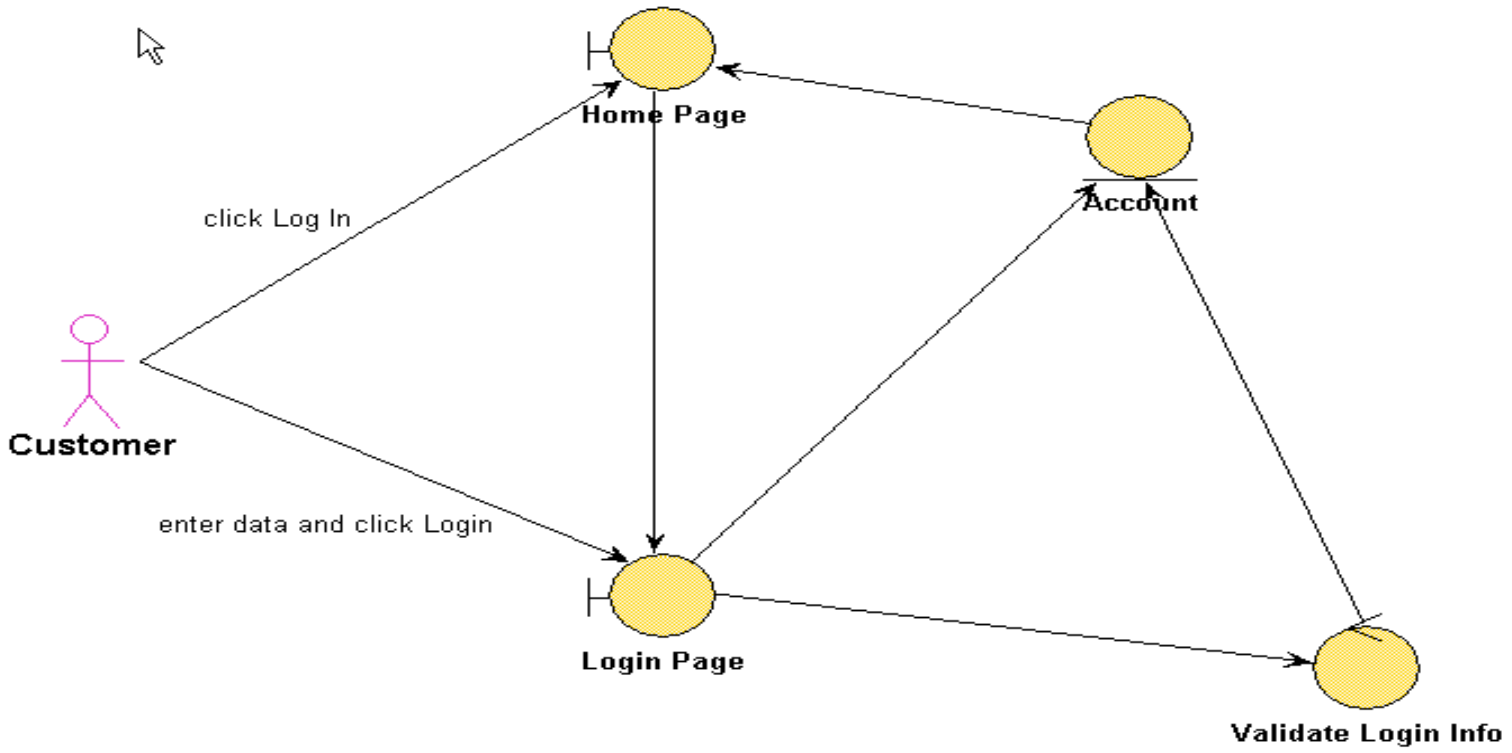
- (Exercise 1) The Customer enters his or her user ID and password, and then clicks the Log In button. The system returns the Customer to the Home Page.
- (Exercise 2) Name: Log In
- Goal: To log a customer into the system.
- Precondition: The Customer is not already logged into the system.
- Basic Course: The Customer enters his or her user ID and password, and then clicks the Log In button....
- Alternate Courses: ...
- Postcondition: The Customer is logged into the system.

Login: good use case text

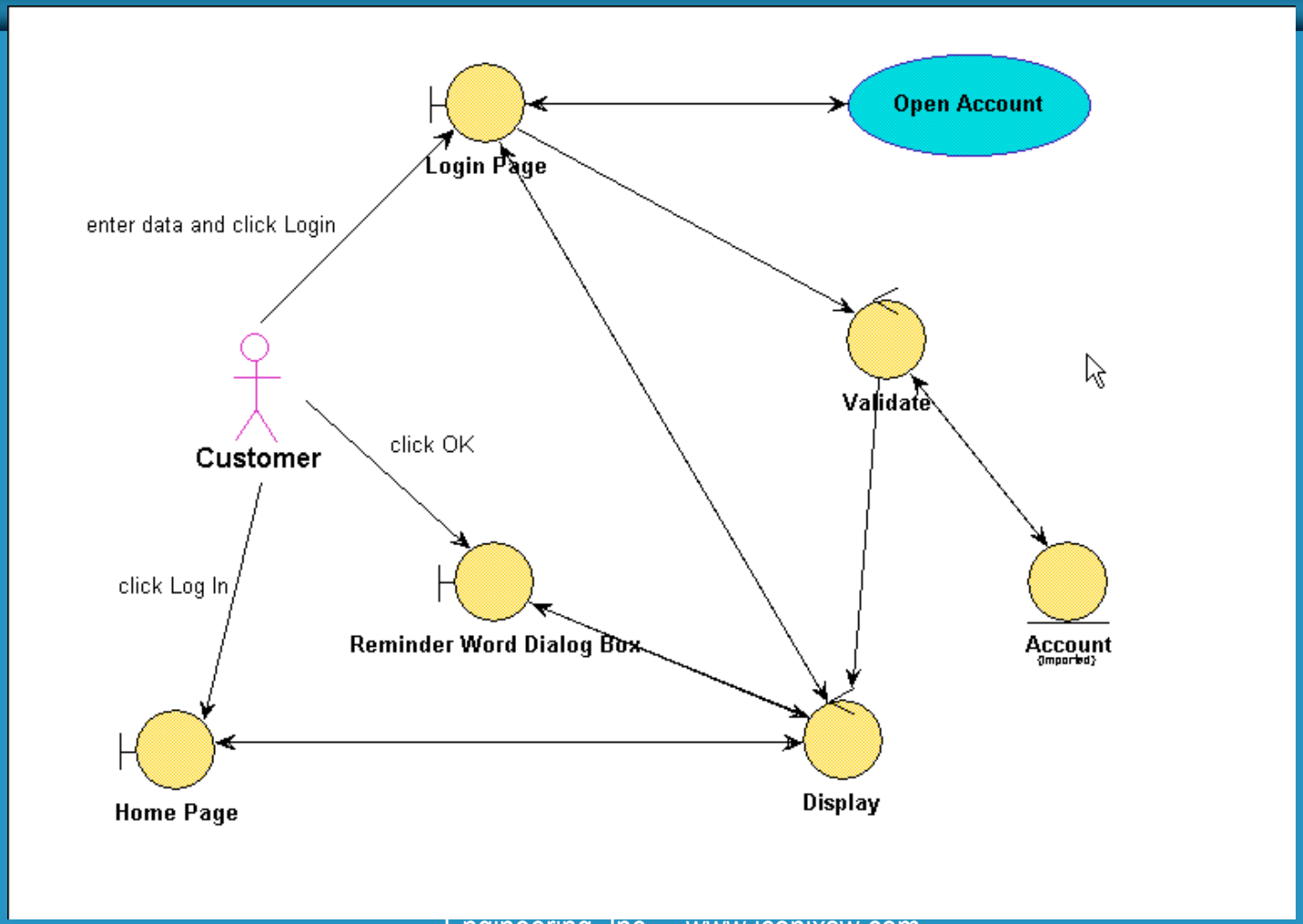
- (Exercise 1) Basic Course: The Customer enters his or her user ID and password, and then clicks the Log In button. The system validates the login information against the persistent Account data, and then returns the Customer to the Home Page.

