Object-Oriented Analysis and Design

Isara Anantavrasilp

Lecture 4-5: Use Cases and Scenarios

Writing Requirements

- Users have goals and requirements of the system must support these goals
- During and interview or a workshop, when the user explains their goals, you might want to ask them to elaborate
  - Who is the user?
  - What does he have to do?
  - What are the inputs or results?
- Or, you can simply write a story about it

System-Usage Scenario

Process Sale: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.

- Do you understand what the system should do?
- Do we miss anything?

More **scenarios**

Scenario

- **Scenario**: A specific sequence of actions and interactions between actors and the system
  - Exact description of one specific trail of interactions
  - Uses the terms and language of the user / customer
  - Only describes interactions between the user and the system, no system internal behavior
  - Sometimes it is called a **use-case instance**

More Usage Scenarios

Process Sale:

Main Success Scenario:

A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item…

Alternate Scenarios:

- If the customer wants to pay by credit card, but his card is rejected, inform the customer and let him select other methods of payment.

More Usage Scenarios (2)

Process Sale:

Alternate Scenarios:

- If the item identifier is not found in the system, notify the Cashier and suggest manual entry of the identifier code (perhaps the code is corrupted).
- If the system detects failure to communicate with the inventory system, store the sales record within the POS. Once the communication can be re-established, send the sales data to the inventory system
- If the Cashier records a purchased item twice…

Happy or Sad.. or Sadder

- In software design, we can separate usage cases into happy and sad ones
  - **Happy path**: Default scenario comprises the sequence of activities executed if everything goes as expected, without mentioning exceptional or error conditions
  - **Sad path**: Exceptional or error-handling usage cases, e.g., invalid password, unknown login
  - **Bad path**: Sadder or error path. This is the case where the system is not capable to handle some cases, e.g., using Thai characters in an ASCII text-field or extra long user name that is not supported by the system

In use-case design, we do not care about bad path

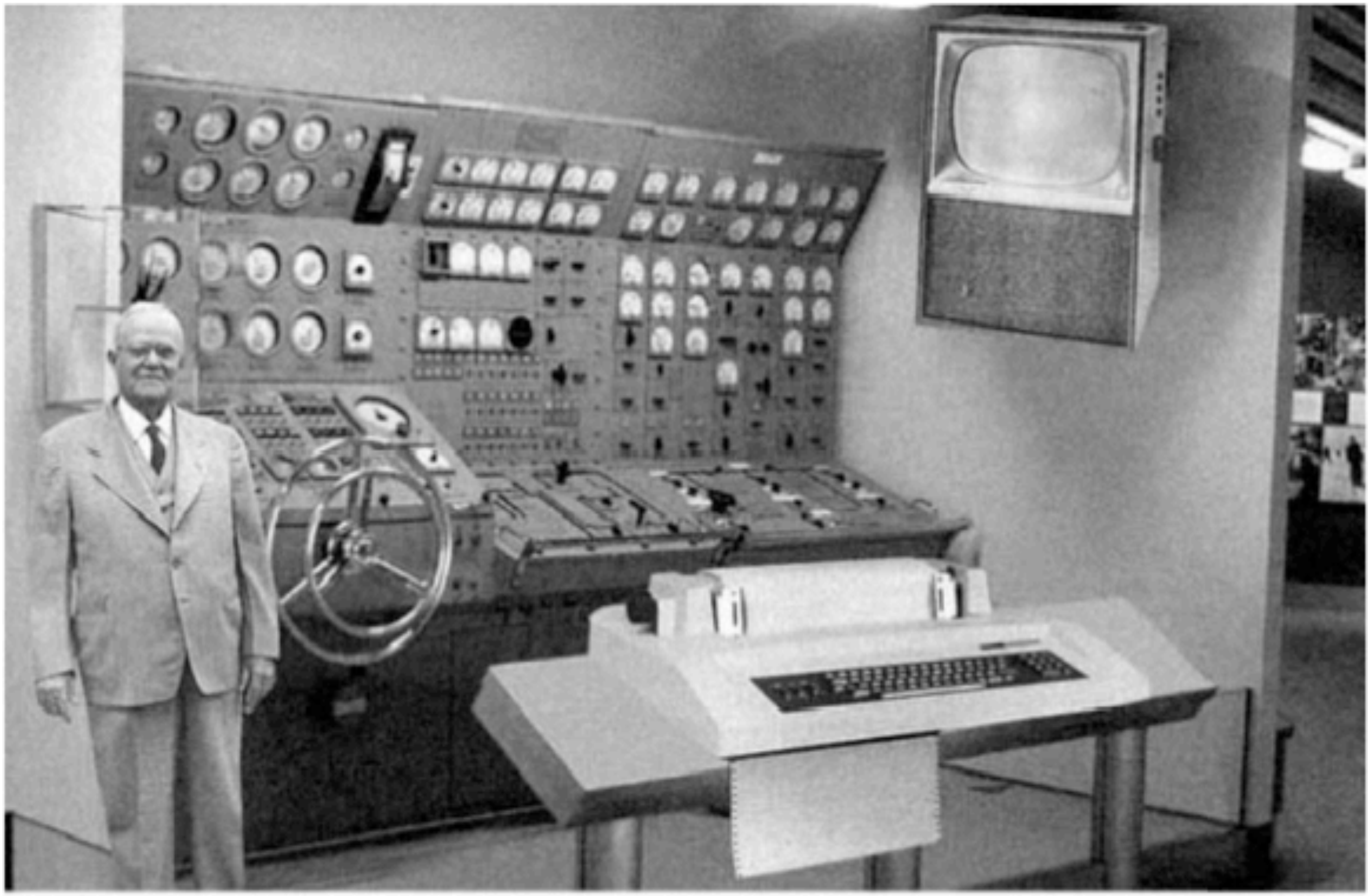
Scenario-Based Design

- **Scenario-Based Design**: Design paradigm that focuses on one sequence of actions (scenario)
  - Focuses on concrete descriptions and particular instances
  - It is work driven not technology driven
  - It is open-ended
  - It is informal
  - Is about envisioned outcomes

Type of Scenarios

- **As-is scenario**: Describes a current situation. Usually used in re-engineering projects. The user describes the system
- **Visionary scenario**: Describes a future system. Usually used in greenfield engineering and reengineering projects

A Visionary Scenario (1954): The Home Computer in 2004



Scientists from the RAND Corporation have created this model to illustrate how a "home computer" could look like in the year 2004. However the needed technology will not be economically feasible for the average home. Also the scientists readily admit that the computer will require not yet invented technology to actually work, but 50 years from now scientific progress is expected to solve these problems. With teletype interface and the Fortran language, the computer will be easy to use.

Heuristics for Finding Scenarios

- Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?
- However, don't rely on interviews and questionnaires alone
- Insist on task observation if the system already exists (interface engineering or reengineering)
  - Ask to speak to the end user
  - Expect resistance and try to overcome it.

Scenario Example: Warehouse on Fire

- Bob, a field officer, driving down main street in his patrol car notices smoke coming out of a warehouse. His partner, Alice, reports the emergency from her car.
- Alice enters the address of the building into her wearable computer, a brief description of its location (i.e., north west corner), and an emergency level.
- She confirms her input and waits for an acknowledgment.
- John, the dispatcher, is alerted to the emergency by a beep of his workstation. He reviews the information submitted by Alice and acknowledges the report. He allocates a fire unit and sends the estimated arrival time (ETA) to Alice.
- Alice received the acknowledgment and the ETA.

