Object-Oriented Analysis and Design

"It is impossible to sharpen a pencil with a blunt ax.
It is equally vain to try to do it with ten blunt axes instead."

–Edsger Djikstra

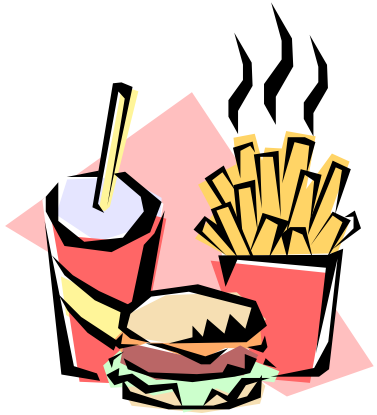
improving
It's what we do. ™

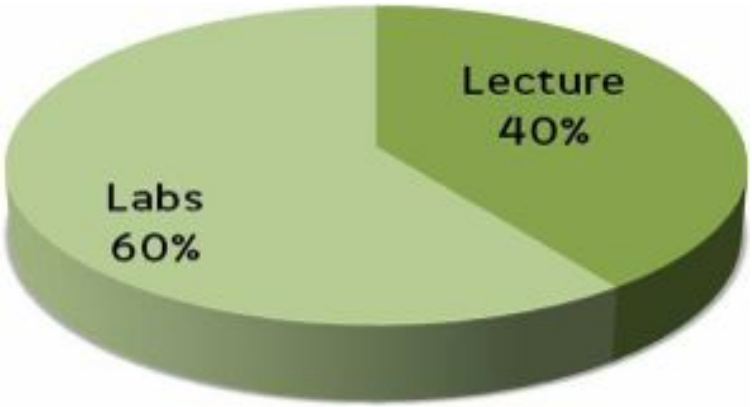
Introduction

- Hello!
- About us
- About you
  - Name
  - Related Experience
  - Goals

improving
It's what we do. ™

Logistics





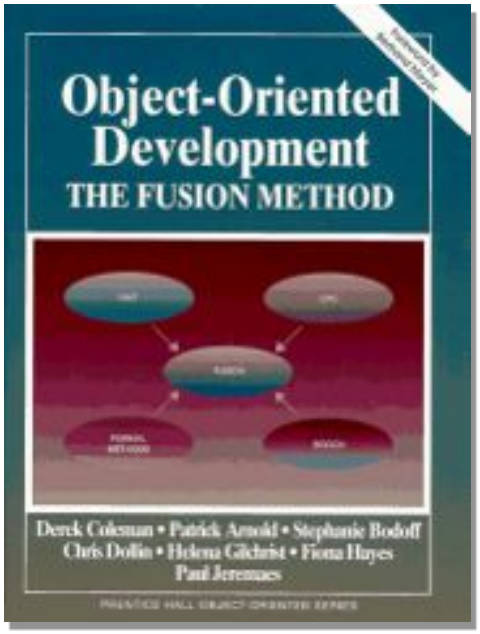
improving
It's what we do. ™

The Course

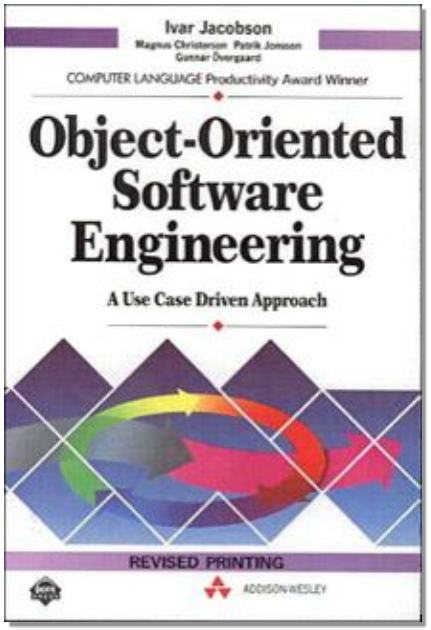
- We Cover:
  - Object-Oriented Analysis
    - Use cases
    - Domain Modeling
  - Object-Oriented Design
    - Key Principles
    - Patterns
  - UML

The Process (Highly Simplified)

- Synthesis of two development methods:



+

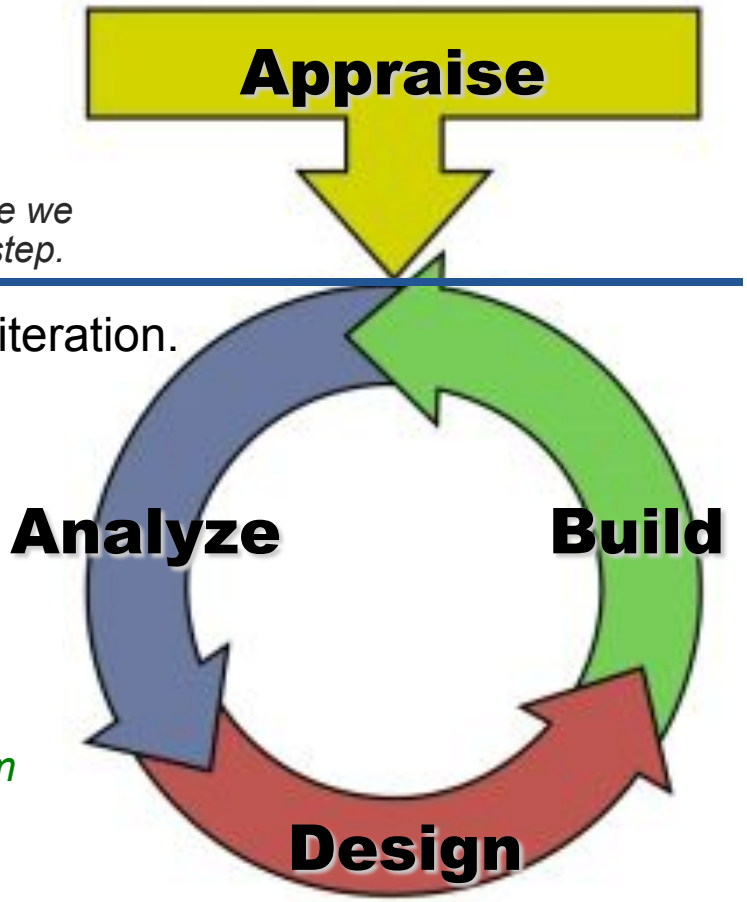


improving
It's what we do. ™

Overview

1. Do a quick appraisal to gain a basic understanding of the system to be built, focusing on external users of the system and their goals.

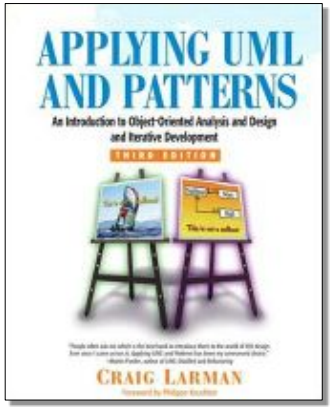
   Try to appraise all the system goals to make sure we don't forget anything important during the next step.

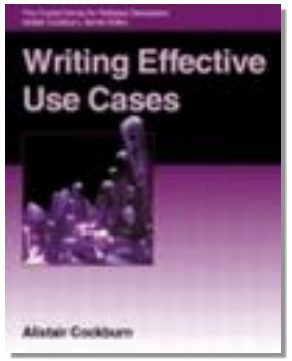


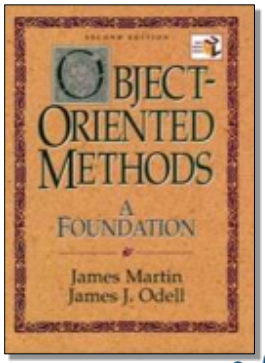
2. Develop each goal (or a piece of it) in an iteration.
   1. *Expand on a goal discovered in Step 1 to create an in depth understanding of what the system must provide, the system's responsibilities.*
   2. *For each individual system responsibility, design a solution of collaborating objects.*
   3. *For each individual design, write the tests and code that provide the expected system behavior.*

improving
It's what we do. ™

Bibliography







- *Applying UML and Patterns*, Craig Larman
  - Basic reference on object-oriented development. Adds Design Patterns to the synthesis of Fusion and OOSE.
- *Writing Effective Use Cases*, Alistair Cockburn
  - The source on use cases.
- *Object-Oriented Methods: A Foundation*, Martin & Odell
  - Excellent book on domain modeling

improving
It's what we do. ™

Course Example

- Build the front-end for a new website selling technical books,

  www.ourbooks.com
- Support customer profiles and ordering books using the shopping cart metaphor.
- Fulfillment will be created by a separate group. They will provide an API that will allow us to indicate what to ship where.
- We're competing based on pricing, so everything we do will be straight-forward. We're not building a book community, and we're not rebuilding Amazon.

What Now?

How do we turn these requirements into software?

improving
It's what we do. ™

Capturing Requirements

Functional Requirements (Purposes)

Use Cases, User Stories, IEEE System Shall's

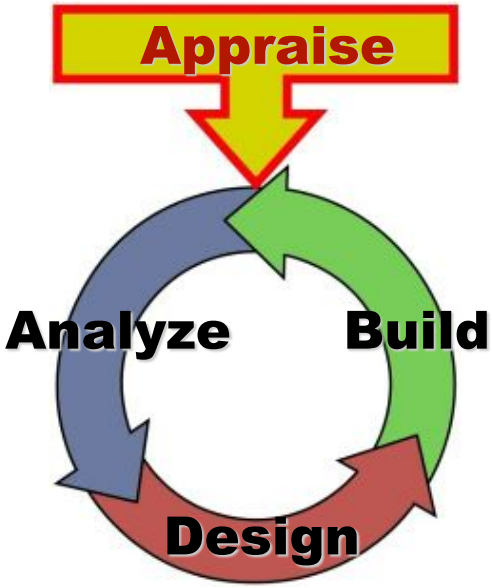
Non-Functional Requirements (Qualities)

Supplemental Specs, User Stories

Business Rules

Spread Sheets, web sites, brains, etc.

improving
It's what we do. ™



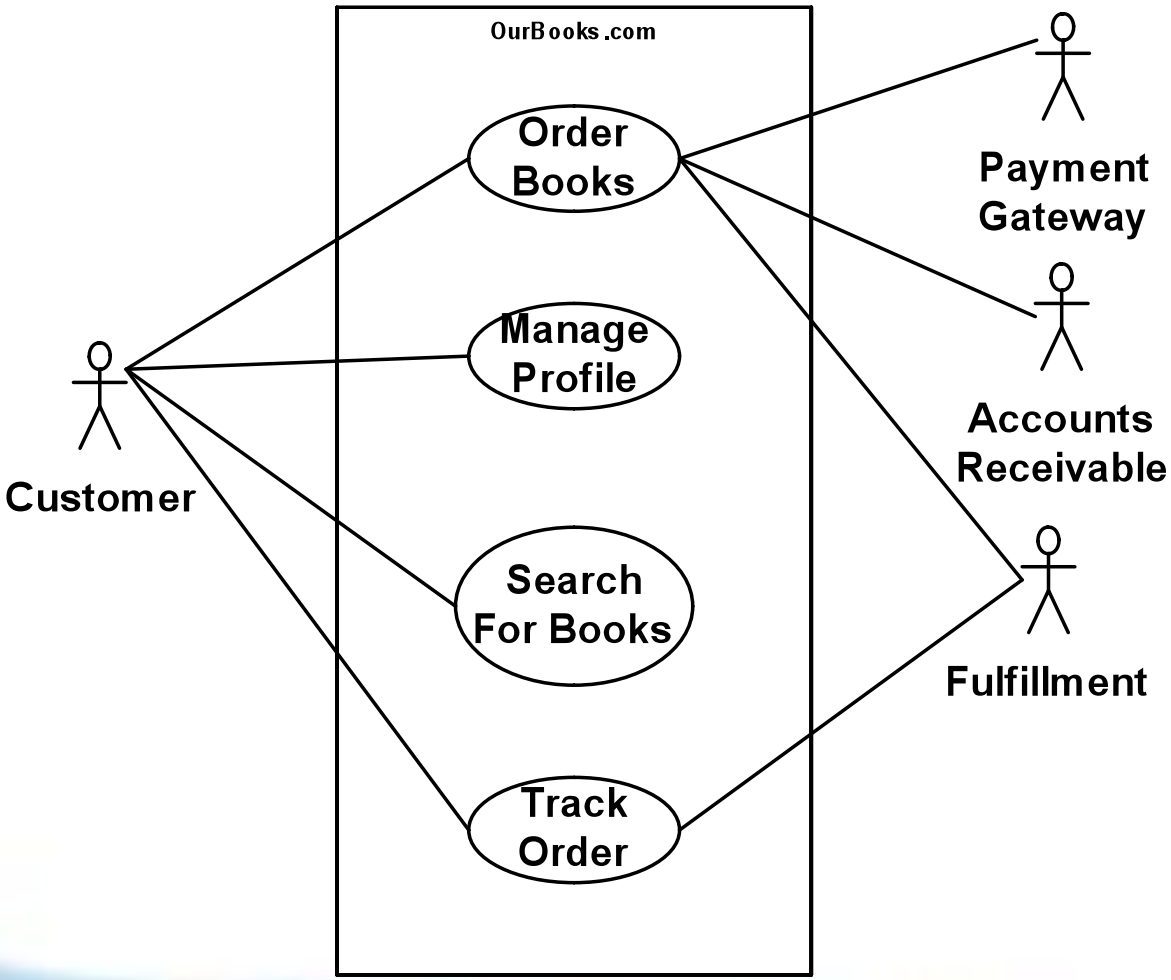
Use Case Survey: An Overview of the System

"The secret of being a bore is to tell everything."

– Voltaire

improving
It's what we do. ™

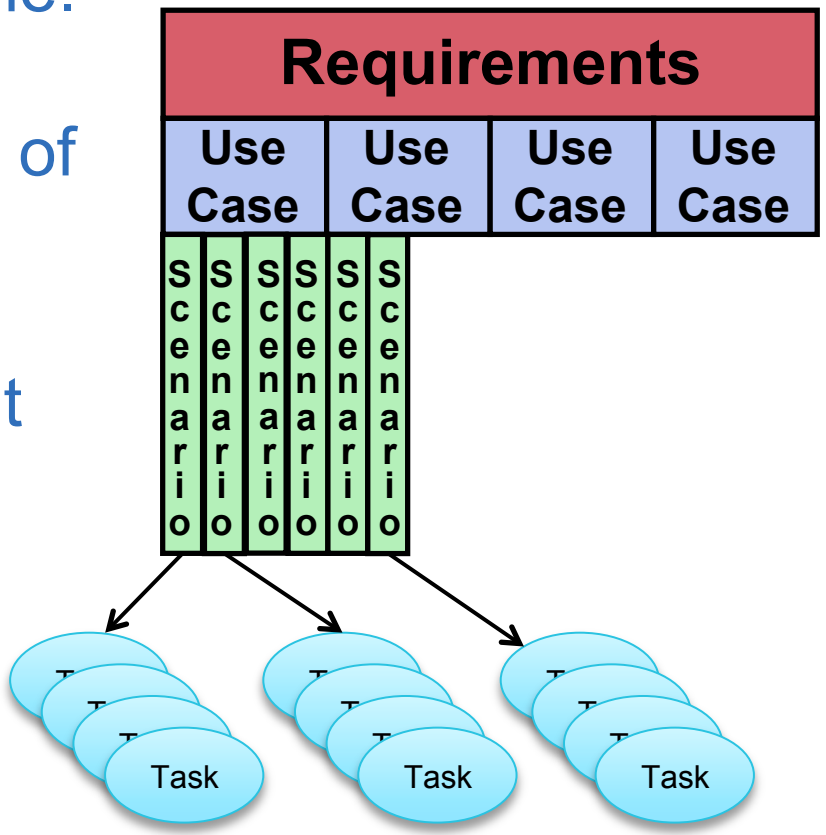
Use Case Diagram



improving
It's what we do. ™

Use Case

- A *use case* is a story of how users and the system work together to get something done.
- Breaks down the functionality of the system.
- Originally developed to exploit the essential value of story-telling to teach and learn.



improving
It's what we do. ™

Use Case: by the book

Use cases are a means for specifying required usages of a system. Typically, they are used to capture the requirements of a system, that is, what a system is supposed to do. The key concepts associated with use cases are *actors*, *use cases,* and the *subject*. The subject is the system under consideration to which the use cases apply. The users and any other systems that may interact with the subject are represented as actors. Actors always model entities that are outside the system. The required behavior of the subject is specified by one or more use cases, which are defined according to the needs of actors.

– UML Superstructure Specification

improving
It's what we do. ™

Actors

Actors are anything (other than the system) involved in the story.

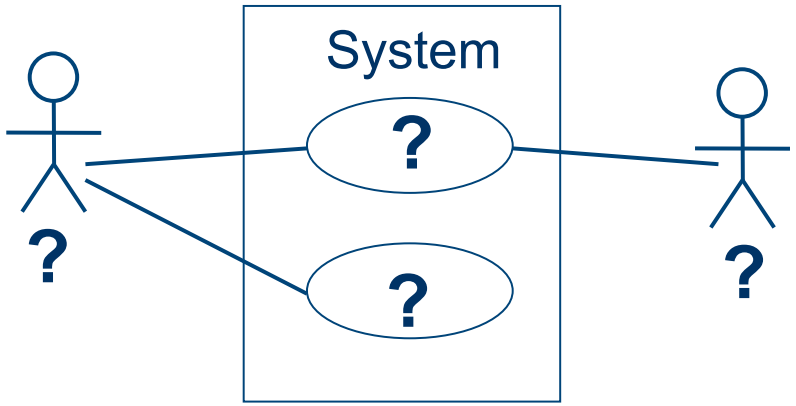
- Roles that people play (e.g., Customer)
- Other systems (e.g., Fulfilment)
- Devices outside the system (e.g., Clock)

Pictures from http://www.ilo.org/dyn/media/images/watermark/cn0568.jpg and http://www.techshout.com/images/wii-tennis.jpg

improving
It's what we do. ™

Exploring Use Cases

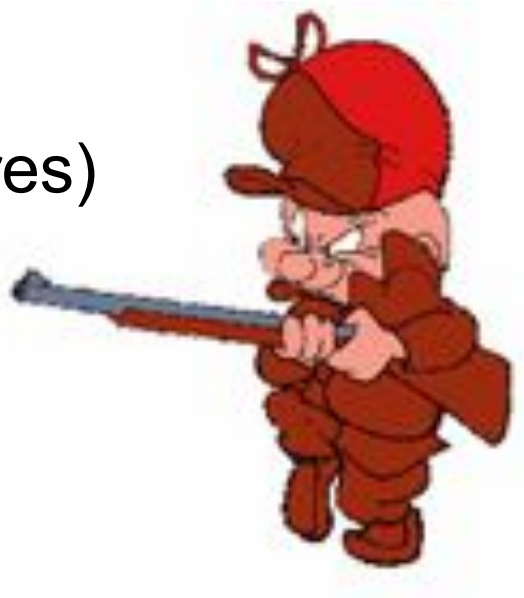
- When trying to find the use cases of a system, start by listing the actors.
  - They may not be explicitly mentioned:
    - System Administrator
    - Help Desk
- For each actor, list what they will try to accomplish using the system.
  - For the Customer:
    - Order Books
    - Manage Profile
    - Search for Books
    - Track Order
    - …



improving
It's what we do. ™

Hunting Use Cases

- Specifically requested
  - Order Books
  - Manage Profile
- Assumed (usually supporting features)
  - Track Order
  - Dispute Order
  - Cancel Order
  - Disable Customer
  - Manage Books
- Behind-the-scenes
  - Startup System, Shutdown System, Migrate System, Upgrade System, Downgrade System, Install System, Backup System, Restore System, Manage Users…

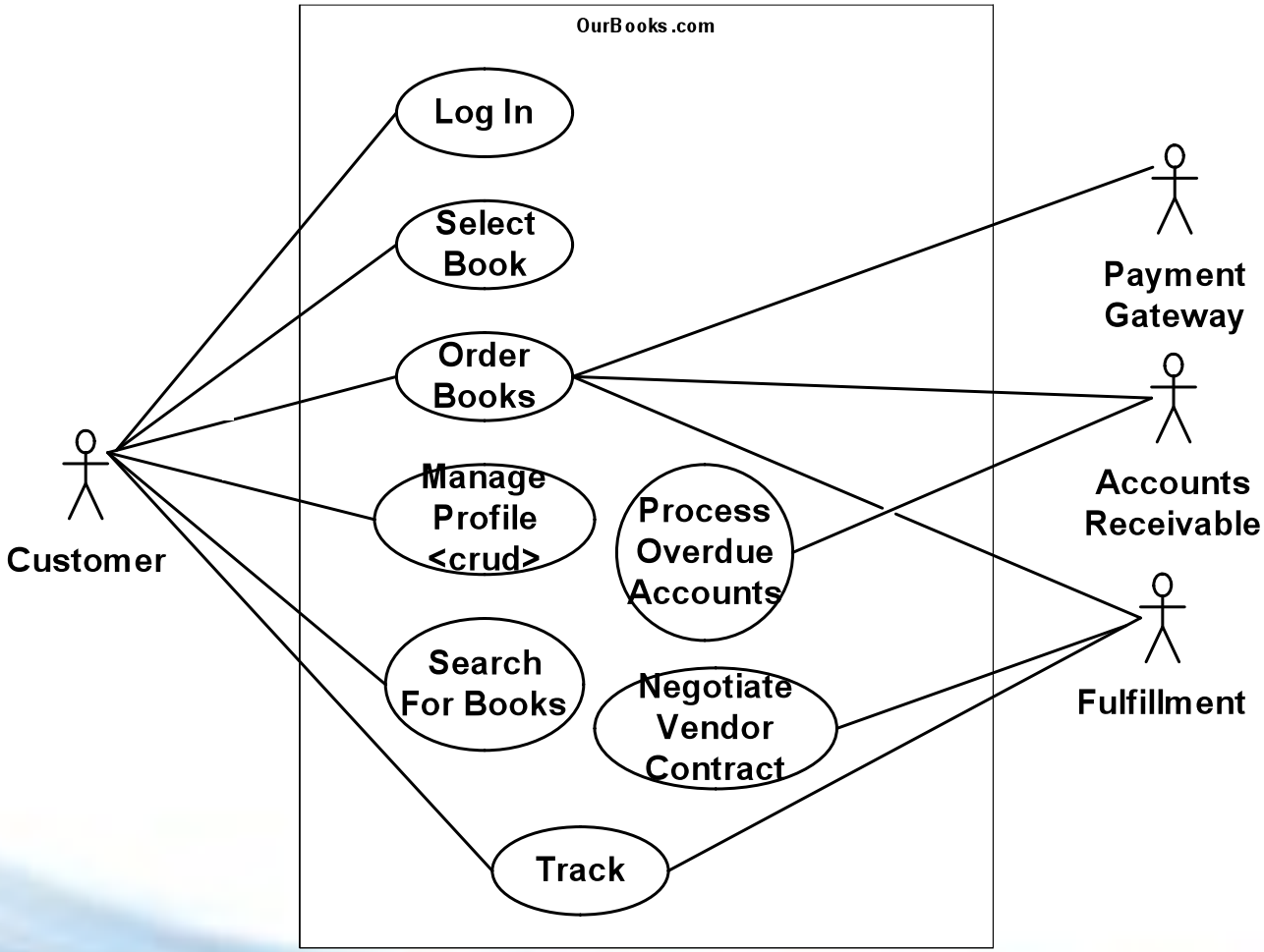


improving
It's what we do. ™

Use Case Guidelines

- User Goal Level
  - Elementary Business Process
  - Boss Test
  - Independently Satisfying!
- The title is short: <*Active Verb*> <*Noun*>
- The text is *technology free*.
- The system is a *black box*.

Exercise: Find Use Case Guideline Violations



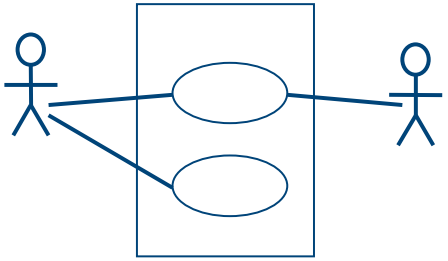
improving
It's what we do. ™

Use Case Briefs

- Use Case Briefs are short summaries of the use case.
  - 3-5 sentences
  - Describes start and end of a use case
  - < 5 minutes to create

Use Case: Order Books

The use case begins when a Customer searches for books. The customer finds books they want, adds them to their shopping cart, and purchases them. The use case ends when the order is sent to the existing Fulfilment system.



Exercise

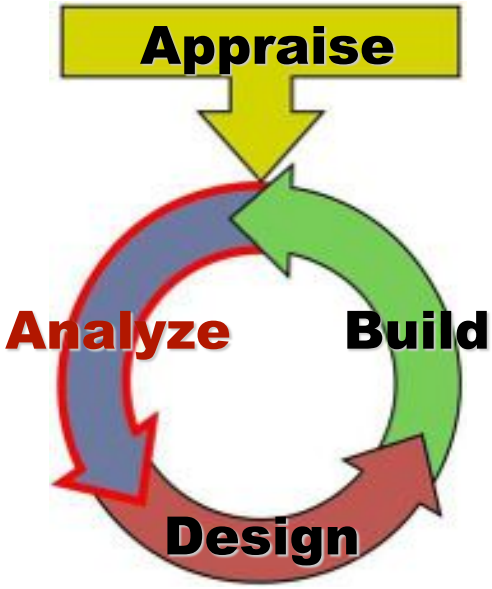
For the course project:

A. Develop a UCD following this procedure:

1. Discuss who will be affected by the development or introduction of the system.
   (people, systems, devices)
2. Select the actors (subset of the affected).
3. For each actor, list use cases they're in.
   (explicitly requested, assumed, behind-the-scenes)
4. Draw the UCD.

B. Create use case briefs for the use cases specified by the instructor.

improving
It's what we do. ™



Use Case Specification: Detailed Requirements

"Progress is substantially slower … with users who don't really know what they want; moreover, they don't realize what they want until they're into the project itself."

– J.D. Aron, 1969

improving
It's what we do. ™

Our Requirements

For the next several chapters we will focus our work on building the software for the first iteration.

Iteration 1:

- Ordering Books on-line, paying only with credit card
- Using the existing fulfillment system to ship books.
- No persistent customer profiles, etc.

