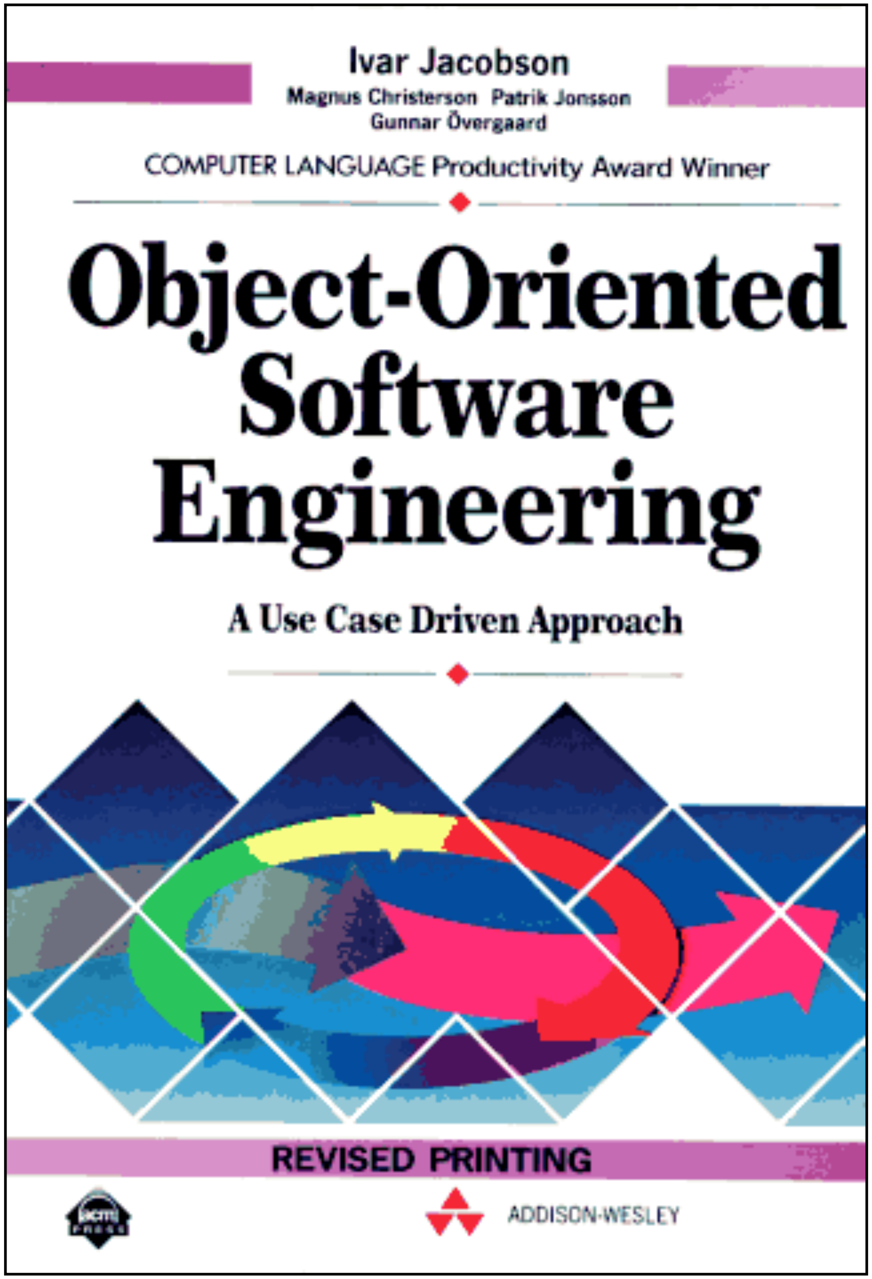
Use-Case Analysis

Robert Keller
Harvey Mudd College

Use-Case Analysis

- What is it?
  - An informal, user-friendly, technique useful for *functional* requirements analysis and specification
- From where did it originate?
  - Ivar Jacobson, a Swedish software engineer at Ericsson, then Rational (now part of IBM), in a *method* called **OOSE** (Object-Oriented Software Engineering). Originally called “Usage cases”.
- Now “part of” UML (Unified Modeling Language), an informal collection of development techniques.



Ivar Jacobson

Definition of "Use Case"

"The specification of sequences of actions that a system, subsystem, or class can perform by interacting with outside actors"

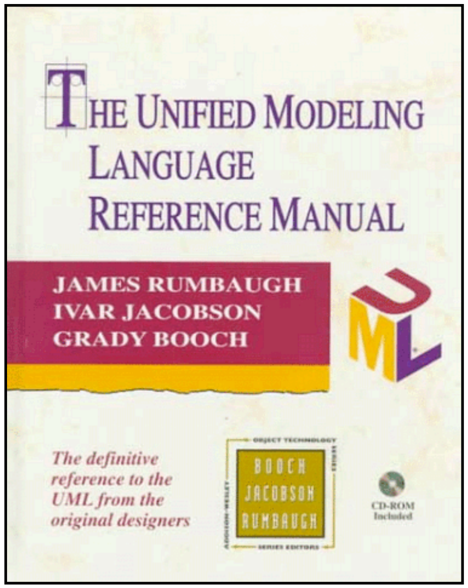
(*UML Reference Manual*, Rumbaugh, Jacobson, and Booch).

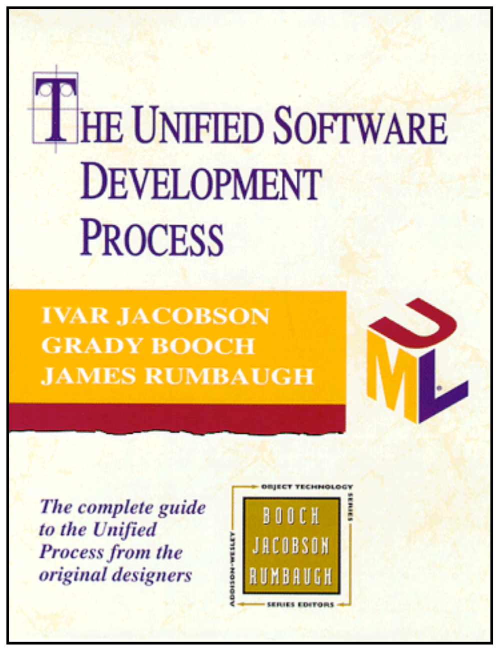
Purpose of a "Use Case"

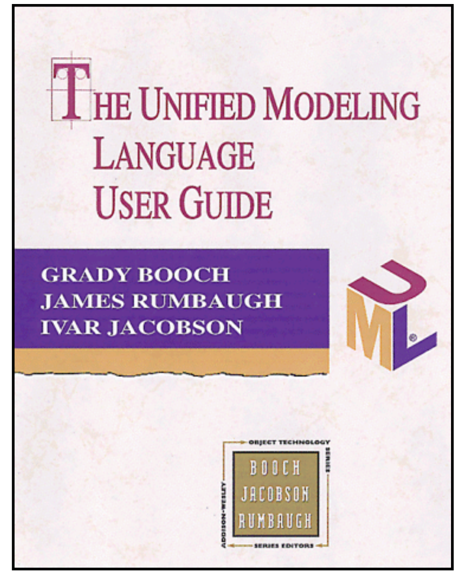
- "to define a piece of behavior of a [system or subsystem or class] ***without revealing the internal structure*** of the [system]

  (UML Reference Manual, Rumbaugh, Jacobson, and Booch).

UML References







Importance of Use Cases

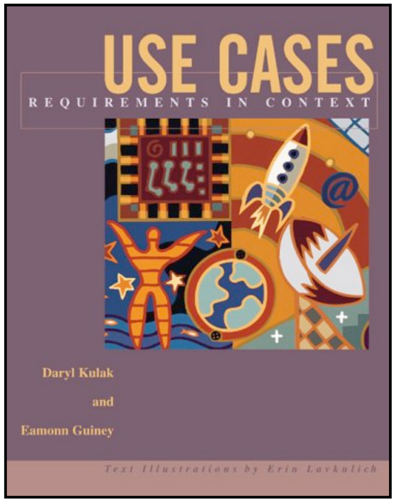
- At least one popular methodology (the Rational Unified Process, based in part on Ivar Jacobson's earlier OOSE) is said to be ***Use-Case Driven***,
- meaning that most development activities are ***traceable*** back to the use cases as defined in agreement with the user or customer.

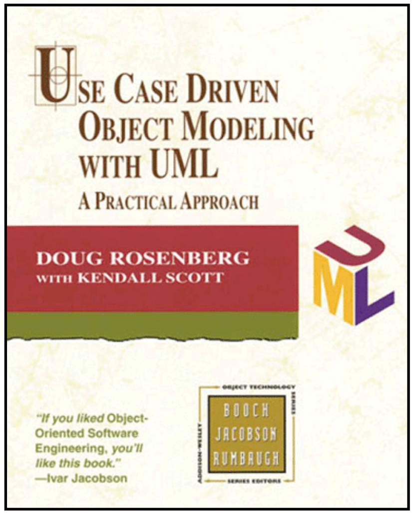
Nonetheless

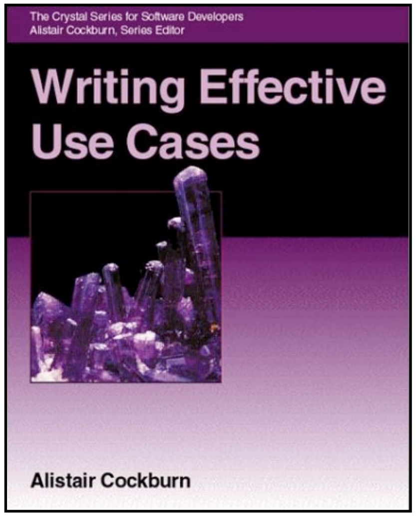
- Use cases alone do not constitute a *complete* SRS.
- For example, use cases focus on the **functional requirements** exclusively.
- Also, their language is **more specific and detailed** than requirements normally would be.

Use-Case References

recommended:







Other Implications

- Use cases could be used for *other* types of design, and system analysis, **not just software**.

- Once you know about them, it is hard to imagine an engineering project or business process of almost any kind starting without them [although in agile development, "**user stories**" are an accepted alternative.]

Characteristics of Use-Case Analysis

- **Use-cases**: The specific ways in which the system is used.
- Each use-case expresses a "compete thought" or end-to-end transaction.
- A "black-box" specification; does not deal with internal structure.

Some Key **Components** of Use-Case Analysis

- **Actors**: Entities that communicate with the system; *typically* people, but could *also* be other *systems* or *devices* as long as they are outside the system being specified.
- **Scenarios**: A sequence of steps taken in the use case.
- **Relationships** between Actors or between Use-Cases

Actors

- Actors are characterized **not by the identity** of the user or other external entity, but rather by the ***role*** played by the actor.
- One person can be several different actors in different roles.
- One actor can be played (at different times) by several different persons.
- An entire **committee** could be **one** actor.
- An actor need not be a living thing; it could be another subsystem.

More on Actors

- Actors are **not part of the system** in question; they supply input to and receive output from, the system.

- In other words, the set of actors collectively defines the **environment** of the system.

- This does not preclude the possibility of an object in the system design **standing for** an actor. Design is a separate issue.

Minimum Requirement for a Use Case

- Verbal description of scenario(s)

Common Components of a Use Case

- Name
- Symbolic label
- List of actors
- Initiator (one of the actors)
- Verbal description

Initiator

- The initiator of a use case is the actor that starts the flow of events.

Brief Use-Case Description

label for this use-case

name of this use-case

- OCI: Order from catalog
  - **Actors**: customer, sales rep., shipping dept.
  - **Initiator**: customer
  - **Scenario**:

Scenario

- An enumerated list of events, e.g.:
  1. Customer calls to order from catalog.
  2. Sales representative identifies item numbers.
  3. Sales representative verifies stock.
  4. Sales representative confirms order.
  5. Sales representative gives order number to Customer.
  6. Sales representative passes order to Shipping.