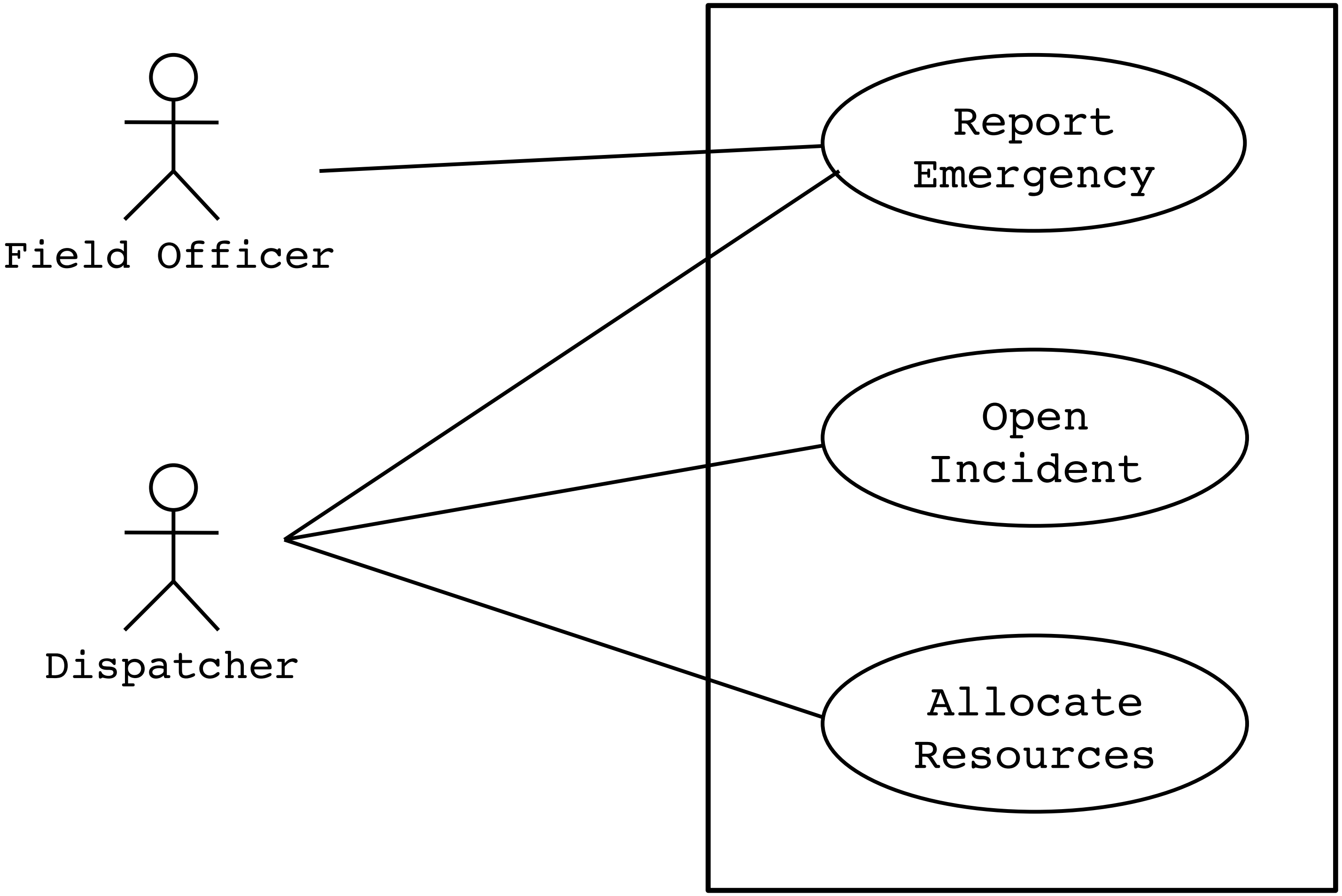
Observations about Warehouse on Fire Scenario

- Concrete scenario
  - Describes a single instance of reporting a fire incident.
  - Does not describe all possible situations in which a fire can be reported.
- Participating actors:
  - Bob, Alice and John

After the scenarios are formulated..

- Find all the use cases in the scenario that specify all instances of how to report a fire
  - Example: “Report Emergency“ in the first paragraph of the scenario is a candidate for a use case
- Describe each of these use cases in more detail
  - Participating actors
  - Describe the precondition
  - Describe the flow of events
  - Describe the post-condition
  - Describe exceptions
  - Describe nonfunctional requirements

Use Case Model for Incident Management



Use Cases

- **Use case**: A set of use-case instances, where each instance of a sequence of actions a system performs that yield an observable result of value to a particular actor
- A use case represents a functionality of a system from an actor's point of view
  - Happy or sad, it is the same functionality
  - It emphasizes on user's goal
- Use cases are requirements!

Writing Use Cases

- [Larman 04] argues that use case diagram does not capture all details needed
  - ***Writing*** use cases is the best way to capture the requirements
- There are three ways to write a use case
  - **Brief**: One paragraph summary, usually of the main success scenario

    Early requirement analysis to get an idea of the functionality
  - **Casual**: Multiple paragraph that cover various scenarios

    Same as brief
  - **Fully dressed**: All steps and variations are written in detail

    Complex/Important scenarios that need to be explained

Brief Use Case Example

Process Sale: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.

Casual Use Case Example

Process Sale:

Main Success Scenario:

A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item…

Alternate Scenarios:

- If the customer wants to pay by credit card, but his card is rejected, inform the customer and let him select other methods of payment.

Fully Dressed Use Case Sections

Use Case Section	Description
Use Case Name	Name of the use case, should start with verb
Scope	System under consideration/design
Level	User-goal or sub-function (Called by user or other use cases?)
Primary Actor	The ones who call on the system
Stakeholders and Interests	Who cares about this use case and what do they want?
Preconditions	What must be true on start and worth telling
Success Guarantee	What must be true on successful completion (Post-conditions)
Main Success Scenario	Happy path scenario
Extensions	Alternate scenario of success or failure
Special Requirements	Related non-functional requirements
Miscellaneous	Other stuff

Fully Dressed Use Case Sections (2)

- **Scope**: Scope of the system(s) under design
  - Use case describes a software system or part of a system
  - Use case describes a business process
- **Level**: Depth of the use case
  - User-goal level: Use case that is activated by an actor
  - Subfunctional level: Use case that is a sub-step of another use case. Maybe shared by many use cases

Fully Dressed Use Case Sections (3)

- **Primary Actor**: The actor that calls upon the system service (activates the use case) to fulfill a goal
- **Stakeholders**: Who cares/benefits from this use case? What exactly do they want?
  - Cashier may activate the service to record correct and quick sales items
  - Sales manager may want to see sales reports
  - Your system features should satisfy all stakeholders' interests

Fully Dressed Use Case Sections (4)

- **Preconditions**: State that must ***always*** be true before a scenario can begin
  - The preconds are not tested within the use case. They are assumed to be true ***before*** entering the use case
  - If the preconds are obvious, do not write them down
- **Success Guarantee (Post-Conditions)**: State that must be true on successful completion of the use case (either main or alternative case)
  - The results or by-products of the tasks that are done during the happy scenarios
  - The guarantees should meet all stakeholders’needs

Fully Dressed Use Case Sections (5)

- **Main Success**: Happy-path scenario or basic flow of the use case
  - Describe the main functionality and steps of the use case
  - Usually does not include exceptions or branches (those are included in the Extensions section
- Example:
  1. Customer arrives at a POS checkout with items to purchase
  2. Cashier starts a new sale
  3. Cashier enters item identifier
  4. The system shows the item name on the screen

  Cashier repeats steps 3-4 until all items are entered

  5. ...