Fully-Dressed Use Cases

Fully-dressed Use Cases are detailed descriptions of the use case.

- ***Title:*** <same as Use Case Brief>
- ***Description:*** <the text of the Use Case Brief>
- ***Main Success Scenario:*** detailed description of the conversation between the actors and the system under normal circumstances
- ***Extensions:*** detailed descriptions of deviations from the norm
    - » Add other sections if they're useful.

Fully-Dressed Use Cases

Examples of other sections:

- Pre-conditions
- Post-Conditions
- Stakeholders and Interests
- Non-Functional Requirements
- Q&A
- Primary Actor
- Level
- Context of Use
- …
- Trigger
- Success Guarantee
- Minimal Guarantee
- Status
- Revision
- Release
- Frequency of Occurrence
- Outstanding Issues

Example: Fully-Dressed Use Case

- Title: Order Books
- Description: The use case begins when a Customer searches for books. The customer finds books they want, adds them to their shopping cart, and purchases them. The use case ends when the order is sent to the existing Fulfillment system.

Example: Fully-Dressed Use Case

Main Success Scenario (page 1 of 2):

Actor Action	System Response
1. The use case begins when a Customer searches for books by supplying a book title.	2. Returns a list of book summaries that match the search criteria.
3. The Customer selects a book summary to see detailed information.	4. Provides detailed information about the book (number of pages, other editions/formats of the book, a link for each author to a list of his books, etc.).
5. The Customer requests the book.	6. Adds the book to the shopping cart.
	7. Returns the new total price.
The Customer may repeat Steps 1-7.	

Example: Fully-Dressed Use Case

Main Success Scenario (page 2 of 2):

Actor Action	System Response
8. The Customer asks to buy the books in his shopping cart.	9. Displays total price, including taxes.
	10. Requests Credit Card and Shipping information.
11. The Customer provides the Credit Card information and shipping address.	12. Charges the credit card for the purchase price (including taxes and S&H).
13. The Payment Gateway approves the credit card charge, providing an approval number for auditing.	14. Logs the credit card payment in the Accounts Receivable System.
	15. Submits the order to the Fulfillment system.
16. The Fulfillment system accepts the order and responds with a tracking number.	17. Provides the tracking number to the customer.

Example: Fully-Dressed Use Case

- Extensions:
- <Step>: If <Condition> then <Response>
- At any time: If the customer abandons their work, then the shopping cart should stay for 30 minutes from the time of the last customer activity before it is cleaned up.
- At any time: If SLA tracking is enabled, log all times to respond to requests, including to the web front-end, as well as the other external systems involved.
- Steps 1-11: The system crashes. The shopping cart is lost.
- Steps 1-7: If the customer requests to remove items from the shopping cart, the cart is displayed and the customer may supply a new quantity for each book, possibly 0. Update the order based on the information provided by the customer.
- Steps 12-end: The system crashed. The order is cancelled, any charge request for the order is reversed, and the request for fulfillment is cancelled.
- 2a: No books match the search criteria. Tell the customer this and request they try again.
- 4a: The customer has selected to see a book that we no longer know about. Tell the customer that a system error has occurred and log the event for later inspection.
- 5a: If the customer requests more than one copy of the book, they may provide the quantity.
- 6a: If the customer has requested > 10 books, deny the request.
- 6b: If the customer has requested > $1000 of books, deny the request.
- 6c: If the customer had requested an age-restricted book, check to see how old they are. If they don't meet the minimum age to purchase the book, deny the request.
- 11a: If the customer elects to use the billing address of the credit card as the shipping address, the billing address is copied into the shipping address.

Example: Fully-Dressed Use Case

- Extensions (cont):

- 12a: If the data provided is invalid, respond with a list of errors. Validate the credit card (see the note on credit card information validation) and the billing & shipping addresses (see the note "Shallow validation of postal addresses").
- 12b: If the Payment Gateway is unavailable, inform the customer of the difficulty and request they wait a few minutes until the situation is resolved. The system should retry automatically after waiting 3 minutes. After the third failed attempt, it should inform the customer that we're giving up and do so. On the first attempt, the system should immediately notify system administrators to resolve the problem.
- 12c: If the Payment Gateway does not respond within 2 minutes, log the event and continue to process the order as a "provisional credit" order. Immediately notify system administrators to investigate the problem and attempt to charge the card. When the charge is successfully made, tell the Fulfillment system to upgrade the order from Provisional Credit to Approved Credit so that it will get shipped.
- 14a: If the Payment Gateway refuses the charge, inform the customer and start again from Step 8. If the user fails 3 times, inform the customer and cancel the session.
- 15a: If the Fulfillment System is unavailable, follow the same process as when the Payment Gateway is unavailable.
- 15b: If the Fulfillment System does not respond within 1 minute, log the event and store the order. Inform a system administrator to investigate the problem and re-submit the orders (the Fulfillment System will recognize duplicates based on our Order Number) when the system becomes available again.
- 17a: The customer is selected to be a part of a customer satisfaction survey. Invoke *Ask Customer About Experience* use case.

Alternate Scenarios

improving
It's what we do. ™

Alternates to Fully-Dressed Use Case

- Use Case Brief.
  - Many use cases are not compelling conversations:

    Manage Profile
    Manage Users
    Manage Books
  - *Manage* may mean: CRUDQIE
    - Create, Read, Update, Delete/Disable, Query, Import, Export

      Manage Profiles <crudq>
      Manage Books <cruqie>
- Diagram.
  - Draw a picture of how various actors interact.
- Storyboards.

Other Requirements

- ***Requirements Memo***—detailed business or technical rules that are application independent.

  Credit Card numbers must be between 12 and 16 characters in length.

- ***Supplementary Specification***—cross-cutting Non-Functional Requirements or points of future variations.

  While the use cases may specify limits (such as no more than ten books per order), all such limits should be run-time configurable.

  Users should be able to find a book in 3 or fewer mouse clicks.

- ***Domain Model***—information-related requirements and constraints.

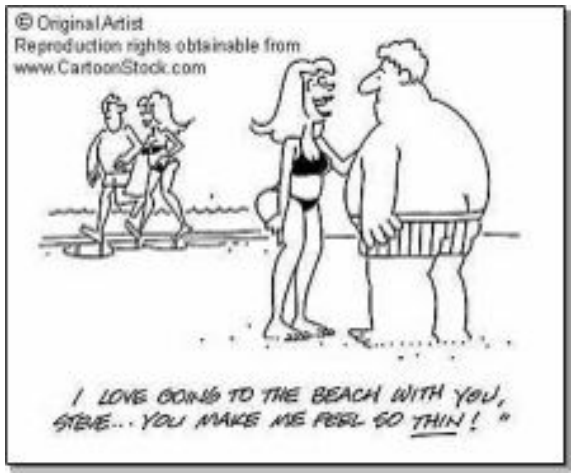
  The domain model shows that all Books have an ISBN, that all Book Summaries have only title, and that each Book has exactly one BookSummary (that is, a BookSummary is not shared across editions).

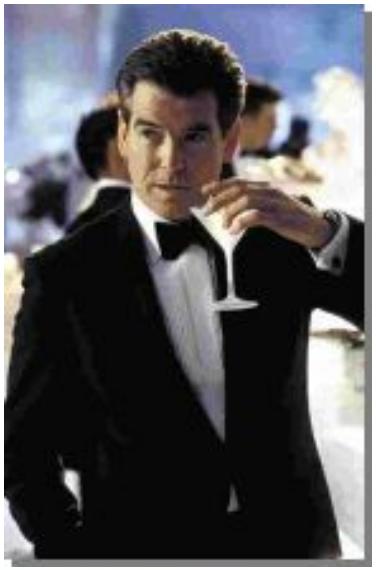
Non-Functional Categories

- Usability
- Scalability
- Portability
- Maintainability
- Availability
- Accessibility
- Supportability
- Security
- Performance
- Cost
- Legal
- Cultural
- …

Exercise: Fully-Dressed Use Case

Create the Fully-Dressed Use Cases for the use cases specified by the instructor.





improving
It's what we do. ™

So how far do we go?

What is your goal?

To Document?

- IEEE 830
- Detailed Use cases

To Represent?

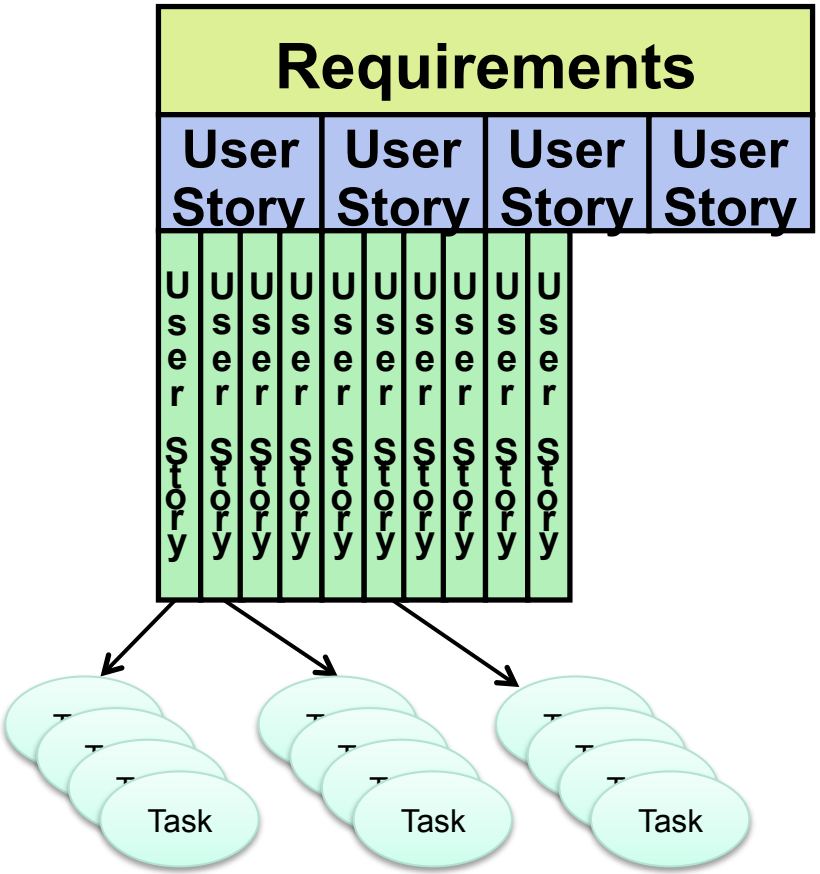
- User Stories
- Use Case Briefs

improving
It's what we do. ™

User Story

Three C's:

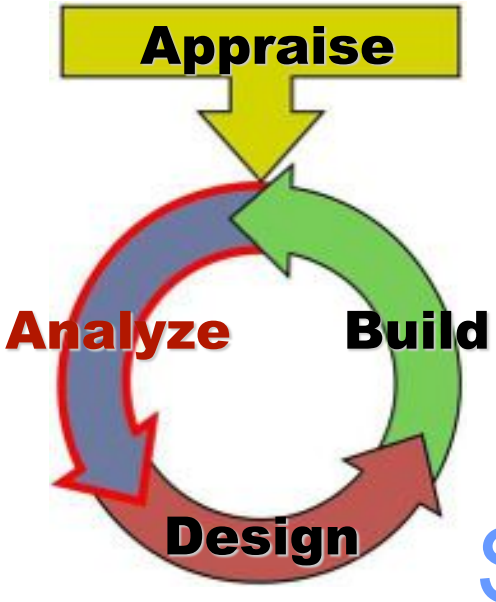
- Card
- Conversation
- Confirmation



improving
It's what we do. ™

User Stories

As a <user>, I want to
<do something>
so that I can
<accomplish some goal>.



System Sequence Diagram: Showing System Responsibilities

"It's a great invention but who would want to use it anyway?"

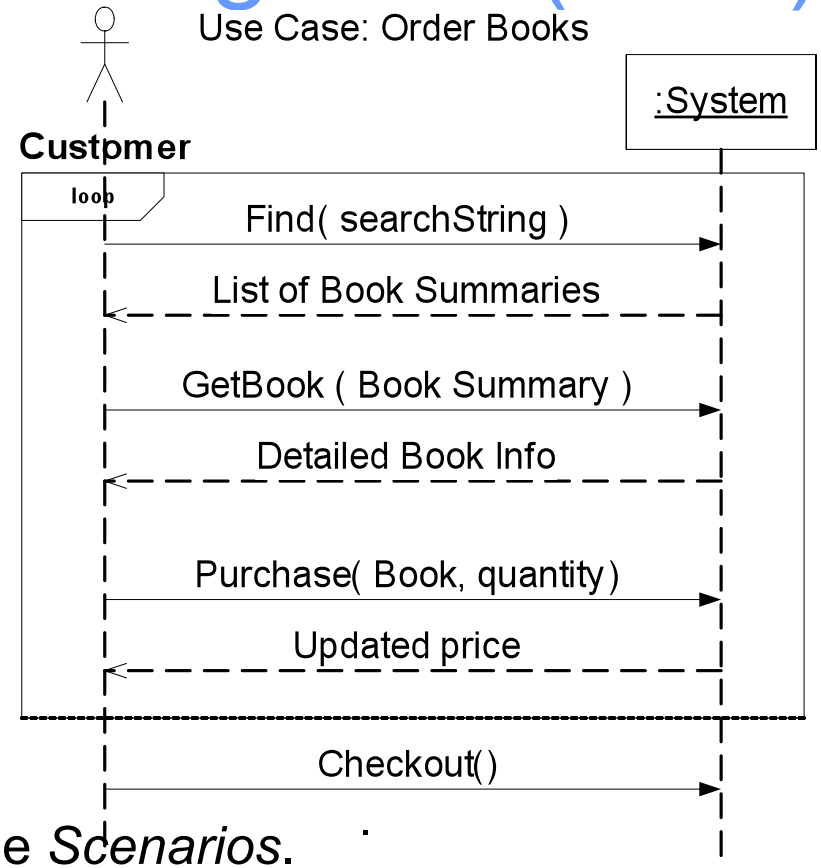
–President Rutherford B. Hayes, about the telephone (allegedly).


1876 - Bell's original telephone

improving
It's what we do. ™

System Sequence Diagrams (SSD)

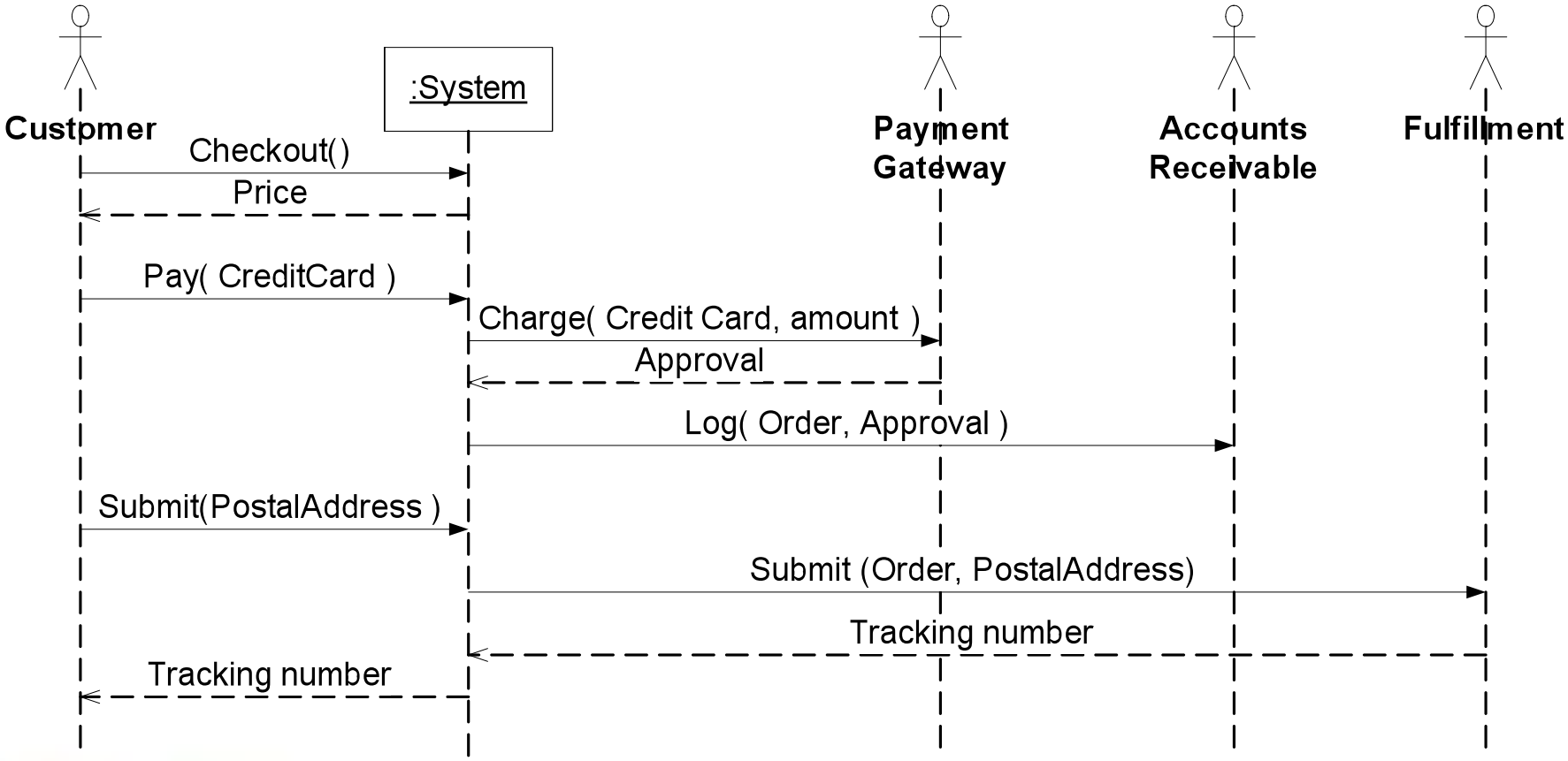
Actor Action	System Response
1. The use case begins when a …	2. … list of book summaries …
3. The Customer selects a book …	4. … provides detailed …
5. The Customer requests the book.	6. … adds the book to the …
	7. Returns the …
The Customer may repeat Steps 1-7.	
8. The Customer asks to buy the …	9. … request Credit Card …



- A Use Case is composed of multiple *Scenarios*.
- Each Scenario is composed of one or more *System Events*.
- Each System Event is supported by the system as a *System Operation*.

improving
It's what we do. ™

Multiple Actors on an SSD

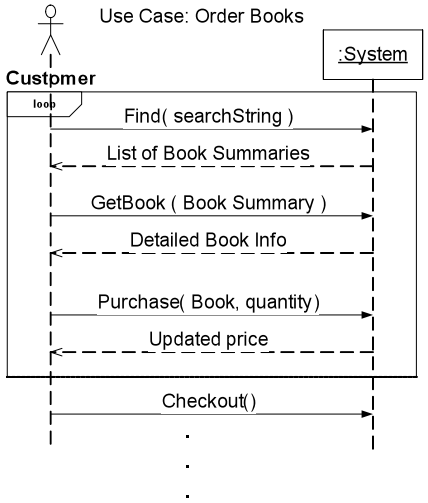


- Note that the System Events don't have to exactly match the text of the use case text.

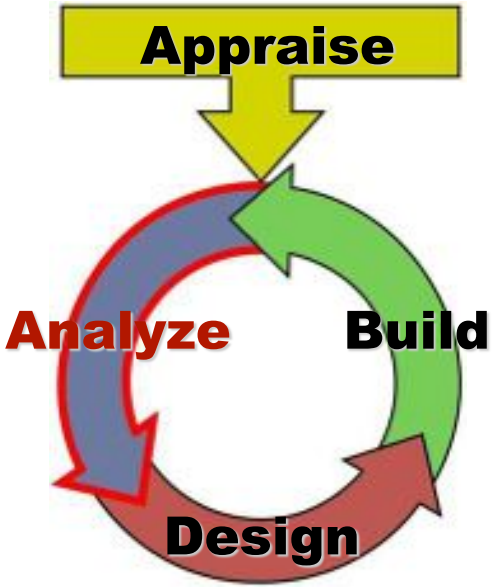
improving
It's what we do. ™

Exercise: System Sequence Diagrams

Create the System Sequence Diagram for the Main Success Scenario of the use cases specified by the instructor.



improving
It's what we do. ™



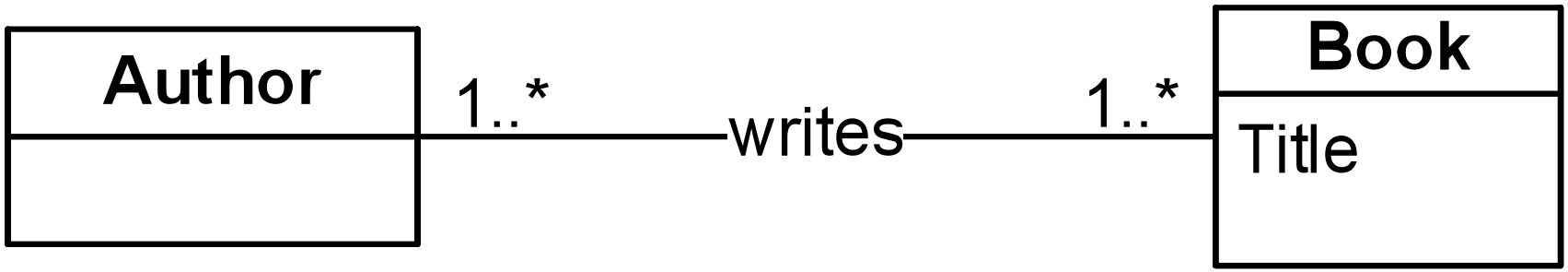
Domain Model: Visual Dictionary

"This is a war of the unknown warriors; but let all strive without failing in faith or in duty…"

—Winston Churchill, on Domain Modeling

improving
It's what we do. ™

Domain Models



- Domain Models are a visual dictionary used to help us speak precisely about our problem.
  - **Concepts** (Candidate Classes)
  - **Relationships** (usually Associations)
  - **Attributes**

Domain Models—Impact

- Contain information-related requirements.
- Precisely defines words we casually use.
- Support design using *Low Semantic Gap*.

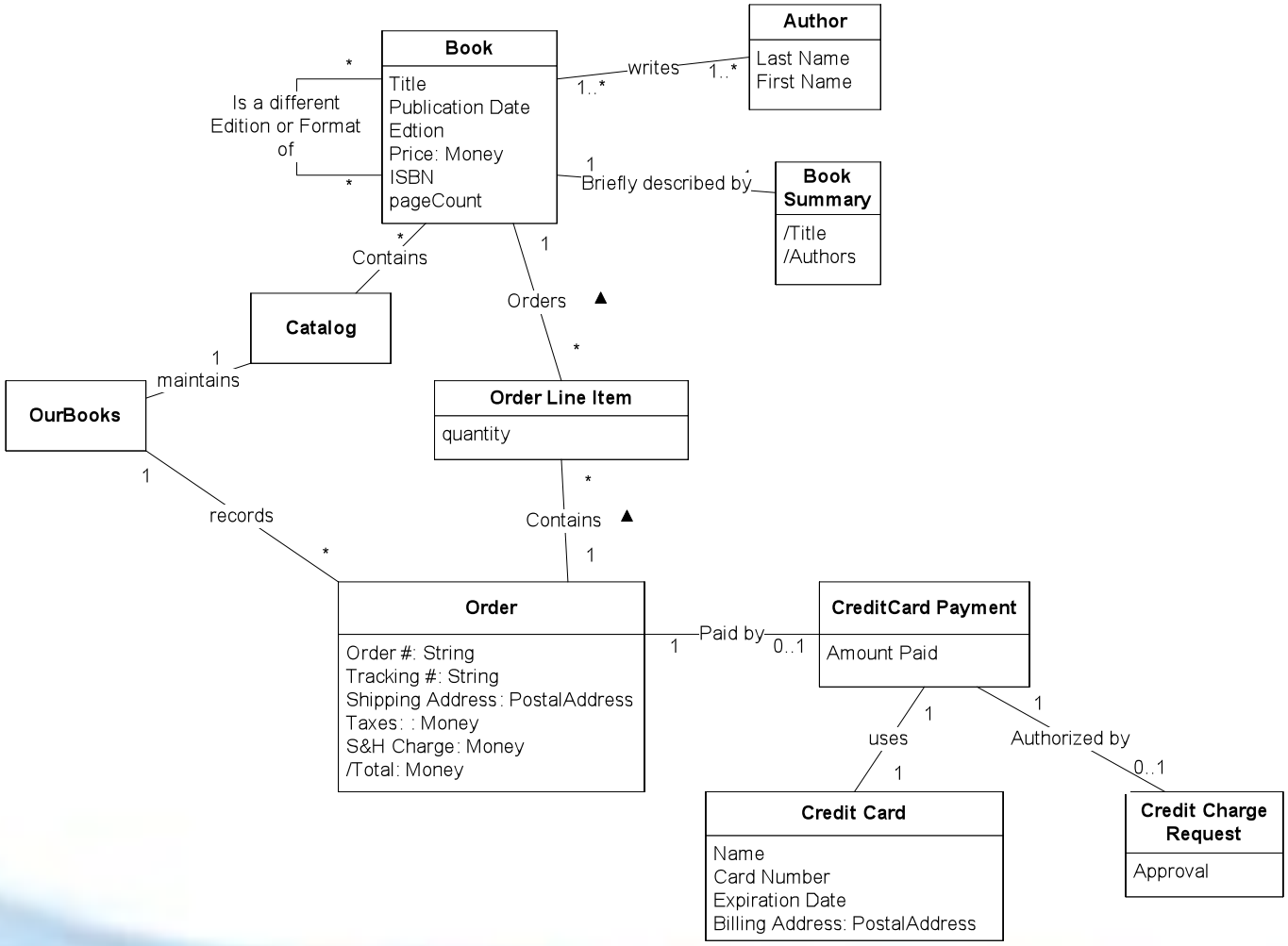
- We often use one term to mean many things, and several terms to mean one thing.
  - Shopping Cart vs Order, Customer vs User

- Domain Models are very difficult to build if we don't understand our problem.
  - ☞ *The harder they are to build, the more useful they'll be to us.*

Domain Modeling: Vocabulary

improving
It's what we do. ™

Domain Model



improving
It's what we do. ™

Domain Models—Concepts

- Concepts are categories of things that are alike in some interesting way.
  - Typically we name them in the singular unless the concept itself represents a grouping of things.
- Concepts can be defined in two ways:
  - By Definition

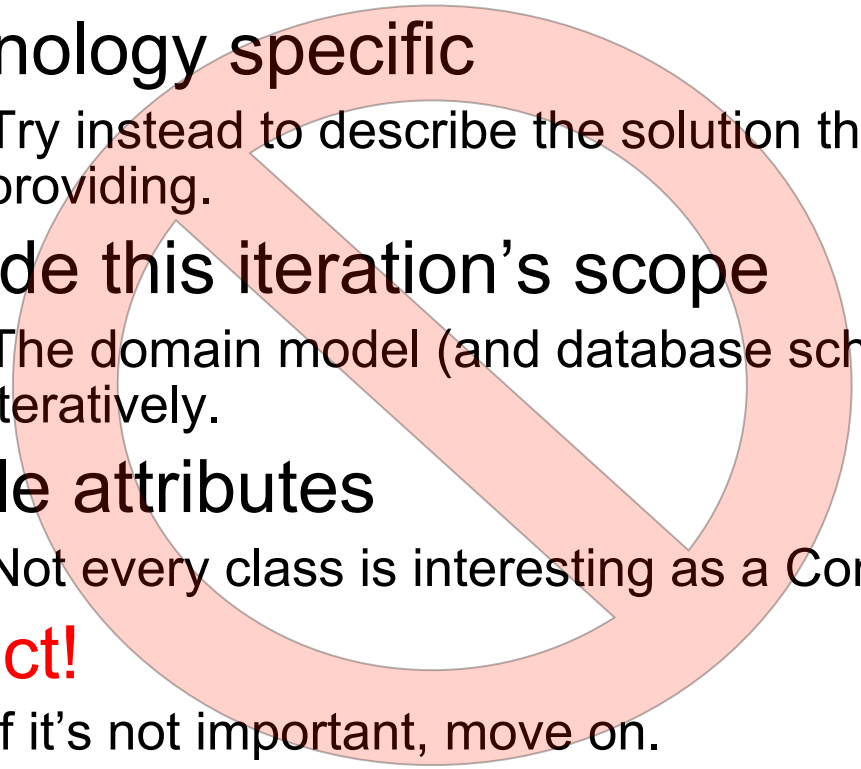
    Order: A request for Books by a specific Customer requested and paid for in one session.
  - By Listing

    Planet: Mercury, Venus, Earth, Mars, Jupiter, Saturn, Uranus, Neptune, ~~Pluto~~

Domain Modeling—Concepts

Concepts should **not** be:

- Technology specific
  - Try instead to describe the solution the technology is providing.
- Outside this iteration's scope
  - The domain model (and database schema) can grow iteratively.
- Simple attributes
  - Not every class is interesting as a Concept.
- Perfect!
  - If it's not important, move on.



