Object-Oriented Analysis and Design

CHAPTER 6: USE CASES, PART II

What will we learn?

How to define use cases, how to find them, how to construct them

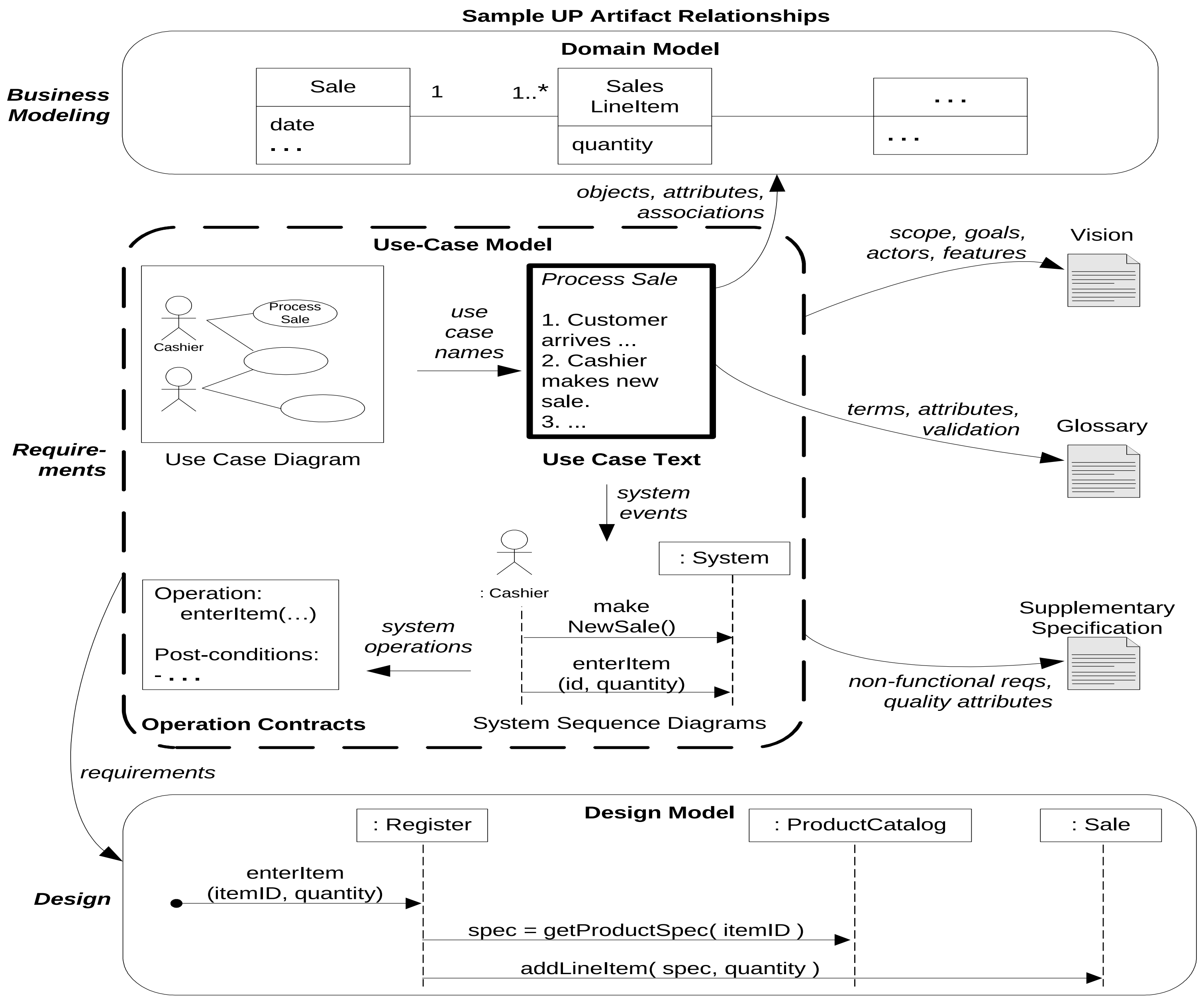
Applying UML use case diagrams

How to work with use cases in iterative methods

Relating use cases

Use Cases

- Use cases are text, not diagrams
  - We may construct diagrams from the text cases later
- These are created as stories, and functional requirements are derived from them
- Actors: something that exhibits behavior in the system (not just a person)
- Scenario: A specific sequence of actions and interactions between the actors and the system (use case instance)
  - One particular story of using the system
- Use Case (informal): A collection of related success and failure scenarios that describe an actor using a system to achieve a goal.
- Use Case (formal, RUP): A set of use-case instances (scenarios), where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor.