- (Exercise 2) Basic Course: The Customer enters his or her user ID and password, and then clicks the Log In button....

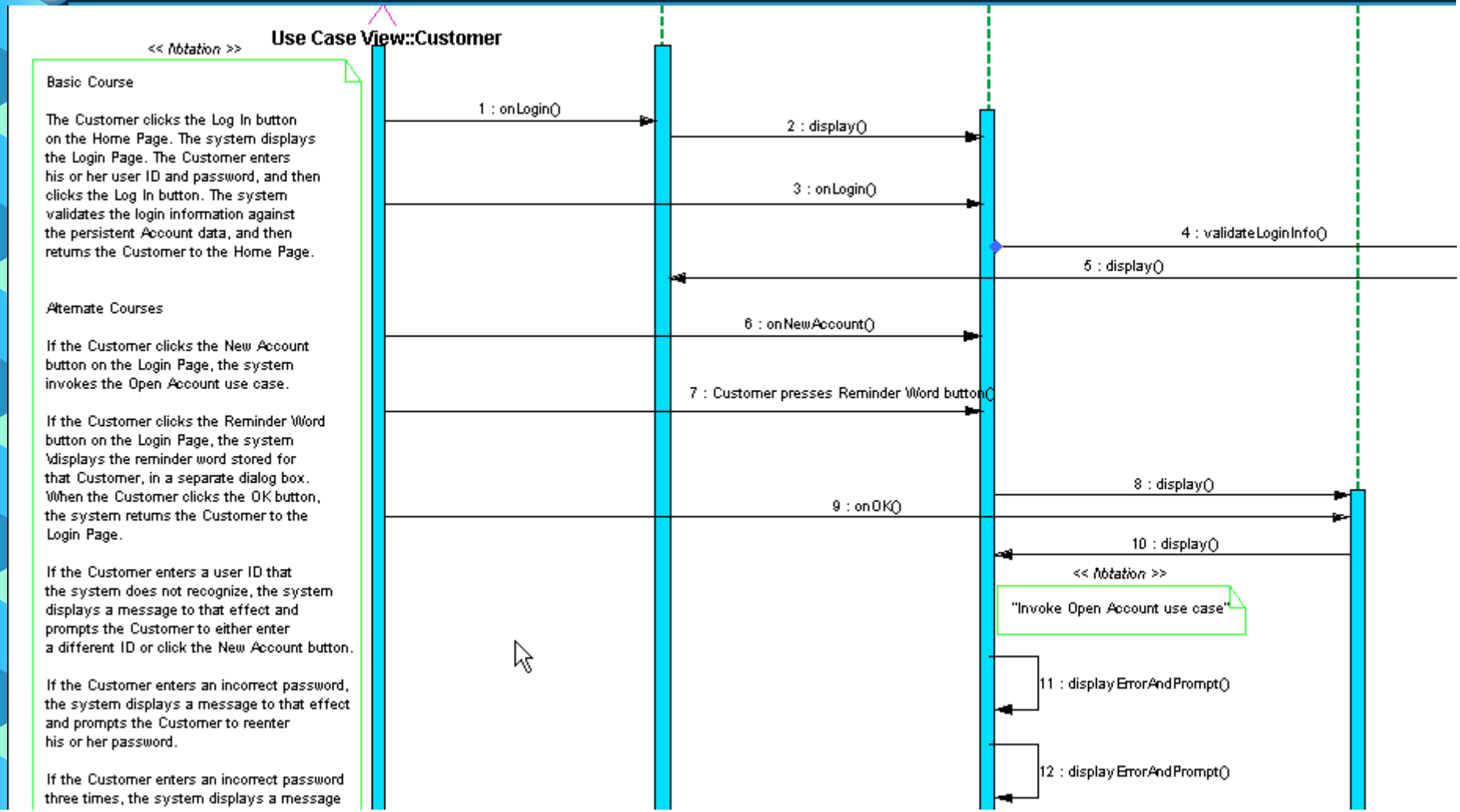
Login: bad robustness diagram



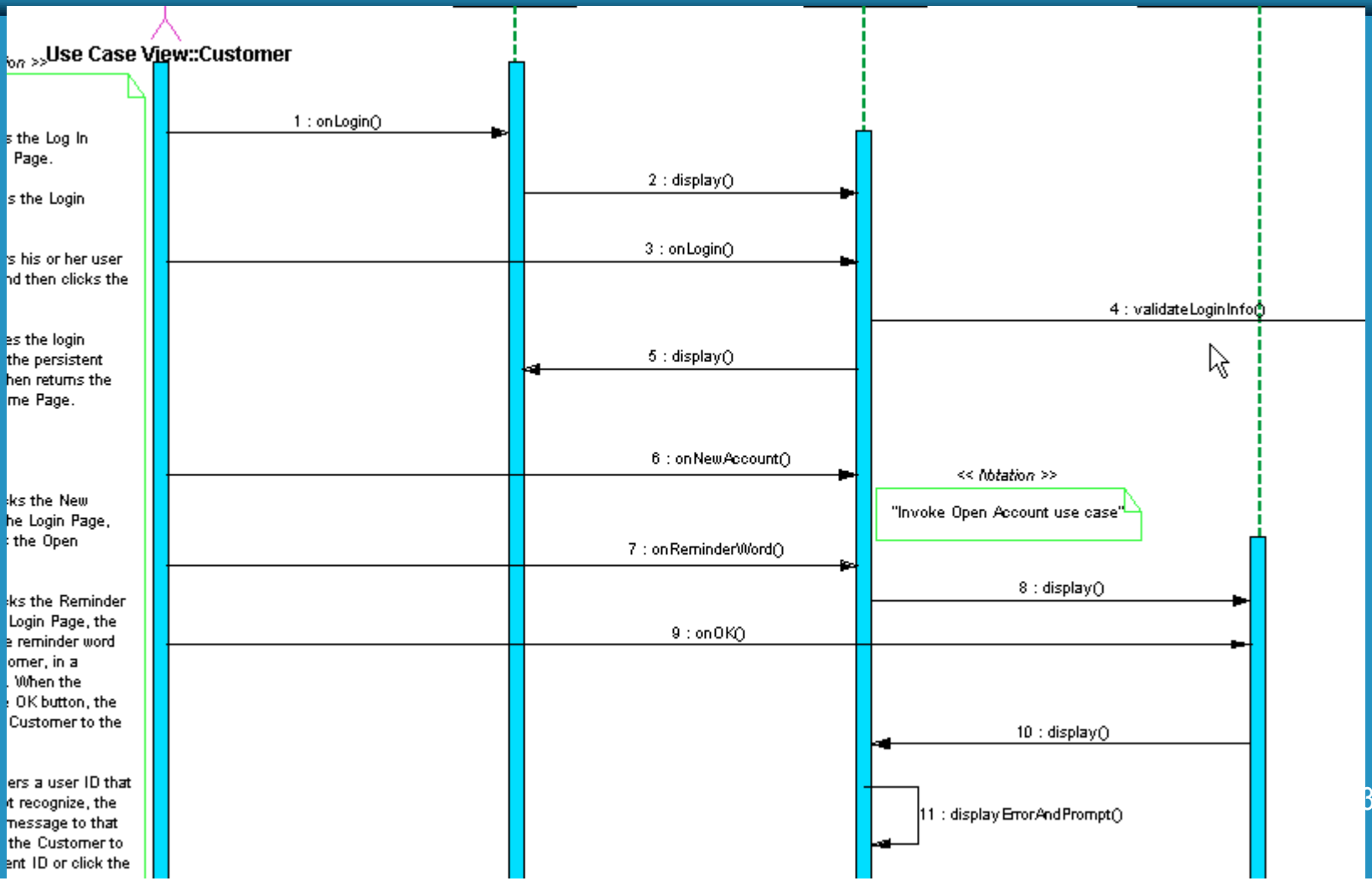
Login: good robustness diagram



Login: bad sequence diagram