Fully Dressed Use Case Sections (5)

- **Extensions**: Indicate all other scenarios corresponding to the use case
  - Branches from the main scenario
  - Can be denoted with branched step (1,2,3…)
  - It should be branched of detectable condition
    - The system cannot connect to the database (detectable)
    - The database server is down (***inferred*** from the detected issue)
  - Consume most of your use case description
- Example:

  3a. Invalid identifier:

  1. System signals error and rejects entry

  3b. Product cannot be sold:

  1. System signals error and rejects entry

Fully Dressed Use Case Sections (6)

- Sometimes a step in the use case may refer to (or perform) another use case
- You can underline the referred use case name

  3a. Invalid identifier:

  1. System signals error and rejects entry
  2. Cashier performs Search Product Code to obtain actual product ID and price

- In practice, you can specify in your use case text how ever you like for example,

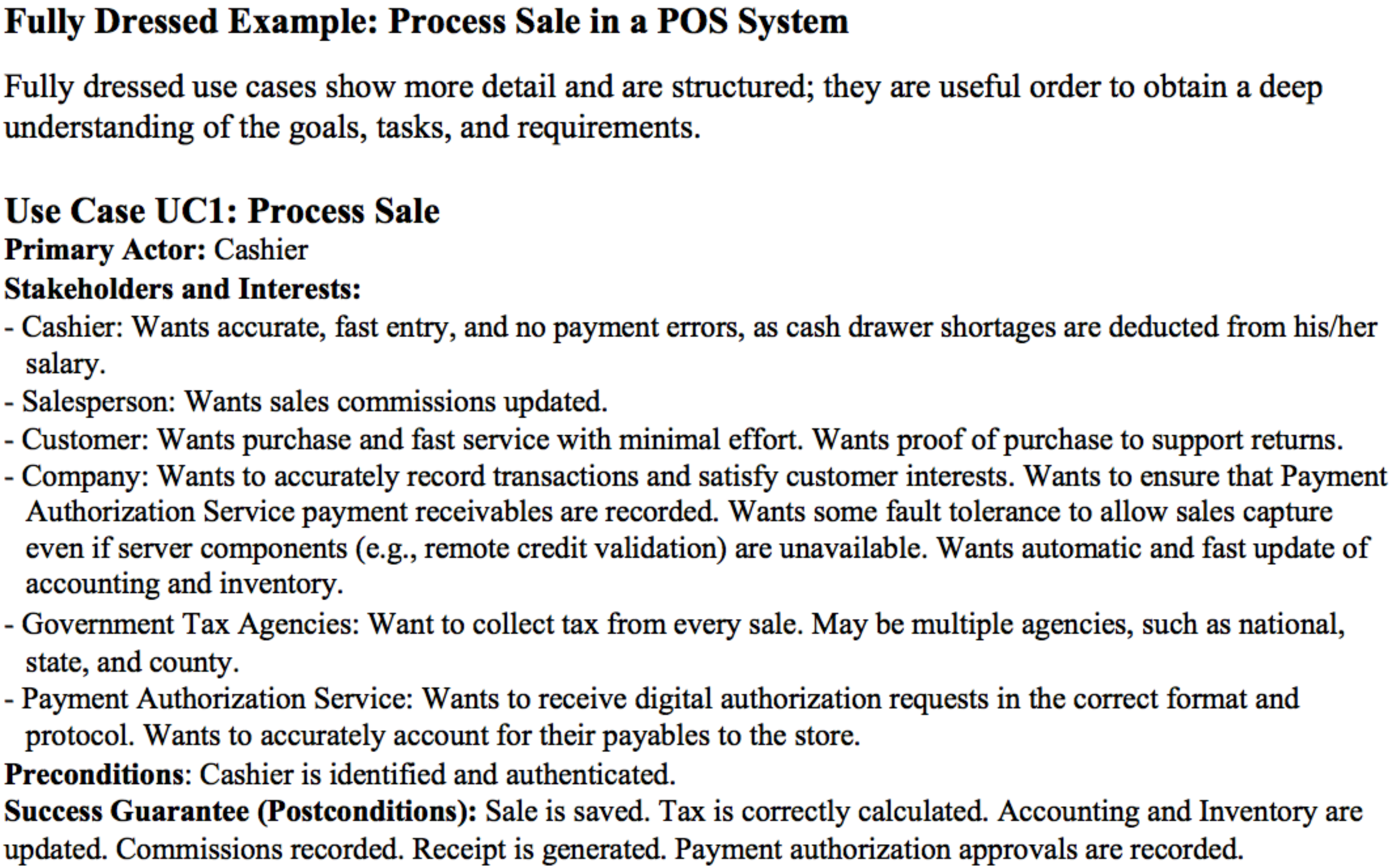
  Cashier performs `Search Product Code` to obtain actual product ID and price

- You have to explain your notation to the reader and be consistent with it

Fully Dressed Use Case Sections (7)

- **Special Requirements**: Non-functional requirements that are related to the use case.
  - When you first write the use case, these requirements should be with the use case
  - However, all requirements could be moved and kept together in a separated section
- Example:
  - The POS UI must be readable from 1 meter
  - The screen is a capacitive touch screen
  - Credit authorization response within 30 seconds 90% of the time

Fully Dressed Use Case Example



Fully Dressed Example: Process Sale in a POS System

Fully dressed use cases show more detail and are structured; they are useful order to obtain a deep understanding of the goals, tasks, and requirements.

Use Case UC1: Process Sale

Primary Actor: Cashier

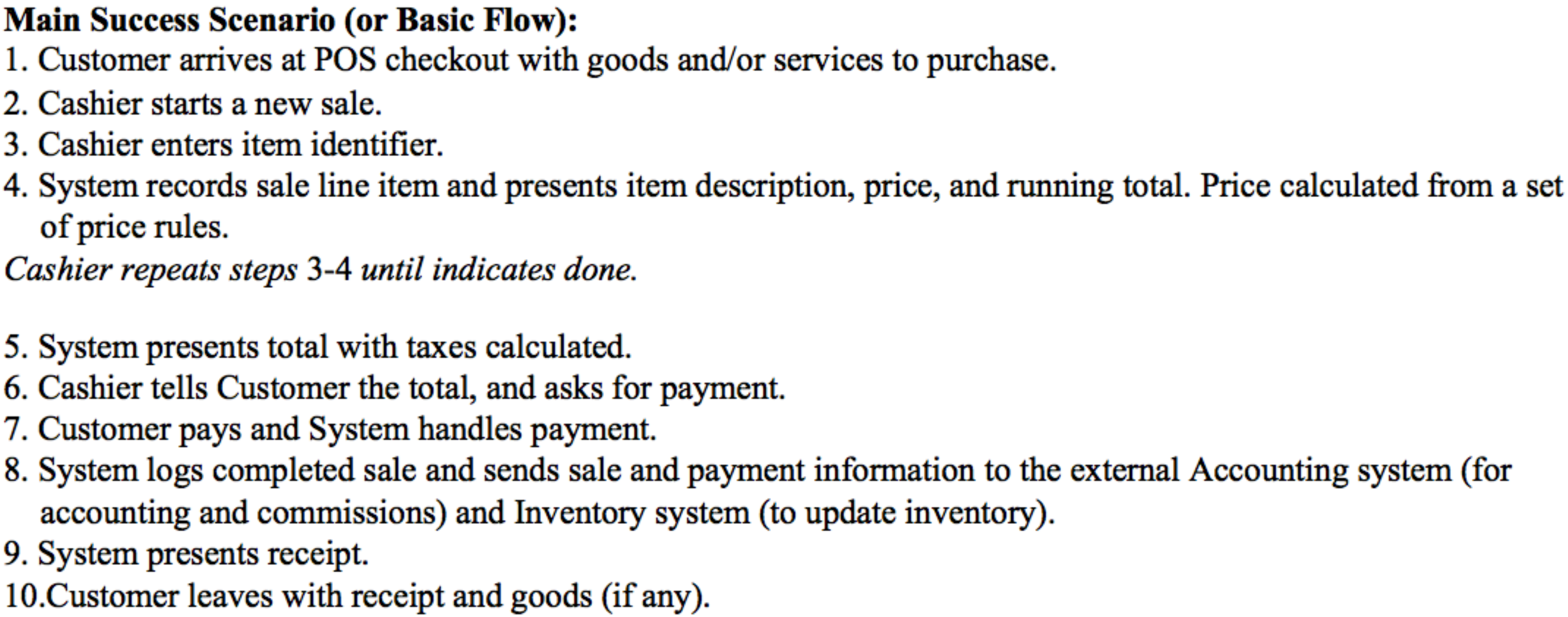
Stakeholders and Interests:

- Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.
- Salesperson: Wants sales commissions updated.
- Customer: Wants purchase and fast service with minimal effort. Wants proof of purchase to support returns.
- Company: Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.
- Government Tax Agencies: Want to collect tax from every sale. May be multiple agencies, such as national, state, and county.
- Payment Authorization Service: Wants to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.