Domain Models—Concepts

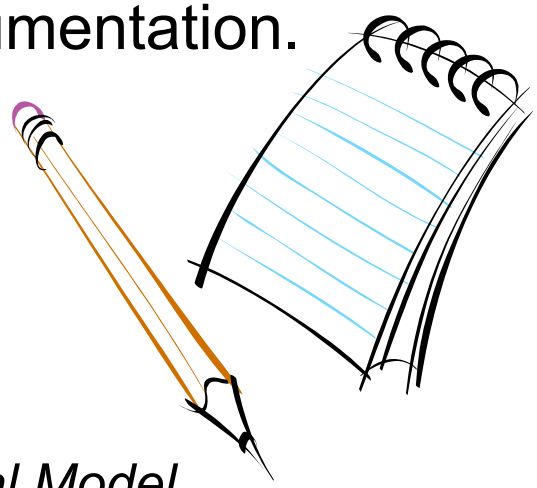
To discover Concepts:

A) Look for noun or noun phrases in documentation. Underline them in your copy.

B) Double-check against a Common Concepts list.

C) Investigate existing analysis patterns.

- Martin Fowler, *Analysis Patterns.*
- C.J. Date, *Temporal Data & the Relational Model.*
- …



Domain Models—Underlining Nouns & Noun Phrases

Actor Action	System Response
1. The use case begins when a Customer begins searching for books by supplying a book title.	2. Returns a list of book summaries that match the search criteria.
3. The Customer selects a book summary to see detailed information.	4. Provides detailed information about the book (number of pages, other editions/formats of the book, a link for each author to a list of his books, etc.)
5. The Customer requests the book.	6. Adds the book to the shopping cart and shows the new total price.
	7. Returns the new total price.
The Customer may repeat Steps 1-7.	

Common Concepts List

Transaction-oriented Concepts

Conceptual Class Category	Examples
Business transactions	Sale, Payment, Reservation
Transaction Line Items	Sales Line Item
Product/Service used in a transaction	Item, Flight, Seat, Meal
Records of finance, work, contracts, legal matters	Receipt, Ledger, Maintenance Log
Roles of people or organizations; actors	Cashier, Customer, Store, Passenger, Airline
Place of Transaction or Service	Store, Airport, Plane, Seat

* from *Applying UML and Patterns*, pp 140-141

Common Concepts List

Other typical categories

Conceptual Class Category	Examples
Noteworthy Events (often with a time or place)	Sale, Payment, Flight Departure
Physical Objects	Item, Board, Airplane, Drivers License
Description of things (Catalog-type entries)	Product Description, Flight Description, SKU
Catalogs / Collections / Containers	Product Catalog, Store, Shelf, Barrel
Things in a container	Item, Apple (in a Barrel)
Other collaborating systems	Payment Gateway, Air Traffic Control
Financial Instruments	Cash, Check, Line of Credit, Tranche
Documents that are regularly referred to	Daily Price Change List, Repair Schedule

* from *Applying UML and Patterns*, pp 140-141

Common Concepts List

From 3rd Grade

Conceptual Class Category	Examples
Who? (Roles, Organizations)	Cashier, Company, Department
What? (Tangible and Intangible things)	Airplane, Seat, Drivers License, Sale
When? (Interesting periods of time)	Effective Period, Admittance, Discharge
Where? (Places events happen)	Store, Address, URL
How? (Existing solutions to be used by System)	Protocol, Rickshaw, Cog
Why? (Goals and Objectives)	Sale, Exercise Outcome

Exercise: Creating a Domain Model Step 1

1. Brainstorm a list of Concepts for your project.
   - It's better to have extra items in the list than to forget some.
   - Use the 3 methods to find candidates. [In class, you will not have access to *analysis patterns*, so just use the first two methods.].

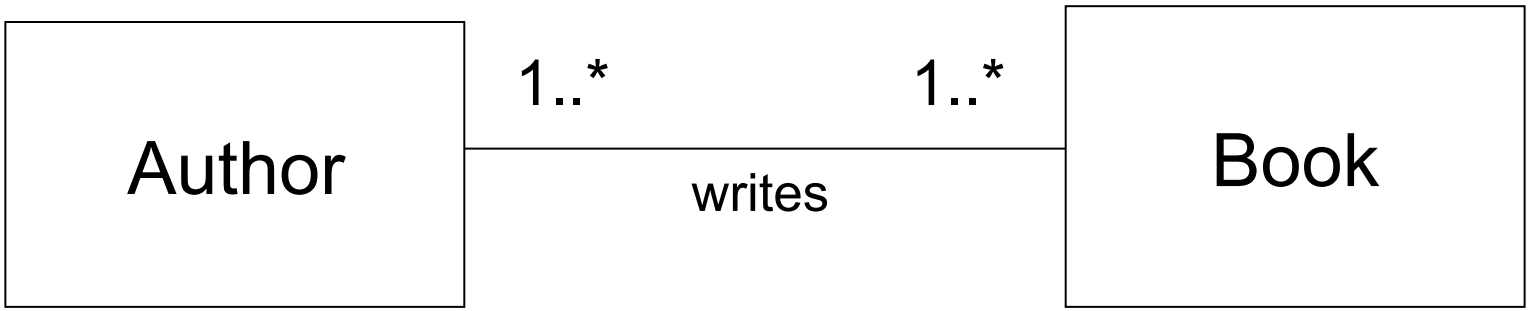
Order	OrderLineItem
Customer	Payment
OurBooks	CreditCard
Book	CCV2
BookSummary	Fulfillment
Author	…
…	

improving
It's what we do. ™

Domain Models—Relationships

- Relationships denote an interesting linkage between two or more Concepts.
- Only add Relationships for which there is a business reason for remembering the linkage.
  - "Need to Know" or "Noteworthy".
- A Relationship is a truth at *one moment in time*.

Multiplicity

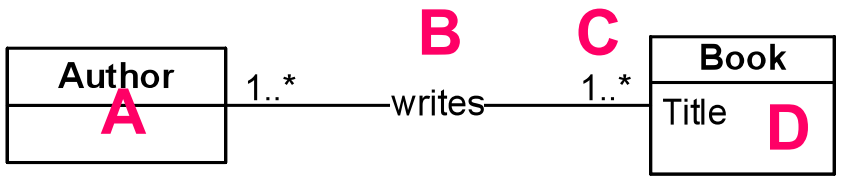


*	Zero or more	1..*	One or more
0..*	Zero or more	5	Exactly five
0..1	Zero or one	2,4	Two or four

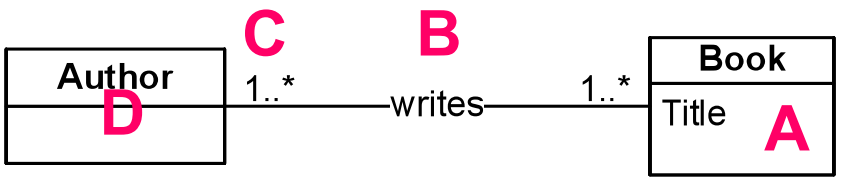
Domain Model—Relationships

Each Relationship makes 2 statements:

1 Author writes one or more Books.



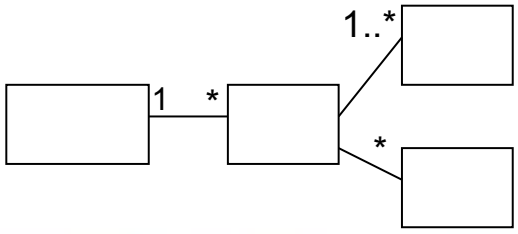
1 Book is written by one or more Authors.



improving
It's what we do. ™

Exercise: Creating a Domain Model Step 2

2. Draw Relationships between Concepts.
   - Start drawing the picture.
   - Don't add Concepts to the picture until you find a compelling Relationship.
   - Start with a central Concept (e.g., *Order*, or *Book*).
   - Leave room in the Concept boxes for Attributes.

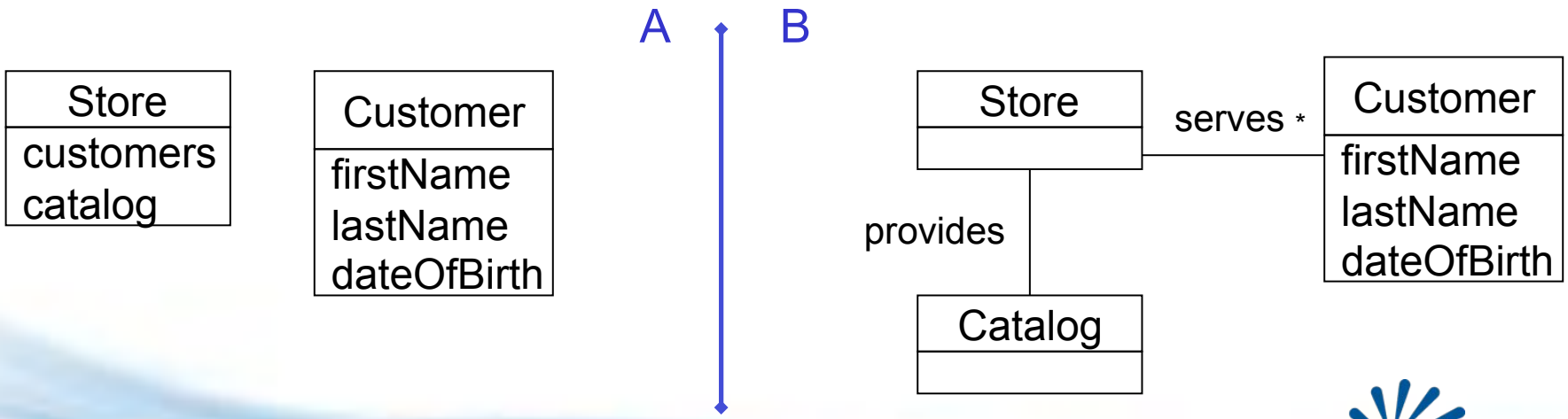


improving
It's what we do. ™

Domain Models—Attributes

- Attributes specify essential data belonging to each instance of a Concept.
- Something you could type into a form.

Which is better—A, or B?



improving
It's what we do. ™

Domain Models—Attributes

An Attribute should be “promoted” to a Concept if it has:

- Identity or Structure that you care about

  Should *Billing Address* be an attribute, or should there be a relationship between *Credit Card* and *Postal Address*?

- Relationships to other Concepts

  If we care about the possible *Delivery Services* available for an *Order* based on the *Postal Address*, should it be related to *Order* or *Postal Address*?

Exercise: Creating a Domain Model Step 3

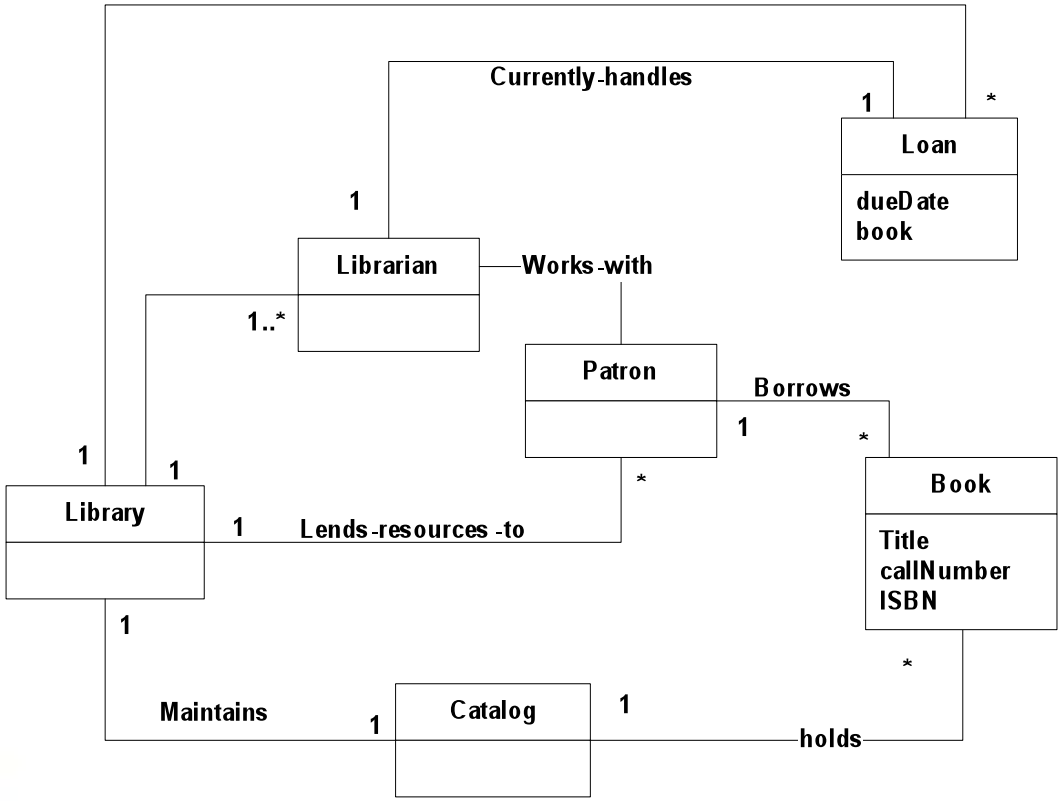
3. Add Attributes to Concepts.

- Add the obvious Attributes, especially those that help define the Concept.
- Provide type information only to make a strong statement (Total: Money). Otherwise, leave it off.
- Postpone design considerations (e.g., foreign keys, database ids).
- Don't worry about missing some (CCV2 on Credit Card).

Order
Order #: String Tracking #: String Shipping Address: PostalAddress Taxes: : Money S&H Charge: Money /Total: Money

Exercise

Improve this domain model:



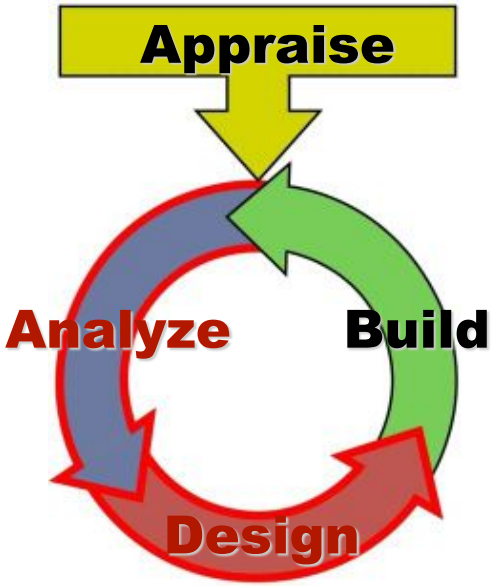
* Derived from ObjectSpace's *Object-Oriented Analysis and Design*, 1996

improving
It's what we do. ™

Creating a Domain Model

1.
    (a)
    (b)
    (c)
2.

3.



Acceptance Criteria
Setting Expectations

"Begin with the end in mind."

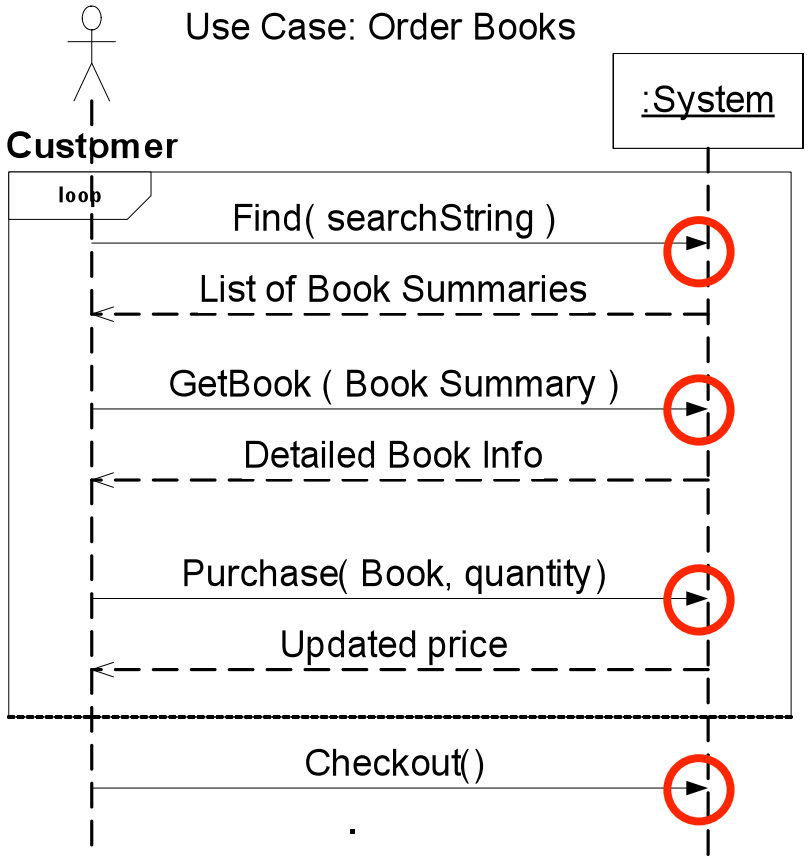
—Steven Covey

improving
It's what we do. ™

So you have a diagram…

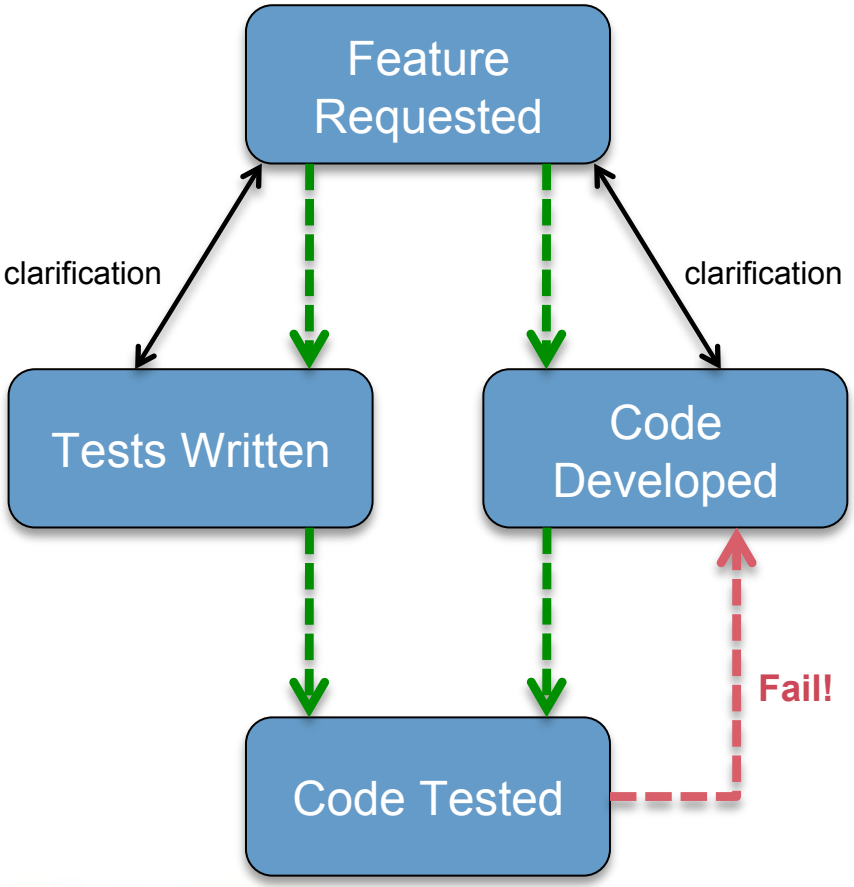
Now what?