Scenario descriptions could contain *iteration*

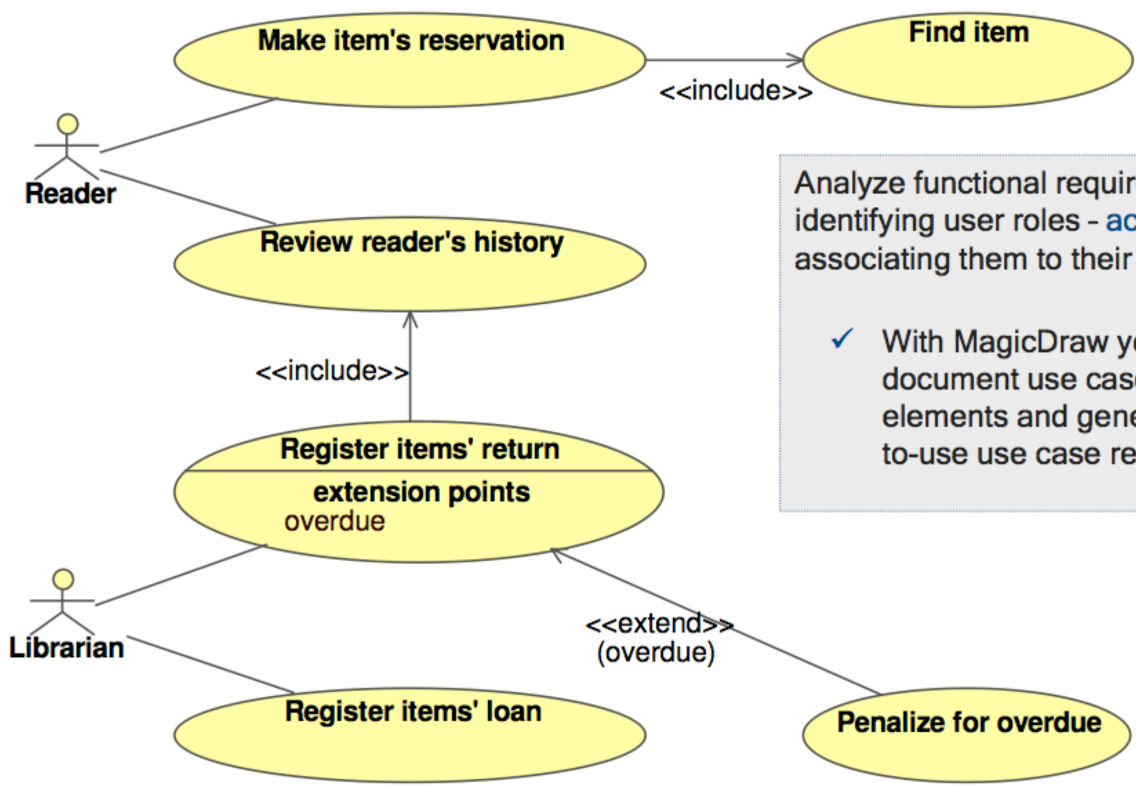
- An order could contain **multiple items**. In this case, the event flow for steps 2-4 should show something like:
  - ***For each*** item to be ordered:
    - Sales representative checks catalog number.
    - Sales representative verifies stock.
    - Sales representative records item.
- Similarly, flow of events could contain **conditional** (if-then-else) behaviors.
- Still, the scenarios are **not** the program.

Use Case Diagrams

- For *visualization* of use case interactions; diagrams are not the use cases themselves.
- Don't tell the whole story
- Useful in brainstorming and documentation
- Used in software tools, such as:
  - Rational Rose
  - iLogix Rhapsody
  - MagicDraw

Example from magicdraw

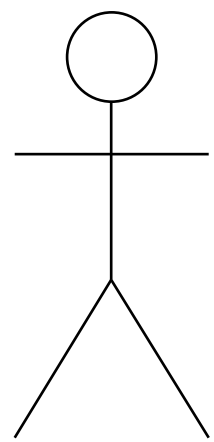
Use Case Model



Analyze functional requirements by identifying user roles - actors - and associating them to their use cases.

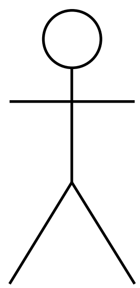
- ✓ With MagicDraw you can document use case model elements and generate ready-to-use use case report.

Icon for an Actor

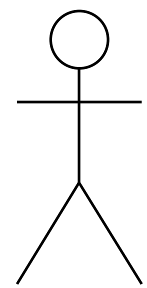


Note: Actors are typically drawn in this “anthropomorphic” way even when the actors aren’t people.

Examples of Actors

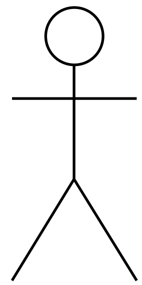


Customer



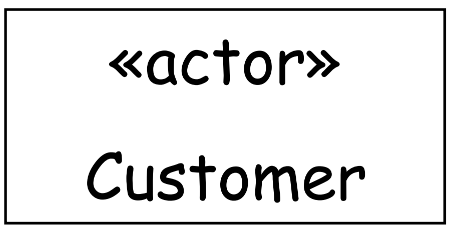
Shipping Dept.

Alternate Actor Icons in UML

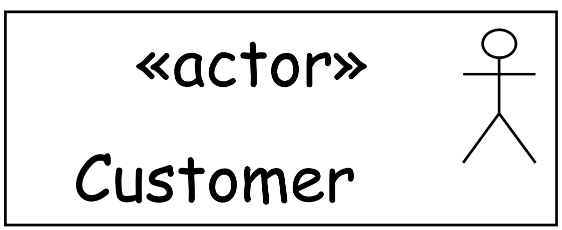


Customer

Visual Icon



Textual Stereotyping

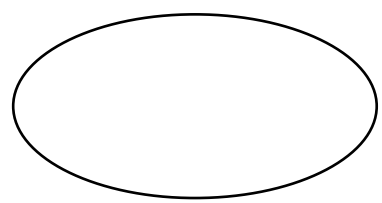


Textual & Visual Stereotyping

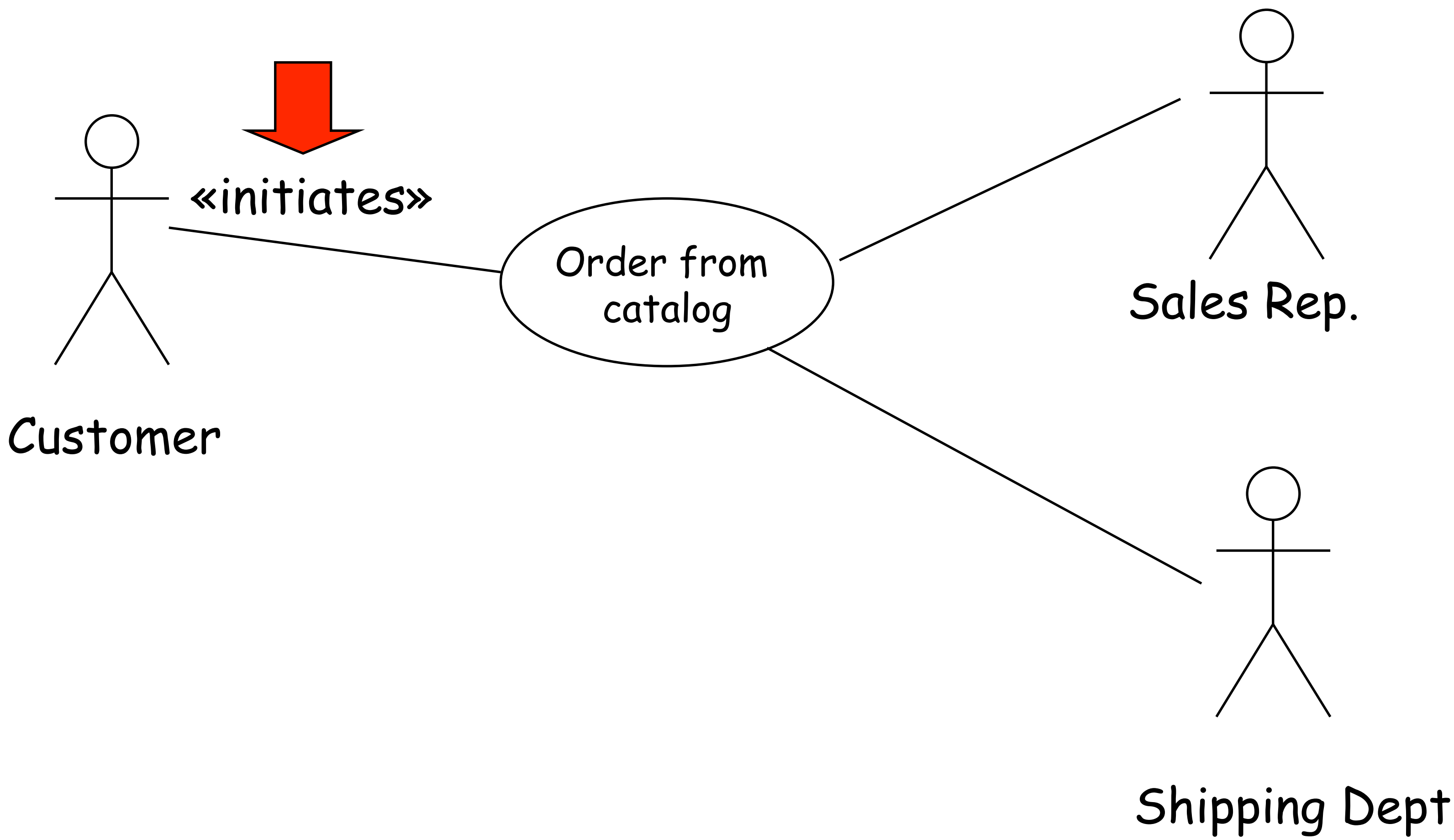
UML Use of "Stereotypes"

- The « … » notation called "guillemets", (used for quoted speech in French, Italian, Spanish (in Spain), etc.). [Unicode U+00AB and U+00BB].
- In UML these indicate the name of a "stereotype", defined as an ***informal extension*** of basic UML concepts.

