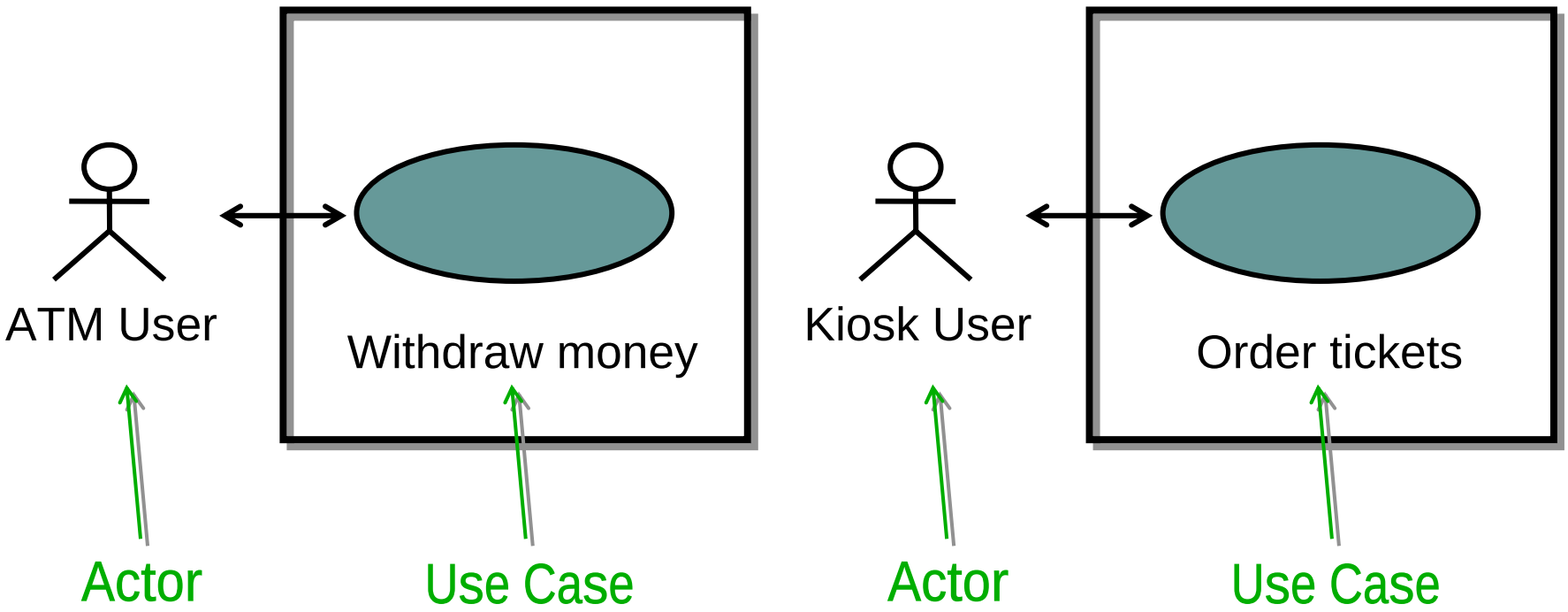
Requirements Engineering

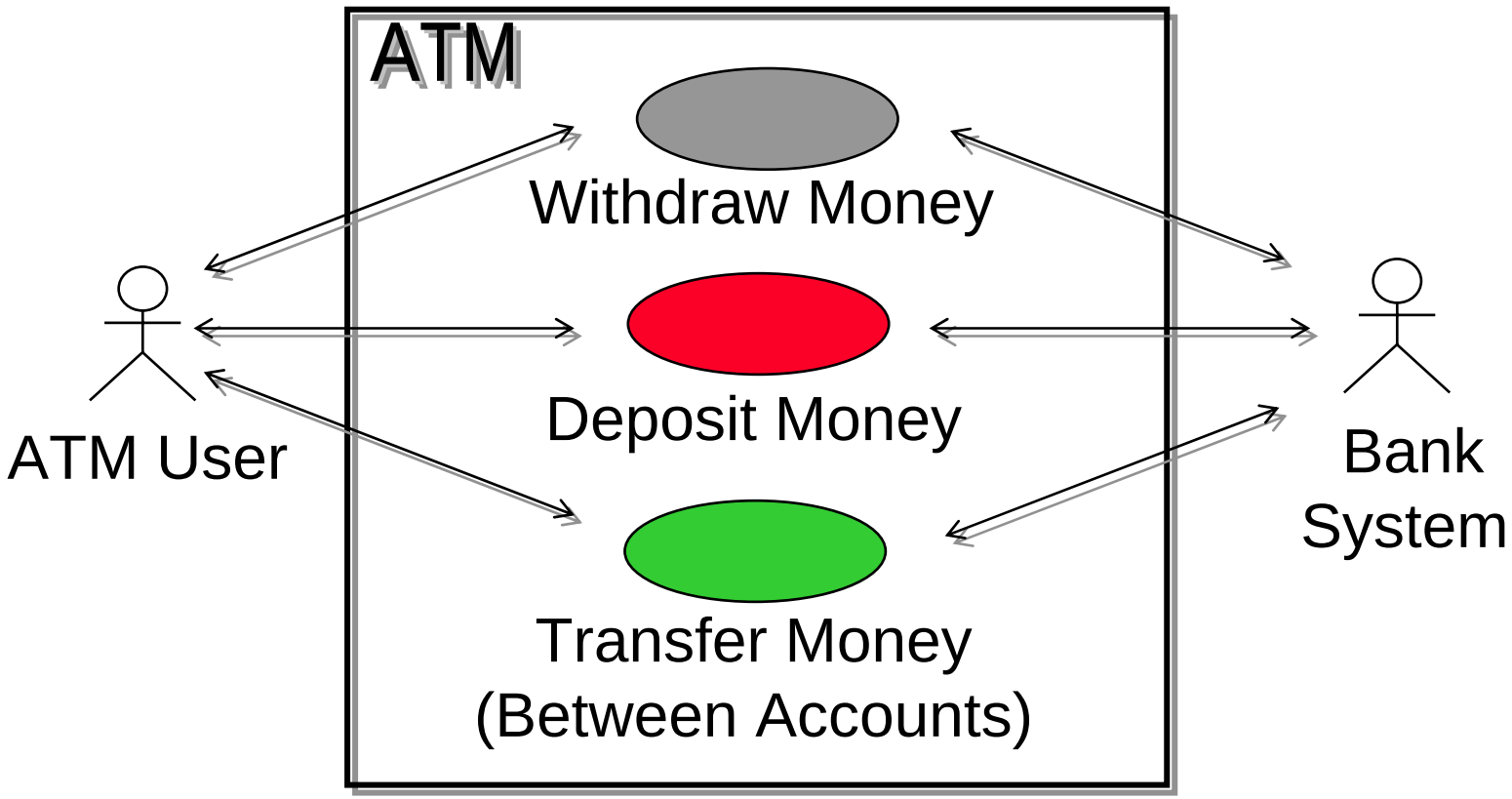
- Requirements engineering methods have many limitations
  - Do not necessarily map well to design/code
  - Do not translate well to acceptance tests
  - Require additional work/effort/thought
  - Are difficult for non-experts/other stakeholders to understand
- Use cases attempt to bridge the understandability gap
  - Describe system behavior, flows of events
  - Describe user requests and system responses
  - Useful in formulating test steps and verification points

Use Cases: Keep it Simple

Use cases are a simple and powerful way to define requirements for software behavior



Example: ATM Use Case Diagram



What is a Use Case (1)

- Use cases are textual descriptions of
  - Major functions the system will perform for its users
  - Goals the system achieves for its users along the way