Sample UP Artifact Relationships
Domain Model
Business Modeling
Sale
date
. . .
1
1..*
Sales LineItem
quantity
. . .
. . .
objects, attributes, associations
Use-Case Model
Process Sale
Cashier
Use Case Diagram
use case names
Process Sale
1. Customer arrives ...
2. Cashier makes new sale.
3. ...
Use Case Text
Require-ments
scope, goals, actors, features
Vision
terms, attributes, validation
Glossary
system events
: System
: Cashier
make NewSale()
enterItem (id, quantity)
System Sequence Diagrams
Operation: enterItem(…)
Post-conditions: - . . .
Operation Contracts
system operations
Supplementary Specification
non-functional reqs, quality attributes
requirements
Design Model
Design
: Register
: ProductCatalog
: Sale
enterItem (itemID, quantity)
spec = getProductSpec(itemID)
addLineItem(spec, quantity)

How to Find Use Cases

One the hardest, but most important parts of the projects

All understanding of the requirements, and hence system design, will flow from here

Interact closely with customer/client/user

- Here customer means the person purchasing the software

One great strategy – take an actor perspective, role play

- Each use case should be designed to satisfy a goal or a primary actor

The NextGen POS System

Point-of-Sale (POS) system is application used to record sales and secure payment

- Checkout line at store

System includes hardware and software

Has interfaces to various service apps, like tax calculator and inventory control, and the system should work even if access to these external services is down (i.e., at least allow checkout with cash if the credit card processing interface goes down)

Needs to support multiple client-side interface types, like thin web-browser, touchscreen, wireless phone, etc.

We plan to sell this to many types of businesses which may have different business processing rules – we need flexibility and the ability to customize

Defining Use Cases: System Boundaries

- Choose a System Boundary
  - Determine the edges of the system being designed – is it the software, the software and hardware? Does it include the person using the system?
  - We are determining the Domain for the system
- Think about our POS Case Study … what is the system boundary?
  - In this case, the *system under design* is the software and hardware associated with it
  - The cashier, store databases, payment authorization services, etc. are outside the boundary of the system
  - Note that does not mean that they are not important – they are simply outside the boundary
- Having trouble identifying the boundary? Identify the external actors, and that should help define it

Use Cases: Identify Primary Actors and Goals

Actors and goals are usually defined together, because in order to identify the primary actor, you usually must know what his/her goal is

Ask the fundamental questions: How is the system being used, and by whom?

- Who starts the action, who provides what, who monitors, etc.
- Remember, actors can be other systems, like a payment system or a database – not necessarily human users!

This is usually done in the requirements workshop brainstorm sessions

One useful approach: create a table, list all actors and goals, and then identify the primaries

- Remember, the primary actor is the one that has a need or goal fulfilled by the system
- Secondary (supporting) actors provide a service to the system
- Offstage actors have an interest in the behavior of the system, but are not primary or supporting