Preconditions: Cashier is identified and authenticated.

Success Guarantee (Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

Fully Dressed Use Case Example (2)



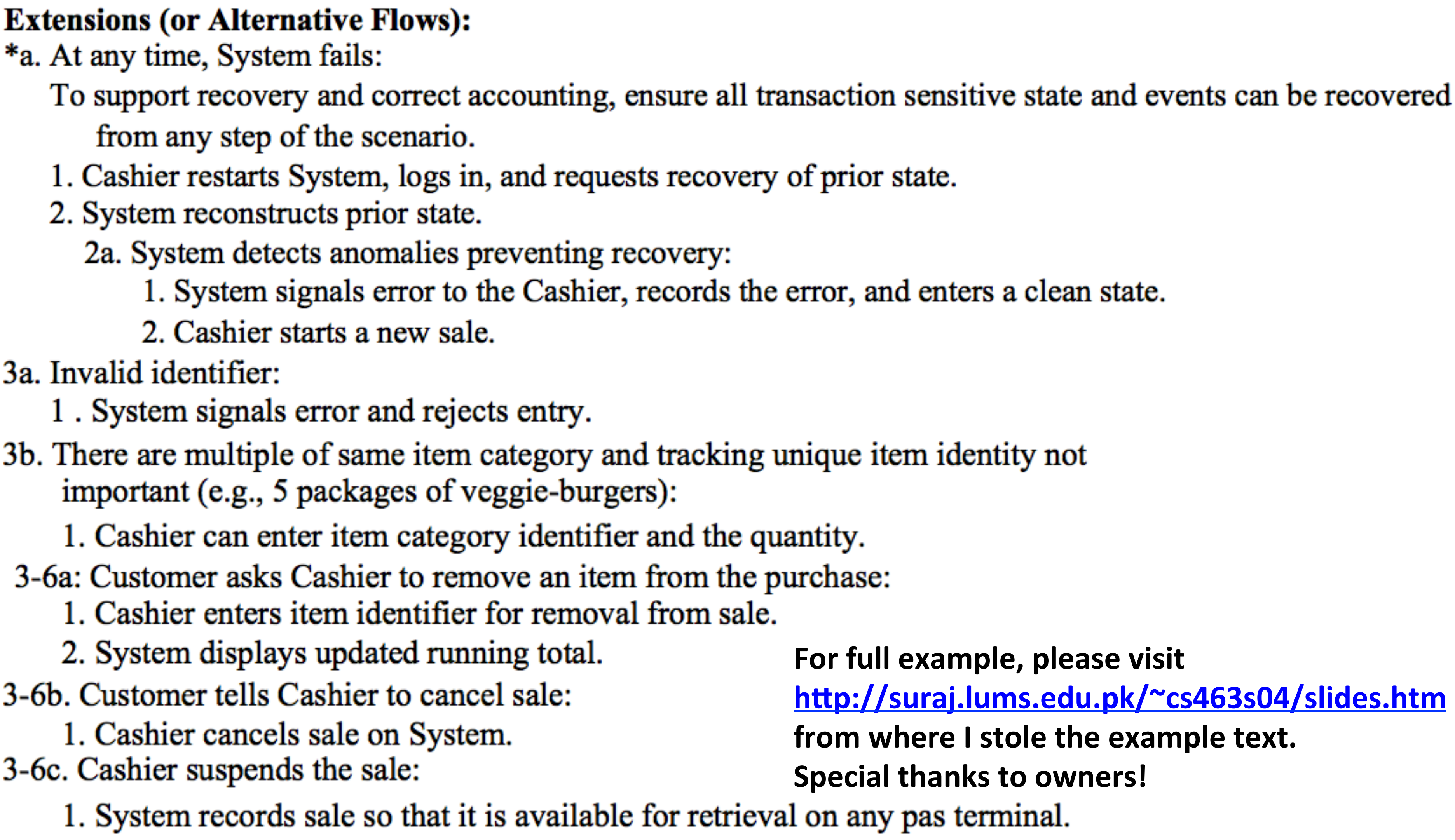
Main Success Scenario (or Basic Flow):

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 *until indicates done.*

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Fully Dressed Use Case Example (3)



Extensions (or Alternative Flows):

*a. At any time, System fails:

To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.

1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.

2a. System detects anomalies preventing recovery:

1. System signals error to the Cashier, records the error, and enters a clean state.
2. Cashier starts a new sale.

3a. Invalid identifier:

1 . System signals error and rejects entry.

3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers):

1. Cashier can enter item category identifier and the quantity.

3-6a: Customer asks Cashier to remove an item from the purchase:

1. Cashier enters item identifier for removal from sale.
2. System displays updated running total.

3-6b. Customer tells Cashier to cancel sale:

1. Cashier cancels sale on System.

3-6c. Cashier suspends the sale:

1. System records sale so that it is available for retrieval on any pas terminal.

For full example, please visit
http://suraj.lums.edu.pk/~cs463s04/slides.htm
from where I stole the example text.
Special thanks to owners!

The Two-Column Format

Use Case	**Process Sale**	
Primary Actor	Cashier	
Precondition	Cashier is identified and authenticated	
Post-condition	Sale is saved. Accounting and inventory updated	
Main Success Scenario	**Actor Input**	**System Response**
	1. Customer arrives..	
	2. Cashier starts..	
		3. Show item names…
Extension	**3a. Invalid identifier**	
	Actor Input	**System Response**
		1. System signal errors..
	2. Perform search..	
	3b. Product cannot be sold	
		1. System signal errors..

PROTIP

Effective Use Case Model

- When you start, keep the UI out of the context
  - Focus on the user intention or goals rather than the concrete actions that that must be done
  - Example:

Cashier enters item identifier

vs

Cashier scans product barcode

  - Writing concrete steps is useful as a guideline for UI design and should be conducted during later design phase

PROTIP

Effective Use Case Model (2)

- Write short concise use case: TL;DR
- Write black-box use case
  - Do not describe the internal workings of the system
  - Focus on the responsibilities or functions of the system, i.e. **what** the system does, not **how**
- Remember that the use case is system functionality in the actor's point of view
  - Focus on actors' goals and needs
  - Try to find what they need the most