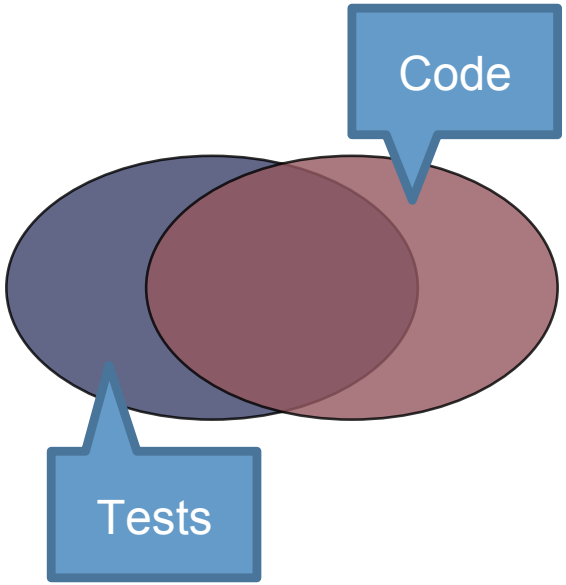
How do you know you're done programming?



improving
It's what we do. ™

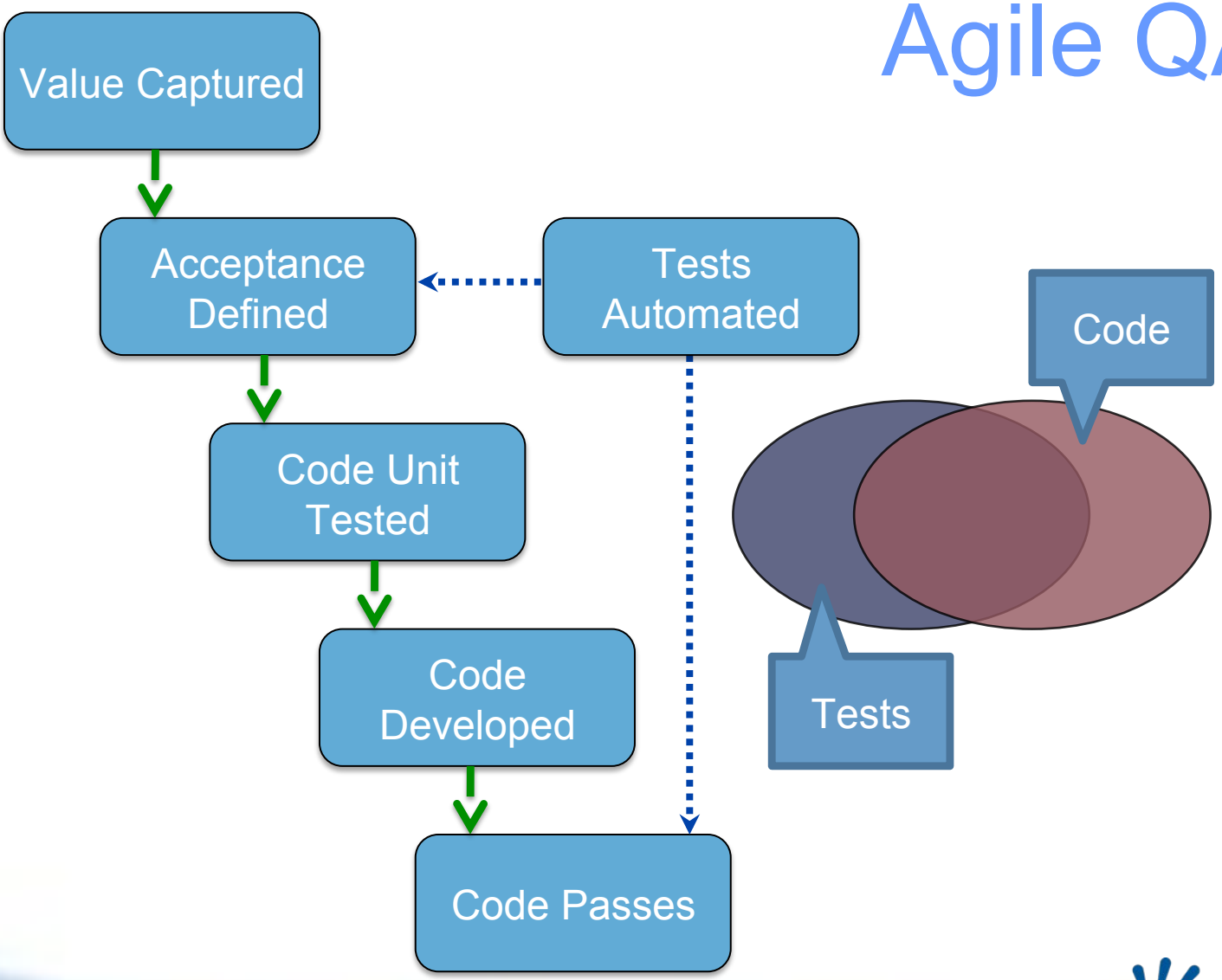
Traditional QA





improving
It's what we do. ™

Agile QA



improving
It's what we do. ™

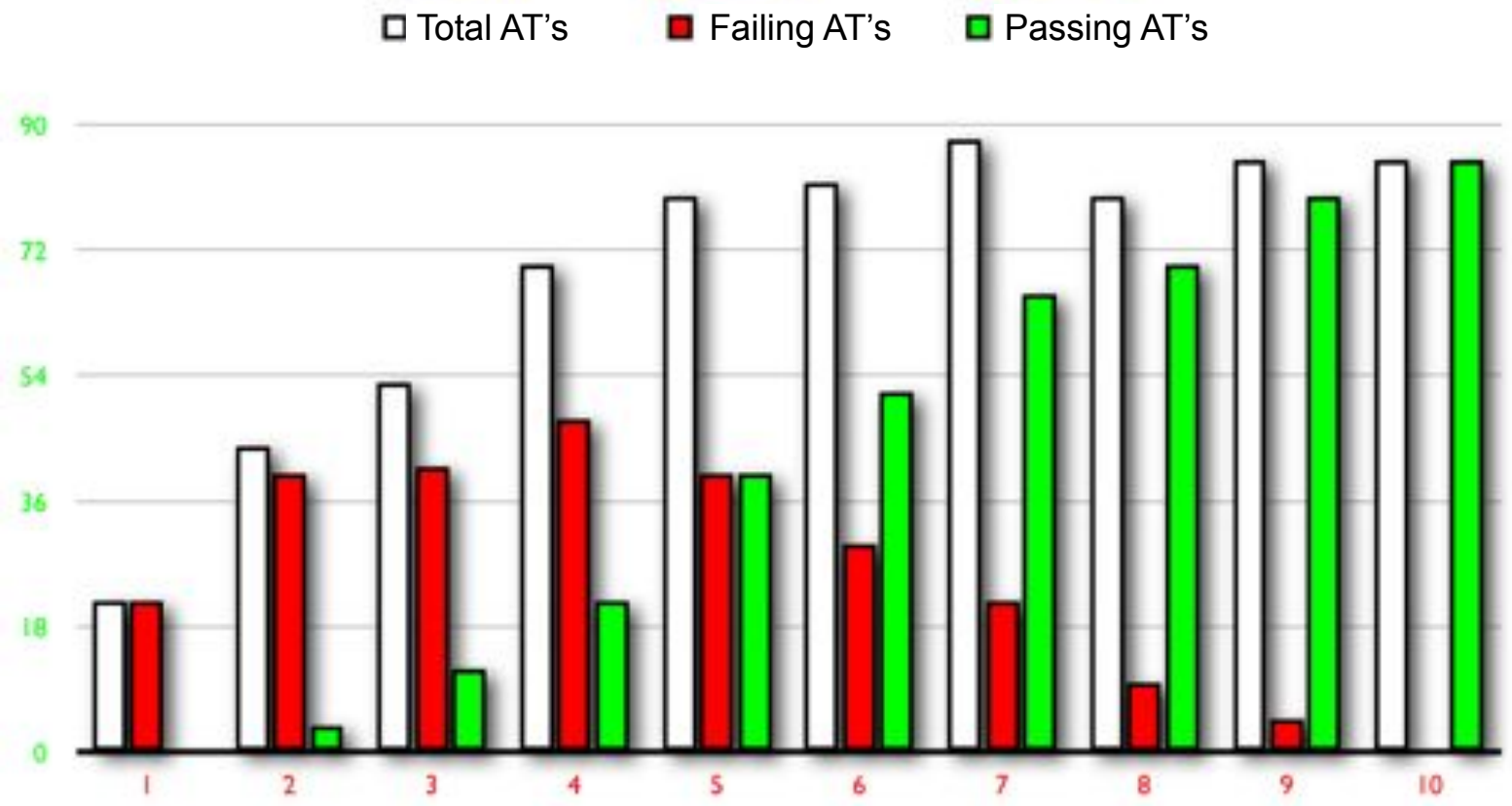
Acceptance Criteria

Acceptance Criteria define what the customer will see in order to approve the work as being complete.

- Scenarios
- Test Cases
- Acceptance tests
- Operation Contracts
- Before and After picture

improving
It's what we do. ™

The Goal



improving
It's what we do. ™

Scenarios

- Briefly describe the scenario to be implemented:

  Customer buys 1 book with valid credit card.

  Customer buys 2 books with valid credit card.

  Customer is prevented from buying 0 or too many books.

  Customer is prevented from buying books with invalid credit card.

  Order Books, Happy Path Steps 1-7

  Order Books, Steps 1-3, including extensions

Acceptance Tests

- Create all the test cases that, when passed, specify the system is acceptable to the customer.
- Each test case describes in a step-by-step manner a specific interaction with the system and the expected response.

Test 1:

1. The user visits the home page.

   The home page displays a banner, …, search text box, … .

2. The user searches for "O'Dell".

   The search results page with 1 book, Object-Oriented Methods, should display.

3. The user selects to buy Object-Oriented Methods.

…

Operation Contracts

- Describe the affects of a System Operation using a Domain Model.

Operation: Purchase(Book book, int quantity)

Pre-conditions: book is contained in OurBooks.Catalog

Post-conditions:

if needed, Order order was created.

OrderLineItem oli was created.

oli was associated with order.

oli.quantity was set to quantity

oli.book was associated with book.

5 Changes

- Instance created
- Instance destroyed
- Association formed
- Association broken
- Attribute modified

improving

It's what we do. ™

Operation Contracts

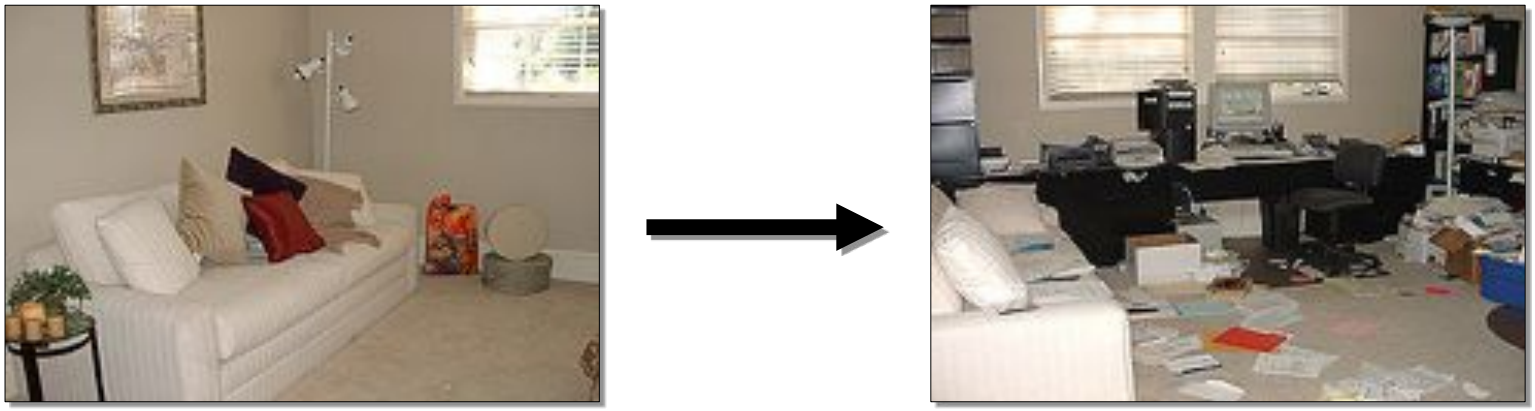
Operation: Find(String searchString): List<BookSummary>

Pre-conditions:

Post-conditions:

Return: List of BookSummary that are associated with Books in OurBooks.Catalog that match searchString

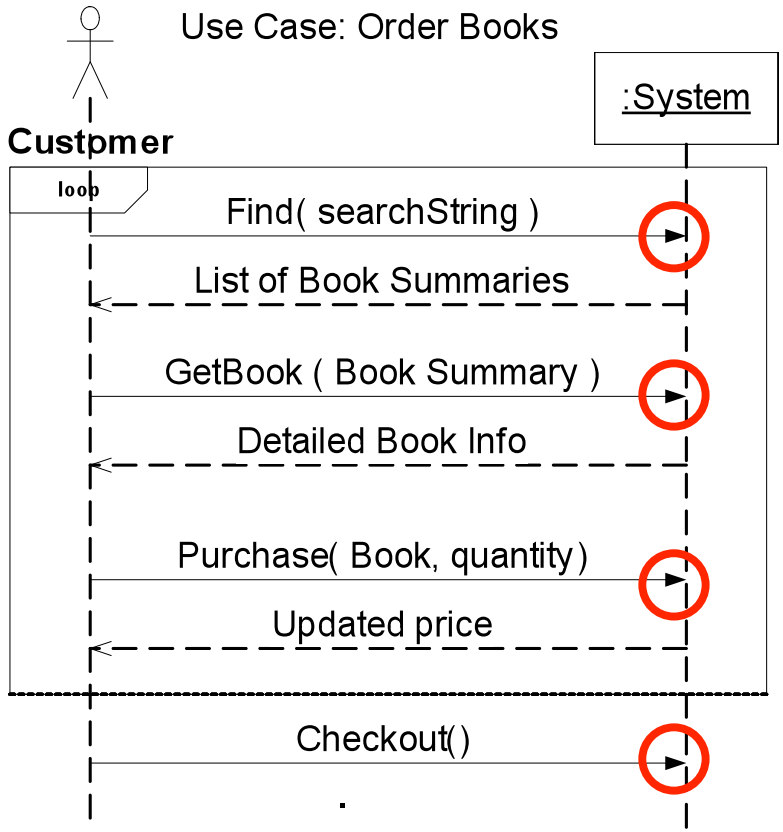
Instance Diagrams



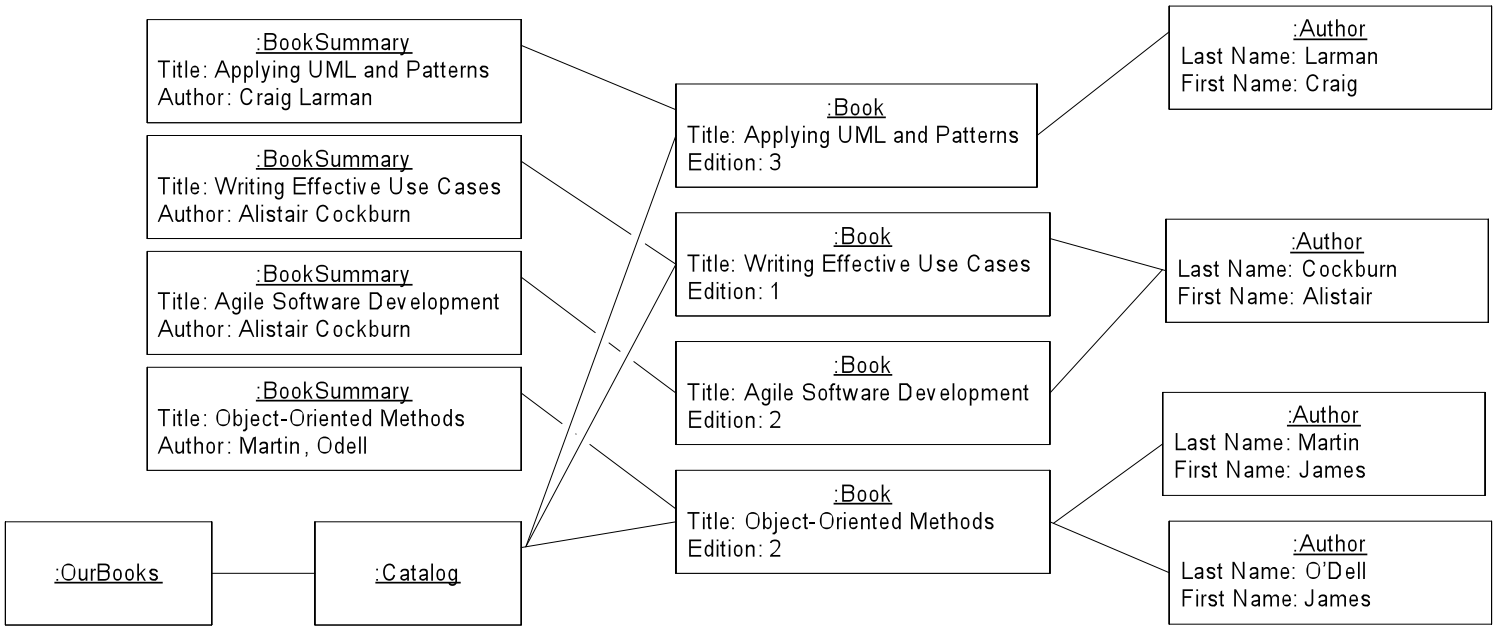
Before and After the Pay Taxes System Operation.

- Photos From http://www.stagedhomes.com

- Instance Diagrams show the network of objects in the system.
- They can be especially helpful for showing a “before and after picture” of a system operation.

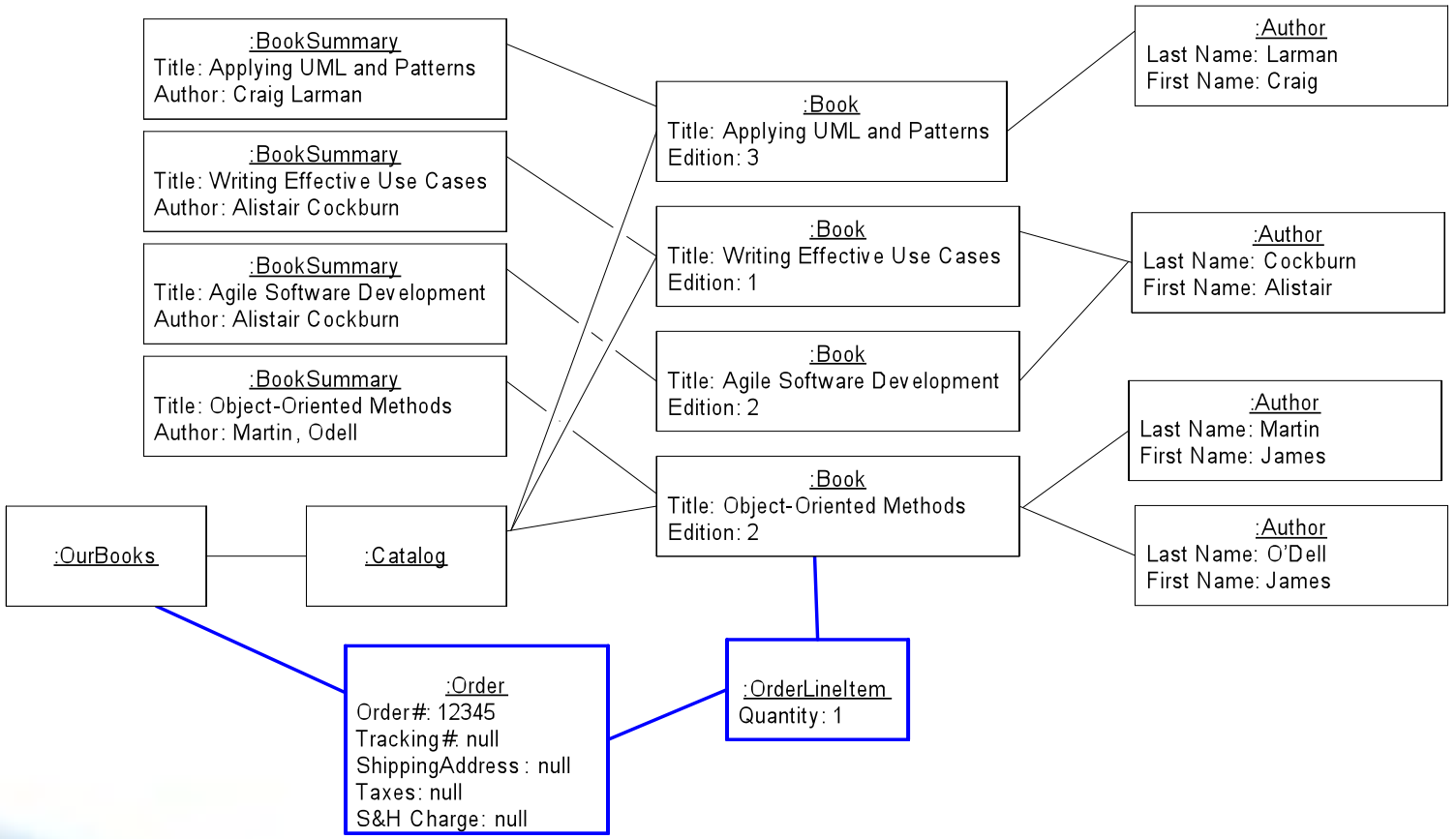


After GetBook(BookSummary) & Before Purchase(Object-Oriented Methods)



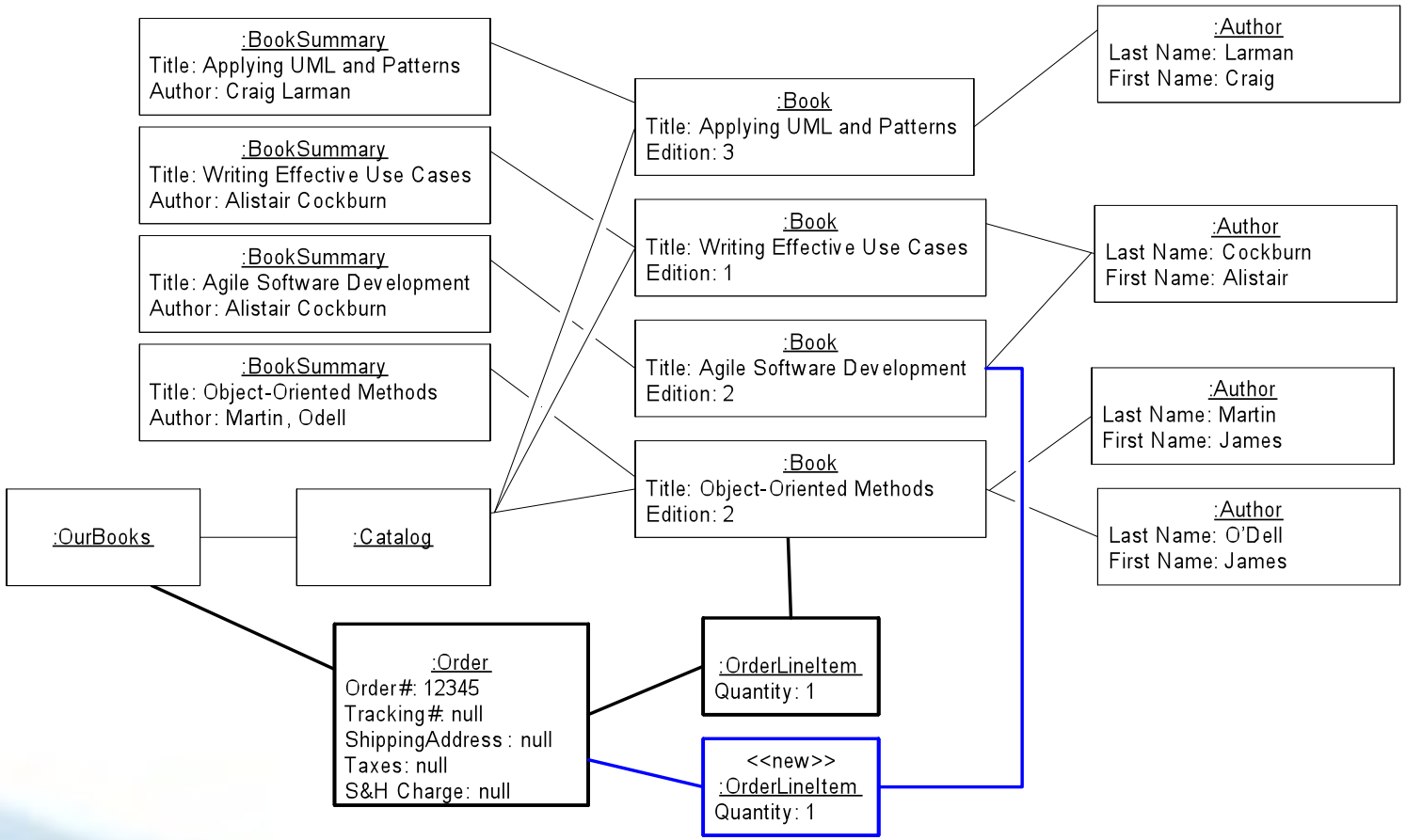
improving
It's what we do. ™

After Purchase(Object-Oriented Methods)



improving
It's what we do. ™

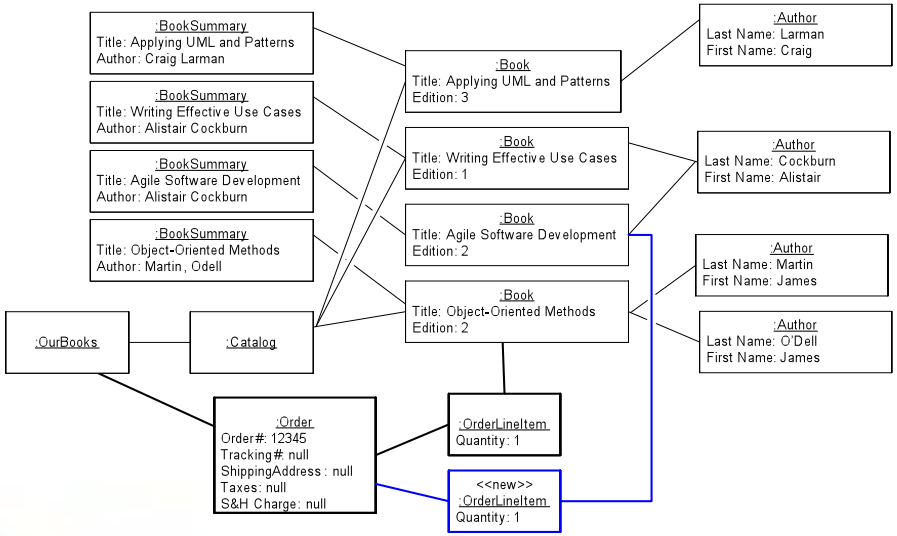
After Purchase(Agile Software Development)



improving
It's what we do. ™

Exercise: Instance Diagrams

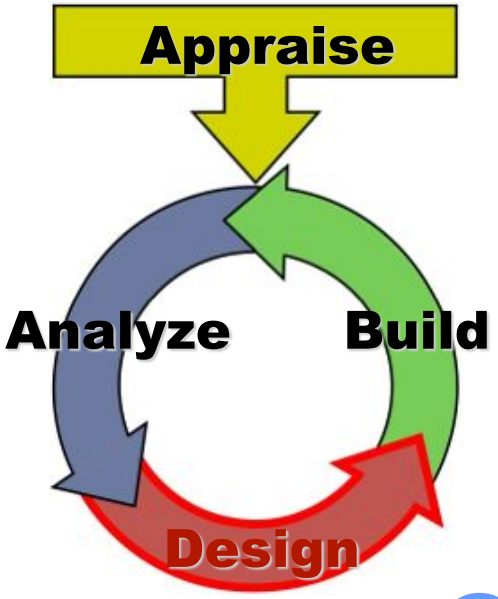
- Create the instance diagrams for the System Operations in the course project.



improving
It's what we do. ™

Exercise: Review

Document	Description	Domain Model
Use Case Diagram	Purpose: Components:	Purpose: Components:
Use Case Brief	Purpose: Components:	
Fully Dressed Use Case	Purpose: Components:	
System Sequence Diagram	Purpose: Components:	
Acceptance Criteria	Purpose: Components:	



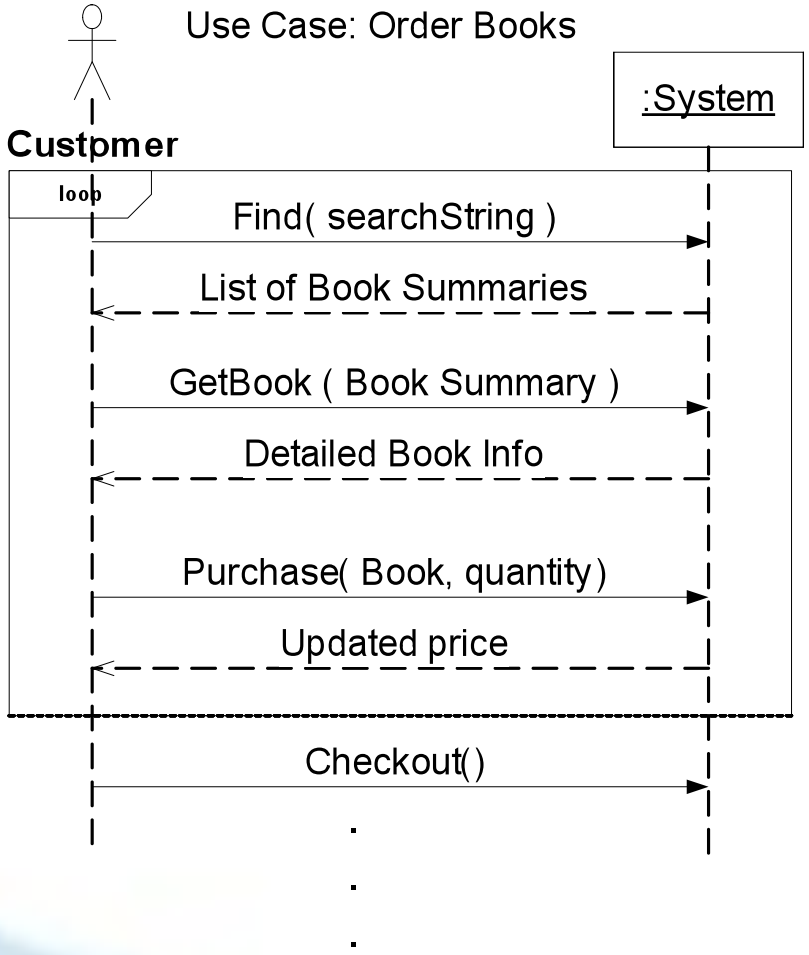
Applying Fundamental Object-Oriented Principles

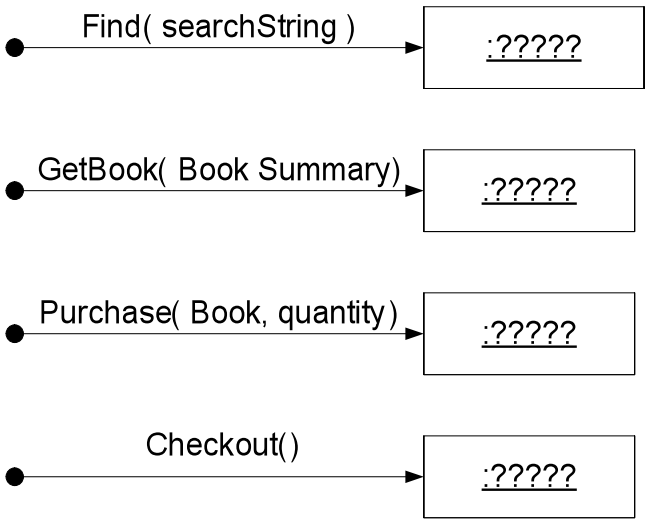
"No complex adaptive system will succeed in adapting in a reasonable amount of time unless the adaptation can proceed subsystem by subsystem, each subsystem relatively independent of the others."