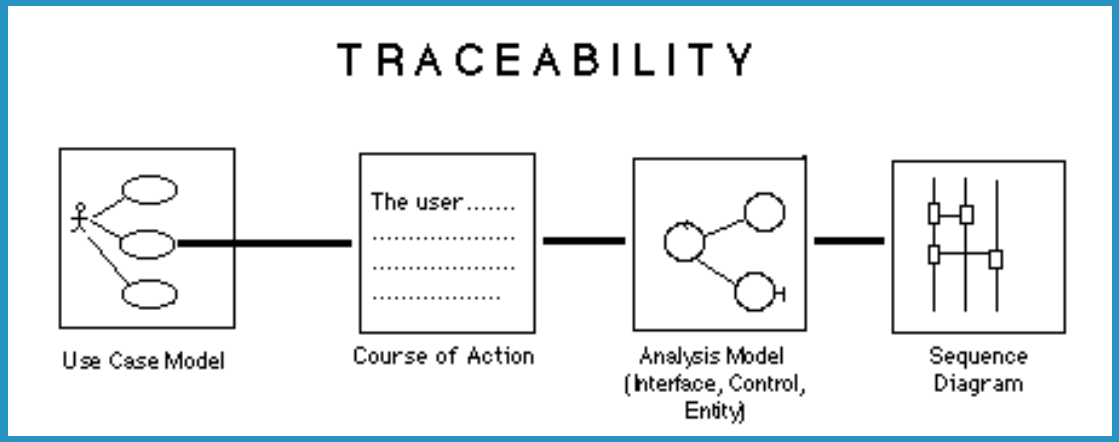


Login: good sequence diagram



High degree of traceability

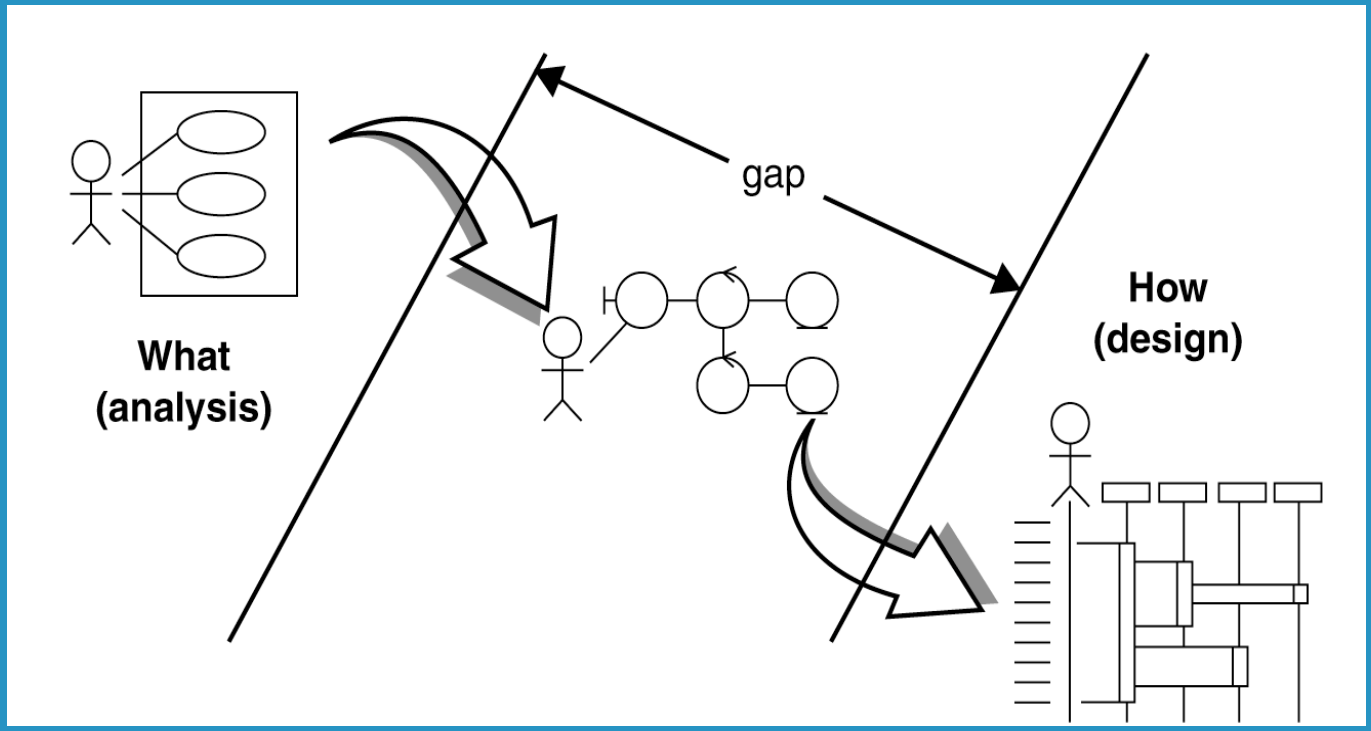
- Courses of action describe what goes on in a use case (normally and in exceptional cases)
- Robustness diagrams bridge the "what/how" gap
- Sequence diagrams are done for each use case