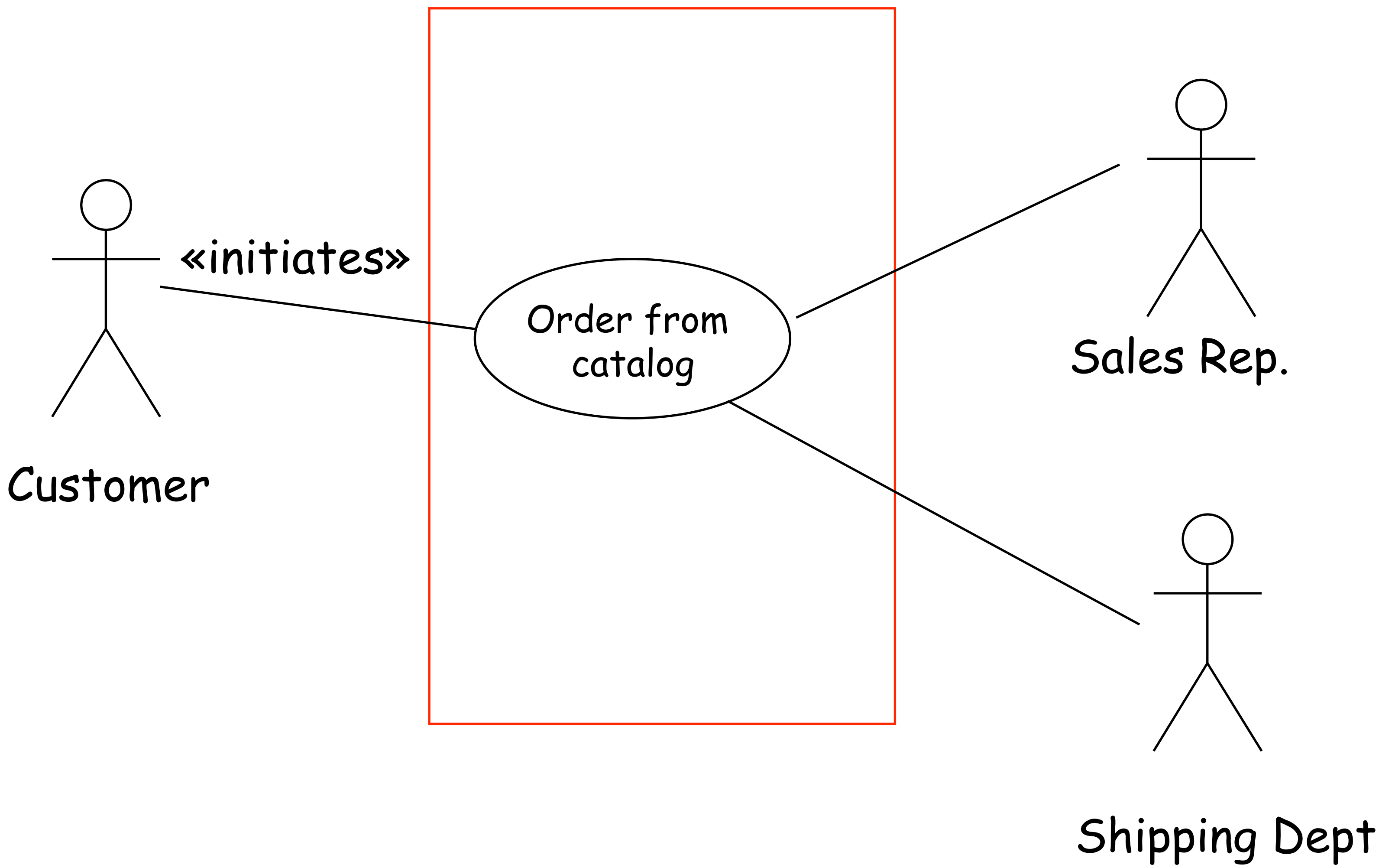
Icon for a Use Case



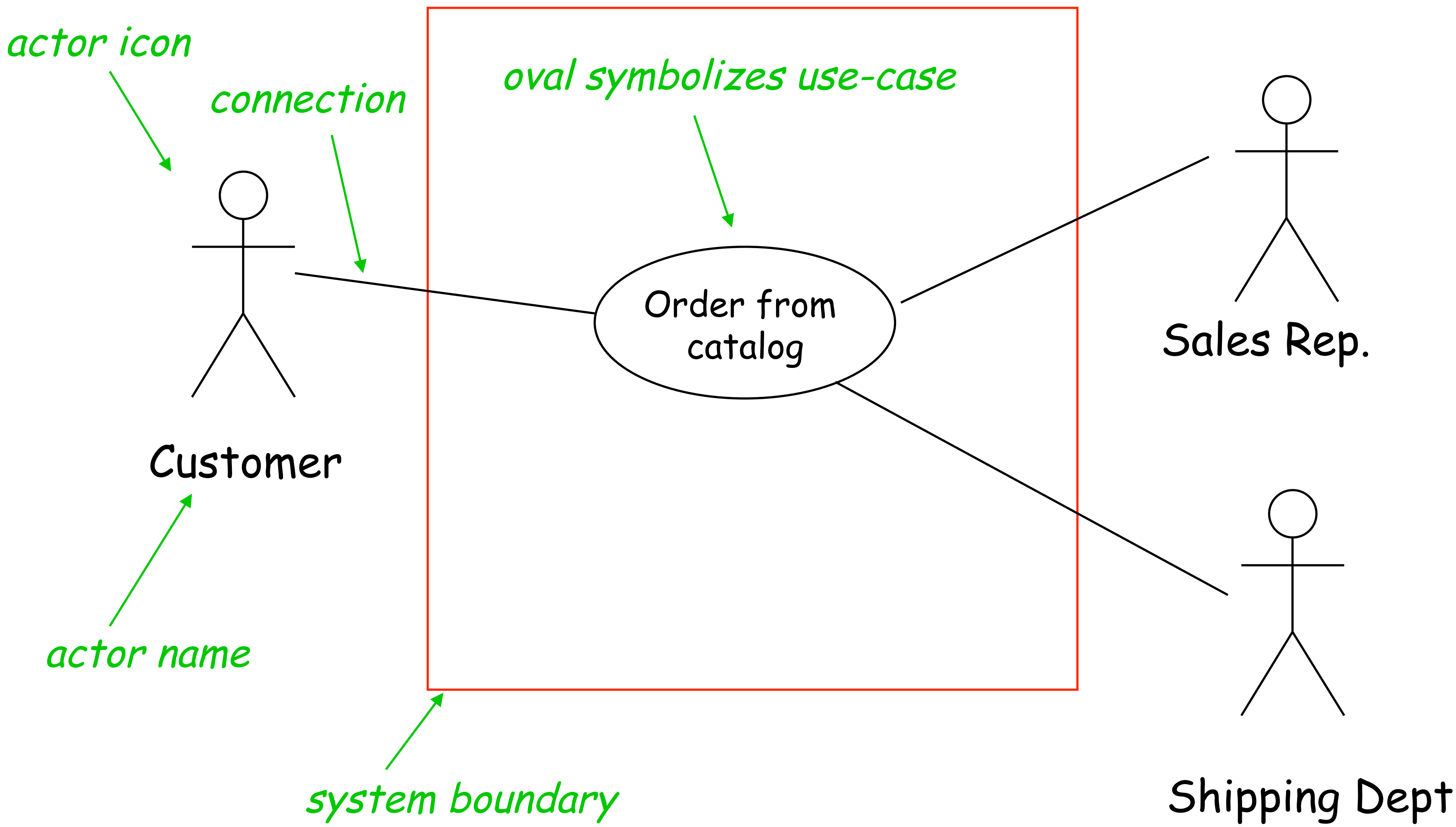
Noting the Initiator



Noting the System Boundary



Symbology for a simple use-case



Class Exercise

- Identify several distinct other possible use-cases in the catalog-order enterprise.
- For one use case, indicate the actors, initiator, and flow of events.

Steps in Use-Case Analysis

- Identify system boundaries
- Identify actors:
  - Recall: an actor is an entity playing a particular role with respect to the system.

Steps in Use-Case Analysis (cont'd)

- Identify use cases themselves:
  - Every use case has at least one actor.
  - A specific actor initiates the use case.
  - The same actor may participate in multiple use cases, as initiator in some and not in others.
- Create the description, including scenarios
- Provide additional information (see later)

Scenarios of a Use Case

- A "scenario" is a *single* path through the event flow. For example, if there is a conditional part, only one branch is taken in the scenario.
- Obviously we can't always **enumerate** all the scenarios; there might be an infinite set of them. If the use case involves iteration, only a finite number of iterations are used in the scenario.

Scenarios (continued)

- Often there will be a “principal” scenario, and several secondary **variations**.

- Some variations may end with **exceptions**, others with normal completion.

A Catalog Order Scenario (1 of 3)

(Alice is a typical customer, Bert a sales clerk.)

- Alice calls company.
- Bert answers the telephone.
- Alice indicates she wishes to place an order.
- Bert asks how the order will be paid.
- Alice indicates via credit card.
- Bert asks for the card number,billing address, and expiration date.
- Alice provides the above info.

A Catalog Order Scenario (2 of 3)

- Bert asks for the first item.
- Alice responds with first item.
- Bert asks for quantity of first item.
- Alice responds with quantity of first item.
- Bert records first item with quantity.
- Bert asks for second item.
- Alice responds with second item.
- Bert indicates second item out of stock; does Alice wish it to be back ordered?
- Alice declines to order item.

A Catalog Order Scenario (3 of 3)

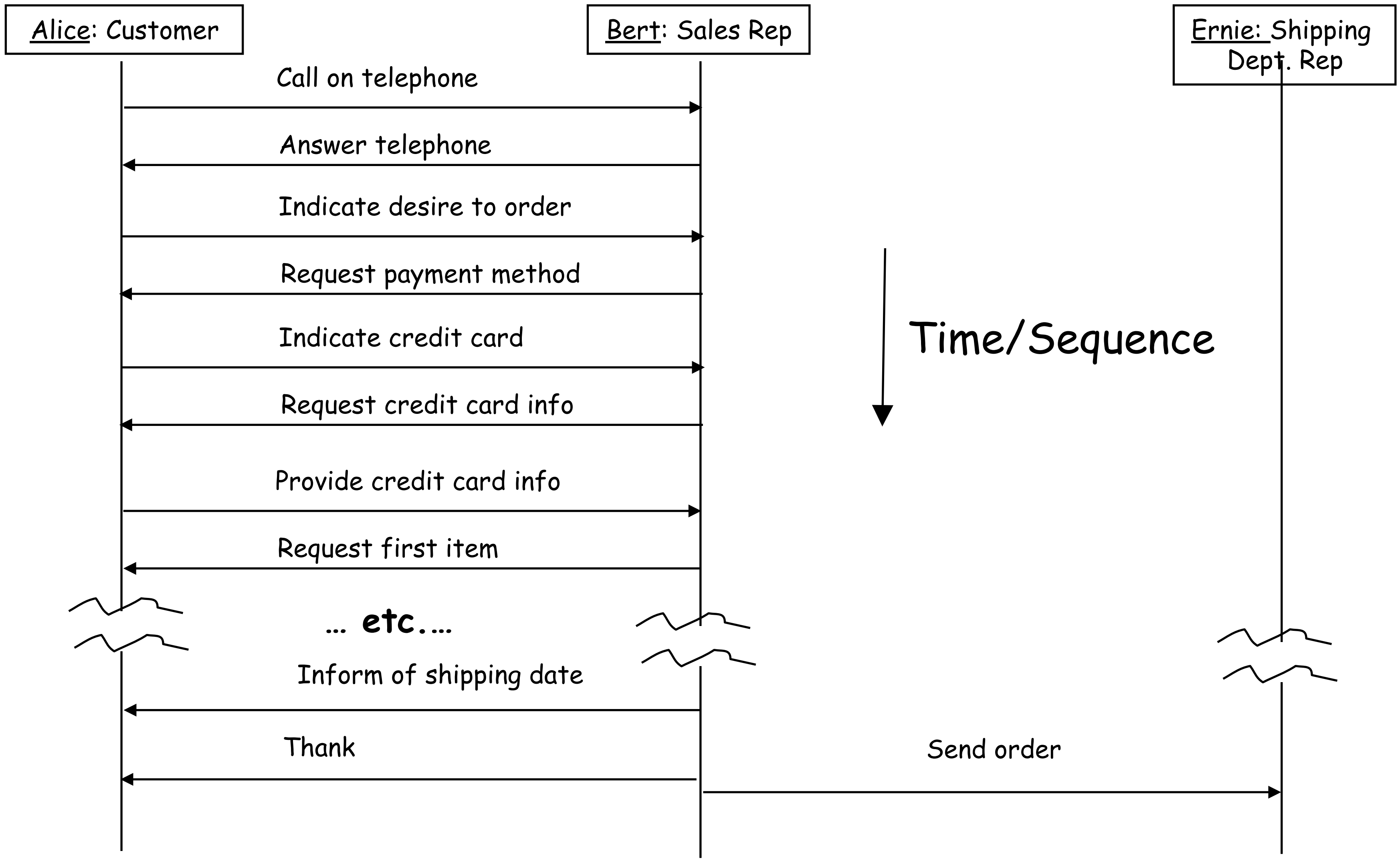
- Bert asks for third item.
- Alice responds that there are no more items.
- Bert asks for shipping address.
- Alice indicates that it is the same as the billing address.
- Bert informs Alice of expected shipping date and provides order number.
- Bert thanks Alice.
- Alice hangs up.
- Bert transmits order to Ernie in the Shipping dept.