- A use case describes a set of flows or scenarios
  - Each scenario is a sequence of steps describing an interaction between a “user” and a “system”
  - The use case is a collection of scenarios that together accomplish a specific user “goal”
  - Each scenario corresponds to a single path or flow through a use case

Example: Buy a Product

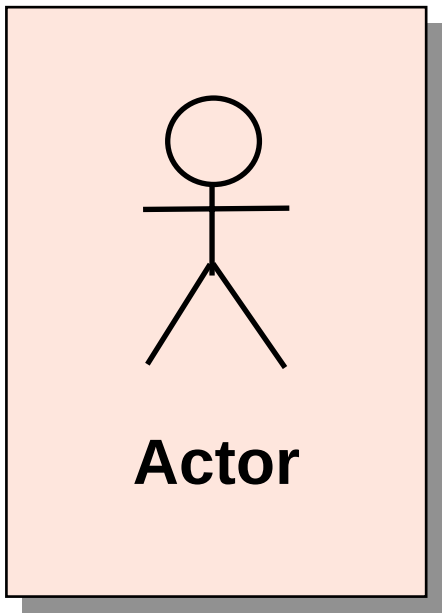
- A scenario is a sequence of steps describing an interaction between a user and a system
- For an online store, we might use narrative text to describe the following Buy a Product scenario*:
  - "The customer browses the catalog and adds desired items to the shopping basket. When the customer wishes to pay, the customer describes the shipping and credit card information and confirms the sale.
  - The system checks the authorization on the credit card and confirms the sale both immediately as well as with a follow-up email."