Edit Shopping Cart use case text

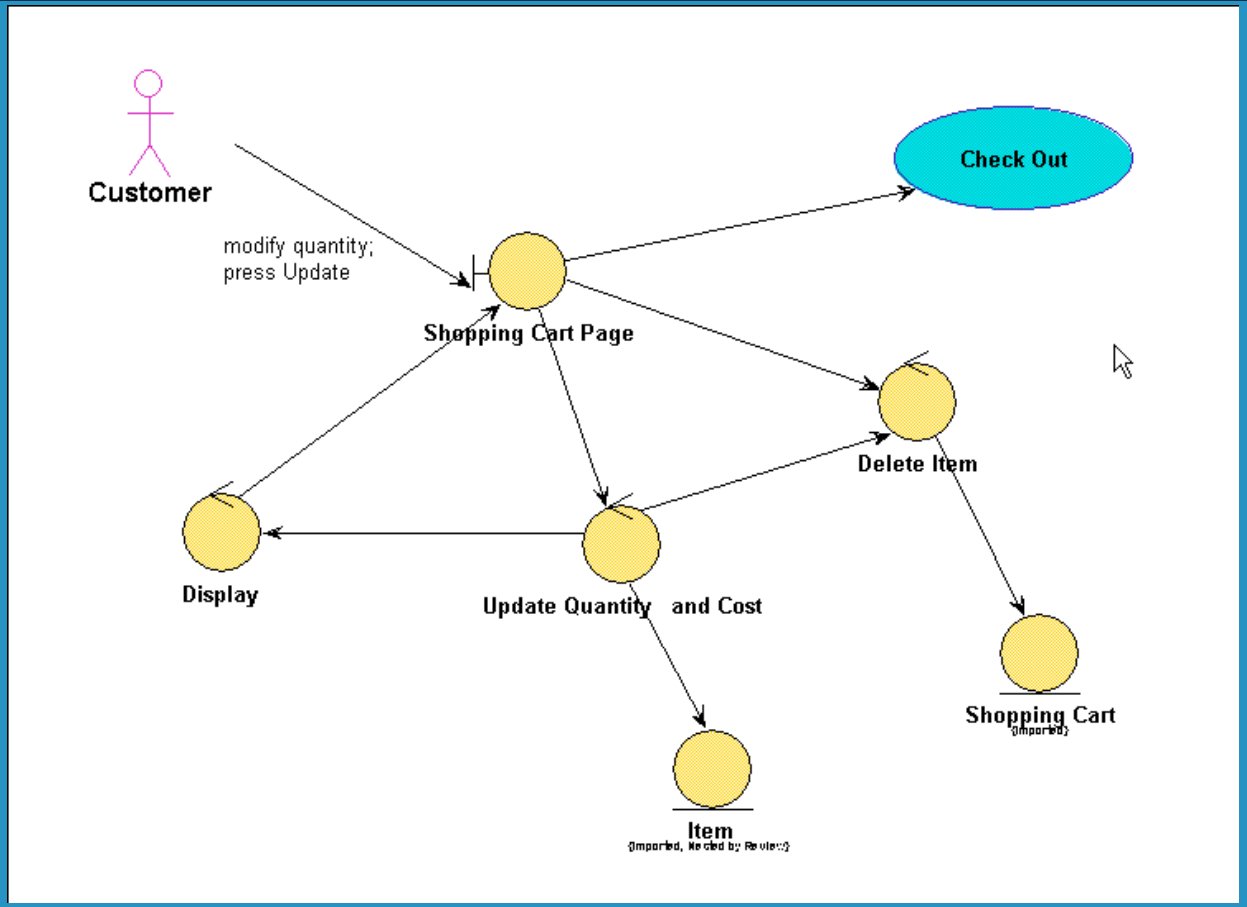
- **Basic Course: On the Shopping Cart Page, the Customer modifies the quantity of an Item in the Shopping Cart, and then presses the Update button. The system stores the new quantity, and then computes and displays the new cost for that Item….**
- **Alternate Course: If the Customer changes the quantity of the Item to 0, the system deletes that Item from the Shopping Cart.**

Robustness diagrams bridge the "what/how" gap

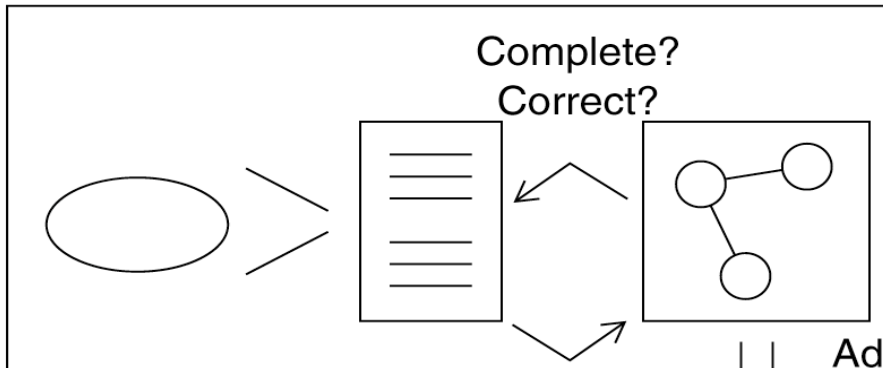


- Most current UML texts do not address crossing this what/how gap.

Edit shopping cart robustness diagram



A closer look inside Robustness Analysis

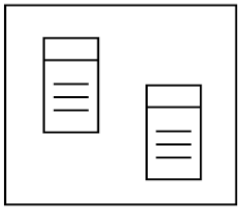


Complete?
Correct?