Use-Case Advice (Larry Constantine and others)

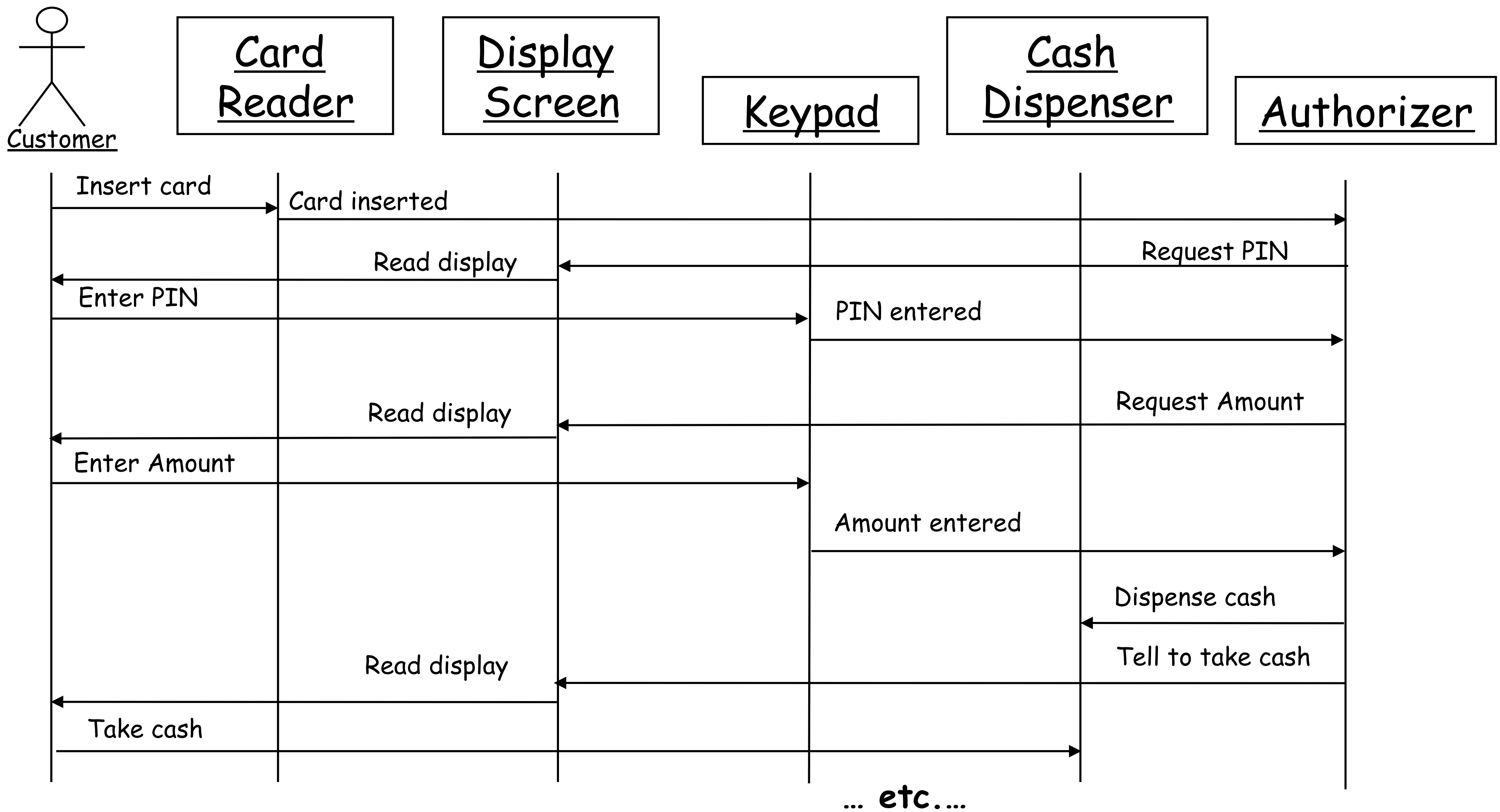
- Write in the active voice.
- Pair responses with the events that invoke them.
- Identify **domain objects** that clearly are part of the application context (such as “catalog”, “inventory”, “fleet” (of automobiles)).

  [A **domain glossary** could be used to prevent ambiguity.]

Sequence Diagram to clarify a Scenario

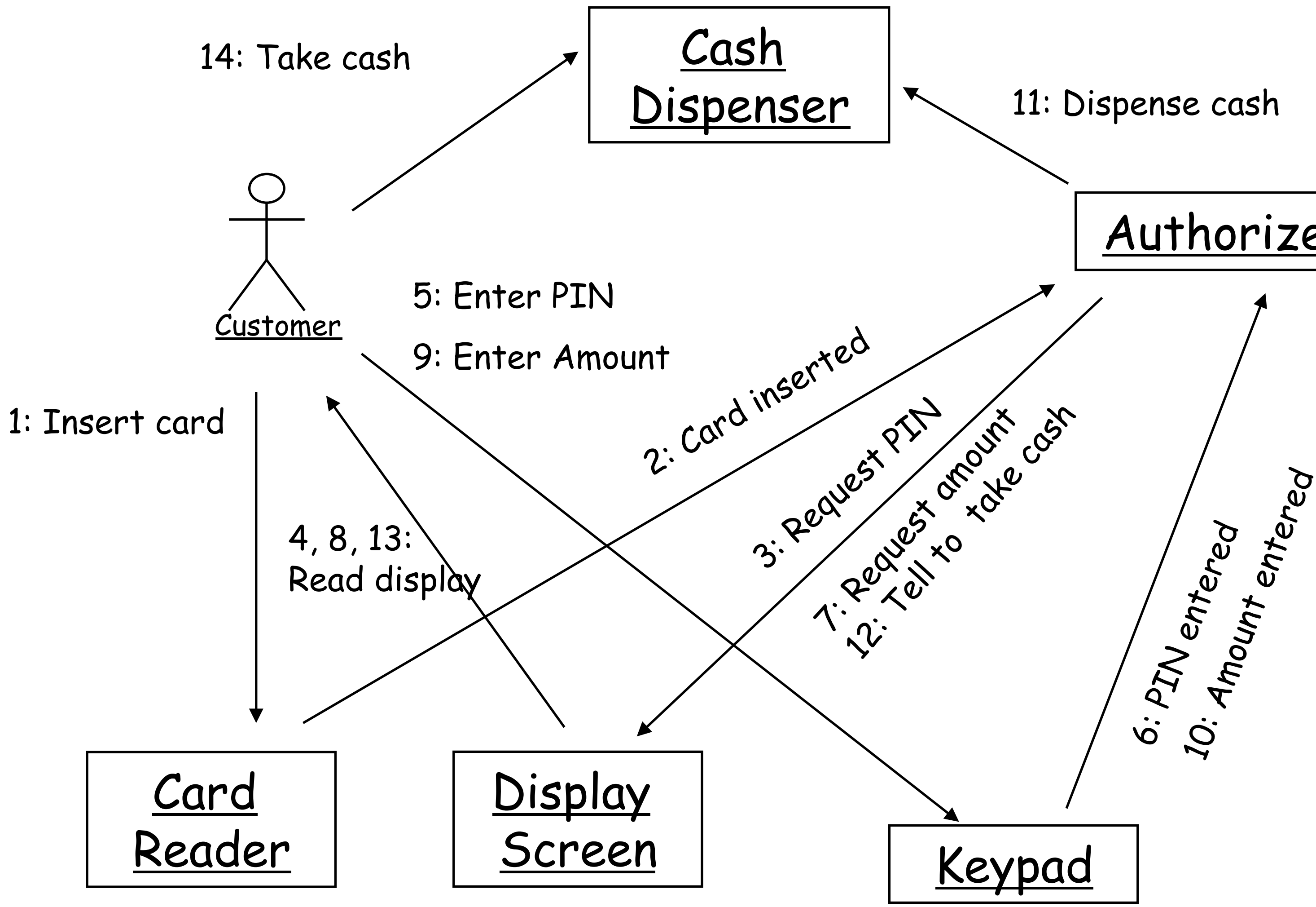


Sequence Diagram
for an ATM Withdrawal Use Case



Collaboration Diagram

(= “folded” sequence diagram with message numbers)



Scenario Types (Bruegge)

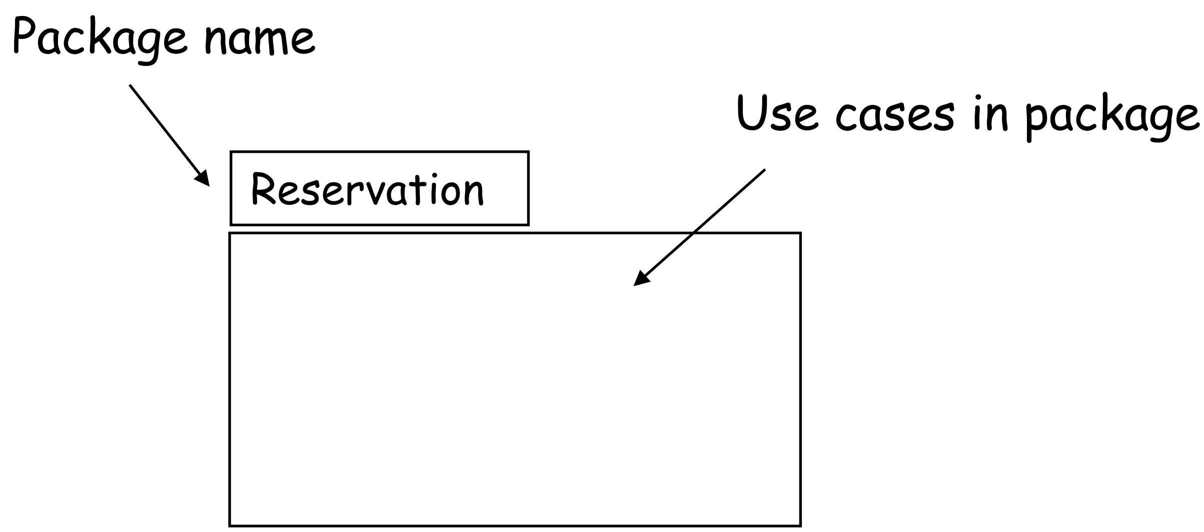
- **Visionary** scenario: Describes future scenario
- **Evaluation** scenarios: Describe user tasks against which system is evaluated
- **Training** scenarios: Used for tutorial purposes
- **As-is** scenarios: Describe current situation (during reengineering)

Structuring Sets of Use Cases

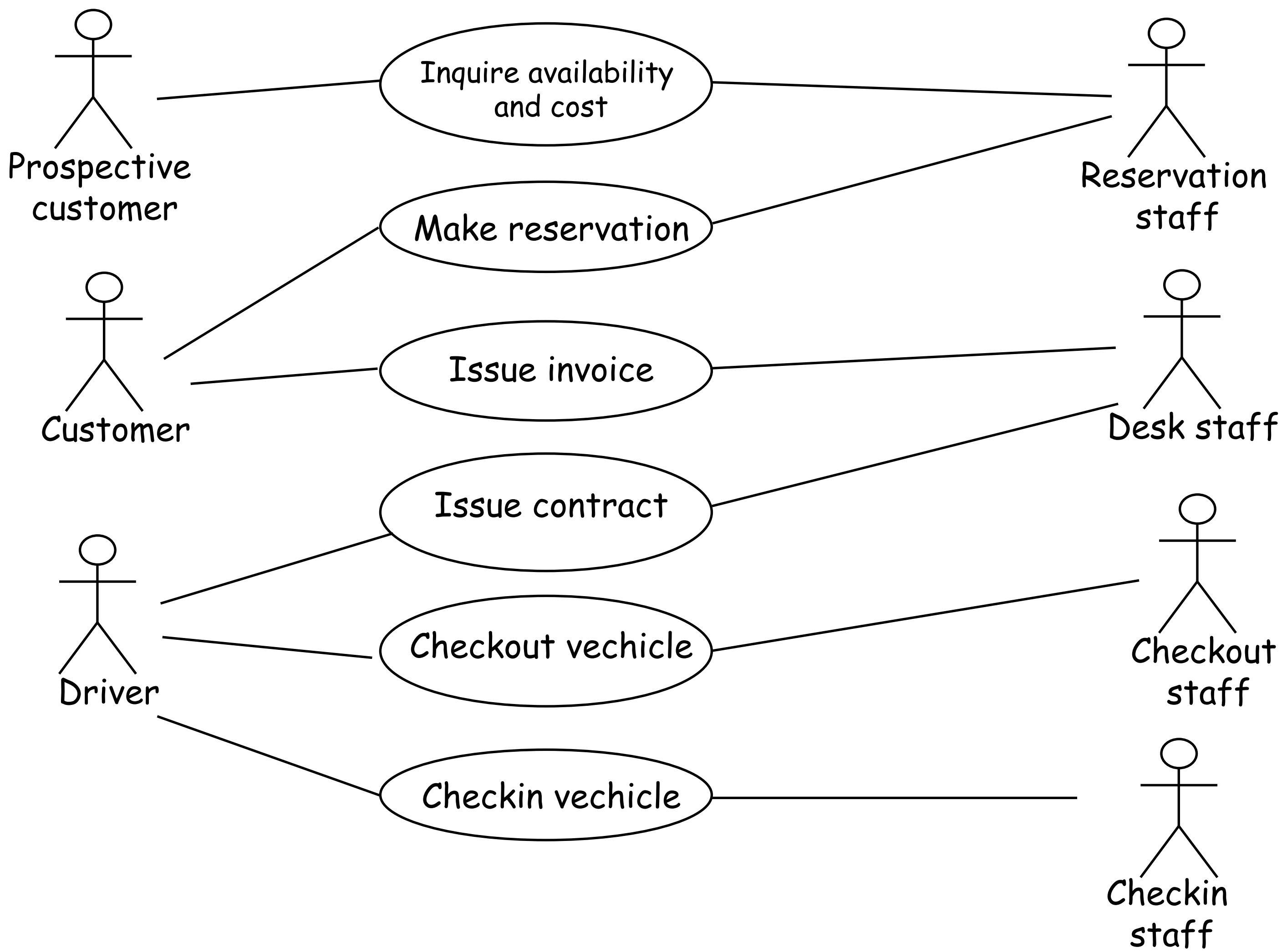
- Packages
- Relationships
  - Inclusion
  - Extension
- Actor Inheritance