* *Adapted from "UML Distilled" by Martin Fowler*

Alternative Flows for Buy a Product

- In our Buy a Product scenario things went well…
  - It is the "happy day" scenario or *Basic Flow*
- But things can go wrong…
  - The credit card authorization might fail
    - » This would be a separate scenario or *Alternative Flow*
  - You may not need to capture shipping and credit card information for returning customers
    - » This is yet another scenario, a second alternative flow
- A use case is a set of scenarios serving a common goal
  - The user does not always succeed but the goal remains

What Is An Actor?



- Actors are not part of the system, they represent roles a user of the system can play
- An actor may actively interchange information with the system
- An actor may be a provider of information, receive information, or both
- An actor can represent a human, a machine or another system (for example, software, hardware, DB)

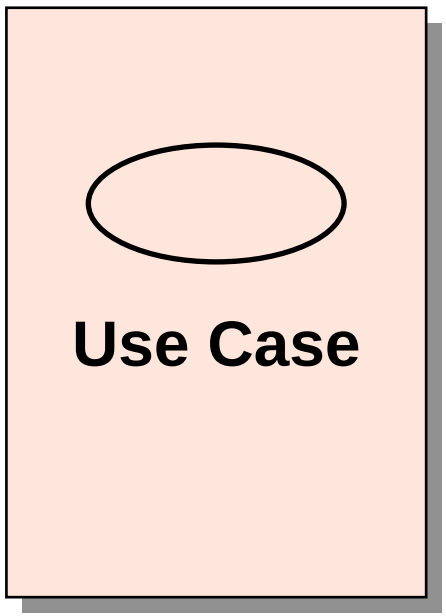
Identifying Actors (1)

- Actors are discovered
  - in any documents describing system scope/definition
  - by talking with customers and domain experts
- Useful questions for identifying actors include:
  - Who uses the system?
  - Who installs the system?
  - Who starts up the system?
  - Who shuts down the system?
  - What other devices and external systems work directly with the system?

Identifying Actors (2)

- Additional questions for identifying actors are:
  - Who gets information from this system?
  - Who provides information to the system?
  - Does anything happen automatically at a preset time?
  - Who is interested in a certain requirement?
  - Where in the organization is the system used?
  - Who will benefit from the use of the system?
  - Who will support and maintain the system?
  - Does the system use an external resource?
  - Does one person play several different roles?
  - Do several people play the same role?
  - Does the system interact with a legacy system?

What is A Use Case? (2)



- A use case is a dialogue between actors and the system
- A use case is initiated by an actor to invoke a certain functionality in the system
- A use case is a complete and meaningful flow of events
- A use case captures the contract or “guarantees” that will hold at the conclusion of the use case

Benefits of Use Cases

- Use cases
  - Capture the intended behavior of the system you are developing
    - » Without having to specify how that behavior is implemented
  - Allow developers to come to a common understanding with your system's end users and domain experts
  - Realized by design and implementation elements working together to carry out each use case
  - Help verify and validate your design and implementation features
    - » Against user goals and requested functions

A Variety of Readers

- **Human Factors, Marketing & Clinical Engineering** -- Approve what the system should do
- **System Engineers** – Ensure system requirements are met by the use cases
- **Reviewers** --Examine the flow of events
- **Software Developers** -- Provide basis for analysis, design and implementation
- **System & Software Testers** – Provide basis for test cases
- **Project Leads** -- Project planning
- **Technical Writers** -- Writing the user's guide for the end users

Identifying Use Cases

- Useful questions to identify use cases in a system:
  - What functions will the actor want from the system?
  - Does the system store information? What actors will create, read, update, or delete that information?
  - Does the system need to notify an actor about changes in its internal state?
  - Are there any external events the system must know about? What actor informs the system about those events?
  - What are the tasks of each actor?
  - What use cases will support and maintain the system?
  - Can all functional requirements be performed by the use cases?