All alternate courses covered?
All functions identified?
Where does the data come from?

Object discovery is driven by analyzing the use cases.

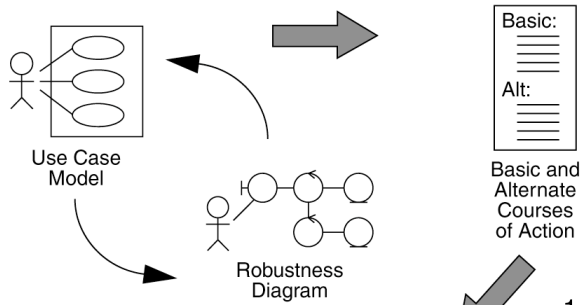
Add new classes as needed.
Assign attributes to classes.



The domain model evolves into a detailed static model.
This evolution is driven by working through the use cases.

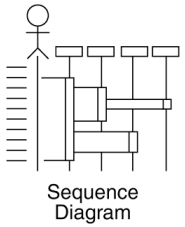
Use the robustness diagram to get the sequence diagram started

Use case text is refined during robustness analysis and reviewed during the preliminary design review.



1. Copy the use case text to the left margin of the sequence diagram.

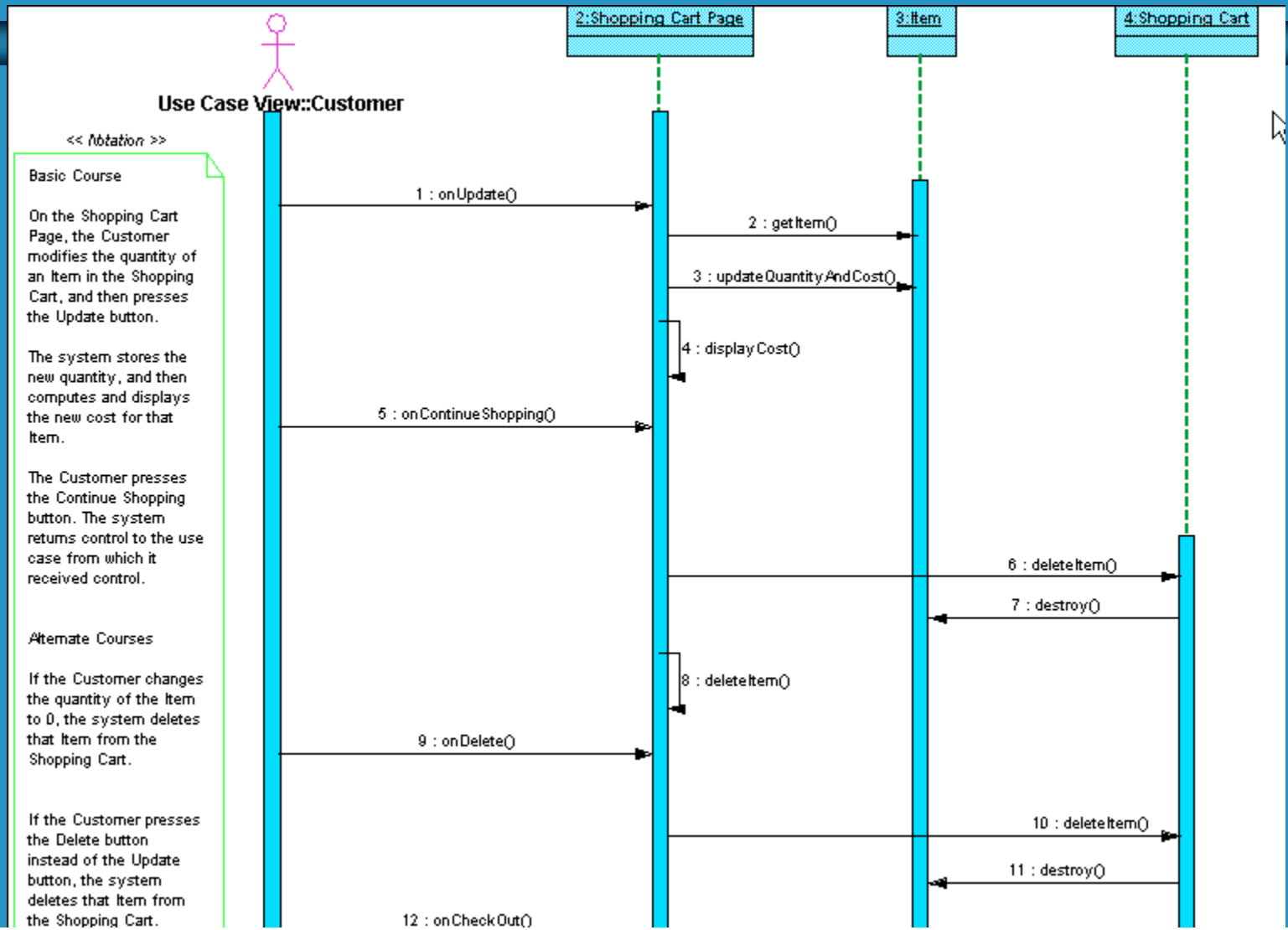
2. Add the entity objects.
3. Add the boundary objects.



4. Work through the controllers, one at a time, and figure out how to allocate the behavior among the collaborating objects.

The user requirements are always visible as we work through the design of the system.