UML Package Notation

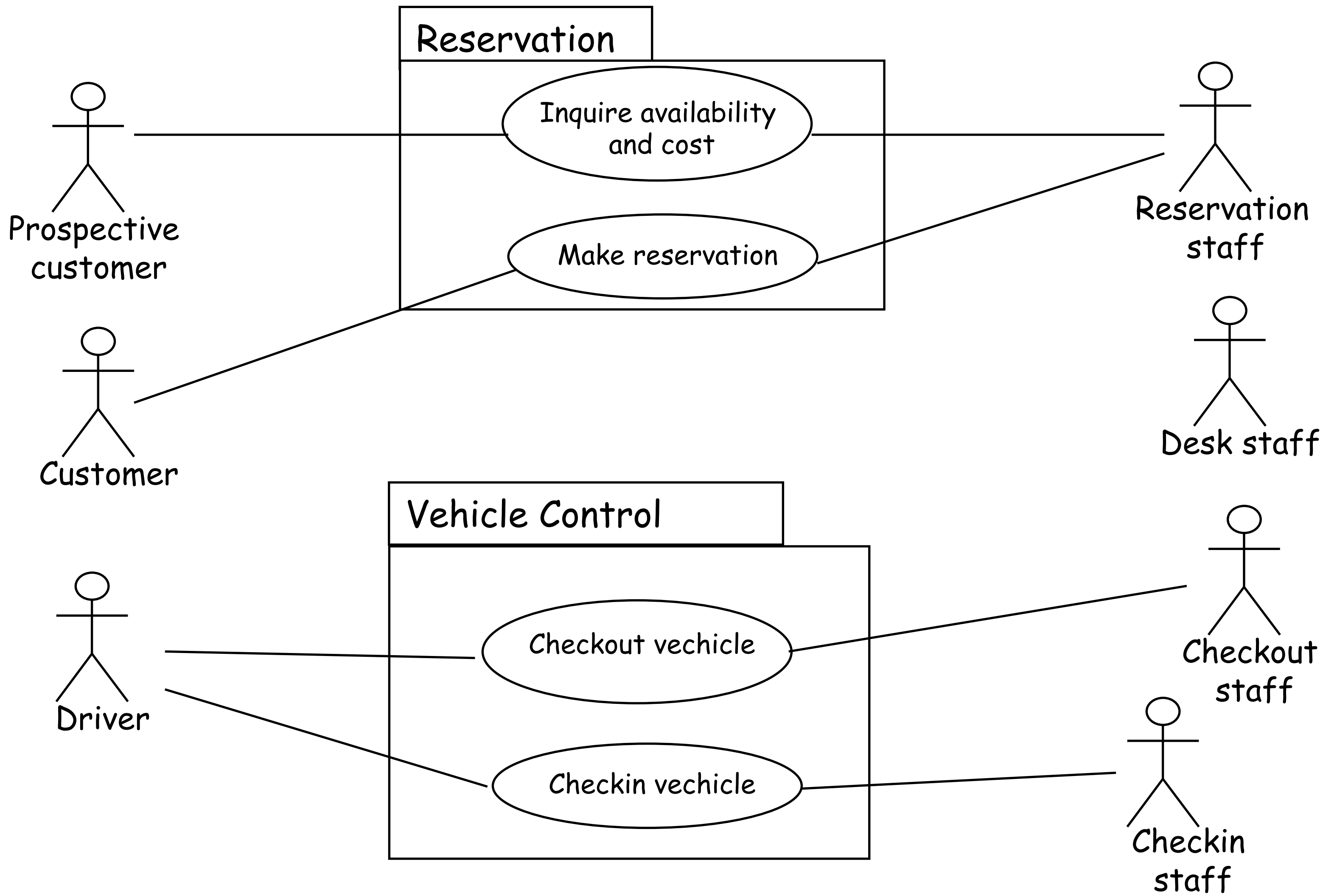
Sometimes used to **Group** Use Cases



Car Rental Example
How might the use cases be packaged?



Car Rental Example showing Two Packages



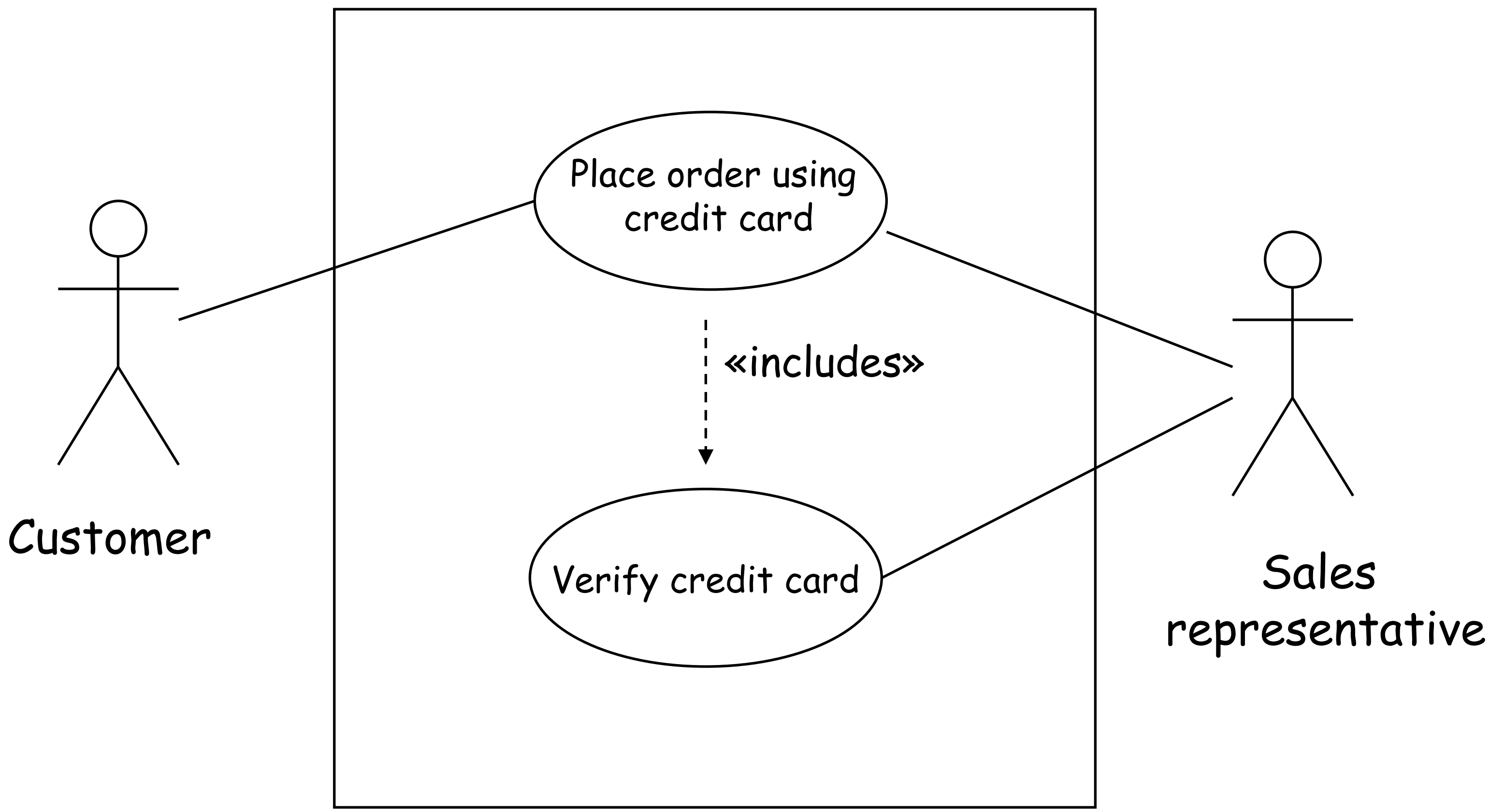
Use of Packages

Packages may be used in the transition to a design, and ultimately coding.

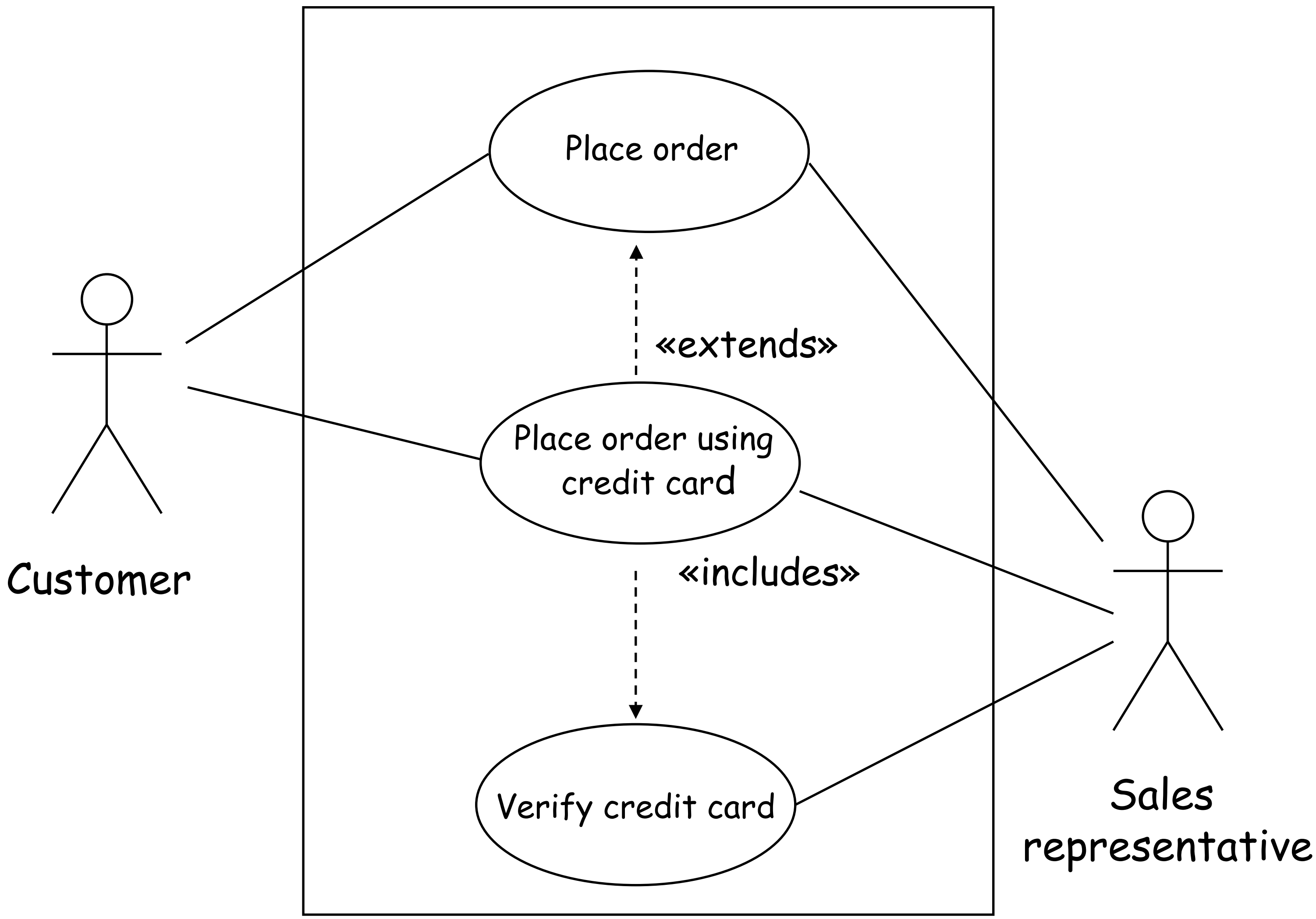
However, these connections would not normally be part of the initial discussion with the customer.