–Christopher Alexander, 1964

improving
It's what we do. ™

Design





improving
It's what we do. ™

The Object-Oriented Paradigm

1. The principle of recursive design is that the parts have the same power as the whole.
    - » Bob Barton, 1968.
2. Objects 101.
    - Everything is an object.
    - Objects communicate by sending and receiving messages.
        - » Alan Kay, formulated by 1973.
3. Information Hiding.
    - Modules hide design decisions. Interfaces reveal as little as possible about inner workings.
        - » David Parnas, 1972
4. Low Semantic Gap.
    - Classes are named and behave congruently with real-world counterparts.
        - » Unknown, used since the mid ’70s.

Classes

So we have a definition of *Object*.

What is a *Class*?

What responsibilities does a class have (if any)?

Object-Oriented Design Principles

Low Coupling

– Modules should have low dependency on other modules.
  » Larry Constantine, mid-1960s; Julius Caesar, 100BC - 44BC

- Example of higher (poor) Coupling

```
public String getAuthorsFullName() {
   String name = this.getAuthor().getLastName();
   name = name + ", " + this.getAuthor().getFirstName();
   return name;
}
```

- Example of lower (better) Coupling

```
public String getAuthorsFullName() {
   String name = this.getAuthor().getFullName();
   return name;
}
```

Object-Oriented Design Principles

High Cohesion

- Modules should have a single purpose.
  - » Larry Constantine, mid-1960s

- Example of lower (poor) Cohesion

```
public Book(String title) {
  if (CONNECTION == null)
    CONNECTION = createDatabaseConnection();
  this.title = title;
}
```

- Example of higher (better) Cohesion

```
public Book(String title) {
  this.title = title;
}
```

Exercise: Coupling & Cohesion

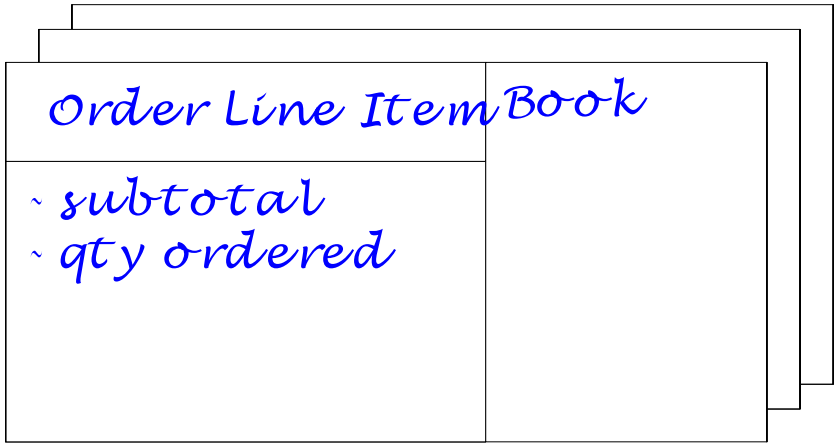
```
public class AtmClient {
   public boolean withdraw( String token, int amount ) throws Exception {
   // FIXME: clean this code up later
   Socket kkSocket = new Socket("127.0.0.1", 10001);
   PrintWriter out = new PrintWriter(kkSocket.getOutputStream(), true);
   BufferedReader in = new BufferedReader(
                  new InputStreamReader(kkSocket.getInputStream()));
   String msg = "ATM6281|" + token + "|1|W|" + amount;
   String eMsg = Crypto.encrypt("Rijndael", msg);
   out.println(eMsg);
   String fromServer = in.readLine();
   out.close();
   in.close();
   kkSocket.close();

   return fromServer.equals("0");
   }
}
```

Object-Oriented Design

- Object-oriented design is assigning responsibilities to objects and deciding which other objects they'll work with.

- CRC
  - **C**lasses
  - **R**esponsibilities
  - **C**ollaborators



GRASP Patterns

A set of fundamental patterns and principles giving advice on assigning responsibilities.

1. Low Coupling
2. High Cohesion
3. Controller
4. Creator
5. Expert

…plus 4 others

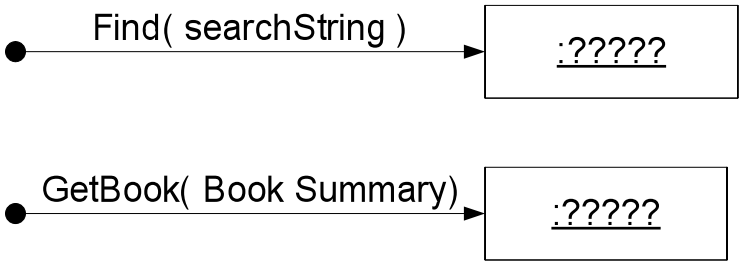
General
Responsibility
Assignment
Software
Patterns

* from *Applying UML and Patterns*

Design Patterns: GRASP basics

Controller

- Choose an object to receive the System Operation based on reducing Coupling and maintaining Cohesion.



- Candidates include objects representing:
  - a single run of a use case — OrderBooksHandler
  - the entire system — OurBooksWeb
  - the business — OurBooks
  - a device — Printer, Scanner

Design Patterns: GRASP basics

Controller

To Answer: Who should receive Find()?

1. List candidates.

   OrderBooksHandler
   OurBooksWeb
   OurBooks

2. Evaluate the candidates based on Coupling & Cohesion.

3. Select the best candidate.

   OrderBooksHandler

Design Patterns: GRASP basics

Creator

- Choose an object that creates another object based on reducing Coupling and maintaining Cohesion.

- Candidates include:
  - Objects that contain the to-be-created object
  - Objects that closely use (will already be highly coupled with) the to-be-created object
  - Objects that have the data necessary to construct the object

Design Patterns: GRASP basics

Creator

To Answer: Who should create OrderLineItems?

1. List candidates.

   Order
   OrderBooksHandler

2. Evaluate the candidates based on Coupling & Cohesion.

3. Select best candidate.

   Order

Design Patterns: GRASP basics

Information Expert

- Assign the responsibility to the object that has most of the information needed to fulfill the responsibility.
- “He who knows, does.” (Seive)

Design Patterns: GRASP basics

Information Expert

To Answer: Who should calculate a Customer's Age?

0. List the information needed to fulfill a responsibility.

   Birth Date
   Today

1. List candidates, based on who knows the information.

   Customer
   Calendar

2. Evaluate the candidates based on Coupling & Cohesion.

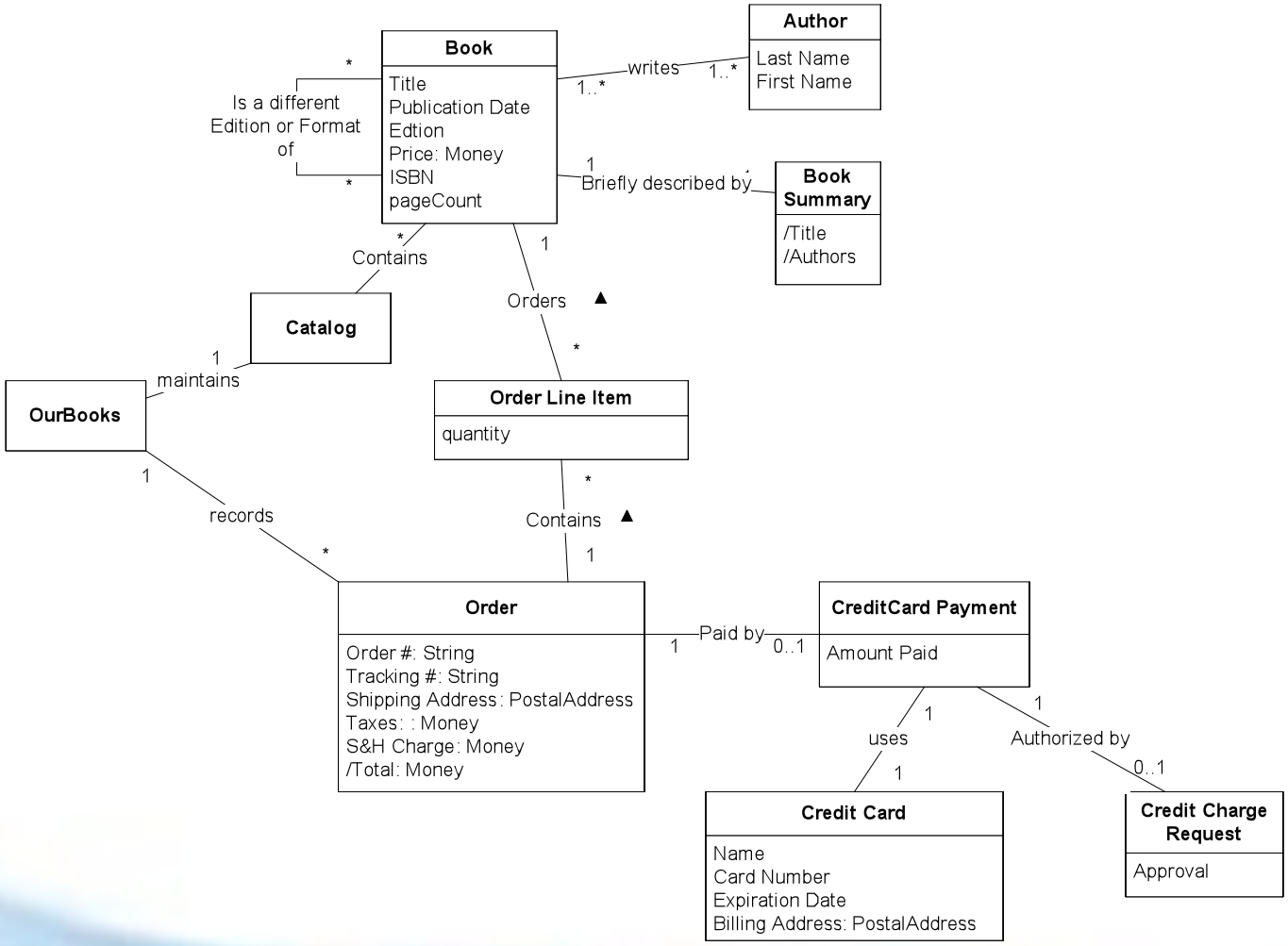
3. Select the best candidate.

   Customer

3 Step Simple Design Process

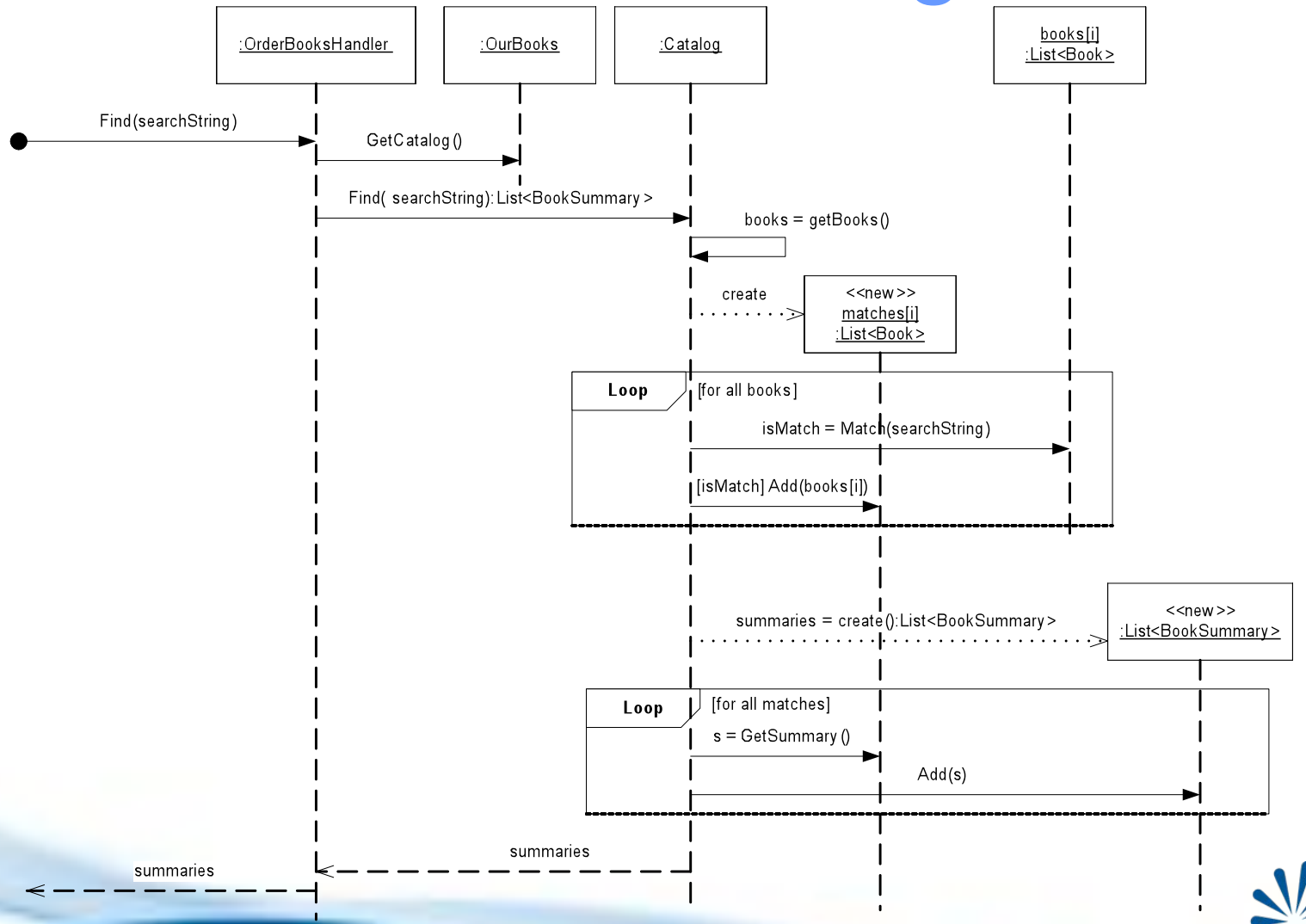
1. List candidate objects.
   - Low Semantic Gap
   - Advice from patterns
2. Evaluate each candidate.
   - Low Coupling
   - High Cohesion
3. Select the best candidate.

Domain Model



improving
It's what we do. ™

Design of Find()



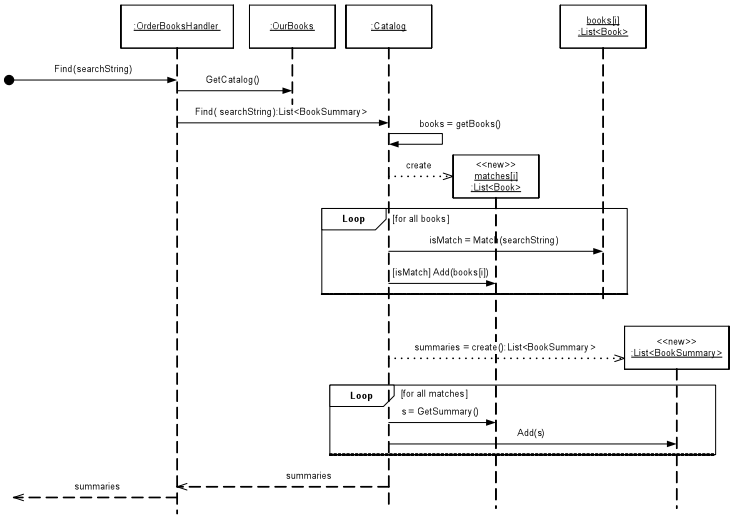
improving
It's what we do. ™

Design of Find()

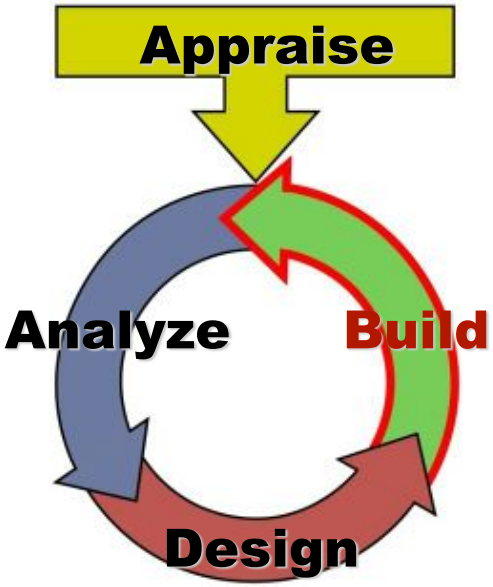
- When coding this, what changes would you make to it to improve Coupling and Cohesion at a statement or block level?
- Our design naïvely assumes all the Books in the Catalog are in memory.
  - How many Books do we sell?
  - How much memory will that take?
  - What is the arrival and departure rates for books?
  - How often does the information of an existing book change?
  - How bad is the assumption?

Exercises

- Design GetBook(BookSummary)
- Design Purchase(Book)
- Design the System Operations for the course project.



improving
It's what we do. ™

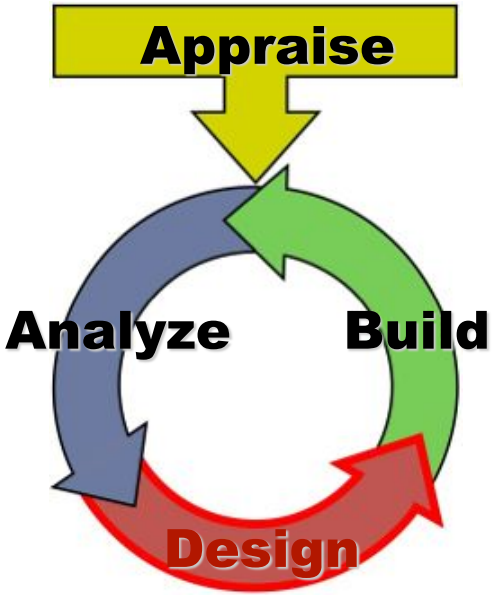


Exercise

Write the code for selected methods.

```
public class OrderBooksHandler {
private OurBooks theStore;
public OrderBooksHandler(OurBooks ourBooks) {
    this.theStore = ourBooks;
}
public List<BookSummary> find(String searchString) {
    return theStore.getCatalog().find(searchString);
    }
}
```

improving
It's what we do. ™



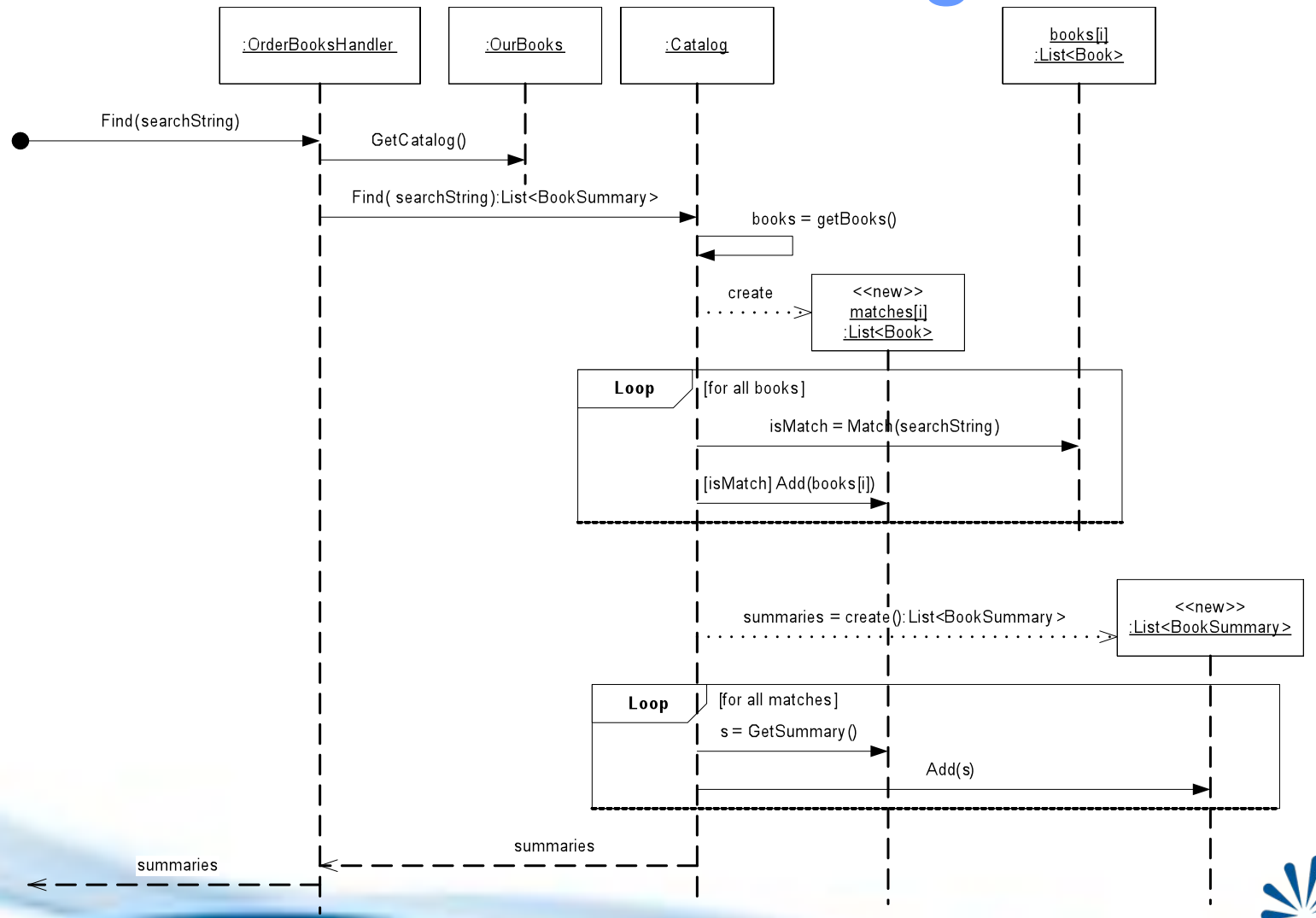
Design Class Diagram: Summarizing Design Decisions

improving
It's what we do. ™

Design Class Diagram

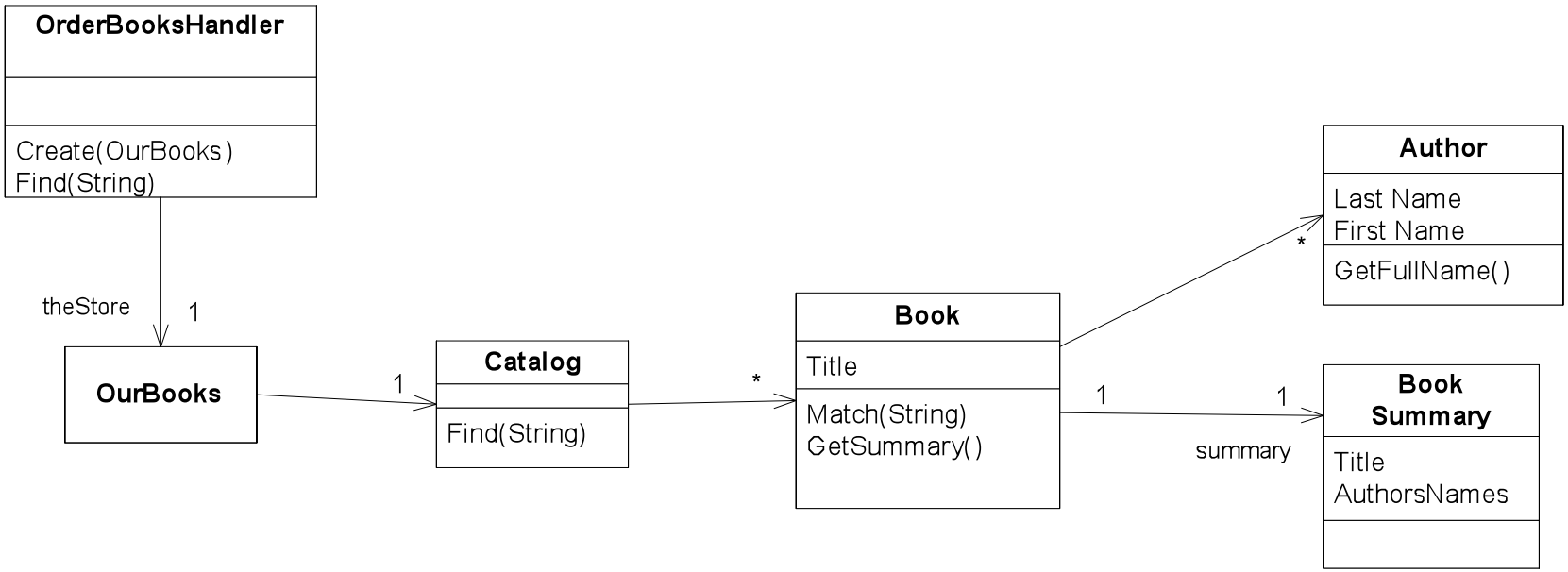
- The Design Class Diagram (DCD) summarizes the design decisions shown or assumed in Interaction Diagrams.
- Looks like a Domain Model except:
  - Behaviors are added to classes
  - Relationships are directed
  - Role names more important than association names