Note the primary actor may be defined based upon the choice of system boundary

Use Cases: Identify Actors, Goals

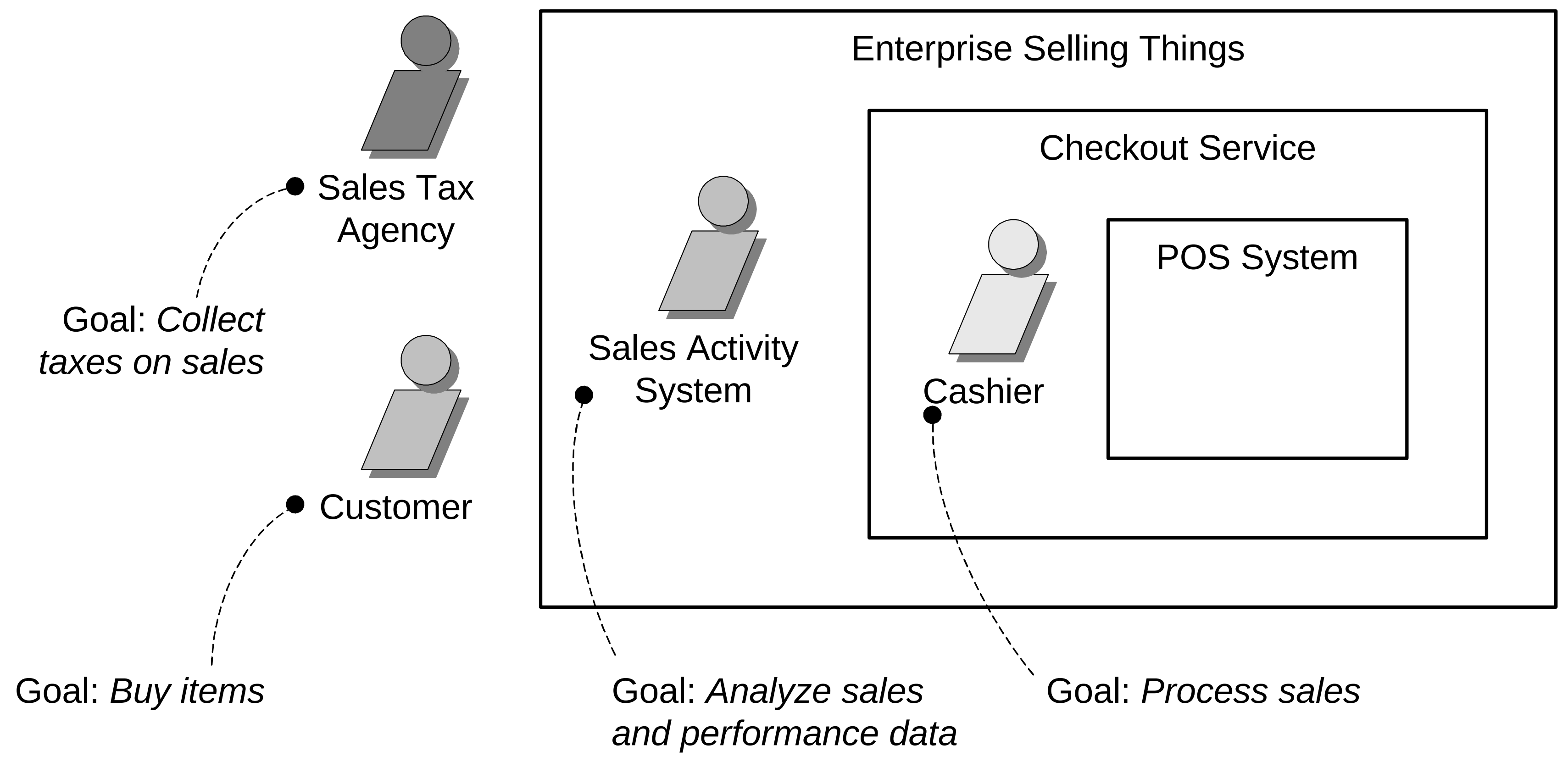
In Use Case Modeling, in order identify the actors and their goals, we need to understand the system from the actors' perspectives

- Who is using the system? What are they doing with the system?

Best approach to this is to identify the actors and then ask "What are your goals? Are they measurable?"

- This gets to the heart of what the stakeholders really want from the system
- Also helps to avoid slipping into design too early – if we concentrate on the end goals, everyone stays open to new solutions to meet those goals

Use Cases: Primary Actors and System Boundaries



POS Case Study: Primary Actor

The previous diagram implies the cashier is the primary actor

What about the customer?

- Customer is an actor, but not primary. Why not?