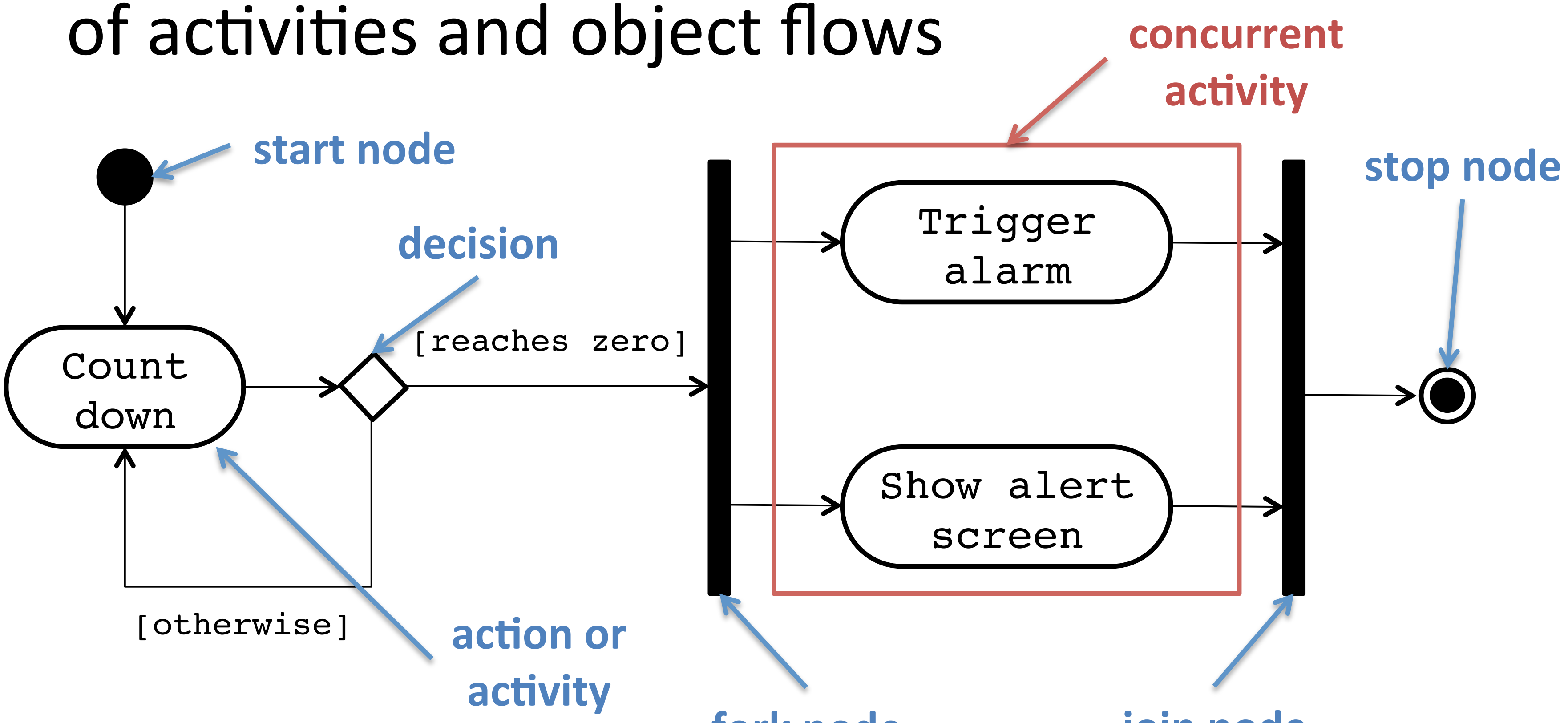
THIS IS WHERE THE CLASS ENDS.
COME BACK NEXT WEEK!

Describing Use Case with Diagram

- A work process or **workflow** can be described by **UML activity diagram**
  - Both computational or organizational processes
- **Workflow**: Sequence of processing steps that completely handles one business transaction or customer request
- Instead of textual content, we can use such diagram to describe use case steps
- You can also use both to explain a flow
- You can also skip the entire step-wise explanation altogether (brief style)

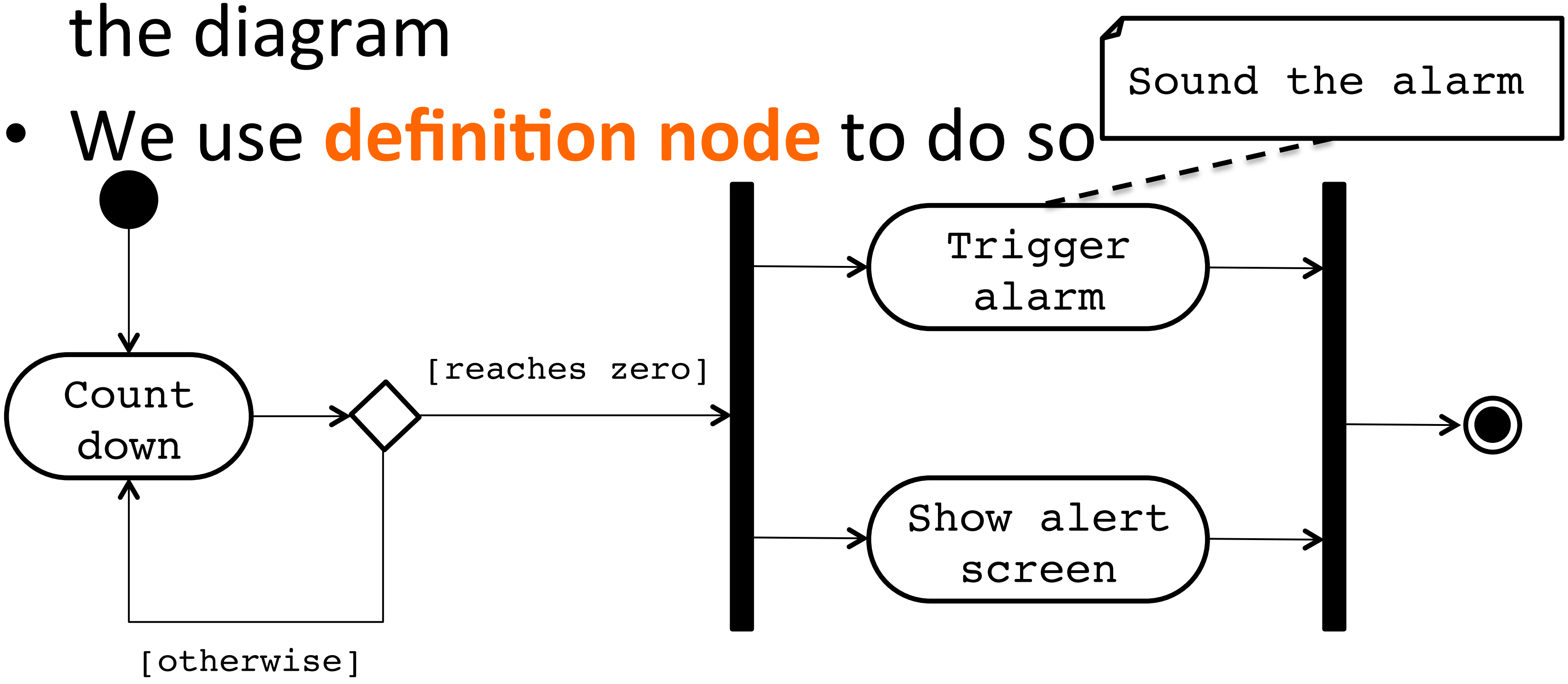
Activity Diagram

- Represents the sequencing and coordinating of activities and object flows



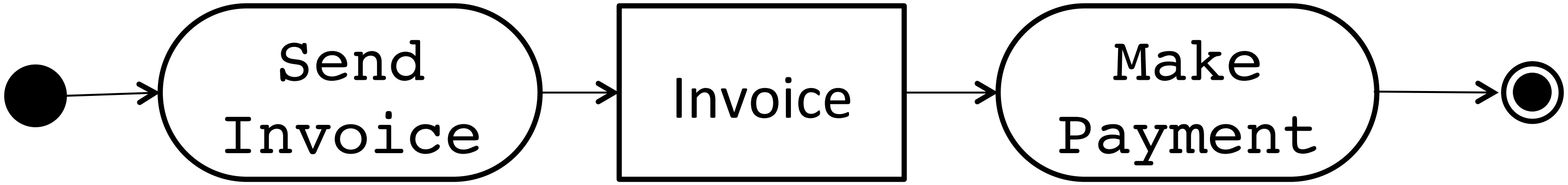
Activity Diagram: Definition Node

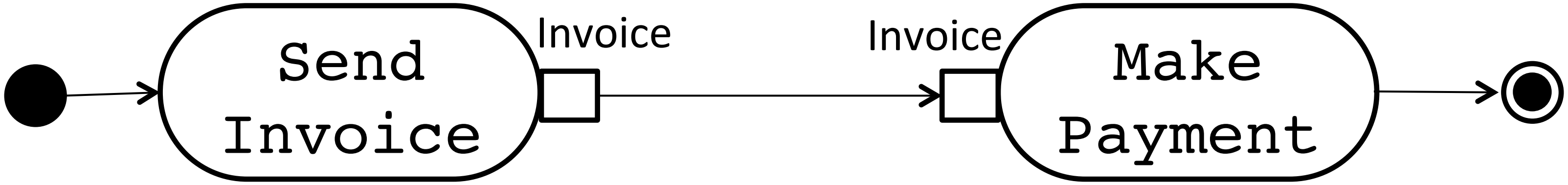
- You can add comments at each component in the diagram
- We use **definition node** to do so