Design of Find()



improving
It's what we do. ™

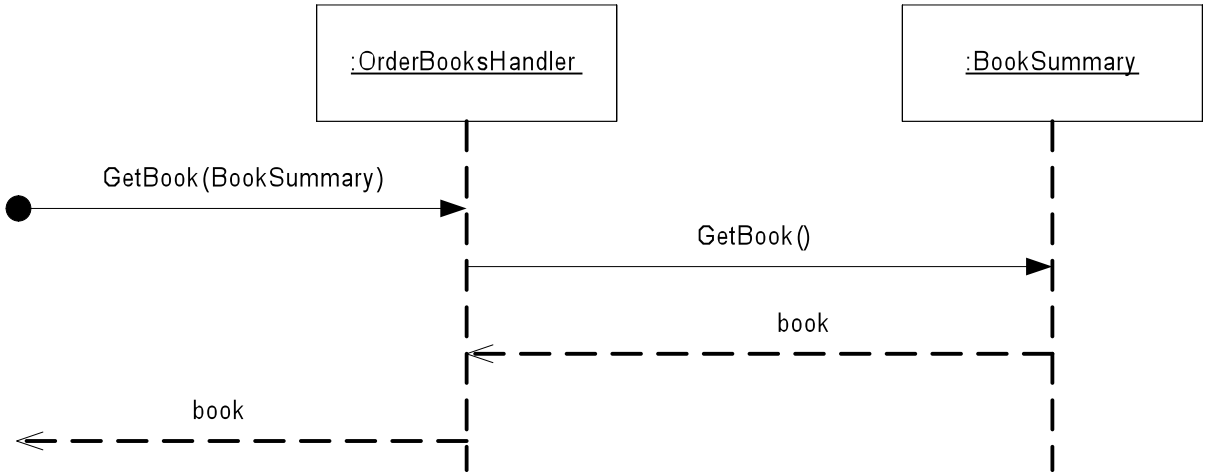
Growing the DCD: From Find()



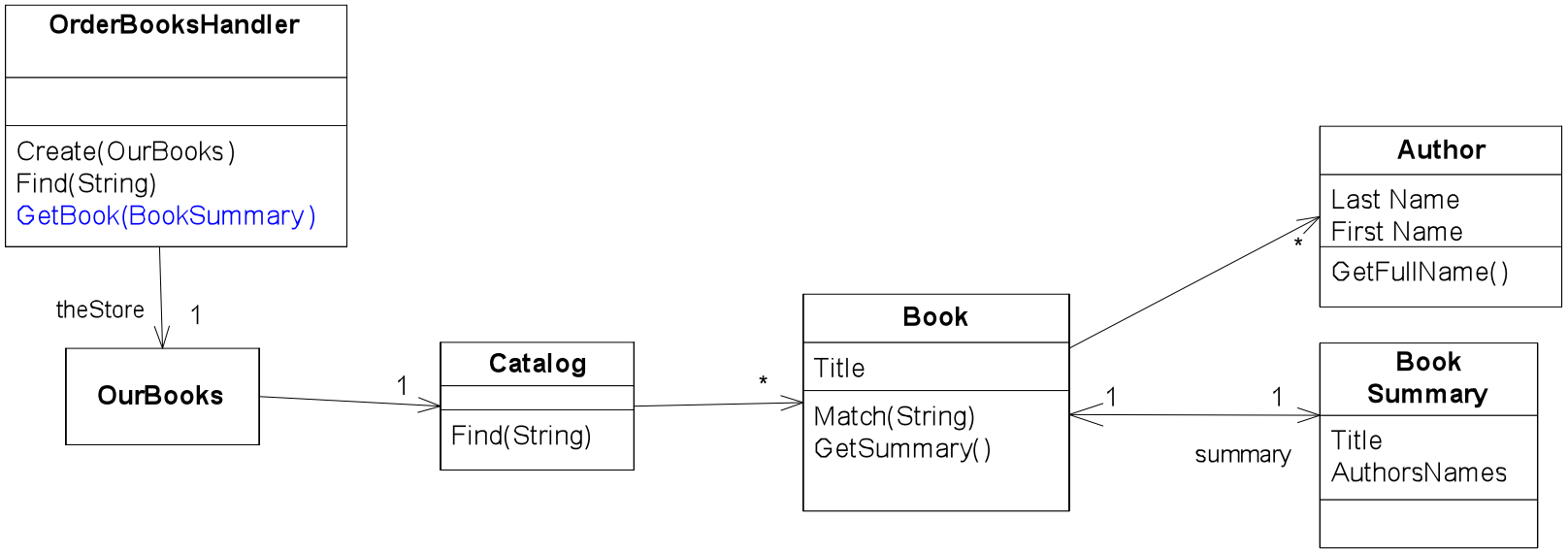
- Note the addition of *OrderBooksHandler*.
- Only add attributes used in design so far.
- Usually don't show `Get<Attribute>`.
- Usually avoid `Set<Attribute>` (focus on responsibilities, not mechanics).

improving
It's what we do. ™

Design of GetBook()

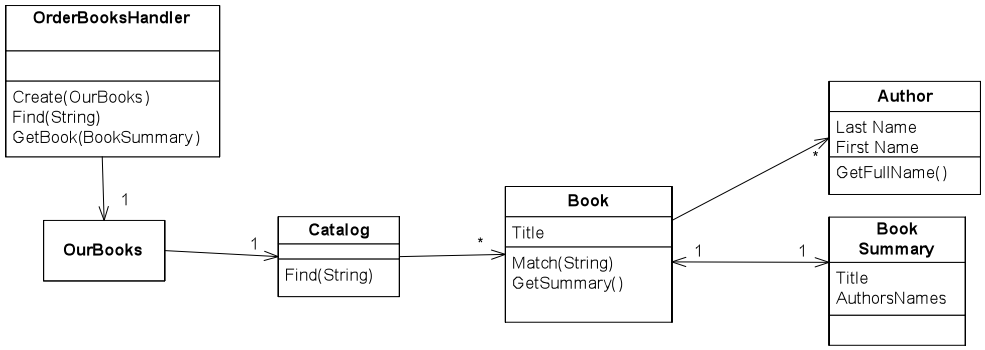


Growing the DCD: From GetBook()



Exercise

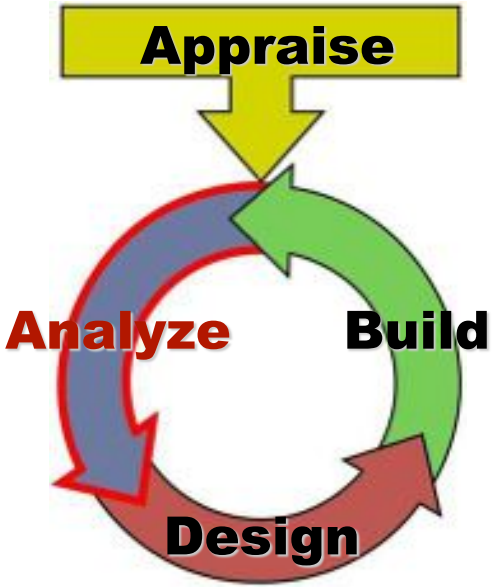
Create the DCD for the design so far.



Exercise

Create a drawing showing how the various activities and diagrams interact in analysis and design.

improving
It's what we do. ™



More on Domain Modeling: Inheritance

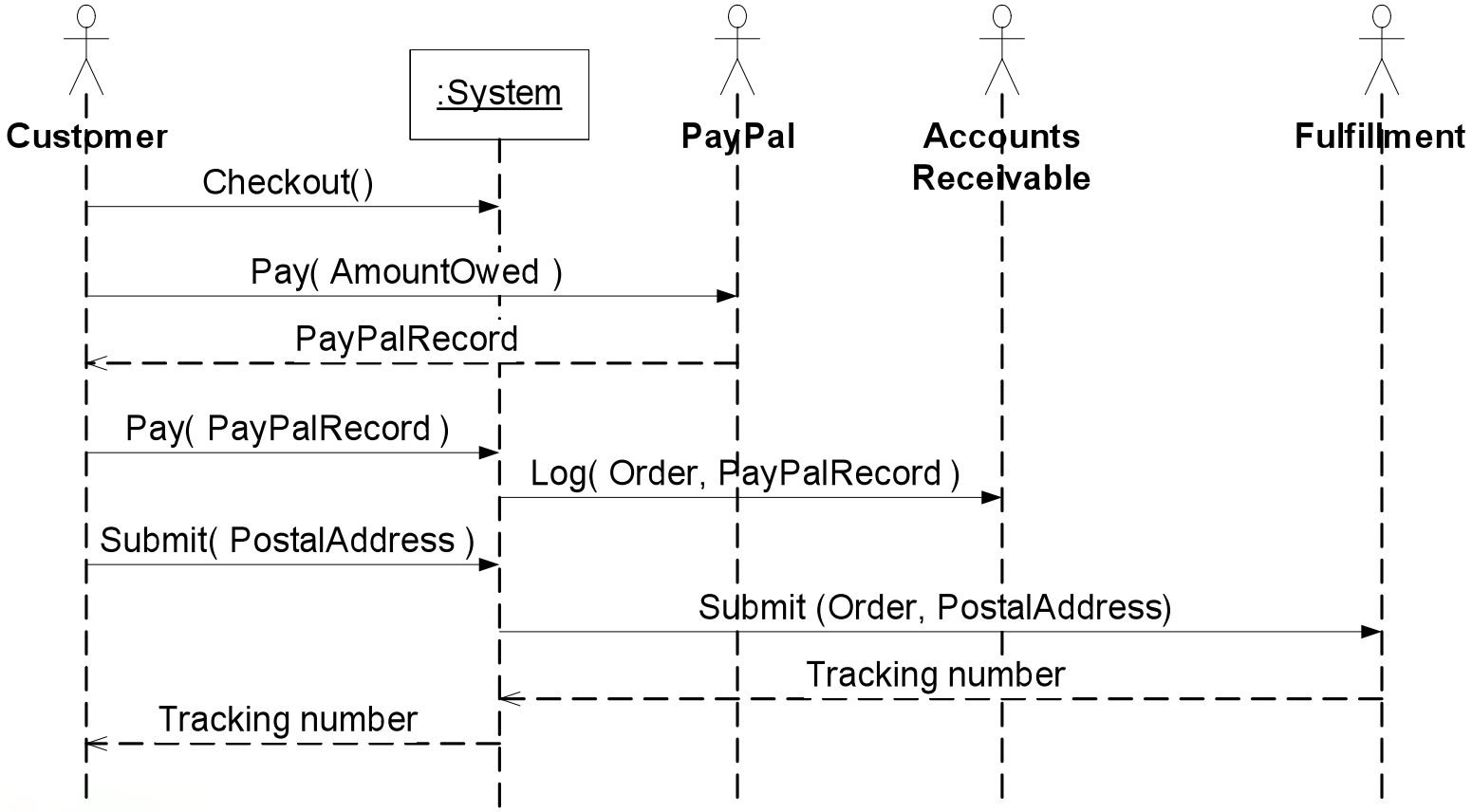
improving
It's what we do. ™

Domain Modeling: Inheritance

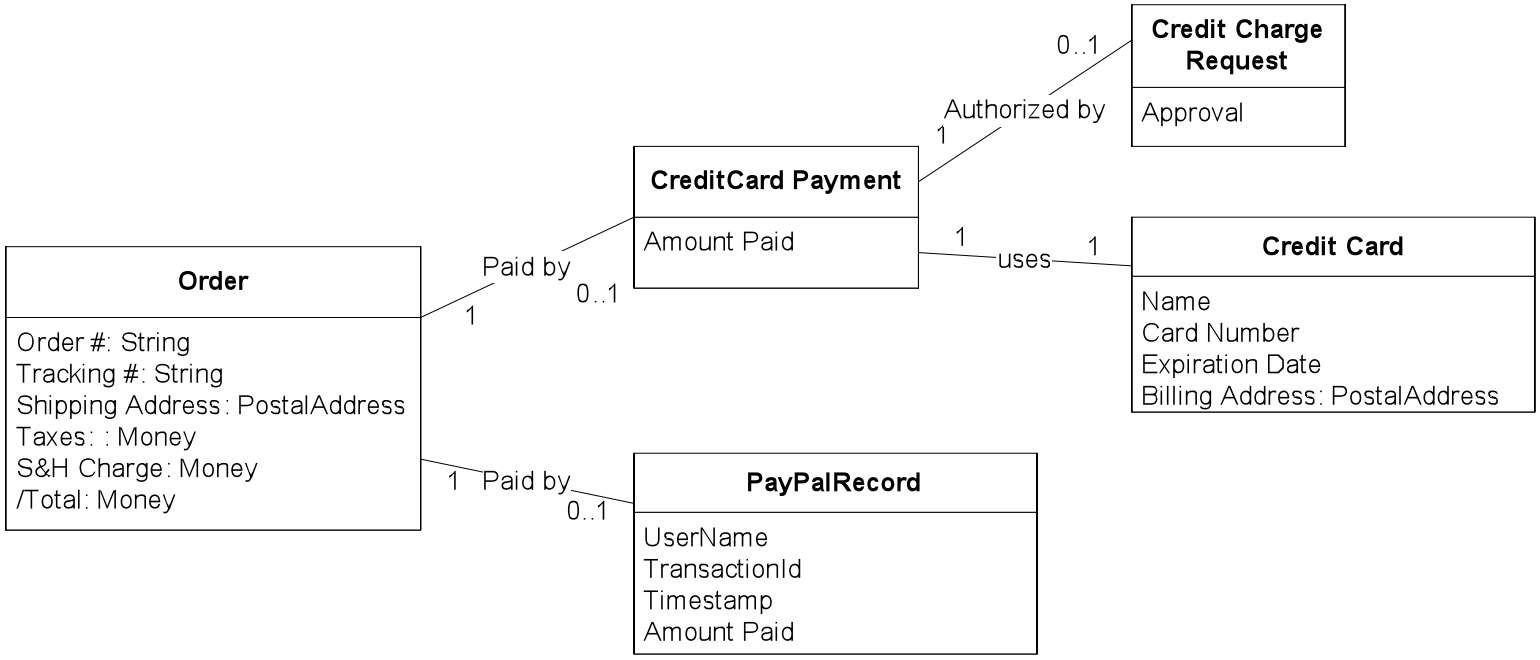
- During Iteration 2, we want to extend the system so that customers can pay using Paypal.
- Updating the use case involves adding the following extension:

  Steps 11-14: The user may also pay by Paypal. Use PayPal to pay, using the Accounts Receivable system.

Domain Modeling: Inheritance



Domain Modeling: Inheritance

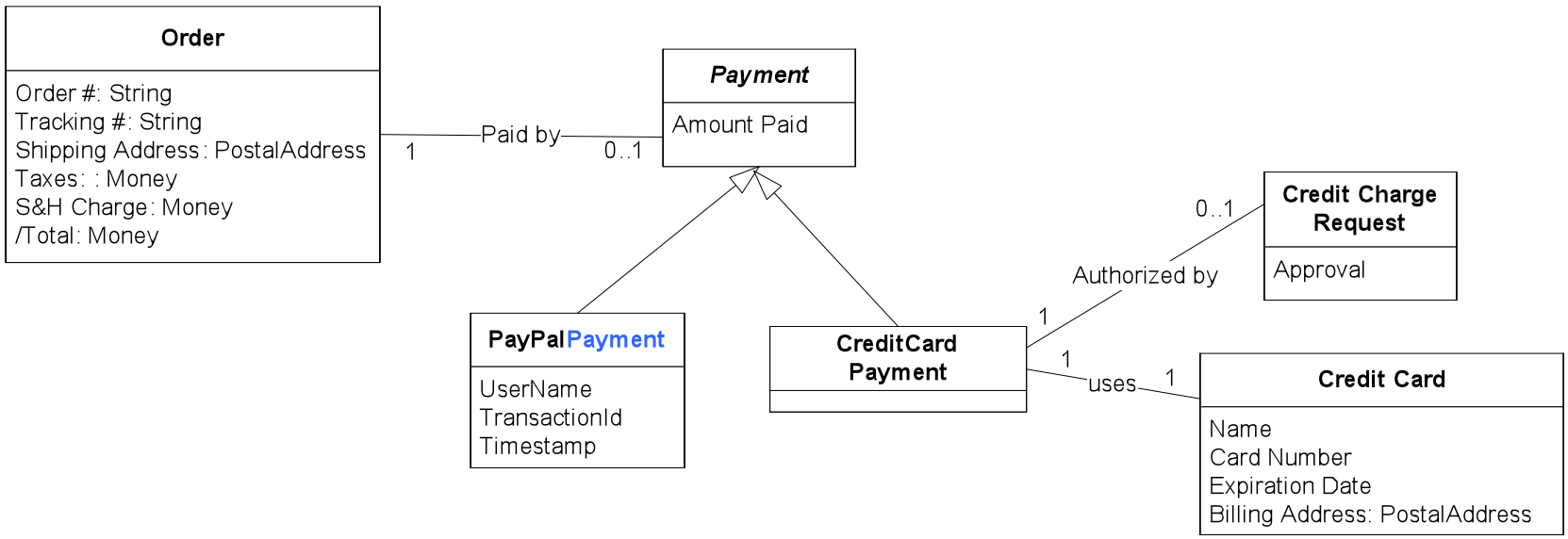


improving
It's what we do. ™

Domain Modeling: Inheritance

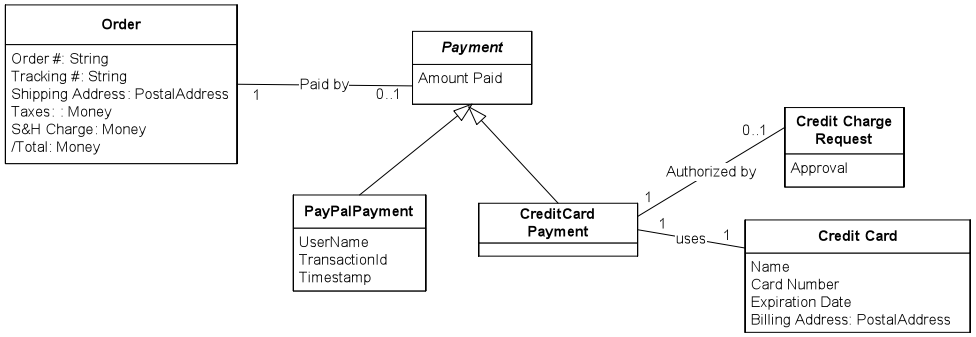
- Generalizations should be *discovered*, not predicted.
  - In analysis, observe common relationships and common attributes.
  - In design, observe shared behavior for which delegation cannot be used to provide the common implementation.
- *100% Rule*: All statements about the generalization are valid for the specialization.
  - Graham Glass, also Liskov Substitution Principle
- Keep Superclasses abstract so they'll be more stable.

Domain Modeling: Inheritance

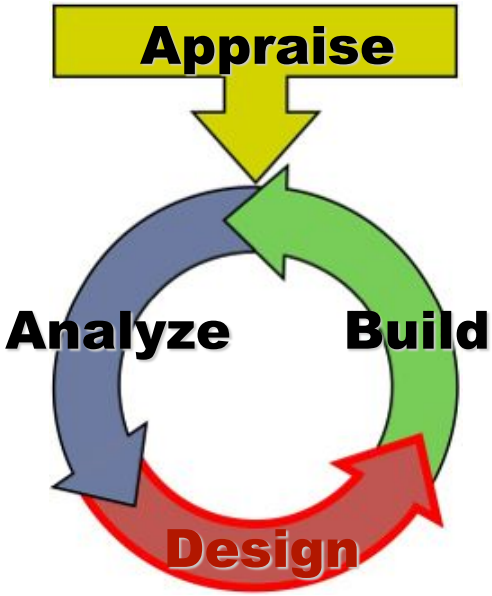


Exercise

Extend the Domain Model of your project to incorporate the requirements in Iteration 2.



improving
It's what we do. ™



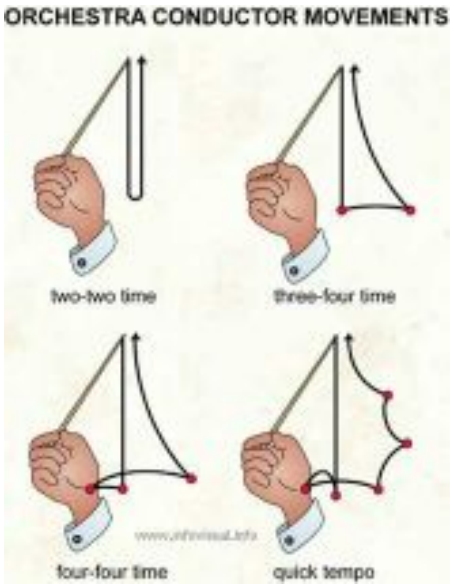
More on Design: Polymorphism

improving
It's what we do. ™

Polymorphism

- Polymorphism is a key object-oriented technology that allows a function name to have multiple implementations.
    - allows the receiver of a message to determine how to respond to it
    - improves Information Hiding
    - reduces code complexity

Polymorphism



One request handled by receiver in different ways.

- the requestor doesn't have to get involved in details
- the receiver does what it best knows how to do

improving
It's what we do. ™

What is wrong with this?

```
public Money getTotalPrice() {
   Money price;

   // calculate prices and tax
   …

   if (shipping.equals("Standard")) {
       price += //calculate standard shipping
   } else if (shipping.equals("Ground")) {
       price += //calculate ground shipping
   } else if (shipping.equals("Priority")) {
       price += //calculate ground shipping
   }
   // FIXME: support Int'l Shipping!

   return price;
}
```

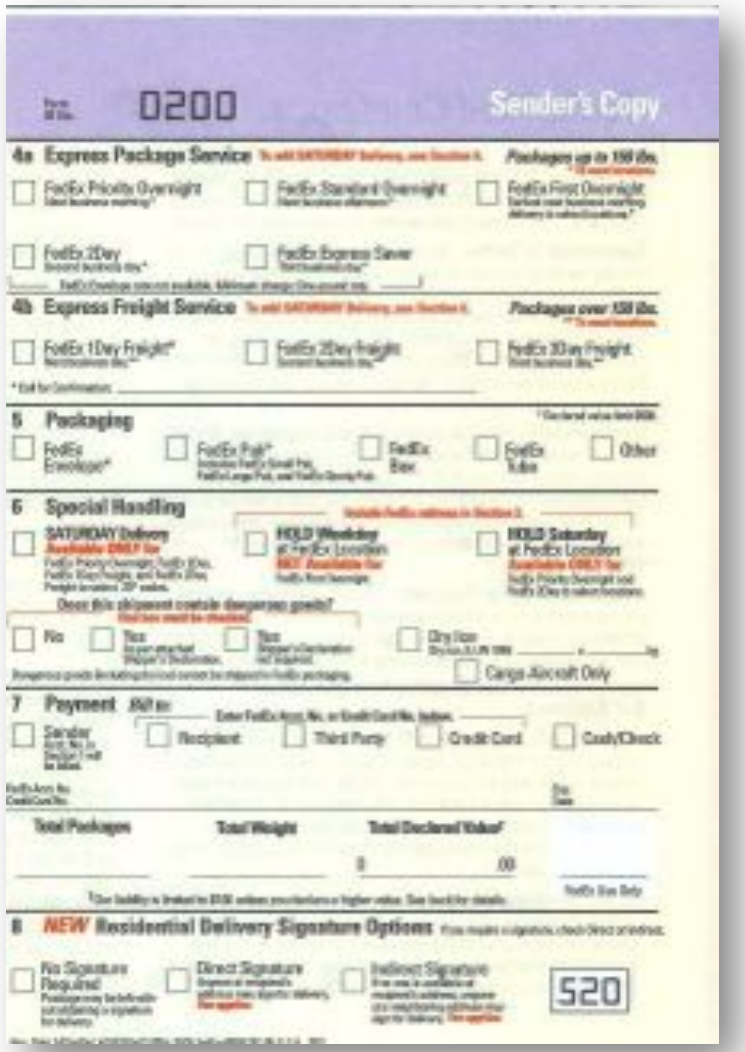
What is wrong with this?

```
public Date getEstimatedDeliveryDate() {
   Date dueDate;

   if (shipping.equals("Standard")) {
        dueDate = //calculate standard shipping
   } else if (shipping.equals("Ground")) {
        dueDate = //calculate ground shipping
   } else if (shipping.equals("Priority")) {
        dueDate = //calculate ground shipping
   }
   // FIXME: support Courier

   return dueDate;
}
```

What now?