Edit shopping cart sequence diagram



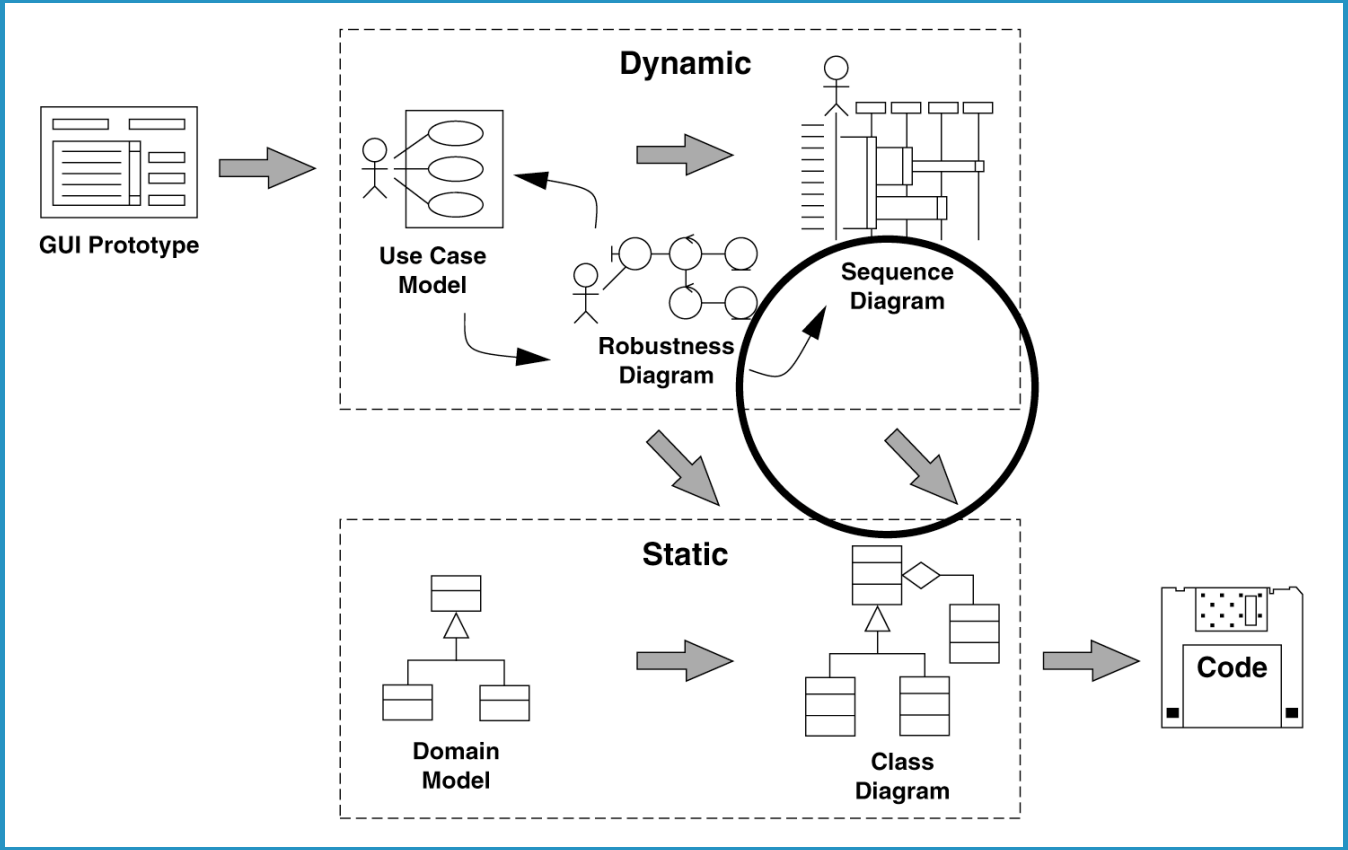
Use the Sequence Diagram to Allocate Behavior

- ***Which class does an operation belong in?***

Halbert and O'Brien criteria:

- **Reusability: does it make this class more general?**
- **Applicability: does it fit? Is it relevant?**
- **Complexity: is it easier to build it here or elsewhere?**
- **Implementation knowledge: does it rely on internal details?**