Look at system boundary: For this case study, the system is the POS system. Unless self checkout, the customer's goal is not the principle goal here

- The cashier's goal (look up prices, get the total, process payment) is the main goal to be fulfilled by the system. **The system is being designed for use by the cashier – to meet his/her primary goal.**

Goes back to the basic domain definition – what are we designing?

Note: If the system under design is the entire sales enterprise system (inventory, sales tracking, POS), then the customer is the primary actor

Use Cases: Putting it all Together

- First, define the system boundary. This involves defining the limits of what services the system will provide
- Next, define the actors and their goals, including the primaries
  - Role play, actor/goal tables
  - Can also do this by analyzing events that may take place, and then identifying the actor generating the event and the goal of the actor
- Finally, define the use case
  - Generally do one use case for each user goal
  - Use a name that describes the goal, start with a verb
  - Often the CRUD goals are collapsed into one goal called *Manage X*, where X is whatever is being managed (e.g., “Manage User Accounts”)

Use Cases: How Big or Small?

It really depends on context

Generally, the use case should define a task performed by one actor in one place at one time, in response to an event.

If the use case is growing to many pages in length, consider creating sub-tasks

If the use case is only one sentence or step, probably too small and not worth exploring

Is there a basic business value in the use case?

- Which actors are impacted and how?
- What is the impact to the overall business?

Applying UML: Use Case Diagrams

Note that diagrams are secondary in use case development, and are often used just as an easy documentation tool. Use Cases are carefully crafted <u>text</u>.

Good to draw these when working out the actors and goals, and the actor-goal list

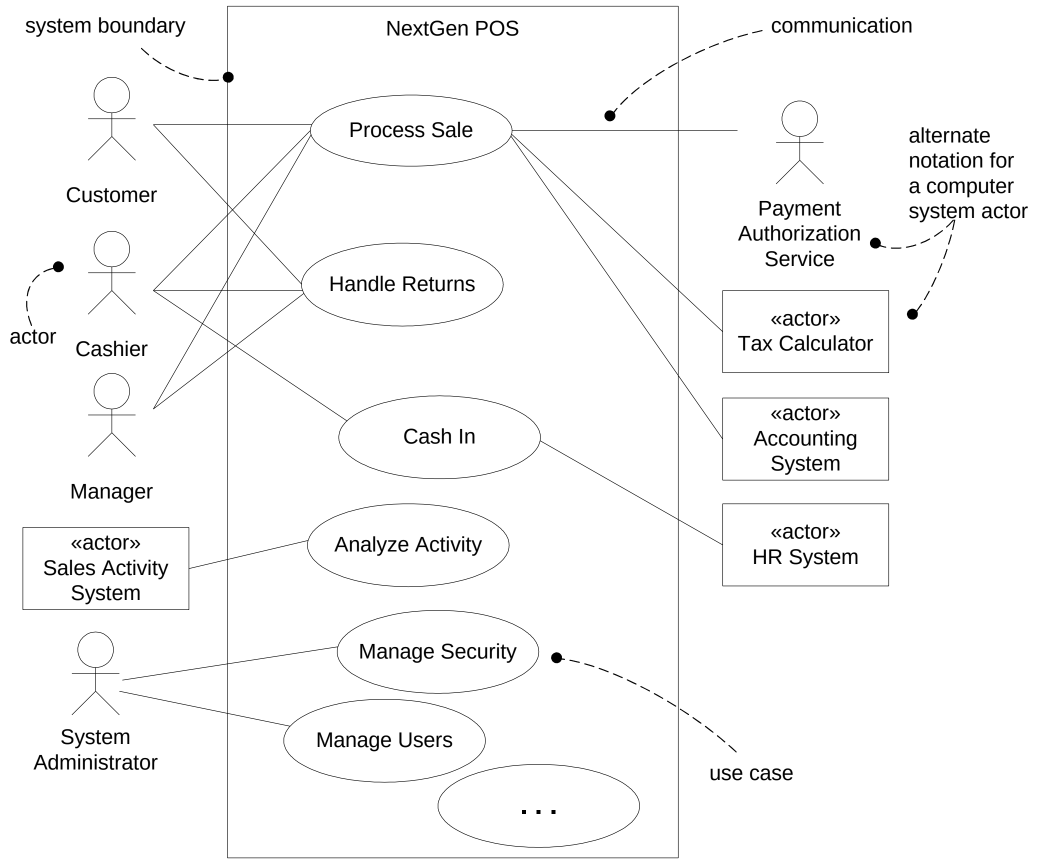
Can also be very helpful when deciding system boundaries – decide what is inside the system and what is outside