Activity Diagram: Object Node

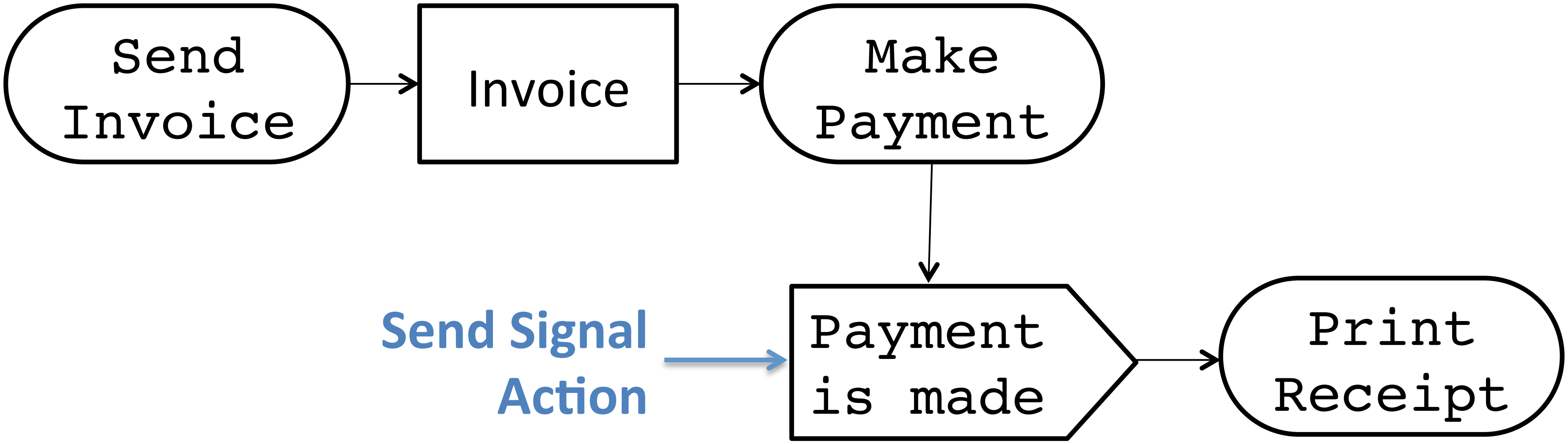
- There could also be object or data being passed among activities
- The data node is called **object node**





Activity Diagram: Sending Signal

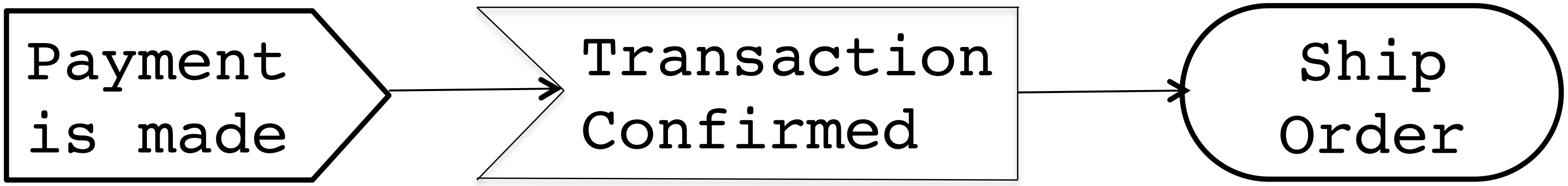
- When an activity is done, we might want to notify other activities about it.



The sent signal is “Payment is made”

Activity Diagram: Accept Event Action

- An activity could be triggered by a specific event. The event listening node is called accept event action
  - If the node **has no incoming edge**, the activity starts immediately and the node remains active (keeps listening for the other events).
  - If the node **has incoming edge**, it will wait until the previous action is complete



Accept Event Action awaits for specific event to be activated

The accepted signal is "Transaction Confirmed"

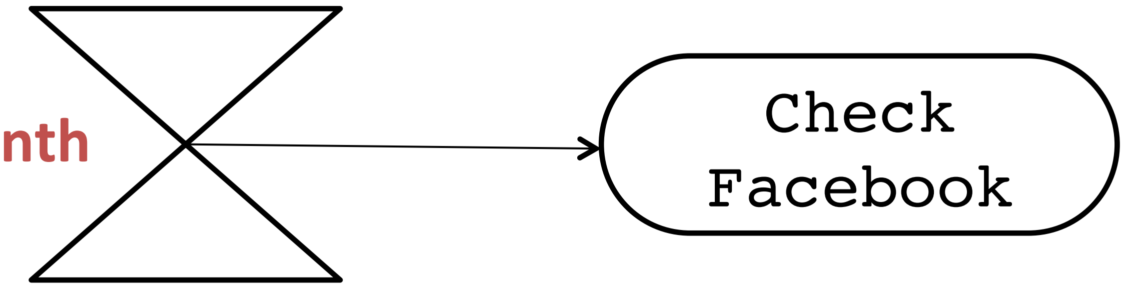
Activity Diagram: Time Event

- Sometimes, we have to execute an action after a specific time interval or at some specific time event

Every hour

End of the month

At 15:00



Check Facebook

Accept Time Event Action or Wait Time Action

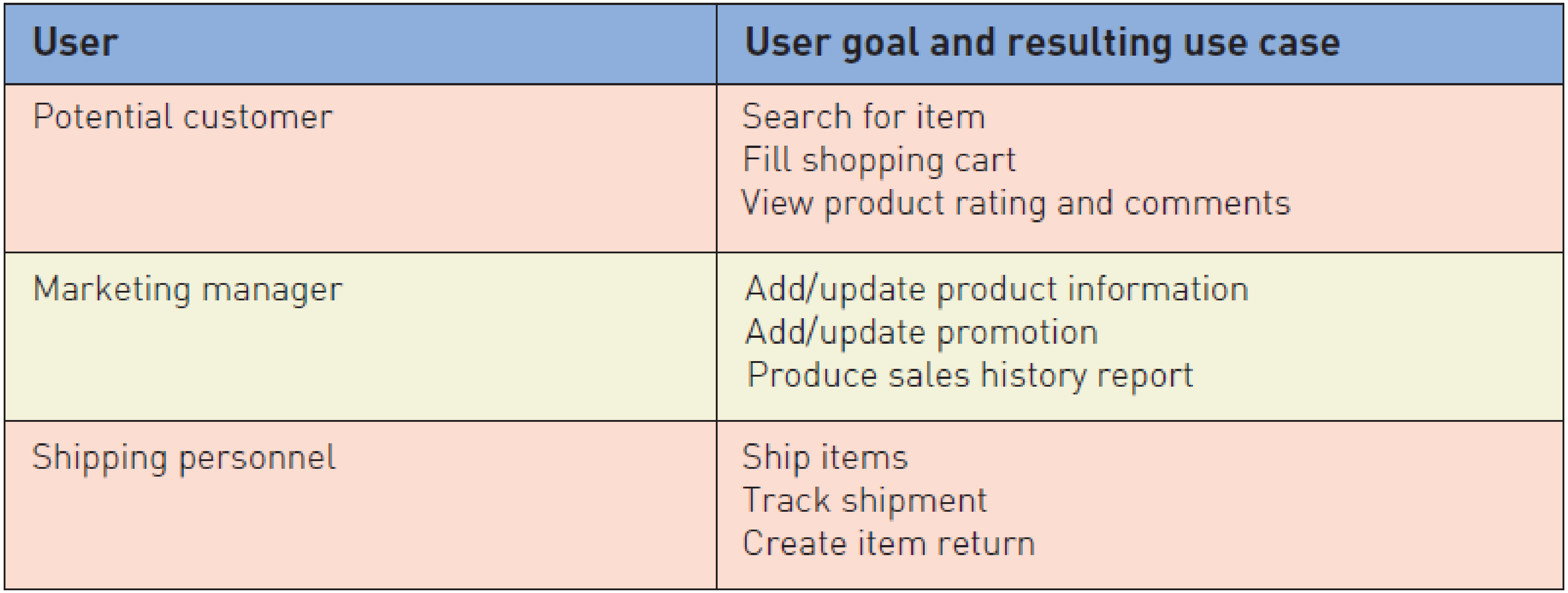
Identifying Use Cases

- User goal technique
  - Ask different kinds of users to describe their goal for using the system
- Event decomposition technique
  - Identify events that will cause the system to respond
- Scenario-based technique
  - Start with user's usual activities