Relations between Use Cases

Inclusion among use-cases



Extension among use-cases



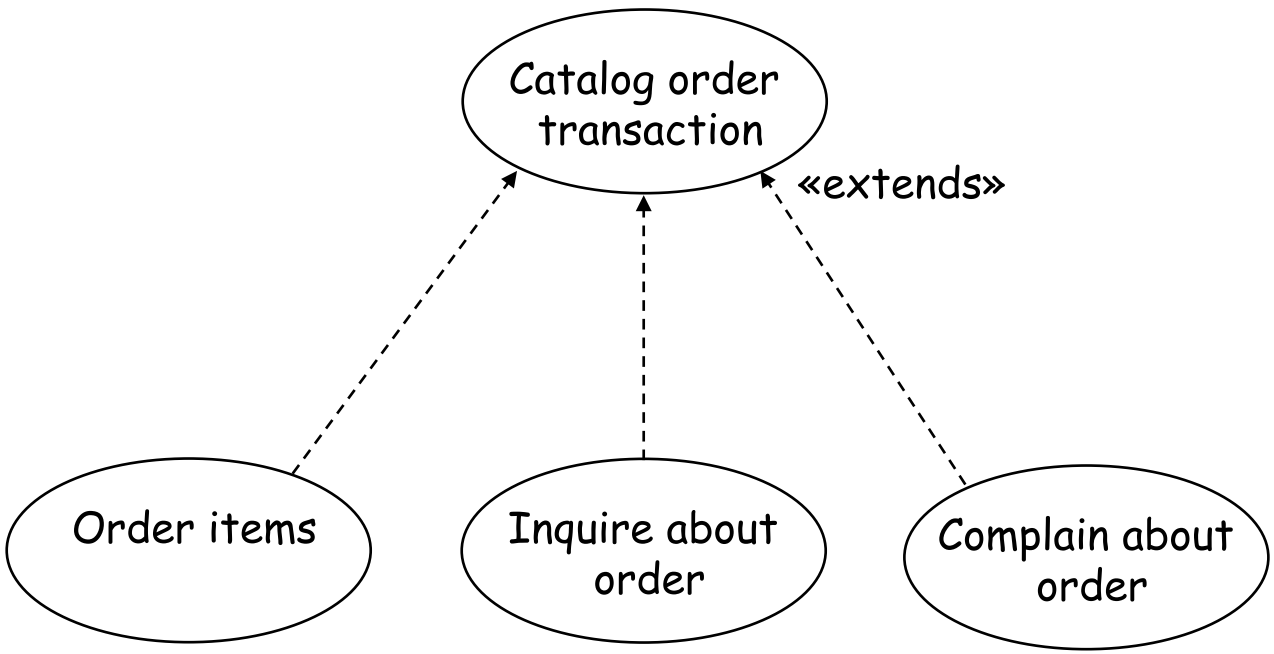
extends vs. *uses*

- **«includes»** means this use-case makes use of another use-case, as a kind of ***subroutine***. This allows us to not have to repeat the included use-case in the description of the including use-case.

- **«extends»** means that this use-case is a ***specialization*** of another use case.

Note: «includes» was formerly called «uses».

«extends» can be used to impart a hierarchical **abstraction structure** to use cases



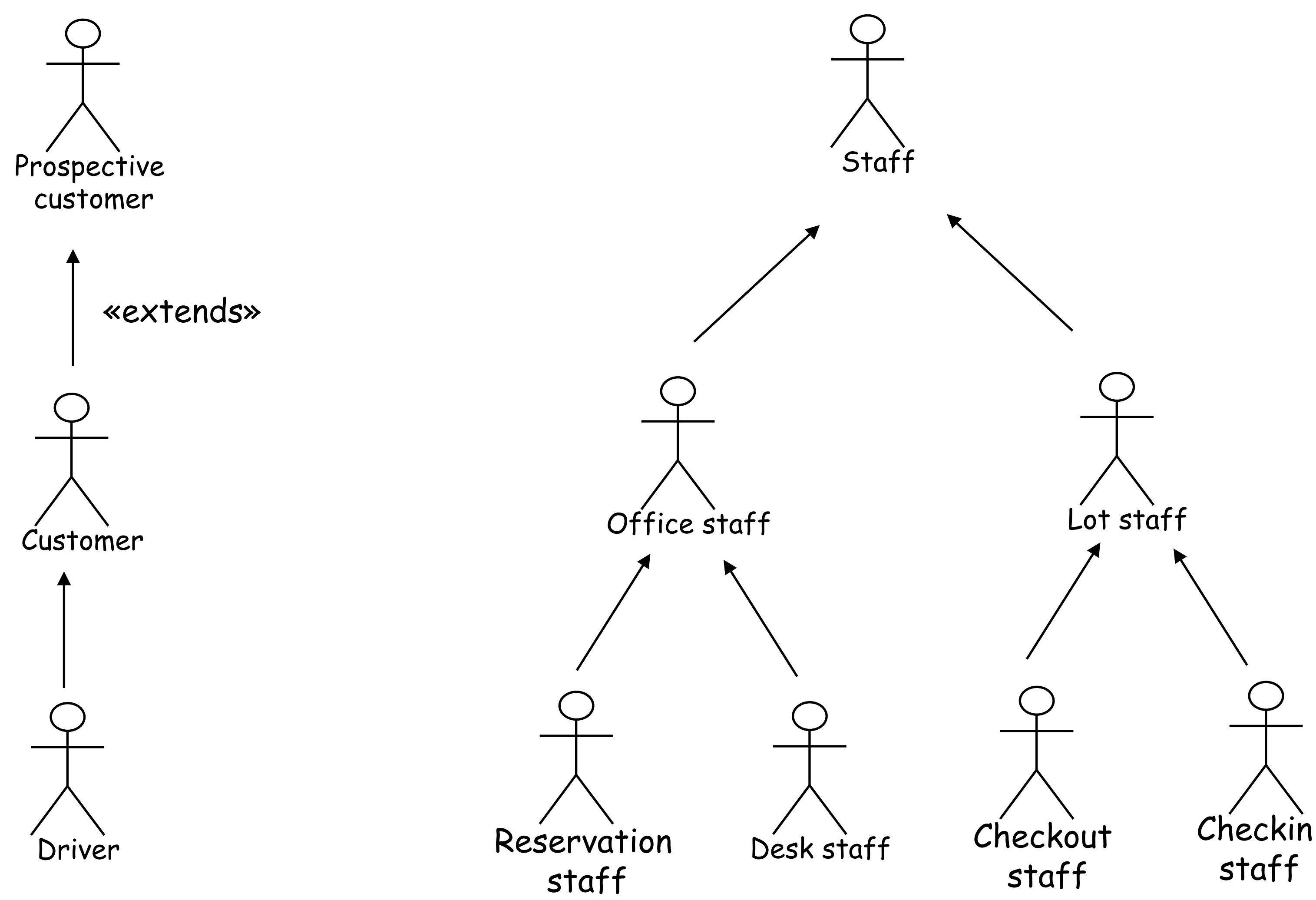
Options of a use case

- **Example**: During order processing, the sales representative offers to tell the customer about current specials.

- Such options should be mentioned as an annex to the other use case items.

Actor Hierarchies

are possible, similar to extensions



Caution about Structuring Use-Cases

- Use-case structuring is obviously *analogous* to structuring in object-oriented systems.

- However, one should ***not*** infer that use-case structure implies anything about *internal* structure of the system.

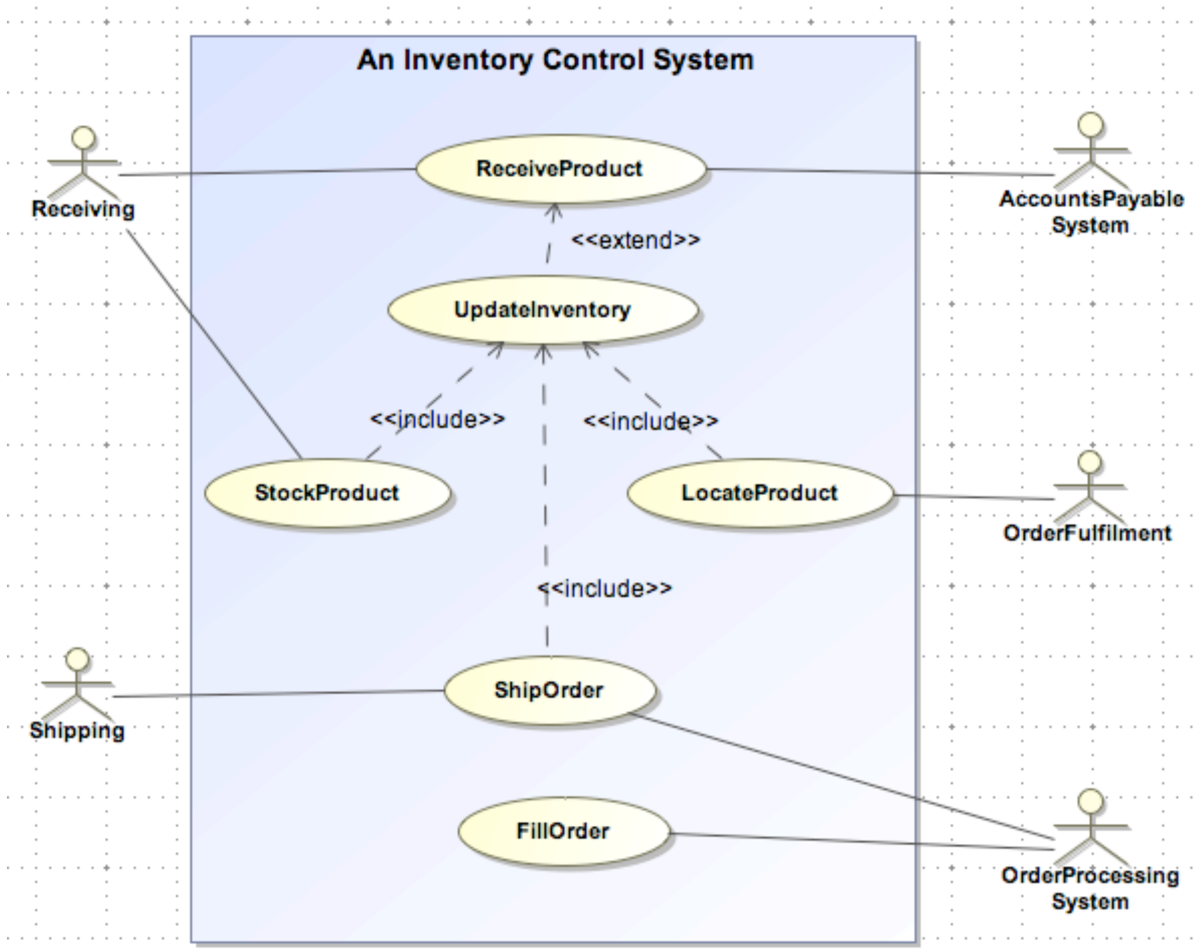
Use-cases vs. Requirements

- A use-case describes one “unit” of functionality.
- A single informally-specified functional requirement could translate into **multiple use-cases**.
- A single use-case could also be involved in satisfying **multiple requirements**.