Use Case Flow Of Events

- Describe only the events needed to accomplish required behavior of the use case
  - » Written in terms of what the system should do, not how it does it
  - » Written in terms the audience (customer/stakeholder/other) will understand
  - » Written using business-domain terminology, not implementation
- The flow of events should describe
  - » When and how the use case starts and ends
  - » The interactions (in sequence) between use case and actors
  - » What data is needed by/exchanged during the use case
  - » The basic flow (normal sequence) of events for the use case
  - » Description of any alternative or exceptional flows of events

Example: Buy a Product

- Level: Sea Level
- Basic Flow (Main Success Scenario)
  1. Customer browses catalog and selects items to buy
  2. Customer goes to check out
  3. Customer fills in shipping information
  4. System presents full pricing information, including shipping
  5. Customer fills in credit card information
  6. System authorizes purchase
  7. System confirms sale immediately
  8. System sends confirmation email to customer
- Alternative Flow

  3a. Customer is regular (repeat) customer
    1. System displays current shipping, pricing and billing information
    2. Customer may accept or override defaults, returns to BF at step 6

Discussion (1)

- Begin by describing the Basic Flow
  - Main Success Scenario
  - Sequence of numbered steps
- Add variations
  - Alternative Flows
    - » Still achieve the goal successfully
  - Exception Flows
    - » Failure to achieve the goal
- Each use case has a primary actor
  - Has the goal the use case is trying to achieve
  - There may be additional, secondary actors
- Each step in the use case flow should be a clear simple statement
  - Show who is engaged and involved in the step
  - Show the intent of the actor, “what” the actor wants, not “how” the system does it
  - Therefore do not describe or include UI details in the text of the use case step

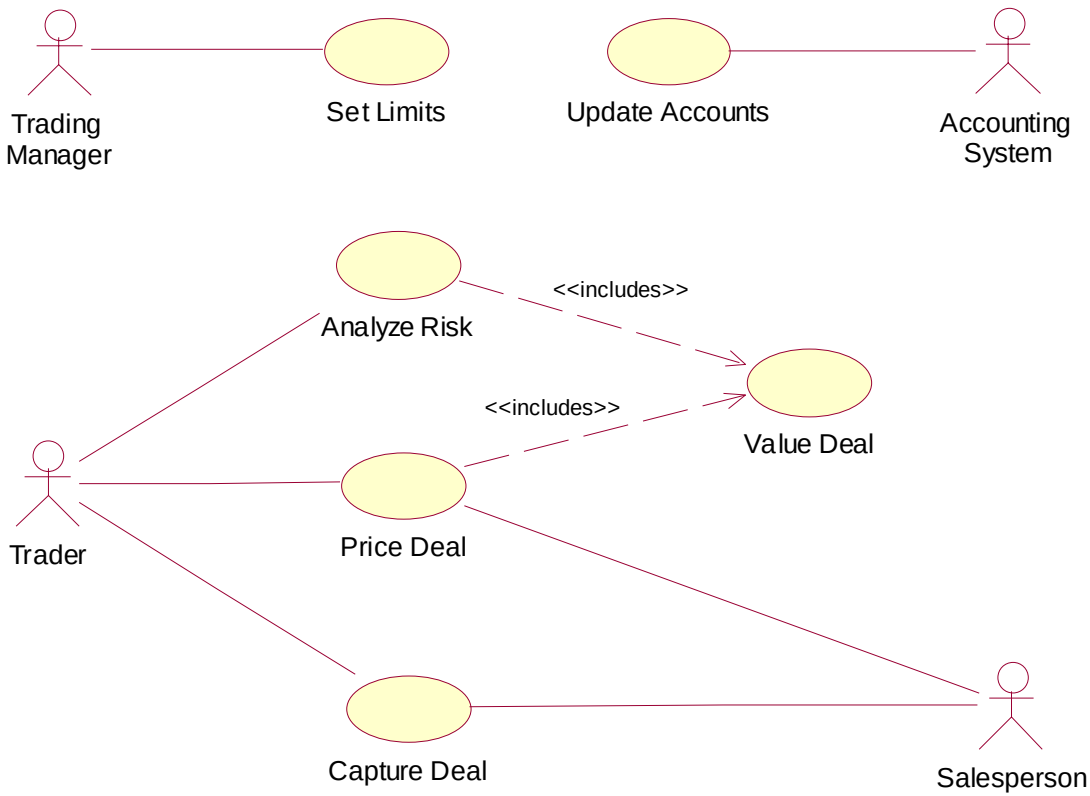
Discussion (2)