Use case diagrams are tools, and should be used to help understand the system and use cases under development

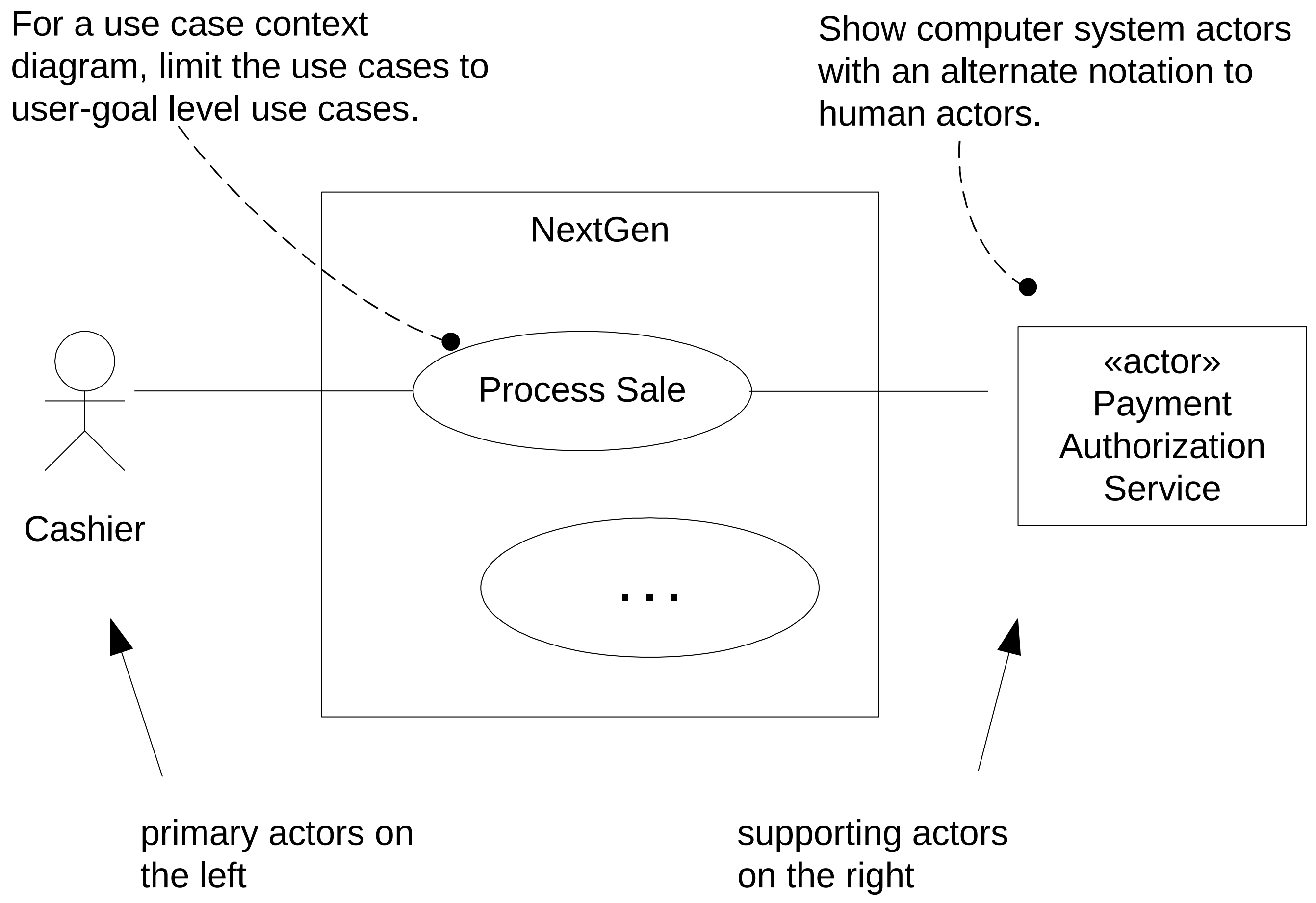
Never accept a set of use case diagrams as the "official" Use-Case Model for a project!