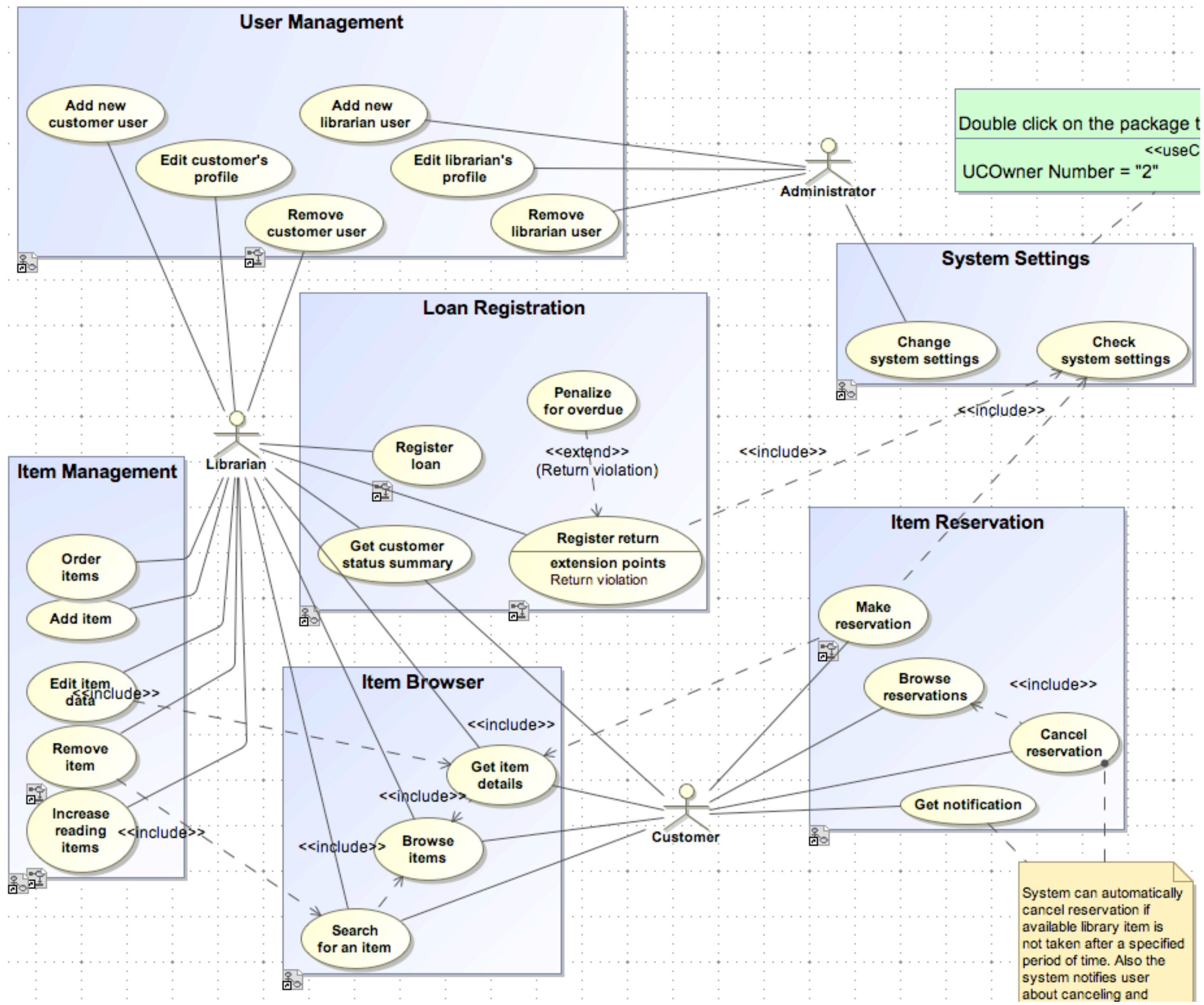
Use-cases vs. Requirements (cont'd)

- **Collectively**, the use-cases ideally should account for **all** of the desired **functional** requirements.
- Non-functional requirements may *annotate* use-cases, but don't get represented as use-cases directly.

Example of UC Automation: "Magic Draw" tool



Library Use-Case Diagram
(Magic Draw)



Further Possible Components of Use Cases

Goals

- A goal describes the **purpose** of the execution of the use case.
- **Goals** are important in the Allistair Cockburn version of use cases.
- Example: Goal for catalog order: "Products are ordered."

Pre- and Post-Conditions

- Some use-cases are not meaningful at arbitrary times, but instead only when the system is in a state with certain properties. Such properties are called ***pre-conditions***.

- Similarly, the use-case might leave the system in a state known to satisfy one or more ***post-conditions***.

Example:

- For the car-rental enterprise, the use case "checkin vehicle" has the **pre-condition:**
  vehicle is rented to driver
  and the **post-condition**:
  vehicle is on site
  & vehicle is not rented to a driver.

- So, in a way, the post-condition should imply that the goal has been realized.

- For the use-case "checkout vehicle", these conditions are reversed.

Invariants

- A condition that is a pre- and post-condition for all use cases is called an ***invariant***.

- Example: Total vehicles =
  vehicles rented
  \+ vehicles available
  \+ vehicles in repair
  \+ vehicles in scrap.

Optional Aspect: Trigger

- A ***trigger*** is an event that causes the use case to be run.
- Example: A catalog order is **triggered** by a phone call.
- This is similar to a pre-condition, but is a dynamic **event** rather than a **condition**.

Exceptions

- If a use case cannot be completed as described, an *exception* is said to occur.
- The description can indicate aspects of the state and output in such cases.

Alternative to Exceptions

- A use case may be allowed **explicit success and failure outcomes**, each with its own post condition.

Precedence among Use-Cases

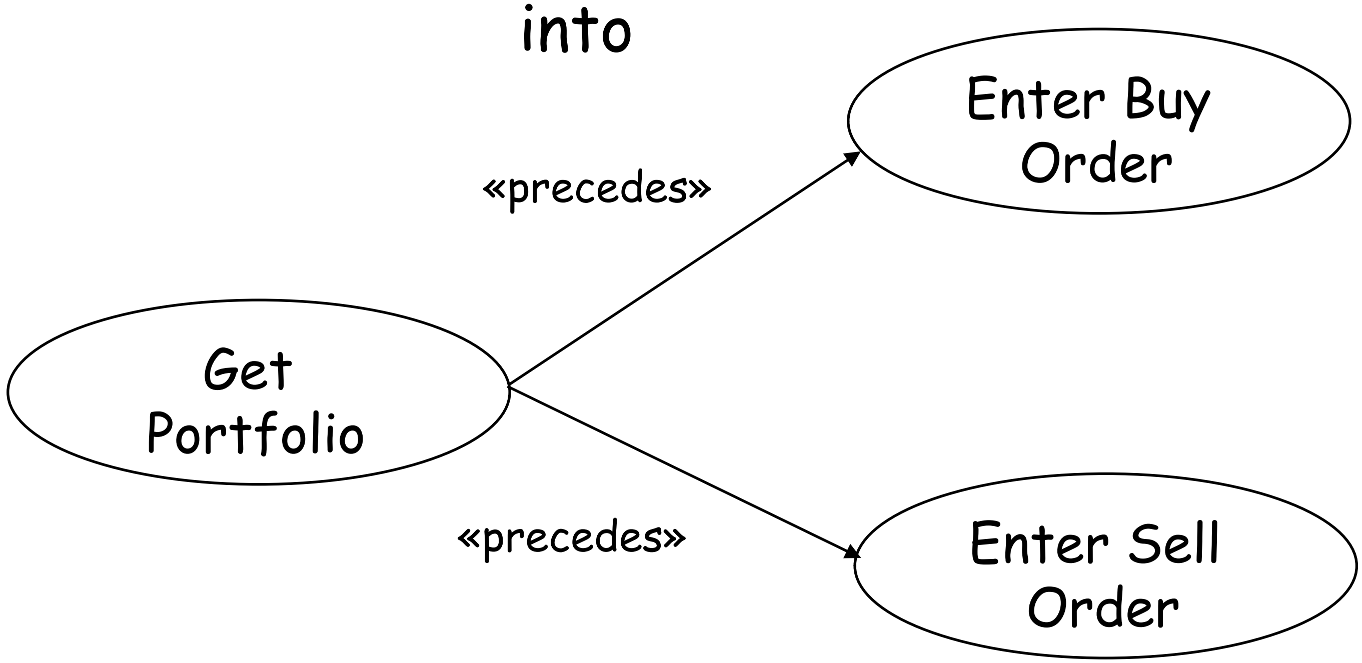
- When one use case is used to establish a pre-condition for another, the two may be linked by a «precedes» relationship.

- One use of precedence is to ***factor*** a use-case into sub-cases, to avoid repetition among different sub-cases.

Precedence Example

Stock-trading example:

could factor into



Subset of "Top 10 Use-Case Pitfalls"

by Susan Lilly, Software Development Magazine, Jan. 2000
ieeexplore.ieee.org/iel5/6380/17059/00787547.pdf?arnumber=787547

- The system boundary is undefined or inconsistent.
- The use cases aren't written so the customer can understand them.
- The use cases are written from the system's (not the actors') point of view.
- The actor names are inconsistent across use cases.
- The use cases don't correctly separate actors based on functional entitlement.
- The use-case specifications are too long or confusing.

The system boundary is undefined or inconsistent [Is “mixed-up scope” an instance?]

Example: A *Kiosk Customer* uses the computer system to order tickets. Alternately, a *Phone Customer* may call the ticket business, and a *Phone Clerk* (an employee of the ticket business) may use the computer system to order tickets. Who are the actors? Figure 1 illustrates a mixed-up system boundary: The modelers have tried to show both the users of the business and the users of the system in the same use case model.

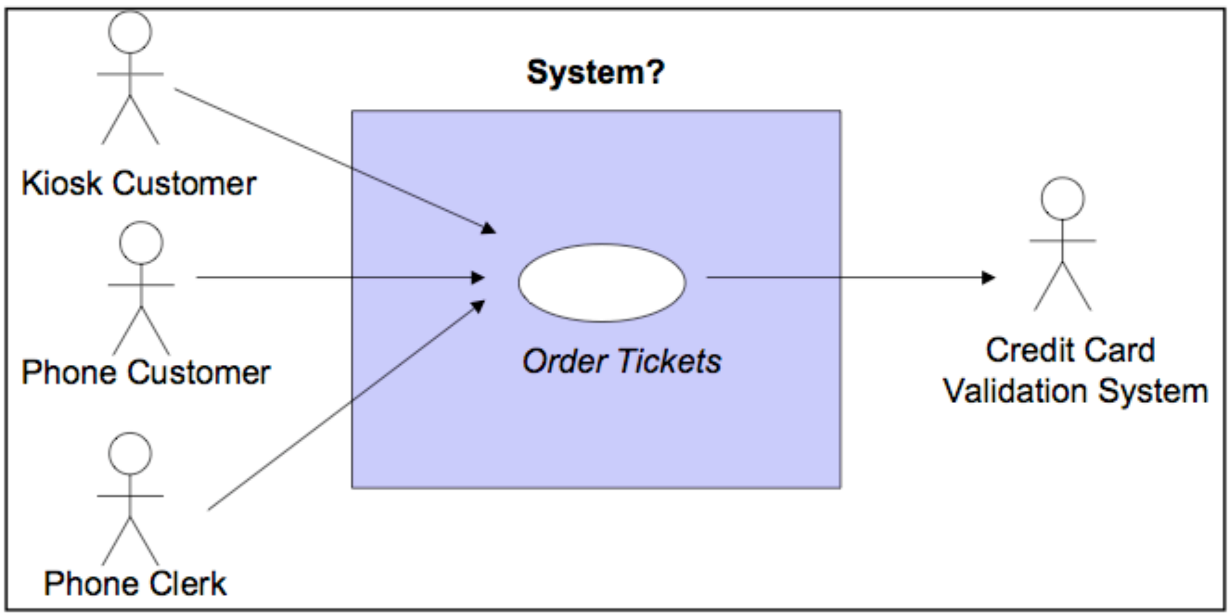


Figure 1: Use Case with Mixed-Up Scope

The use cases don't correctly separate actors based on functional entitlement.