- Use case granularity or “level” is useful but challenging
  - Cockburn* defines a hierarchy of use case levels
  - Core use cases are at “sea level”
    - » An interaction with an actor toward a visible tangible goal
  - Fish Level
    - » Use cases that are included by sea-level use cases
  - Kite Level
    - » Show how sea-level use cases fir into larger business context
    - » Also called summary-level or business-level use cases
- Establish your own conventions for levels for your project

Use Case “Includes”

- One use case includes another use case in its entirety
  - Analogous to a program calling another or a routine using a subroutine
  - The “call” is mandatory
    - The calling/including use case must flow through the included use case
  - Often used for reuse
    - Multiple use cases share the same functionality
    - This functionality is placed in a separate use case
    - Avoids repetition of the same information in multiple use cases
- Examples
  - Logon/logoff
  - User Authentication/Authorization

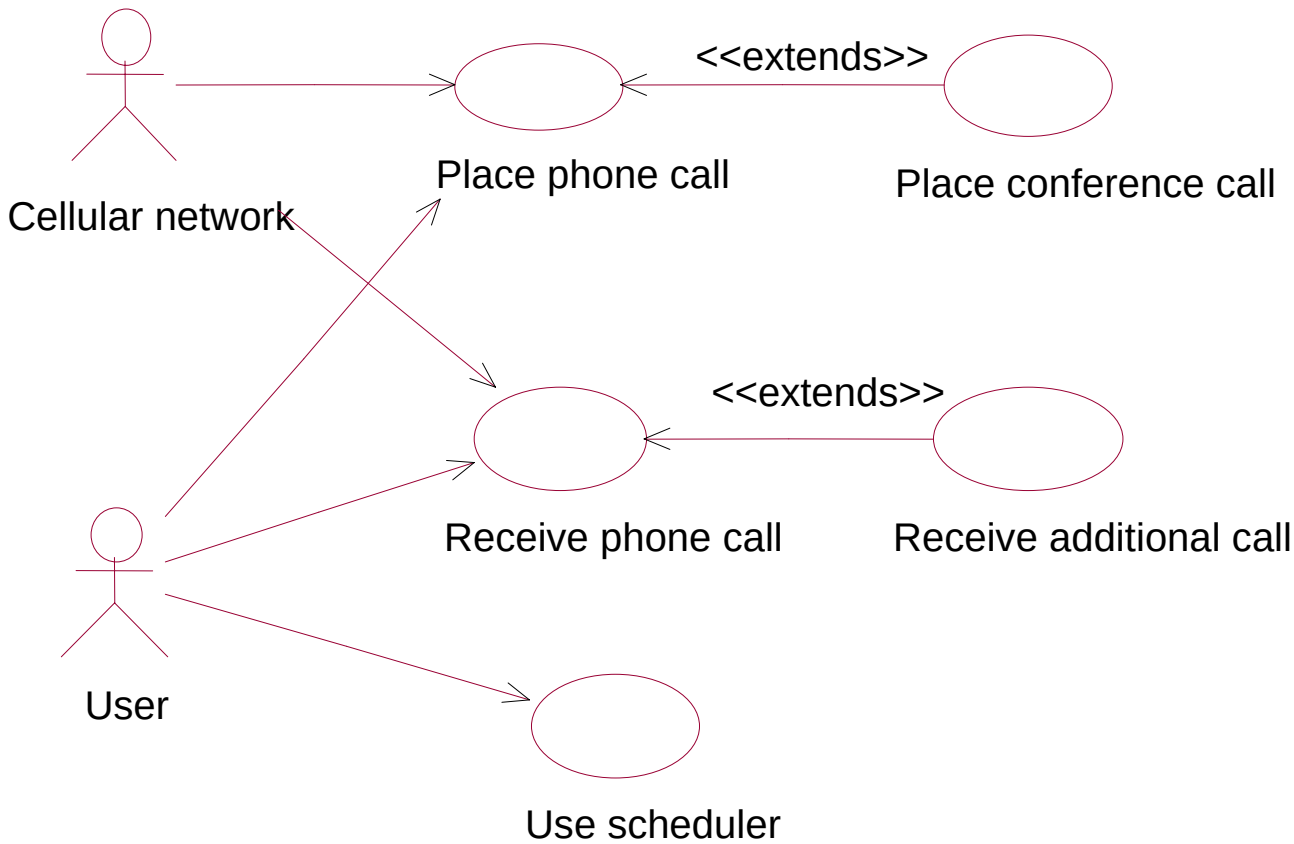
Example: Equity Trading System



Use Case “Extends”