- They are much too vague, and you are leaving yourself open to misunderstanding later down the road
- Get it in writing!

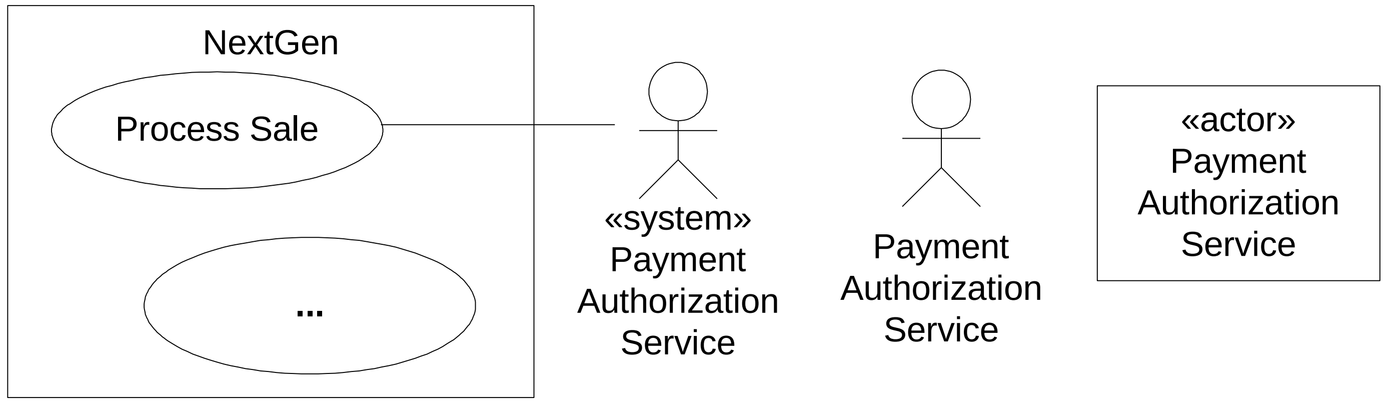
Example: POS


system boundary
NextGen POS
communication
Customer
Cashier
actor
Manager
Process Sale
Handle Returns
Cash In
Analyze Activity
Manage Security
Manage Users
. . .
Payment Authorization Service
alternate notation for a computer system actor
«actor» Tax Calculator
«actor» Accounting System
«actor» HR System
«actor» Sales Activity System
System Administrator
use case

UML Use Case Diagramming


For a use case context diagram, limit the use cases to user-goal level use cases.
Show computer system actors with an alternate notation to human actors.
NextGen
Process Sale
. . .
Cashier
«actor» Payment Authorization Service
primary actors on the left
supporting actors on the right

UML Use Case Diagramming (alt)


NextGen
Process Sale
...
«system» Payment Authorization Service
Payment Authorization Service
«actor» Payment Authorization Service

Some UML alternatives to illustrate external actors that are other computer systems.

The class box style can be used for any actor, computer or human. Using it for computer actors provides visual distinction.

Use Cases – Some Observations

Don’t design, don’t suggest coding. Think about the user story, the actors and goals

UML can also be used to create activity diagrams, which are workflow diagrams. We will discuss later

Avoid detailed function lists – this is old school, tedious, and not efficient. Write user stories.

- The Use-Case Model artifact captures the functional requirements

Remember, there are other UP artifacts that may still collect requirements: non-functional requirements, domain rules, etc. may be captured in the Supplementary Specification artifact.

The Vision document artifact *may* contain a list of high level system functions, as these are necessary to define the system scope

Case Two: The Monopoly Game

Software version of Monopoly game

This will run as a simulation; the user will configure the game (i.e. indicate the number of players, etc.), start the simulation, and let the game run to its conclusion.