0200

Sender's Copy

4a Express Package Service

FedEx Priority Overnight

FedEx Standard Overnight

FedEx First Overnight

FedEx 2Day

FedEx Express Saver

4b Express Freight Service

FedEx 1Day Freight

FedEx 2Day Freight

FedEx 3Day Freight

5 Packaging

FedEx Envelope

FedEx Pak

FedEx Box

FedEx Tube

Other

6 Special Handling

SATURDAY Delivery

HOLD Weekday

HOLD Saturday

No

Yes

Dry Ice

Cargo Aircraft Only

7 Payment

Sender

Recipient

Third Party

Credit Card

Cash/Check

Total Packages

Total Weight

Total Declared Value

8 NEW Residential Delivery Signature Options

No Signature Required

Direct Signature

Indirect Signature

520

improving

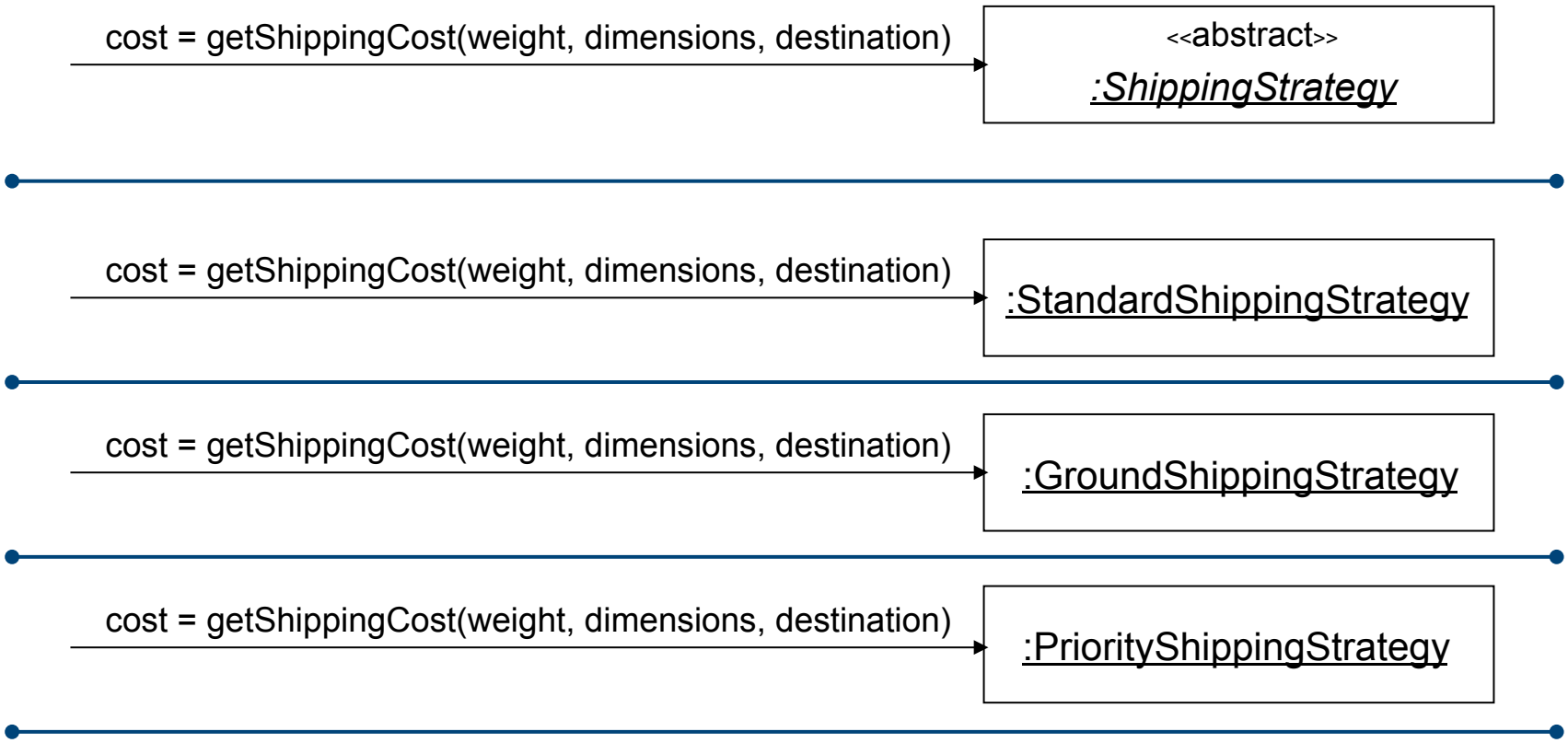
It's what we do. ™

Better?

```
public Money getTotalPrice() {
   Money price;
   // calculate prices and tax
   …
   price += shippingStrategy.getShippingCost(weight,
                                             dimensions,
                                             destination);
   return price;
}

public Date getEstimatedDeliveryDate() {
   Date dueDate =
            shippingStrategy.getEstimatedDeliveryDate();
   return dueDate;
}
```

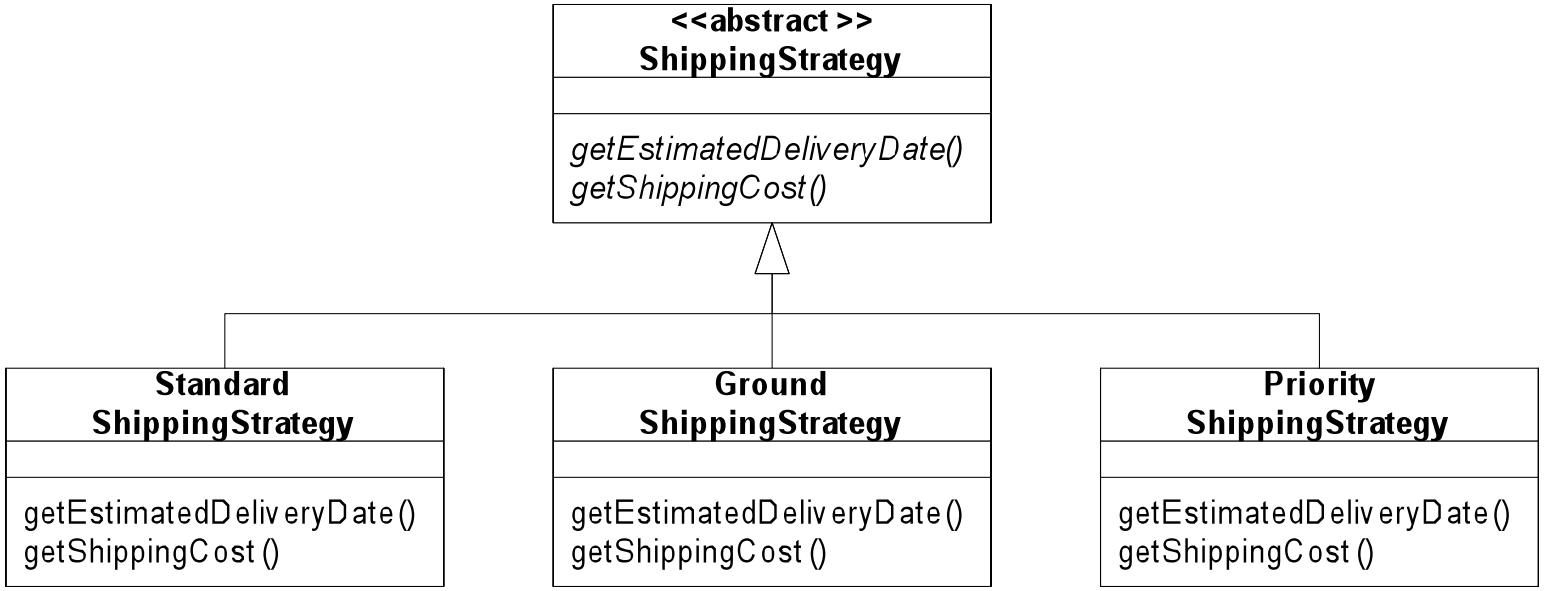
Polymorphism Example



☞ *1 diagram for each different implementation*

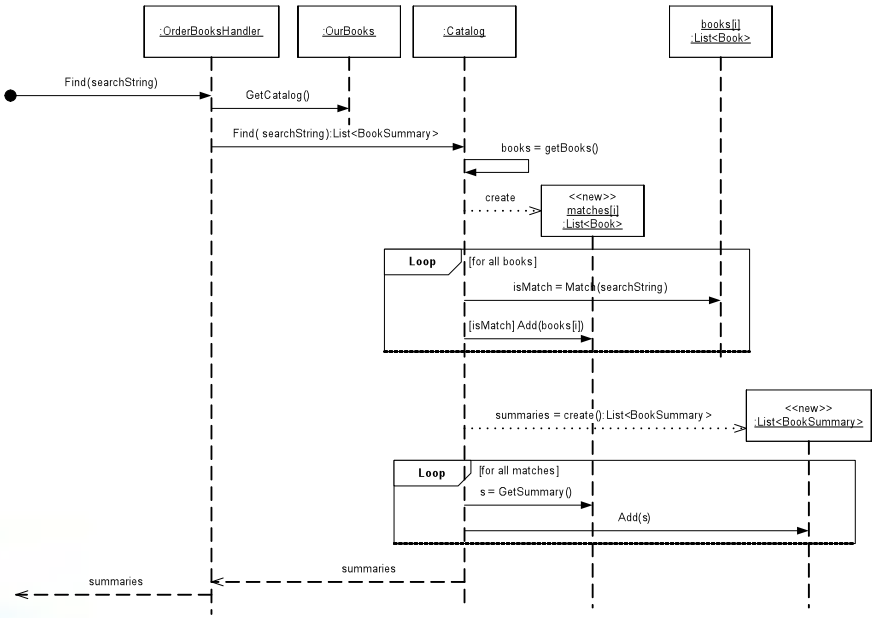
improving
It's what we do. ™

Polymorphism Example

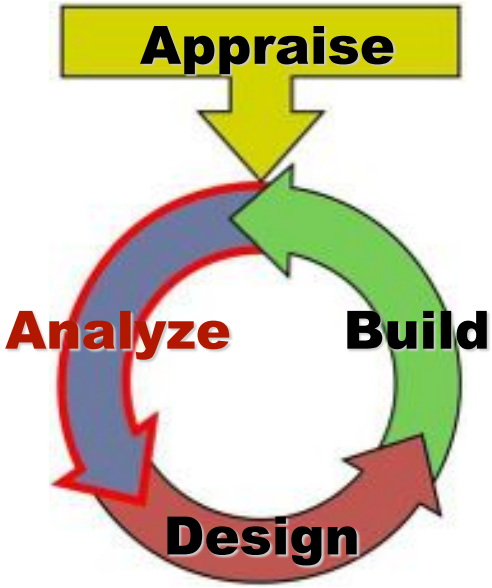


Exercise

Design the next Iteration of the course project.



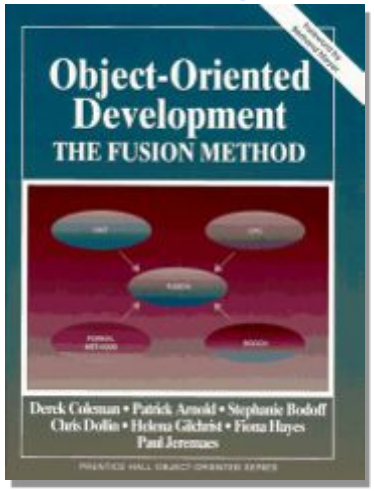
improving
It's what we do. ™



More on Domain Modeling: Associative Types

improving
It's what we do. ™

Associative Types



- During Iteration 1, our tests showed a defect in the system:

  Author names are not ordered properly.

- After some investigation, we learn:

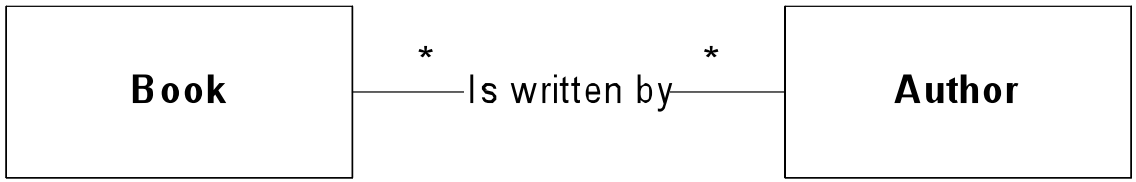
  Book authors are listed in a specific order of priority (often based on the alleged amount of contribution to the text).

  Book "authors" actually play different roles. The same person may be an editor, a contributor, a regular author, etc. Authors are only given one such designation per book. Editors and contributors are marked as such after the name.

- How should we model this additional information?

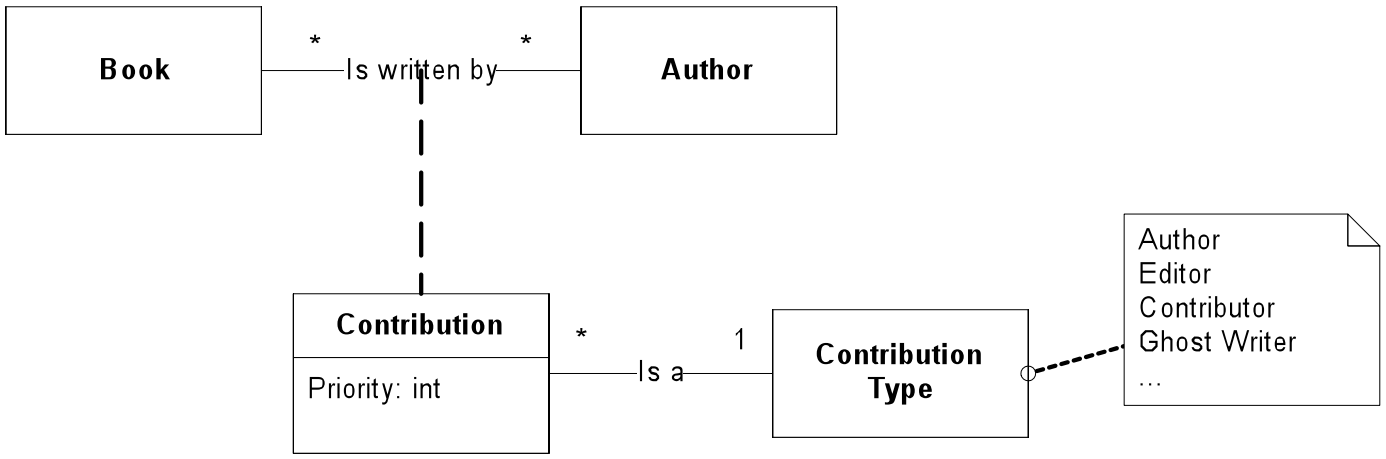
Associative Types

Information is often more closely bound to the relationship between two Concepts than it is to either Concept.



Associative Types

- This leads to the understanding that Relationships can be Concepts too.



Associative Types

- A Standard Example:

  The Instructor will now draw a standard example for associative types.

Exercise

Create a Domain Model from the following domain:

The Specification ("Recipe") for creating an Integrated Circuit is basically a composition of using many different Processes multiple times in different ways.

For example, one Spec may read:

... Step 40: Deposit Silicon-10 Å

... Step 49: Wash-5 minutes

... Step 80: Deposit Silicon-15 Å

while another Spec may read:

... Step 50: Deposit Silicon-2 Å

... Step 53: Etch-7 Å

... Step 59: Wash-4 minutes

Exercise (cont.)

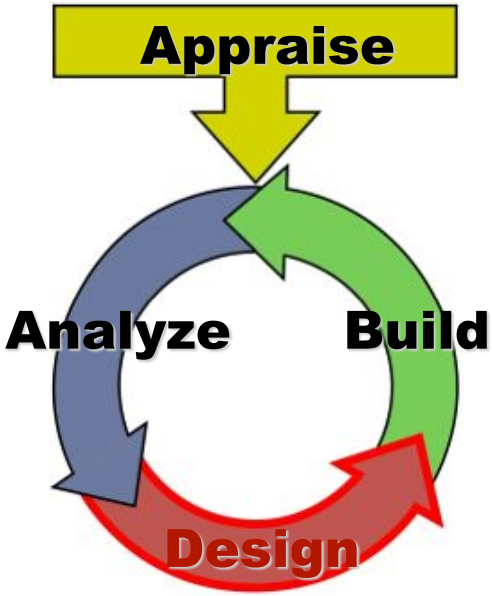
Each Process defines a list of formal parameters whose value is provided when the Process is used.

The Process "Deposit Silicon" requires the SiliconThickness formal parameter.

In addition, the values of the actual parameters may be provided in different Units of Measure.

Each of the following are equivalent:

- Deposit Silicon-32,000 Å
- Deposit Silicon-3.20 μm
- Deposit Silicon-0.000126 inches



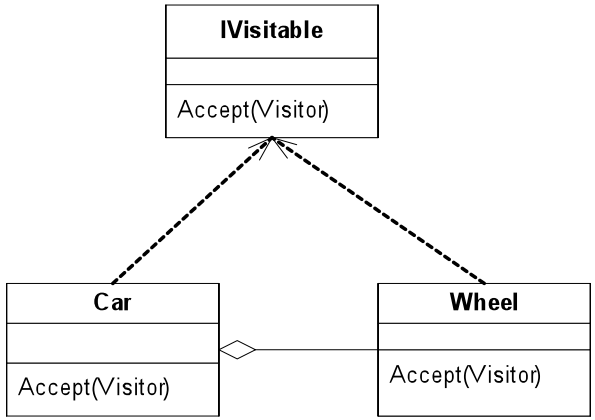
More on Design: GRASP Variations

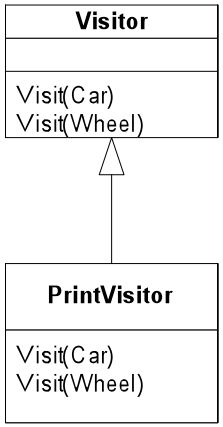
improving
It's what we do. ™

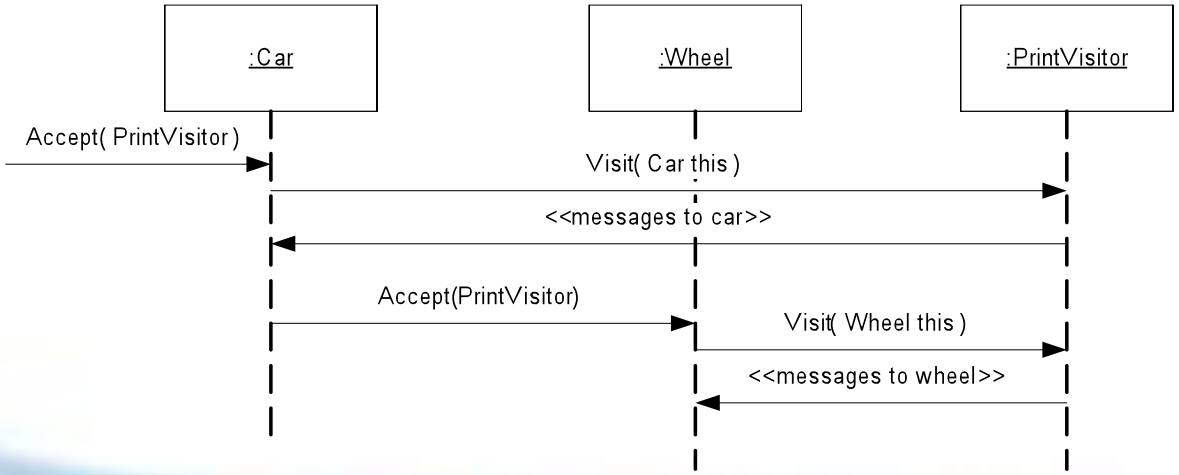
Varieties of Experts

- Information Expert with a Dominant Expert
  - Put the implementation with the Dominant Expert (standard Information Expert).
- Information Expert with 2 equal experts
  - Assign the responsibility to either and split the implementation (Double Dispatch, Collaboration with rich interaction).
- Information Expert with multiple experts
  - Assign the responsibility to a new object created solely to coordinate amongst the experts (Mediator).

Double Dispatch







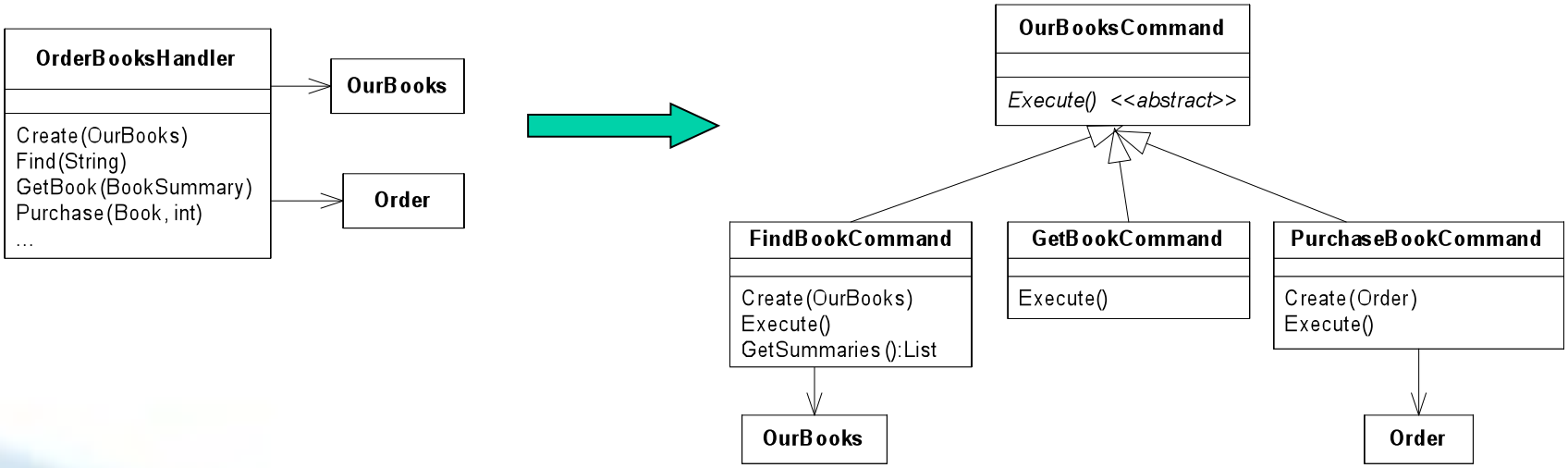
improving
It's what we do. ™

Exercise

- For Bust'Em, we've found out that the fines that apply are magnified for repeat offenses—1x for first offense, 2x for second offense, 3.5x for 3rd offense.
- Our previous design simply had the Law answer the amount owed based on the Violation.
- Re-design the way we get the amount owed (fines and fees) based on our new requirement.

Option to Controller

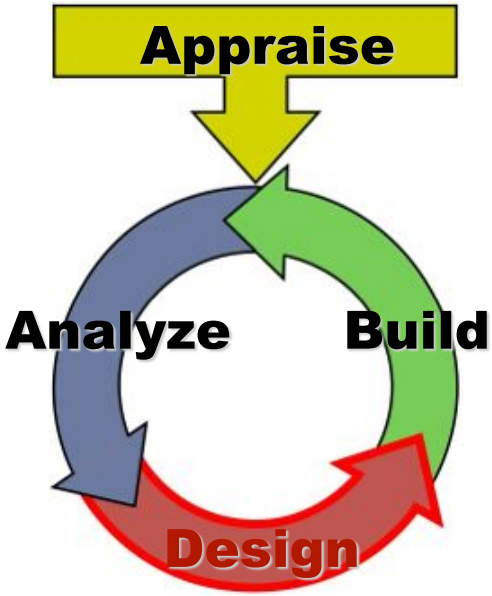
- The Controller typically holds the session state as well as reducing coupling from the client.
- If session state is held within the client, then it is possible to use the *Command* pattern instead of Controller.



improving
It's what we do. ™

Exercise

For the course project, re-design the Controllers to be Commands instead.



More on Design: Indirection

"Any problem in computer science can be solved with another layer of indirection. But that will usually create another problem."

David Wheeler, co-inventor of the subroutine

improving
It's what we do. ™

Indirection

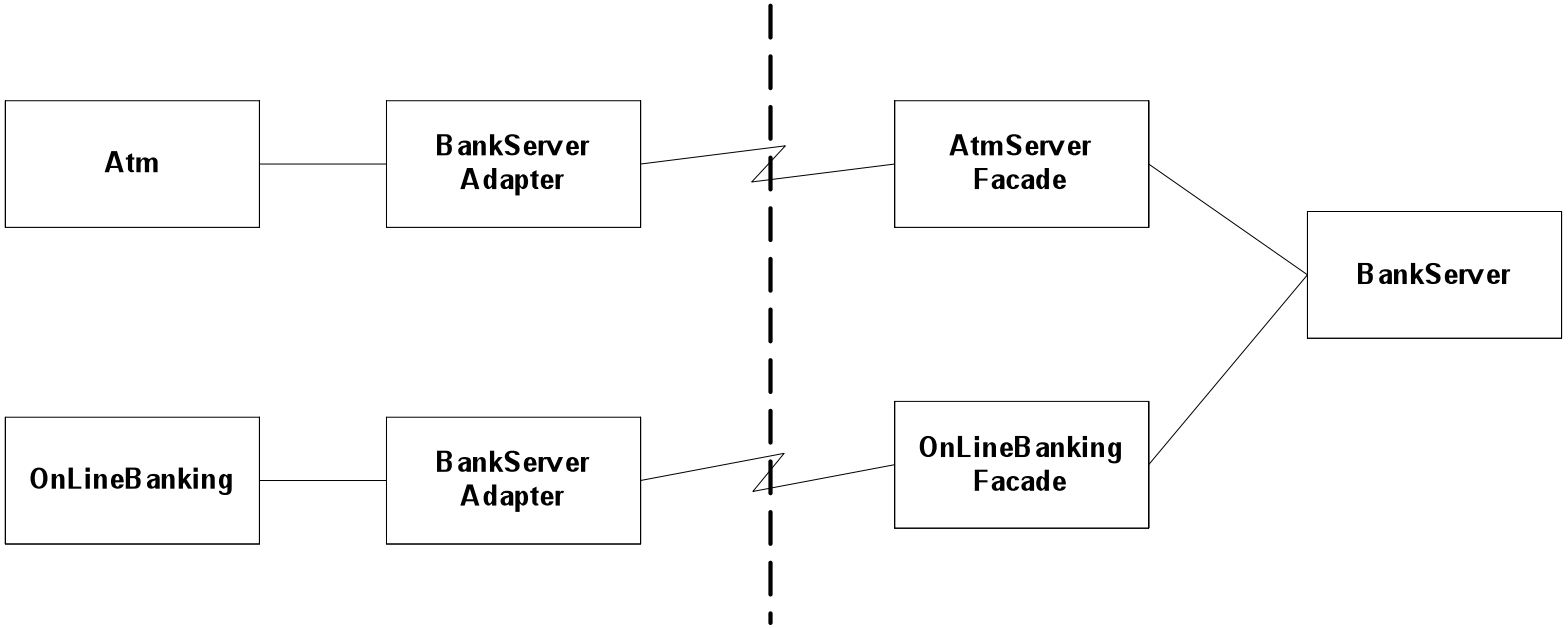
- Solve problems by placing an object in the middle.