- An extends relationship is used to show:
  - Optional behavior
  - Behavior that is only run under certain conditions, such as triggering an alarm
  - Several different flows which may be run based on actor selection
  - For example, a use case that monitors the flow of packages on a conveyer belt can be extended by a Trigger Alarm use case if the packages jam
  - An extends relationship is drawn as a directed (dashed) line with an arrowhead at the end closest to the base use case
  - Several differing schools of thought over “extends”
    - » For example, whether “extends” should return to the main flow

UC Diagram with Extends



Use Case Template

- Use case name/title
- Use case description
- Revision History
- Actors
- System Scope
- Goal
- Level
- Assumptions
- Relationships
  - Includes
  - Extends
  - Extension Points
- Precondition
- Trigger Events

Use Case Template (cont'd)

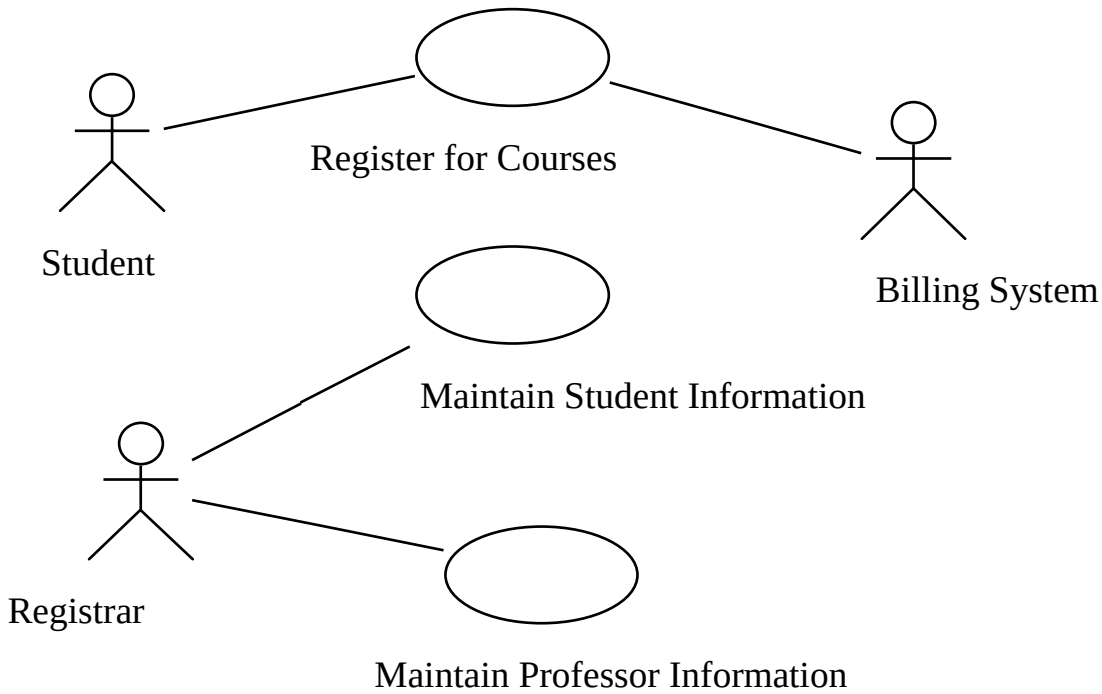
- Basic flow 1 - Title
  - Description (steps), etc.
- Post Conditions
- Alternative Flow 1 – Title
  - Description (steps)
- Alternative Flow 2 – Title
  - Description (steps)
- Alternative flow 3 – Title
  - Description (steps), etc.
- Exception Flow 1 – Title
  - Description (steps), etc.
- Activity Diagram
- User Interface
- Special Requirements
  - Performance Requirements
  - Reports
  - Data Requirements
- Outstanding Issues