A trace log will be created to record each move a player makes

Monopoly Game Use Case

This is an example of a project where most of the requirements are NOT captured by the use cases!

In this case, there is only one (simple) use case …

- The player starts the game (simulation)

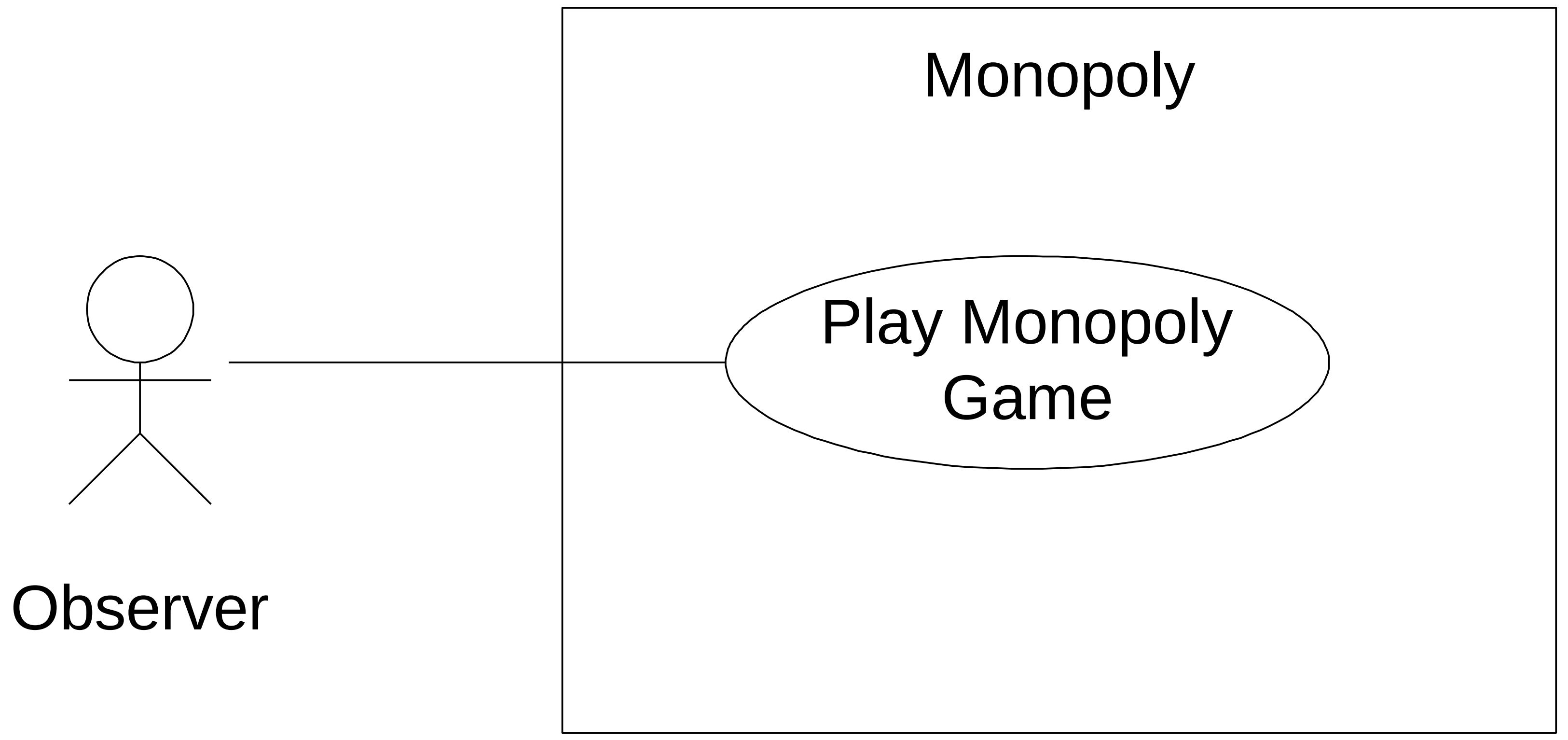
Are there no requirements?