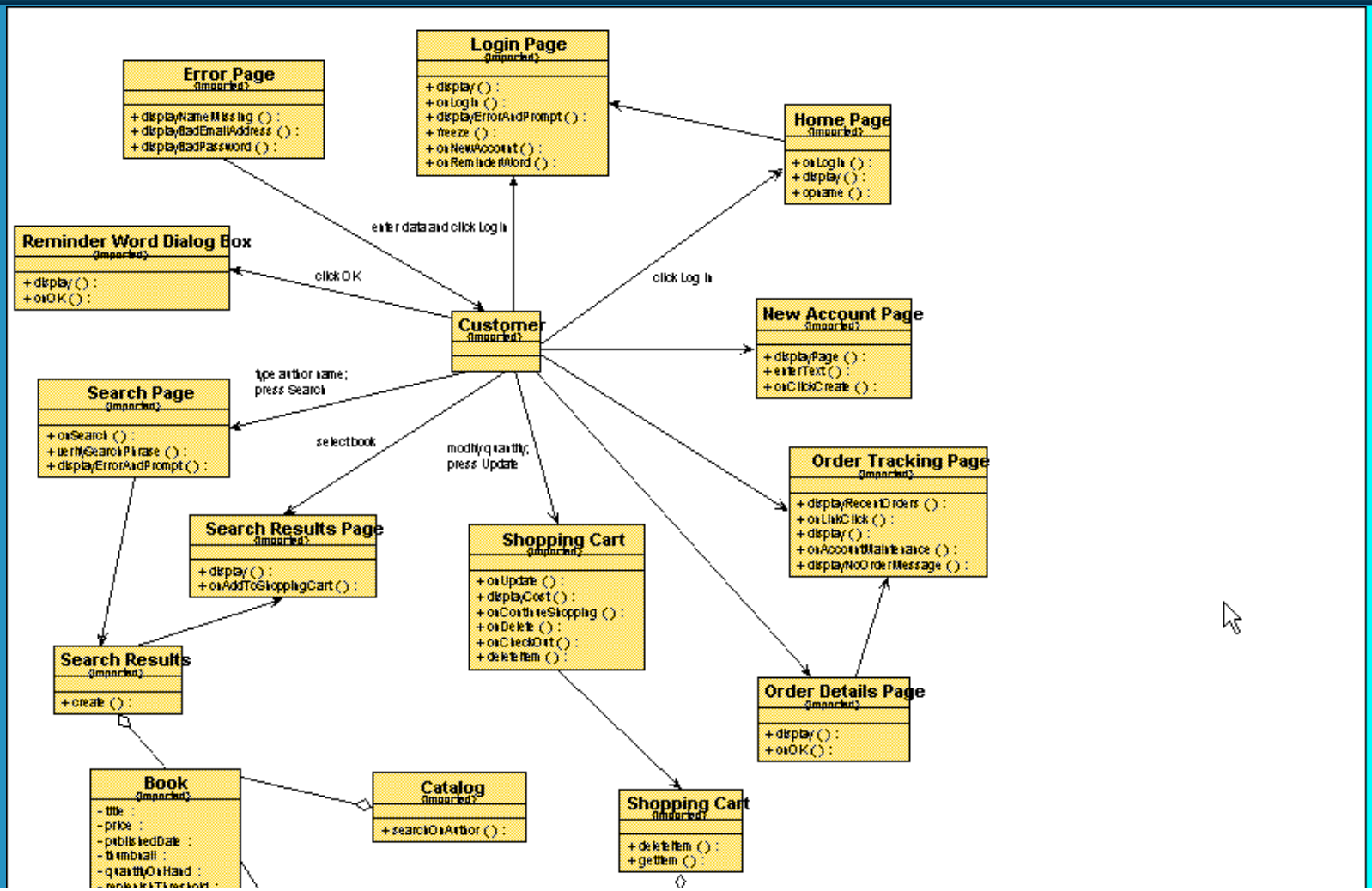
Update your static model, again



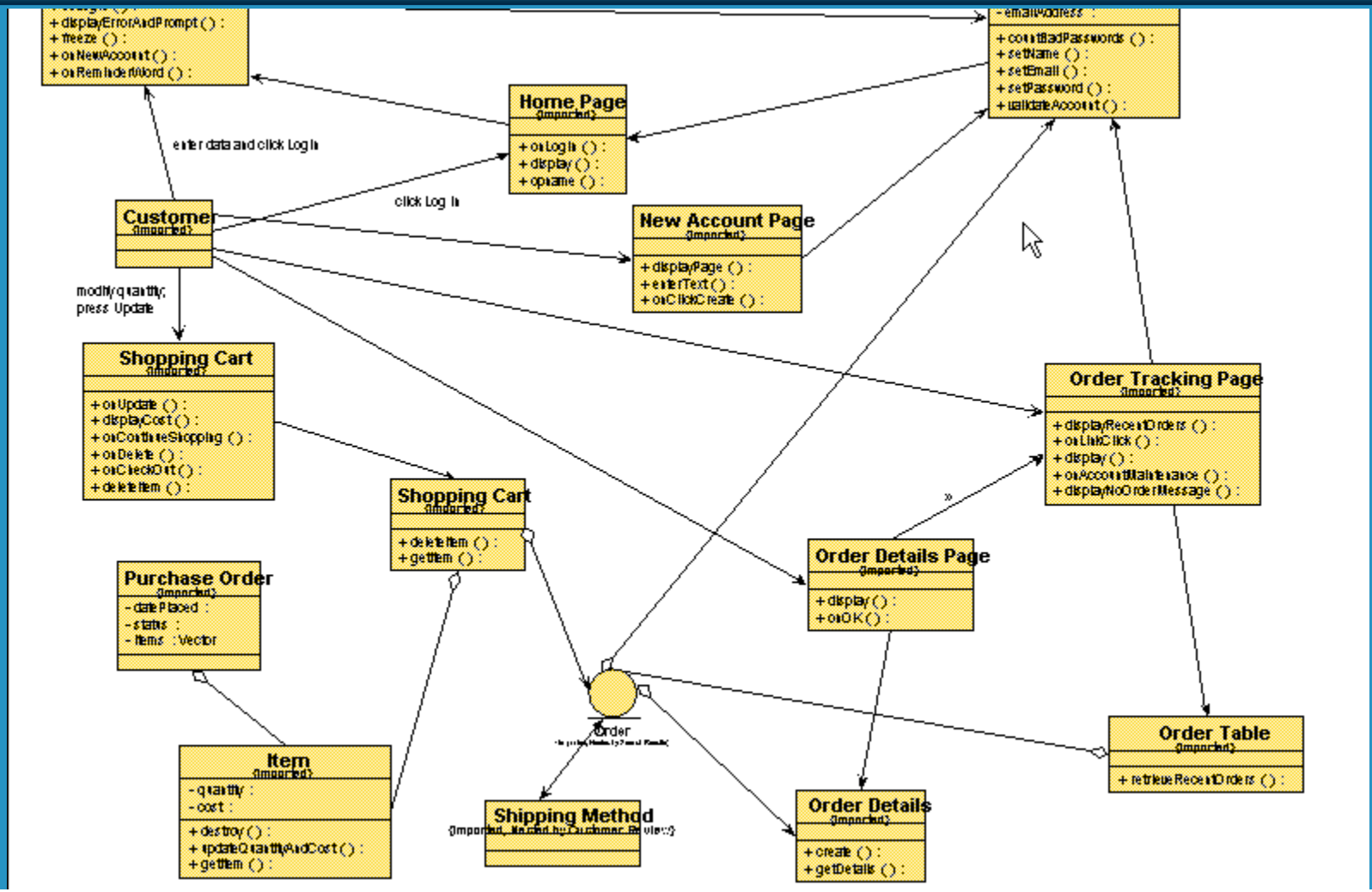
What is a "quality" class?

- Coupling: should be loosely coupled with other classes
- Cohesion: should be highly cohesive
- Sufficiency: does it do enough?
- Completeness: does it cover all the relevant a abstractions?
- Primitiveness: stick to basic operations

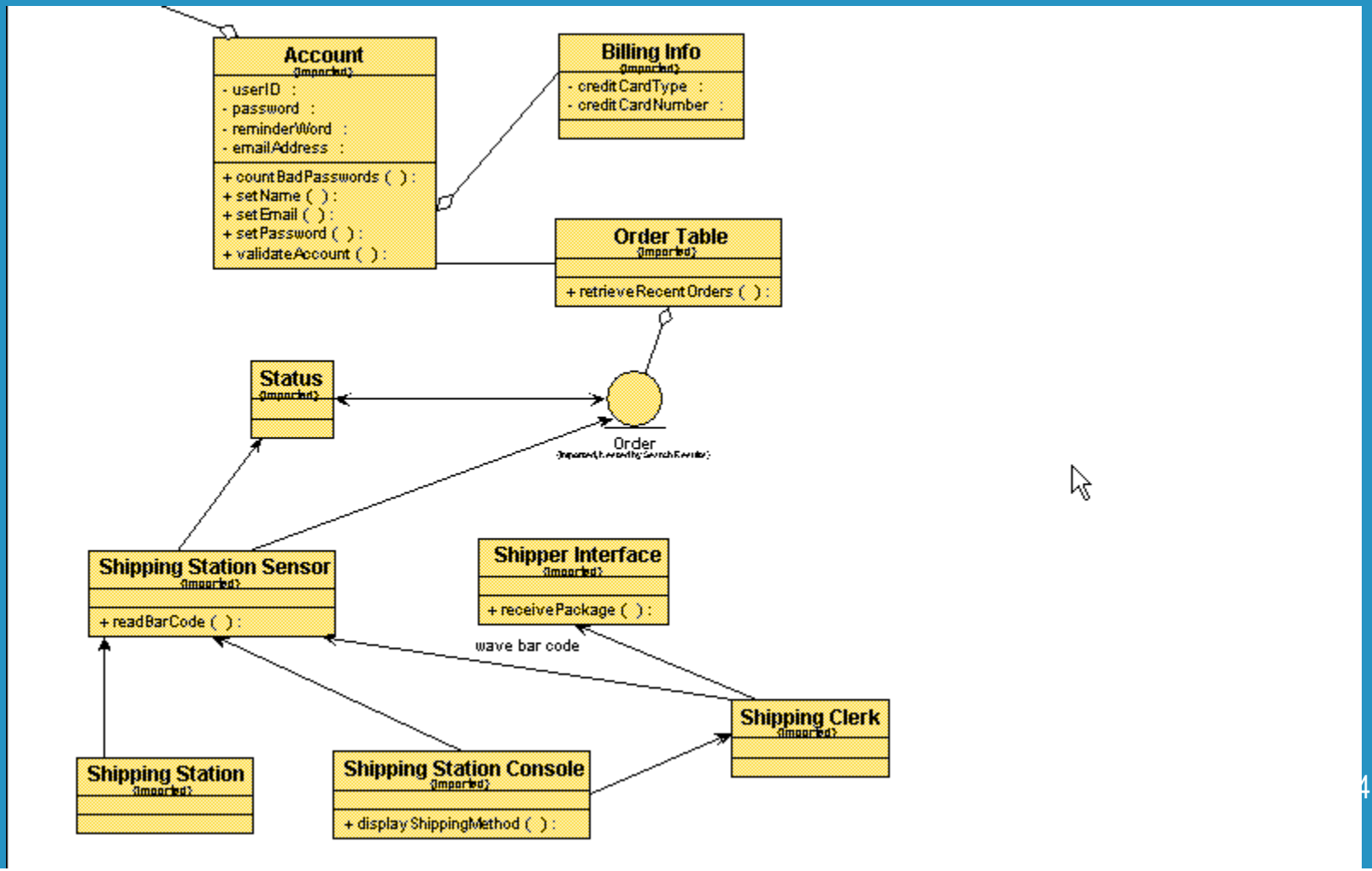
Detailed Static Model (1 of 3)