Use Case Diagrams

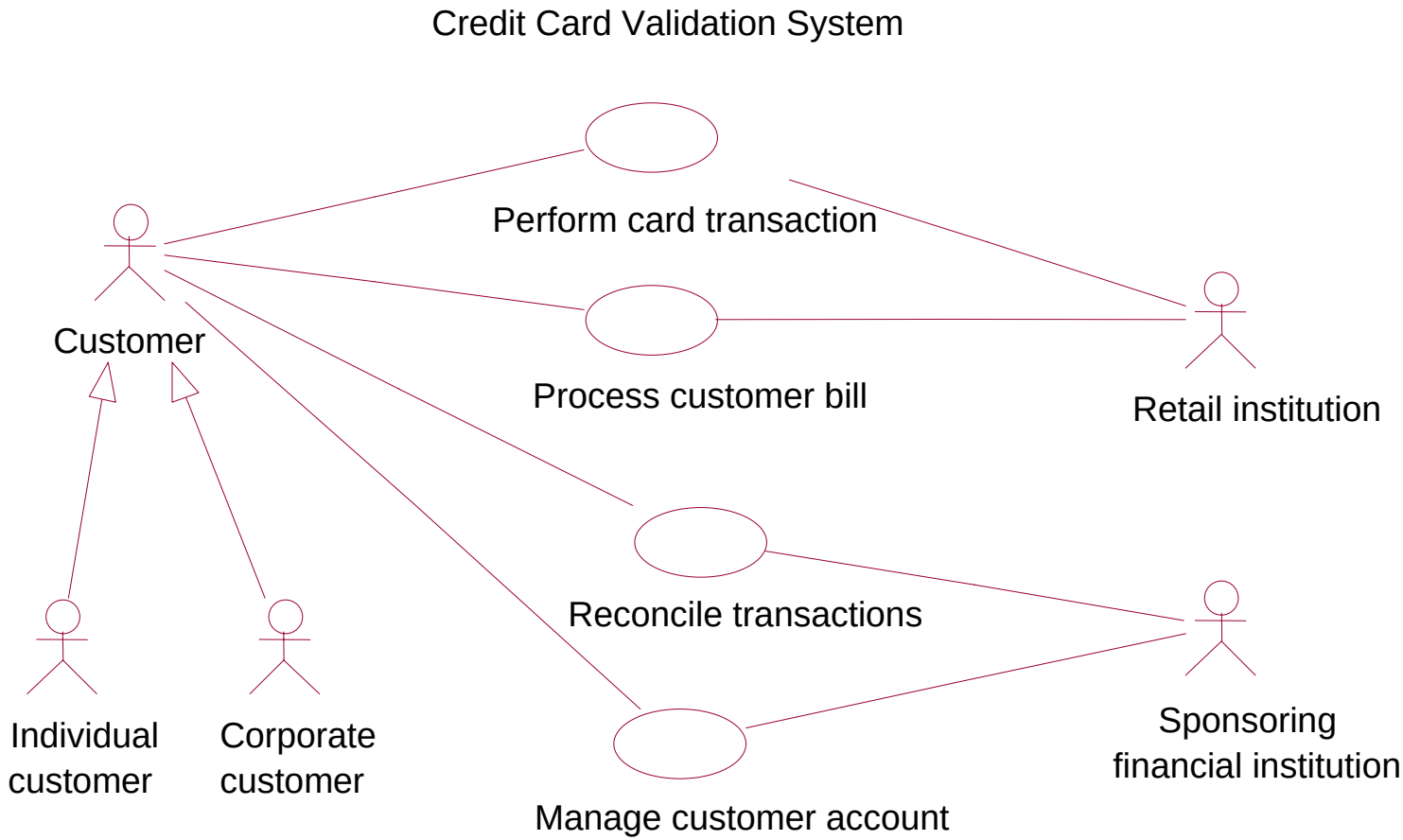
- A use case diagram is a graphical view of
  - Some or all of the actors, use cases, and their interactions identified for a system
- Each system typically has a Main Use Case diagram
  - A picture of the system boundary (actors) and the major functionality provided by the system (use case packages)
  - A Main use case diagram for each package
- Other use case diagrams may be created as needed
  - A diagram showing all the use cases for a selected actor
  - A diagram showing all the use cases being implemented in an iteration
  - A diagram showing a use case and all of its relationships

Example Use Case Diagram (1)

- A use case diagram for university course registration



Example Use Case Diagram (2)



What Is A Use-Case Model?