- The *functional* requirements are simple, and basically captured in the project description on the last slide
- This system will have many *rules* requirements – namely, the rules of the Monopoly game.

These requirements, like most *business logic* requirements, would be captured in the Supplementary Specification (we will talk about this later).

- This type of requirement is not directly functional to the user, so usually not captured in the use case

Monopoly Game: Use Case Diagram



Monopoly Game Use Case

USE CASE UC1: Play Monopoly Game

Scope: Monopoly application

Primary Actor: Observer

Stakeholders and Interests: Observer: Wants to easily observe the output of the game simulation

Main Case Scenario:

1. Observer requests new game initialization, enters number of players
2. Observer starts play.
3. System displays game trace for next player move (see domain rules, and "game trace" in glossary for trace details)
4. Repeat step 3 until a winner is decided or the Observer cancels the simulation

Monopoly Game Use Case

USE CASE UC1: Play Monopoly Game (cont.)

Extensions:

*a. At any time, System fails:

(to support recovery, System logs after each completed move)

1. Observer restarts System.
2. System detects prior failure, reconstructs state, and prompts to continue.
3. Observer chooses to continue (from last completed player turn).

Special Requirements:

- Provide both graphical and text trace modes.

Use Cases and Iterative Methods

UP is use-case driven development

Functional requirements are primarily captured in the use cases (i.e. the Use-Case Model) – this means you would not expect to find many functional requirements elsewhere

Critical to planning iterations

- Planning usually involves selecting a use case to work on, or enhancing an existing use case

In UP, the main goal for each team is to design objects which, when collaborating together, will *realize* a use case. This is true of Agile methods in general.

Use cases often give valuable material for the creation of user manuals

Use cases also provide valuable ideas for testing

Use cases often developed in Requirements Workshops held with the customer

Requirements Workshops

Often attended by System Analyst, developer, software architect, customer, end user, etc.

Held throughout Inception/Elaboration Phases

Inception:

- Start by identifying stakeholders and goals, create actor-goal table mentioned earlier
- Then create a list of use cases *by name only*, and maybe write brief descriptions for a few of the most critical ones
- Finally, by the end of Inception, a few key use cases will be identified as high risk and critical to the core architecture; these are written out in fully dressed form

Elaboration:

- At regular times during each iteration, refine the existing fully dressed use cases based upon feedback
- Create fully dressed versions of remaining use cases, add more use cases as needed

Iterative Development: Disciplines, Phases, and Artifacts

In UP, each artifact is owned by a discipline

- The Requirements discipline owns the Use Case Model artifact

The artifacts are developed over the course of the phases of the project, i.e. Inception, Elaboration, Construction, and Transition

Some artifacts are started later than others, some a revised in multiple phases

This is common in Agile methods in general – don’t “freeze” requirements

Iterative Development: Disciplines, Phases, and Artifacts

Discipline: Requirements. **Artifact:** Use-Case Model

During Inception, hold initial requirements workshop to identify high level use cases (brief descriptions); identify maybe 10% that are high-risk and architecturally important, and do a deep dive to define those

During the iterations of the Elaboration Phase, hold requirements workshops near the end of each iteration of expand upon use cases (based upon feedback received on developed code), and add details to high level use cases

By the end of the Elaboration Phase, 80-90% of the use cases should be clear and written in detail. The rest will be fleshed out during Construction Phase.

Iterative Development: Disciplines, Phases, and Artifacts

Discipline	Artifact	Inception	Elaboration	Construction	Transition
Business Modeling	Domain Model		S, R		
Requirements	**Use-Case Model** Vision Supp Spec Glossary	S S S s	R R R R		
Design	Design Model SW Arch Doc		S S	R	

S – Start, R - Refine

Takeaways from Chapters 6

Understand how to create use cases by defining system boundaries, identifying the actors and goals

Understand when and how use cases are created throughout the iterative process

Next …

How to relate use cases (Chapter 30)

Other requirements, and how to find them (Chapter 7)