Steps for Determining Use Cases with User Goal Technique

- Identify all potential users for the new system
- Classify the users based on their roles (e.g. sales, marketing, etc.)
- Ask each kind of user for their goals. What do they currently want? What would they like to have.
  - Add customers
  - Update order
  - Produce report
- Create a list of use cases for each kind of users
- Identify which use cases are common among users

Determining Use Cases with User Goal Technique



User	User goal and resulting use case
Potential customer	Search for item Fill shopping cart View product rating and comments
Marketing manager	Add/update product information Add/update promotion Produce sales history report
Shipping personnel	Ship items Track shipment Create item return

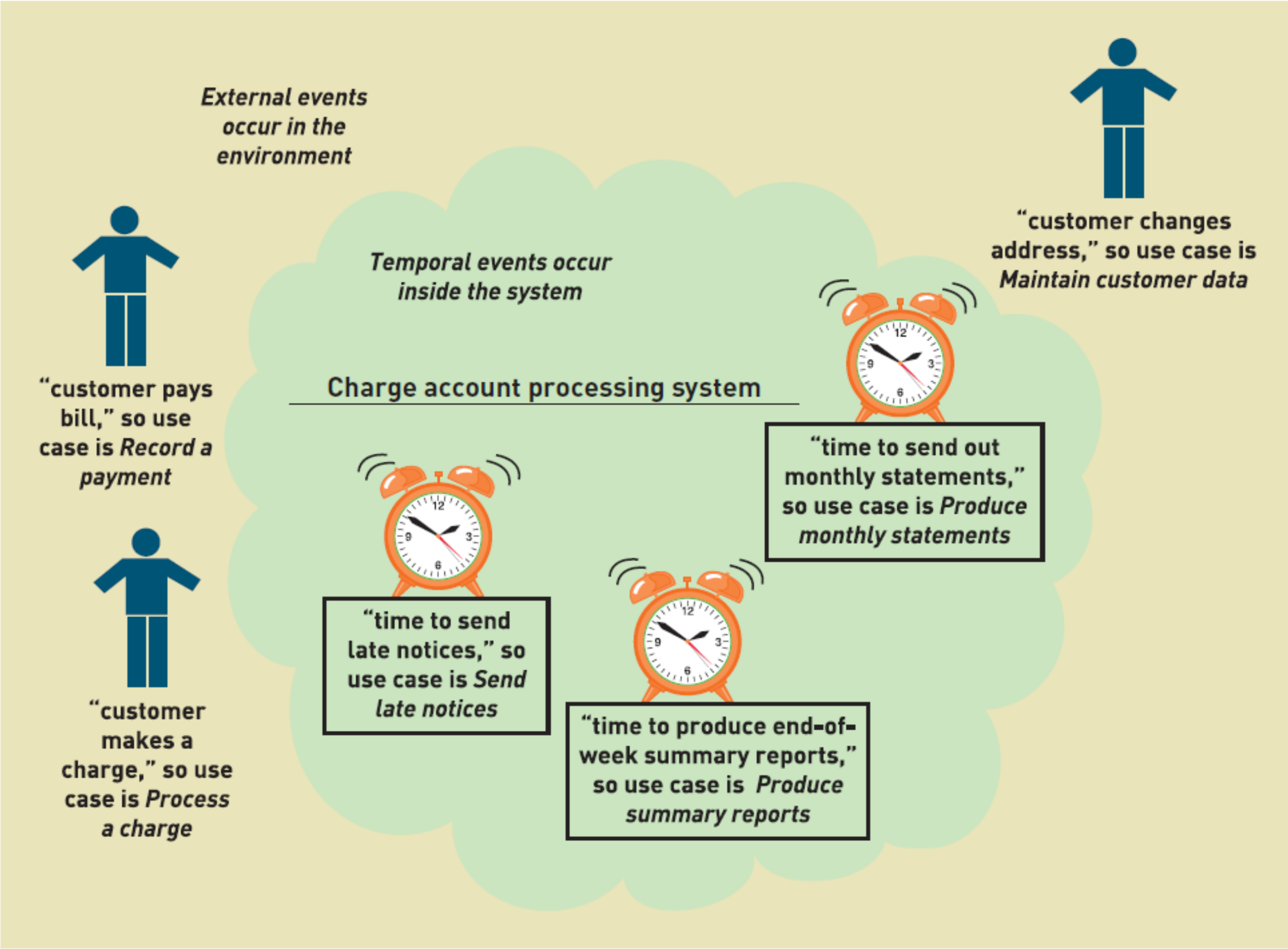
Example of user goals

Image: Satzinger et al, Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Cengage Learning, 2012.

Event Decomposition Technique

- More Comprehensive and Complete Technique
  - Identify the events that occur to which the system must respond.
  - For each event, name a use case (verb-noun) that describes what the system does when the event occurs
- **Event**: something that occurs at a specific time and place, can be described, and should be remembered by the system

Events and Use Cases



External events occur in the environment

"customer pays bill," so use case is *Record a payment*

"customer makes a charge," so use case is *Process a charge*

"customer changes address," so use case is *Maintain customer data*

Temporal events occur inside the system

Charge account processing system

"time to send late notices," so use case is *Send late notices*

"time to produce end-of-week summary reports," so use case is *Produce summary reports*

"time to send out monthly statements," so use case is *Produce monthly statements*

Image: Satzinger et al, Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Cengage Learning, 2012.