- A **use-case model** illustrates
  - The system's intended functions/behaviors (use cases)
  - Its immediate surroundings (actors)
  - Direct links between the system and its surroundings (diagrams)
  - Other related requirements documents may be linked
    - Security, performance, reusability, maintainability, other "ilities"
    - UI, reports, messages, outstanding items, actions, other project management/project tracking issues
- The same use-case model used in requirements
  - Is used in analysis, design, and test
  - Serves as a unifying thread throughout system development

The most important role of a use-case model is to communicate the system's functionality and behavior to the customer or end user

Benefits Of A Use-Case Model

- The use case model can be
  - used to communicate with the end users and domain experts
    - Provides buy-in at an early stage of system development
    - Insures a mutual understanding of the requirements
  - used to identify
    - Who will interact with the system and what the system should do
    - What interfaces the system should have
  - used to verify that
    - All behavioral (system-interaction) requirements have been captured
    - Developers have understood the requirements

Use Cases: Not so Fast…

- If you don't fully understand the ins and outs of use cases
  - It is easy to misuse them or turn them into "abuse cases"
- Ellen Gottesdiener
  - "*Top Ten Ways Project Teams Misuse Use Cases – and How to Correct Them*." The Rational Edge, June 2002 (Part I), July 2002 (Part II).
- Martin Fowler
  - "*Use and Abuse Cases*." Distributed Computing, April 1998.
- Doug Rosenberg
  - "*Top Ten Use Case Mistakes*." Software Development, February 2001.
- Susan Lilly
  - "*How to Avoid Use Case Pitfalls*." Software Development, January 2000.
- Kulak and Guiney
  - "*Use Cases: Requirements in Context*." Second Edition, Addison-Wesley 2003.

Ten Misguided Guidelines (Gottesdiener)

- Don’t bother with any other requirements representations
  - Use cases are the only requirements model you’ll need!
- Stump readers about the *goal* of your use case
  - Name use cases obtusely using vague verbs such as do or process
- Be ambiguous about the scope of your use cases
  - There will be scope creep anyway, so you can refactor your use cases later
- Include nonfunctional requirements and UI details in your use-case text
- Use lots of extends and includes in your initial use-case diagrams
  - This allows you to decompose use cases into itty bitty units of work

Ten Misguided Guidelines (Cont'd)

- Don't be concerned with defining business rules
  - you'll probably remember some of them when you design and code
- Don't involve subject matter experts in creating, reviewing, or verifying use cases
  - They'll only raise questions!
- If you involve users at all in use case definition, just "do it"
  - Why bother to prepare for meetings with the users?
- Write your first and only use case draft in excruciating detail
  - Why bother iterating with end users when they don't even know what they want
- Don't validate or verify your use cases
  - That will only cause you to make revisions and do more rework!

Summary

- System behavior is documented in a use case model
  - illustrates the system's intended functions (use cases)
  - its surroundings (actors)
  - relationships between the use cases and actors (use case diagrams)
- The most important role of a use case model is
  - to communicate the system's functionality and behavior
- Written in concise, simple prose, use cases are understandable by a wide range of stakeholders
- Each use case contains a flow of events
  - The flow of events is written in terms of what the system should do, not how the system does it
- A use case diagram is a graphical representation of some or all of the actors, use cases, and their interactions for a system

Book Recommendations

1. Writing Effective Use Cases, by Alistair Cockburn, Addison Wesley, 2000
2. Use Case Modeling, by Kurt Bittner, Ian Spence, Addison Wesley, 2002
3. Managing Software Requirements: A Use Case Approach, by Dean Leffingwell, Don Widrig, Addison Wesley, Second Edition, 2003
4. Applying Use Cases: A Practical Guide, by Geri Schneider, Jason P. Winters, Addison Wesley, Second Edition, 2001
5. Use Cases: Requirements in Context, by Daryl Kulak, Eamonn Guiney, Addison Wesley, Second Edition, 2003
6. Object Oriented Software Engineering: A Use Case Driven Approach, by Ivar Jacobson, Addison Wesley, 1992
7. Aspect Oriented Software Development with Use Cases, by Ivar Jacobson, Pan-Wei Ng, Addison Wesley, 2004