  Ask your wife to explain to her mom why you won't come over for dinner the night the game is on.

  Add an air-filled rubber tube around a wheel to smooth the ride and reduce damage to the road (call it: tire).

  Mediator Façade Adapter Proxy Decorator …

Adapter / Façade



Exercise

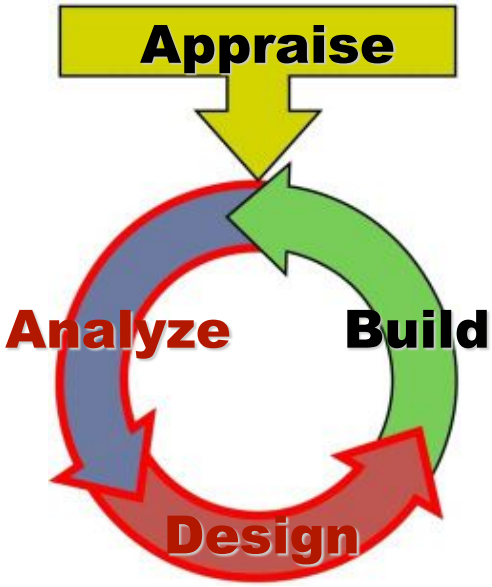
- Develop the software for the system described by the instructor.

improving
It's what we do. ™

SOLID

OO Design Principles, Robert Martin:

- **S**ingle Responsibility/Reason for Change
- **O**pen-Closed Principle
- **L**iskov Substitution Principle
- **I**nterface Segregation Principle
- **D**ependency Inversion Principle

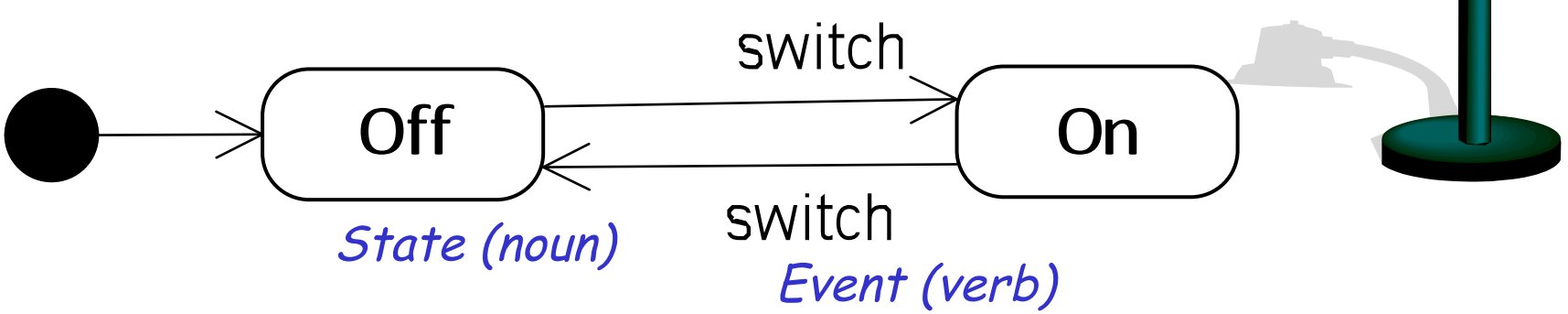


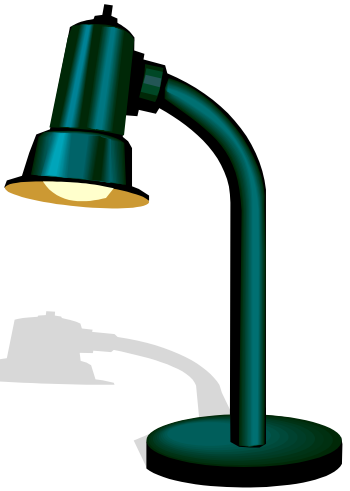
State Machines

State Machine Diagram

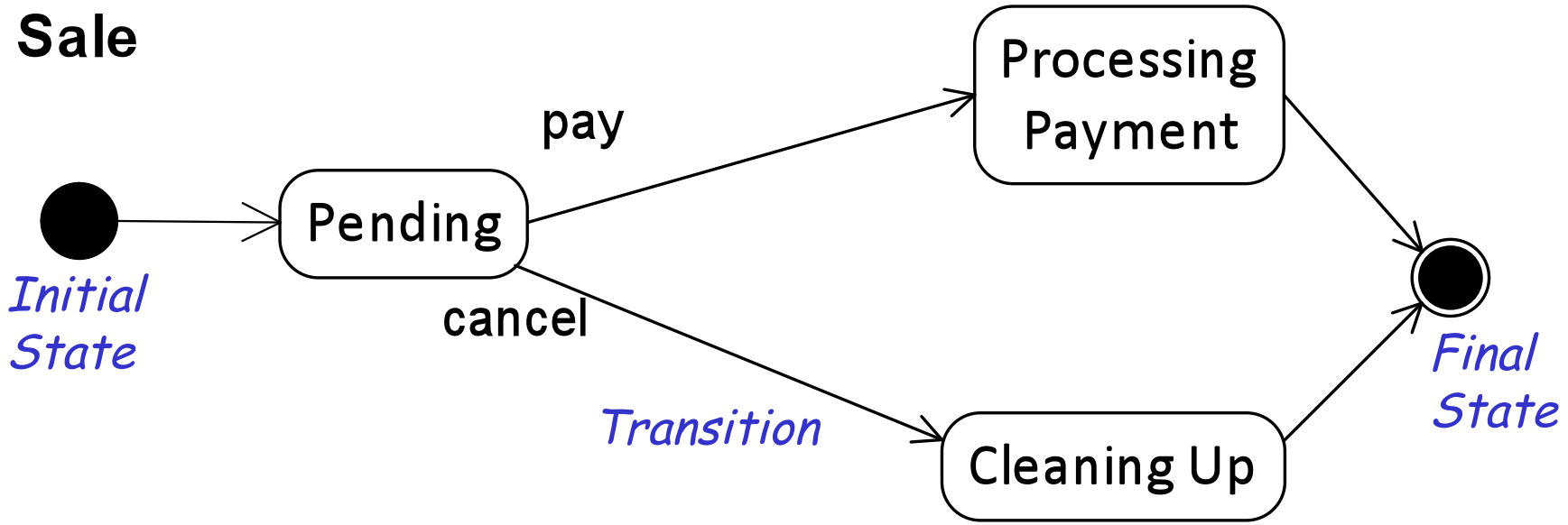
Lamp

Subject





State Machine Diagram



improving
It's what we do. ™

State Machine Diagrams

- Tool to show “dynamic” business rules—those that vary based on specific conditions.
  - ☞ Often overlooked in analysis!
- Adds an additional layer of information about behavior of business entities.

Try One

Essay

Not Started

Drafted

Completed

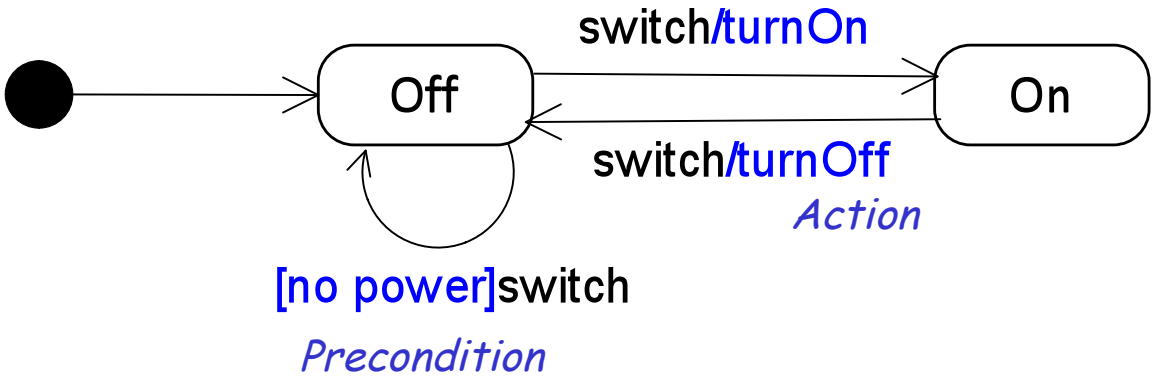
Outlined

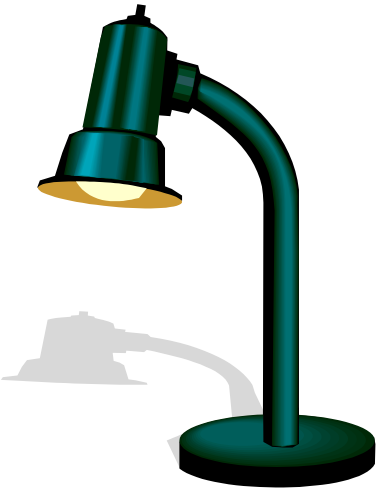
Proofed

improving
It's what we do. ™

Additional Notation

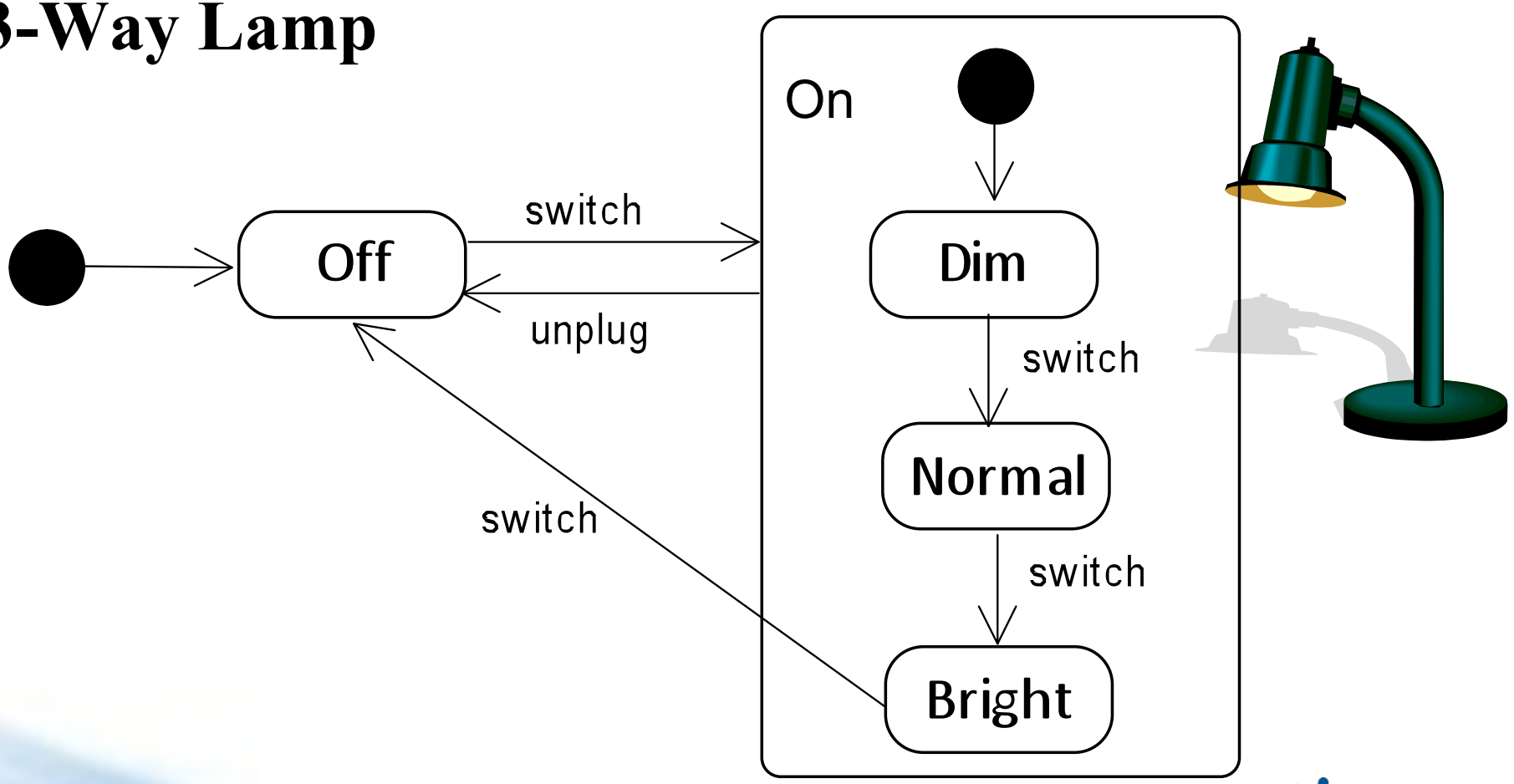
Lamp





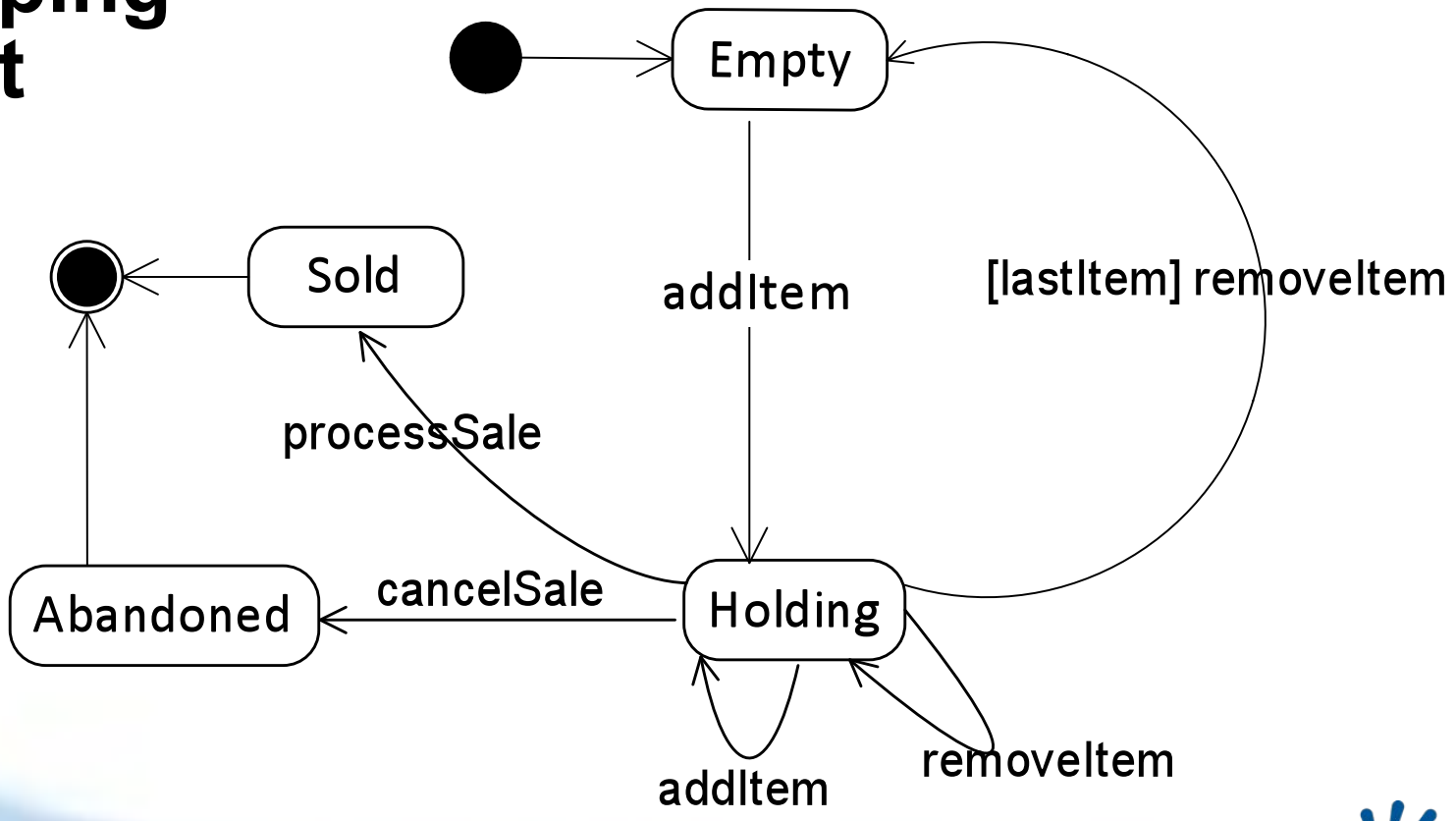
Sub-states

3-Way Lamp



Another Example

Shopping Cart



improving
It's what we do. ™

Exercise

Draw the State Machine for the following system:

Our Document Management system controls which versions of specifications are to be used when developing integrated circuits.

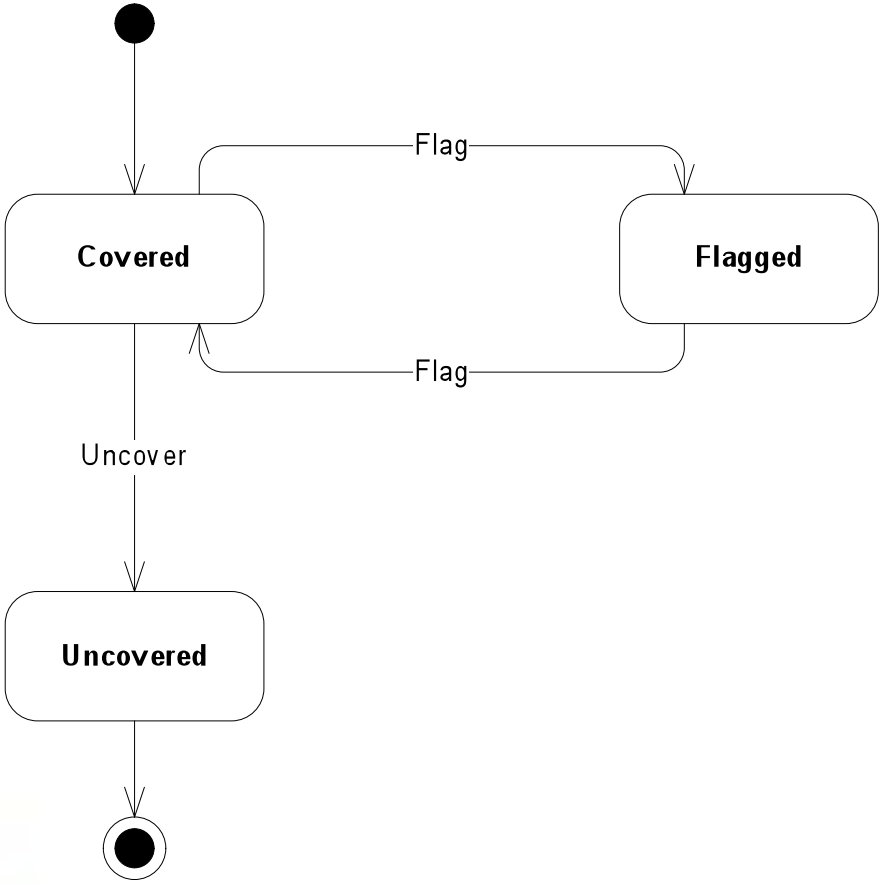
Typically a process or product engineer may spend weeks or months developing a new version of a specification. When asked, they'll say it's "in work".

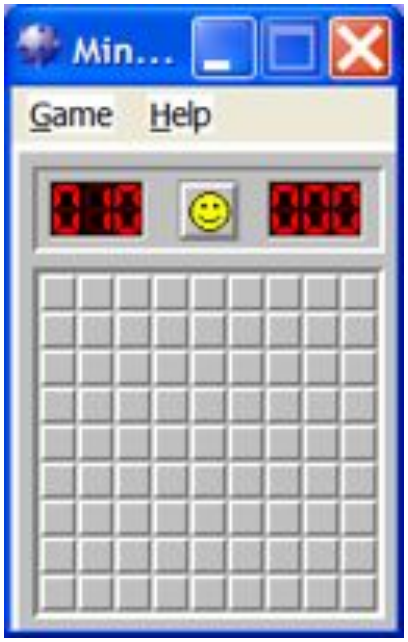
When the new version is ready it must go through the signoff-loop.

The rules are:

- Any specification used in the factory must be approved, meaning that it has completed all the sign-offs required for it.
- Any approved version must be designated as the active version in order to be used (there may be many approved versions of a specification without there being an active version).

Minesweeper





improving
It's what we do. ™

Example—before

```
public class Square {

    private boolean isCovered = true;

    private boolean isFlagged = false;

    public void uncover() {
        if (isCovered && !isFlagged) {
            // uncover the square
        } else {
            // do nothing
        }
    }

    public void flag() {
        if (isCovered && isFlagged) {
            // unflag
        } else if(isCovered && !isFlagged) {
            // flag
        } else {
            // do nothing
        }
    }
}
```

Example—after

```
public class Square {

    private SquareState state = new CoveredState();

    public void uncover() {
        state.uncover(this);
    }

    public void flag() {
        state.flag(this);
    }
}
```

Example—after

```
public class CoveredState extends SquareState {

    public void uncover(Square square) {
       square.reveal();
       square.setState(new UncoveredState() );
    }

     public void flag(Square square) {
        square.setState(new FlaggedState() );
    }
}
```

Example—after

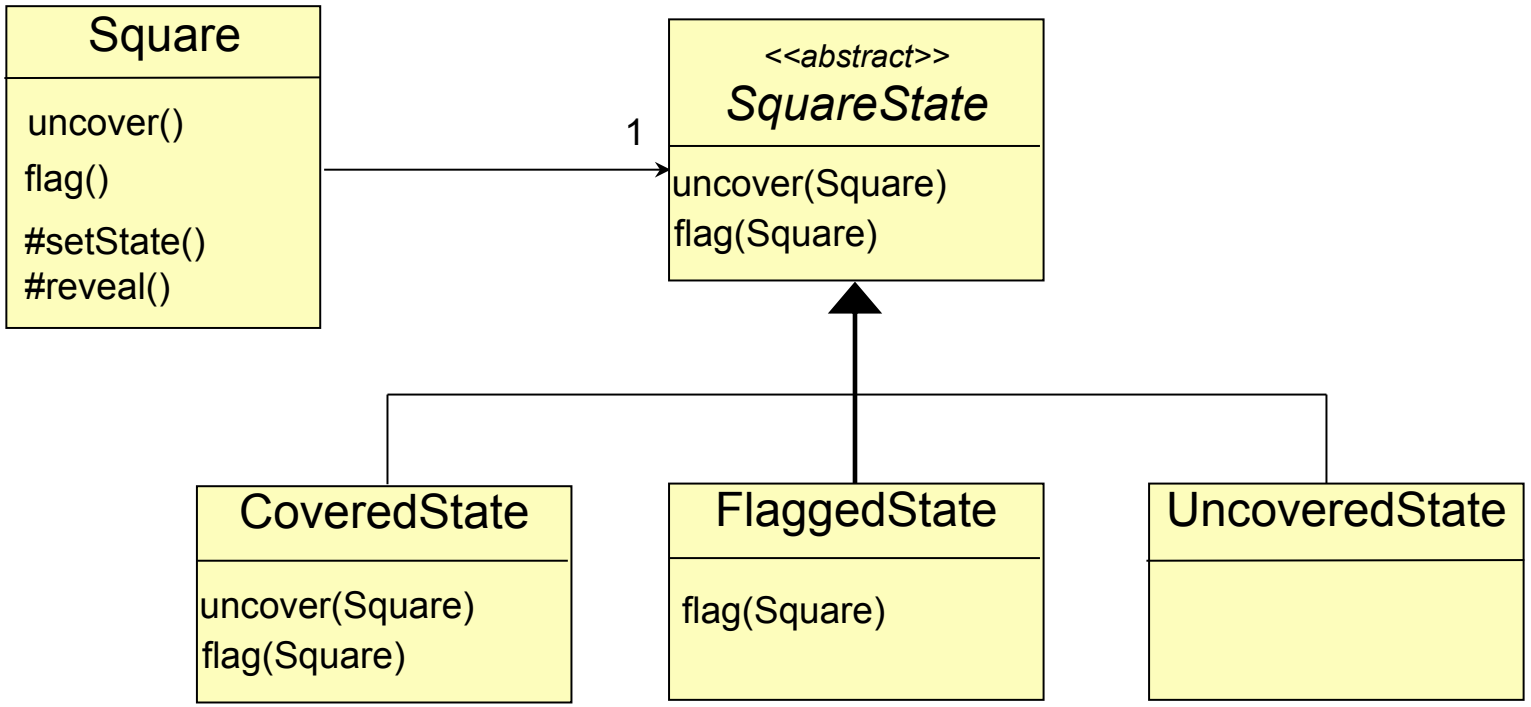
```
public class FlaggedState extends SquareState {

    public void flag(Square square) {
        square.setState(new CoveredState() );
    }
}
```

```
public class UncoveredState extends SquareState {

}
```

State Pattern



Exercise

Using the State Pattern, design the following system:

We use a state machine to control the Order in OurBooks (especially important with enabled Back & Forward buttons).

The Order starts out as "open"—able to have it's contents modified. When the customer is done, and requests Checkout, the order is waiting for payment. Once the payment information is captured, it is processed, and when accepted, the Order is submitted to Fulfillment.

Once a Order is in checkout, any request to modify contents is ignored. A request to "checkout" again is also ignored. Any other events are errors.

Once payment processing has begun, the only valid event is Processing Completed and any other event is an error.

After an Order has been submitted, any event is an error.

References

Object Terminology

- **Object**—a module that provides a public list of responsibilities, typically encapsulating a private set of information. In object-oriented design, everything is an object.
- **Class**—an object primarily responsible for creating objects. It contains the definitions for the responsibilities and the list of information each object has.

More popular pattern languages

- **GoF patterns (Gamma, et. al.)**
  - en.wikipedia.org/wiki/Design_Patterns
- **GRASP (Larman)**
  - en.wikipedia.org/wiki/GRASP_(Object_Oriented_Design)
- **Organizational Patterns of Software Development (Coplien)**
  - www.bell-labs.com/cgi-user/OrgPatterns/OrgPatterns
- **Anti-patterns**
  - www.antipatterns.com
  - www.worsethanfailure.com
- **Refactorings (Fowler)**
  - www.refactoring.com
  - en.wikipedia.org/wiki/Code_smell
- **xUnit Testing (Meszaros)**
  - xunitPatterns.com