Detailed Static Model (2 of 3)



Detailed Static Model (3 of 3)



Code and Test

- Component Diagrams show packaging of classes into distributable units
- Usage scenarios (use cases) become test scenarios (test cases)
- We can link requirements, test cases and other software quality assurance (SQA) information to these models and follow them through the design.

Key features of the ICONIX Process

- Avoidance of analysis paralysis
- Streamlined usage of the UML
- Minimalist yet sufficient
- High degree of traceability
- Based on fundamental OOAD questions
- Work from the outside in
- Work from the inside out

Based on fundamental OOAD questions

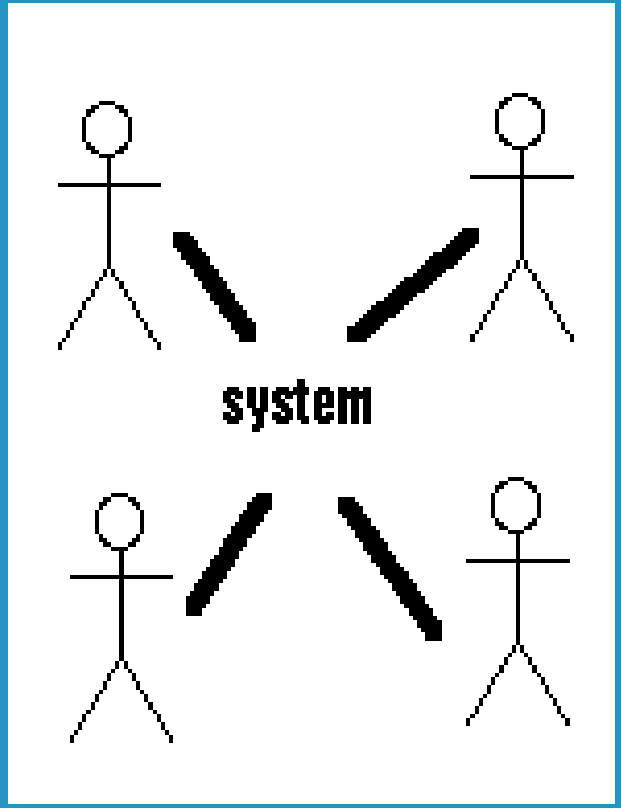
- What are the users doing? (Jacobson)
- What are the objects in the real world? (Rumbaugh)
- What objects are needed for each use case? (Jacobson)
- How do the objects collaborate with each other? (Jacobson and Booch)
- How will we implement real-time control? (state models)
- How are we really going to build this system? (Booch)

Work from the outside in

Objectory and the ICONIX Process are use-case driven (outside-in)

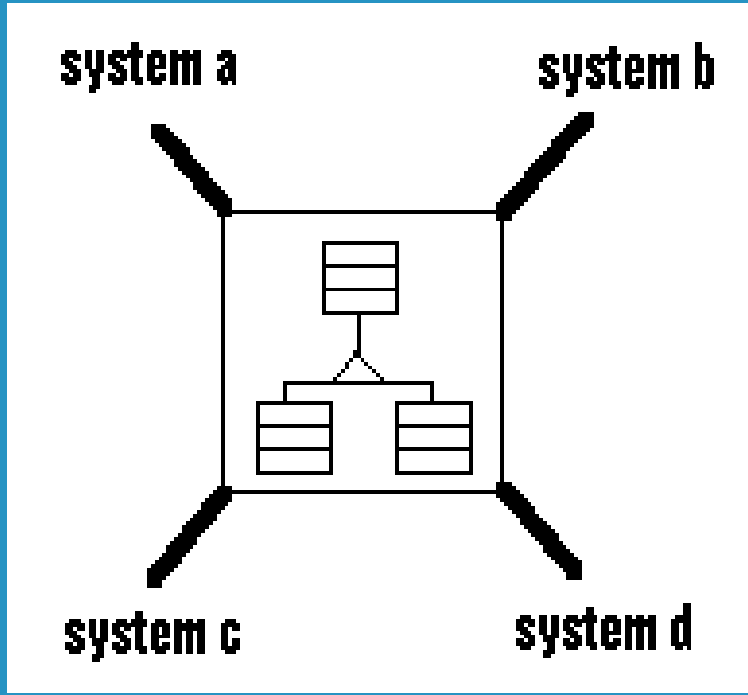
By keeping use cases as the primary unit of system decomposition, we stay user-focused

By using prototyping in conjunction with use cases, we stay user-focused



Work from the inside out

- OMT was object driven (inside-out)
- OMT models == real-world (domain)
- Some upfront thought about the problem domain makes everything easier
- Reuse across systems comes from the domain model



For further information

- EMAIL: doug@iconixsw.com
- http://www.iconixsw.com/UMLBook.html
- http://www.iconixsw.com/UMLTraining.html
- http://www.iconixsw.com/UMLecommerce.html
- http://www.iconixsw.com/UMLworkbook.html
- http://www.iconixsw.com/public_courses.htm
- Phone: 310-458-0092
- FAX: 310-396-3454
- If interested in public class in SF Dec 10, 11 contact Jon Graff
- jon@iconixsw.com 310-458-0092