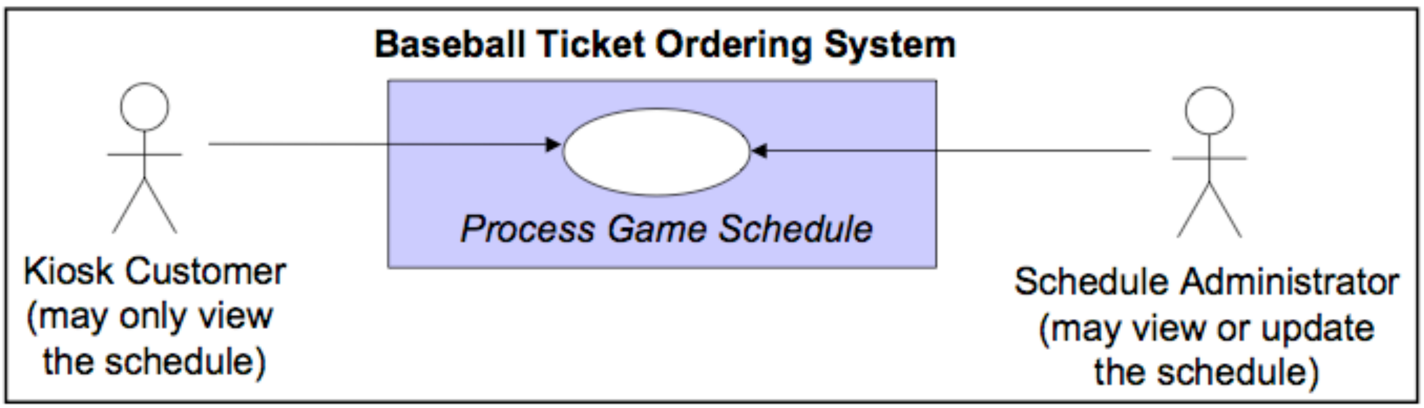


Figure 9: Confusing Functional Entitlement

Fix to the Preceding Flaw

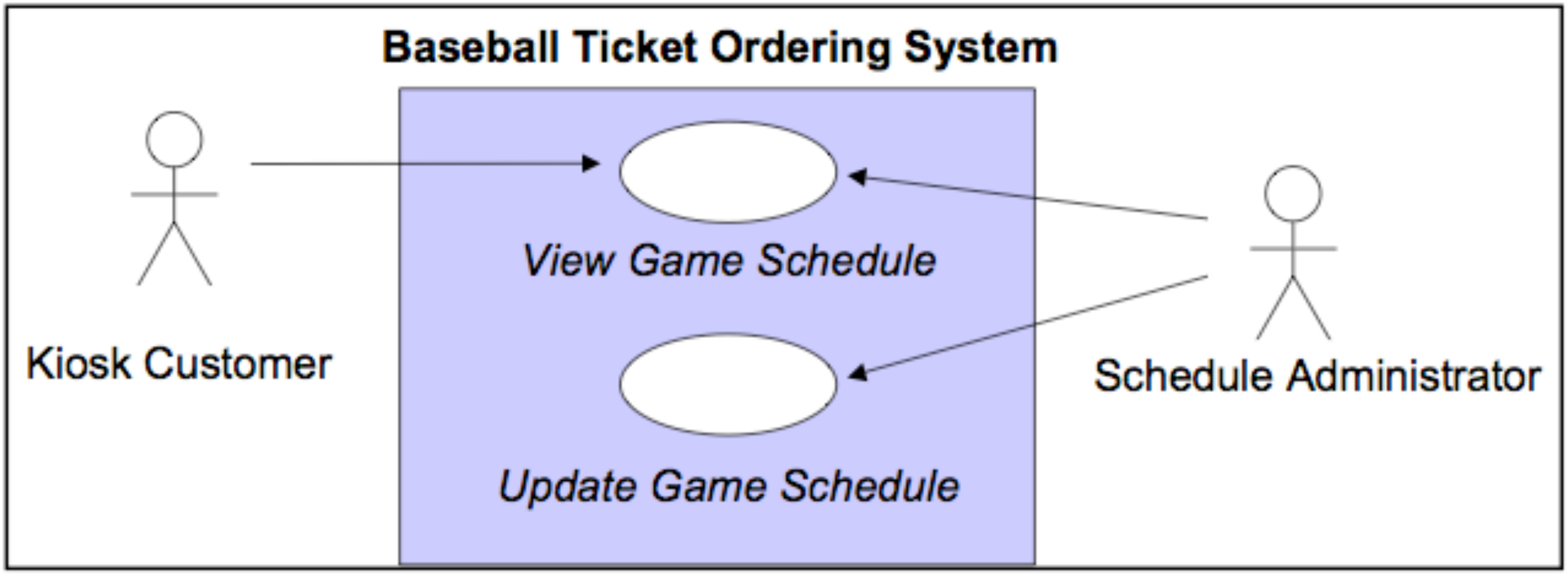


Figure 10: Use Cases with Correct Functional Entitlement

Use-Case Template

Possible template for use cases:

- Label
- Name
- Goal
- Actors
- Initiator
- Description
- Pre-conditions
- Post-conditions
- Options (if present)
- Scenarios

Traceability Matrix

- One form of Traceability Matrix, lists each of the requirements here identifies the use cases that cover those requirements.

Additional Points on Use Cases

Do *Not* use Use-Cases to Fully Decompose into a **Design**

- Factoring should be used to ***simplify the description*** of use-cases.
- Avoid the temptation of turning use-case decompositions into *the* design. [Why?]
- Use-cases are ***customer language***, ***not design language*** or pseudo-code. They describe *what*, not *how*.
- There are other tools that are better-suited to the design phase.

Uses of Use-Cases **across** various Development Phases (Bruce Douglass, “Doing Hard Time”)

- Analysis phase:
  - Suggest large-scale partitioning of the problem domain
  - Provide structuring of analysis objects (i.e. actors and sub-systems)
  - Clarify system and object responsibilities
  - Capture new features as they are added
  - Validate analysis model

Uses of Use-Cases across Development Phases (Bruce Douglass) (cont'd)

- Design phase:
  - ***Validate*** the elaboration of analysis models in the presence of design objects
- Coding phase:
  - ***Clarify*** purpose and role of classes for coders
  - ***Focus*** coding efforts
- Testing phase:
  - Provide test scenarios for ***validation***
- Deployment phase:
  - Suggest ***iterative prototypes*** for spiral development

Levels of Use Cases

- These ideas are from Use Cases: Requirements in Context, Daryl Kulak and Eamonn Guiney, ACM Press, 2000.
- Four **iterative levels** for specifying use cases:
  - **Façade**: Outline and high-level descriptions
  - **Filled**: Broader and deeper descriptions
  - **Focused**: Narrowing and pruning
  - **Finished**: Touch-up and fine-tuning
- See the reference for example worked out at all levels.

Façade Use-Case Components

- Name
- [Goal] (I added this.)
- Summary
- Basic course of events

Filled Use-Case Components

- Name
- [Goal]
- Summary
- Basic course of events
- Alternative paths
- Exception paths

Focused/Finished Use-Case Components

- Name
- [Goal]
- Summary
- Basic course of events
- Alternative paths
- Exception paths
- Extension [Option] points
- Trigger
- Assumptions
- Preconditions
- Postconditions
- Related ***business rules***

Business Rules

- ***Business rules*** are requirements that represent ***constraints*** on behaviors, rather than behaviors themselves.
- Examples:
  - All transactions are in U.S. Dollars.
  - Valid operators license and credit card are required in order to rent a car.
  - Late-fee is assessed for enrollment after the second week of the semester.

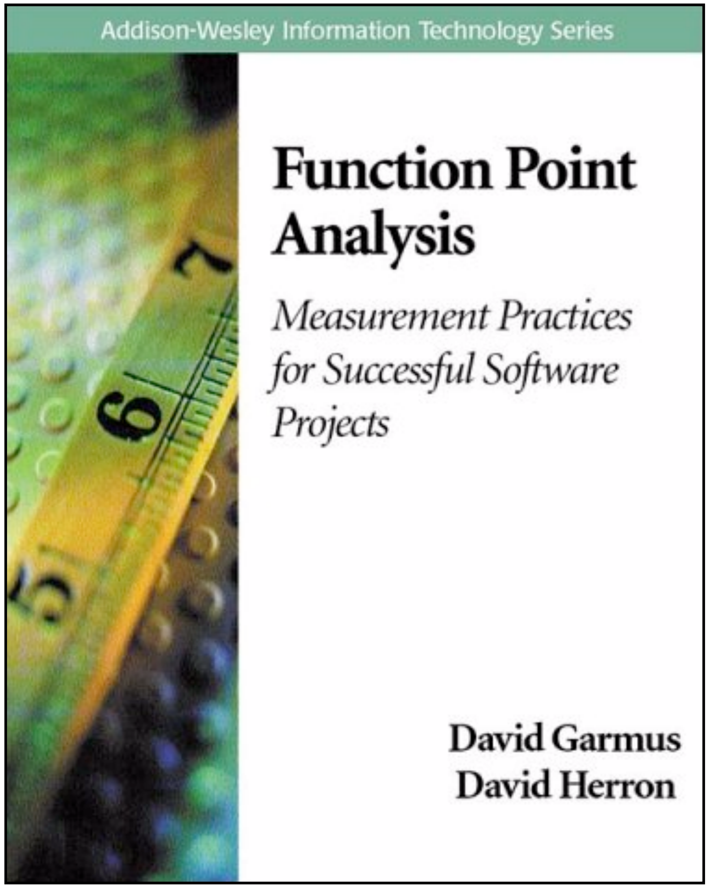
UML Ways of Clarifying Complex Behaviors in Use Cases

- These are more technical and may be more appropriate in the ***design*** phase. However, sometimes they can clarify a use case:
  - **Sequence diagram**: shows messages between actors and sub-systems
  - **Collaboration diagram**: a sequence diagram organized as a directed graph rather than as a linear sequence of messages.
  - **State chart**: Elucidates behavior in terms of properties of state
  - **Timing diagram**: a sequence diagram with a time metric applied to the sequence dimension

Related Idea Earlier than Use Cases

- **Function-point** analysis
- Function points are the set of specific features or operations in a software product.
- Function points are used more for **cost analysis** than for SRS as such.
- Promoted by IFPUG (International Function Point Users' Group)

Function Point Books



Examples of Function Point Categories

- EI: External Inputs
- EO: External Outputs
- EQ: External Query
- ILF: Internal Logical Files
- EIF: External Interface Files

Function-Point Computation

Type of Component	Complexity of Components			
	Low	Average	High	Total
External Inputs	___ x 3 = ___	___ x 4 = ___	___ x 6 = ___	
External Outputs	___ x 4 = ___	___ x 5 = ___	___ x 7 = ___	
External Inquiries	___ x 3 = ___	___ x 4 = ___	___ x 6 = ___	
Internal Logical Files	___ x 7 = ___	___ x 10 = ___	___ x 15 = ___	
External Interface Files	___ x 5 = ___	___ x 7 = ___	___ x 10 = ___	
		Total Number of Unadjusted Function Points		___
		Multiplied Value Adjustment Factor		___
		Total Adjusted Function Points		___

Worked Example

(1&2) PF

All values from FP tables.

Function	**Diff. Level**	**Number**	**Complexity**	**FP**
Input	M	2	3	6
DB	A	4	10	40
Output	S	4	3	12
Total				58

(3) PIF Project-Influence Factors

PIF	**Difficulty Level**
Data performance objectives	2
Multiple sites	5
Total	7

(4) Total Effort

PM = 0.036*58*7 = 14.616 person-months

From an FP Training Manual

When Not to Use Function Points

Function points are not a very good measure when sizing maintenance efforts (fixing problems) or when trying to understand performance issues. Much of the effort associated with fixing problems (production fixes) is due to trying to resolve and understand the problem (detective work). Another inherent problem with measuring maintenance work is that much of maintenance programming is done by one or two individuals. Individual skill sets become a major factor when measuring this type of work. The productivity of individual maintenance programmers can vary as much as 1,000 percent.

Performance tuning may or may not have anything to do with functionality. Performance tuning is more a result of trying to understand application throughput and processing time. There are better metrics to utilize when measuring this type of work.