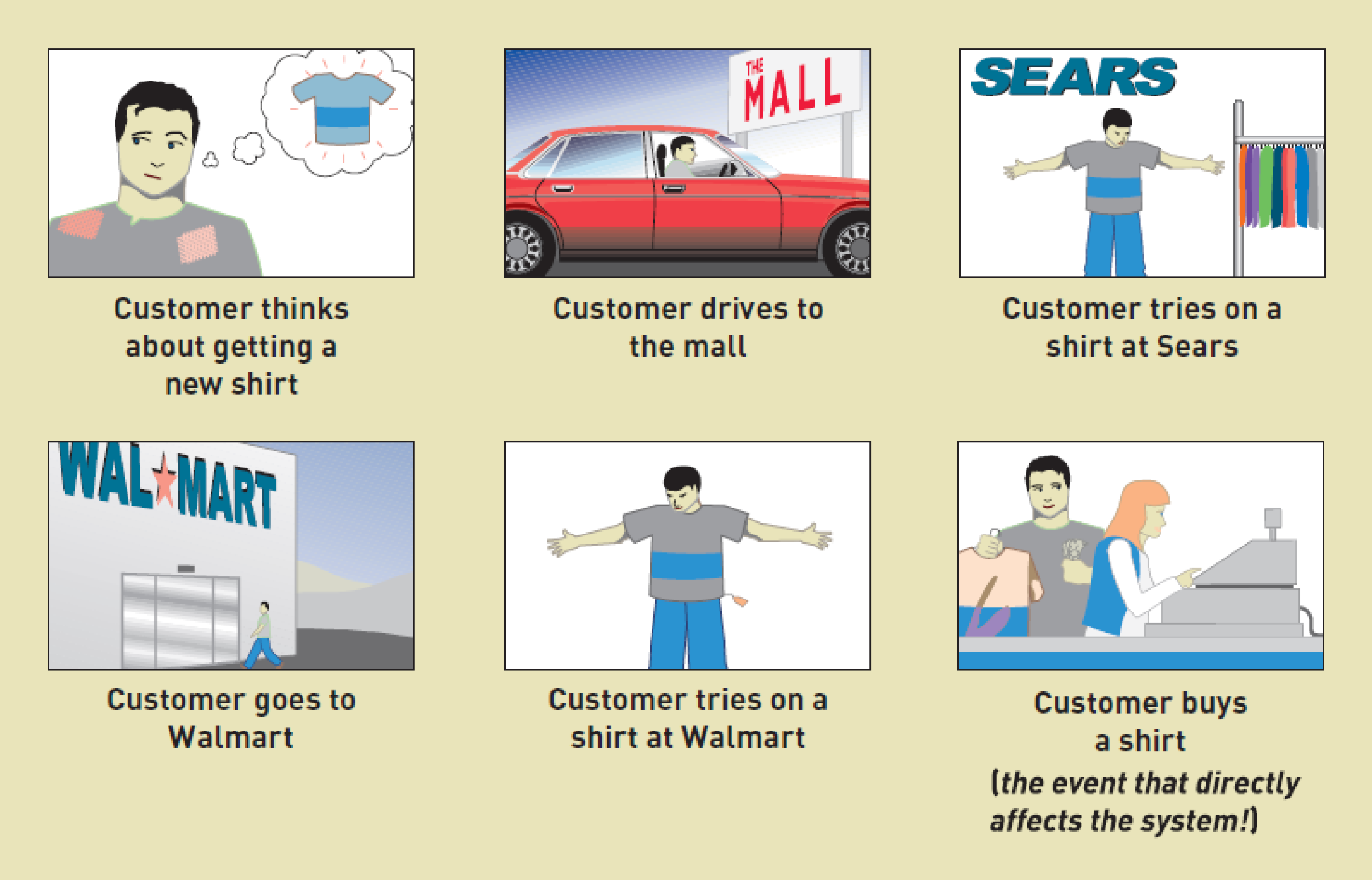
Types of Events

- **External Event**: An event that occurs outside the system, usually initiated by an external agent or actor
- **Temporal Event**: An event that occurs as a result of reaching a point in time
- **State Event**: An event that occurs when something happens inside the system that triggers some process

Identifying Events

- Sometimes it is not easy to identify which events actually cause the use cases
- There may be just sequences of actions that lead to the event
- You have to find the event that ***directly*** affect the system
- That would be the event (or action) that you are interested in

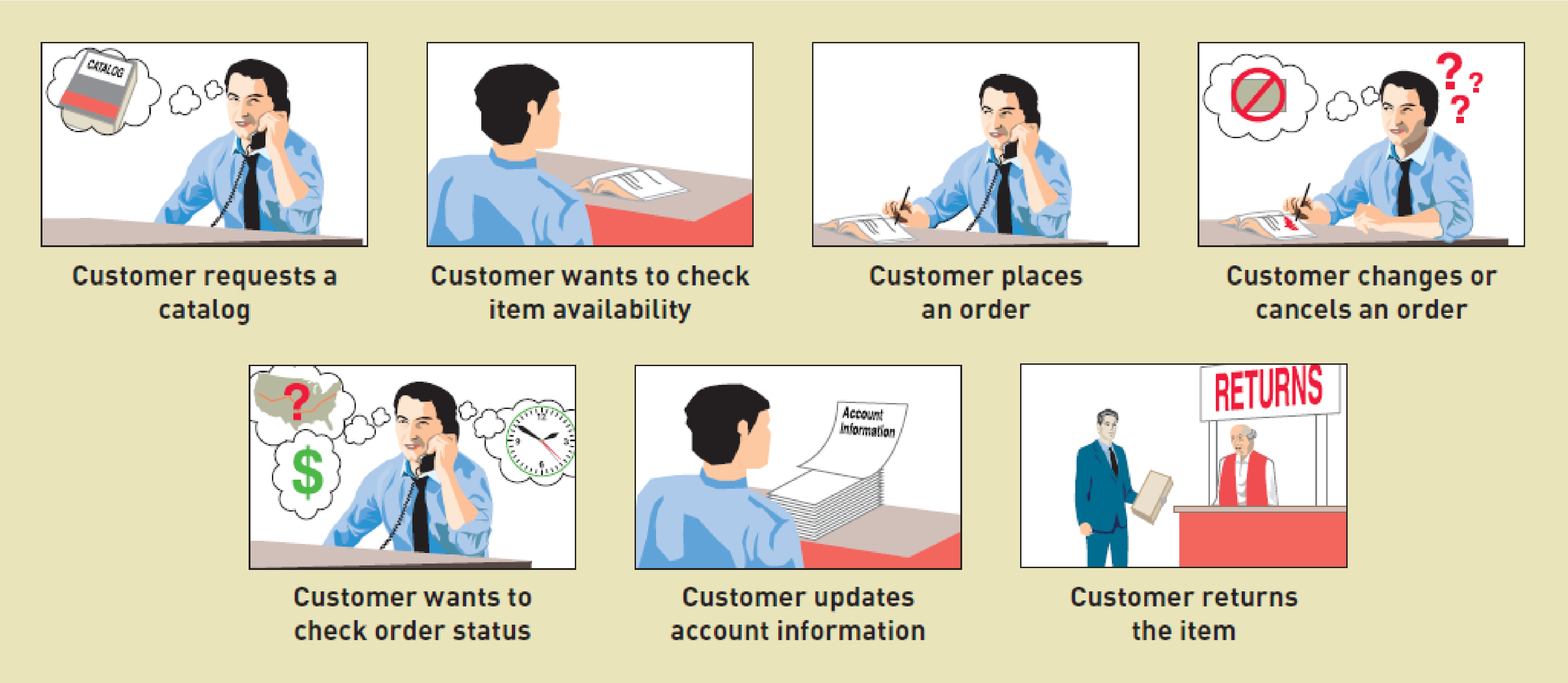
Identifying Events (2)


THE MALL
SEARS
WAL★MART
Customer thinks about getting a new shirt
Customer drives to the mall
Customer tries on a shirt at Sears
Customer goes to Walmart
Customer tries on a shirt at Walmart
Customer buys a shirt
(the event that directly affects the system!)

Sequence of events that lead to “**Purchase product**” use case

Image: Satzinger et al, Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Cengage Learning, 2012.

Sequence of Actions May Lead to Many Use Cases



Customer requests a catalog

Customer wants to check item availability

Customer places an order

Customer changes or cancels an order

Customer wants to check order status

Customer updates account information

Customer returns the item

All these actions may result in many separate use cases

Image: Satzinger et al, Introduction to Systems Analysis and Design: An Agile, Iterative Approach, Cengage Learning, 2012.

Heuristics for Finding Scenarios

- Ask yourself or the client the following questions:
  - What are the primary tasks that the system needs to perform?
  - What data will the actor create, store, change, remove or add in the system?
  - What external changes does the system need to know about?
  - What changes or events will the actor of the system need to be informed about?
- However, don't rely on interviews and questionnaires alone
- Insist on task observation if the system already exists (interface engineering or reengineering)
  - Ask to speak to the end user
  - Expect resistance and try to overcome it.

CRUD Technique

- When we extract use cases, we tend to focused on the users activities
- Thus, we might forget some crucial use cases that happen behind the scene
- **CRUD**: **C**reate, **R**ead/**R**eport, **U**pdate, **D**elete
- CRUD technique can be used to cross check if we miss any use cases, especially the one regarding the data
- They should not be used to identify use cases, only to check if we miss anything

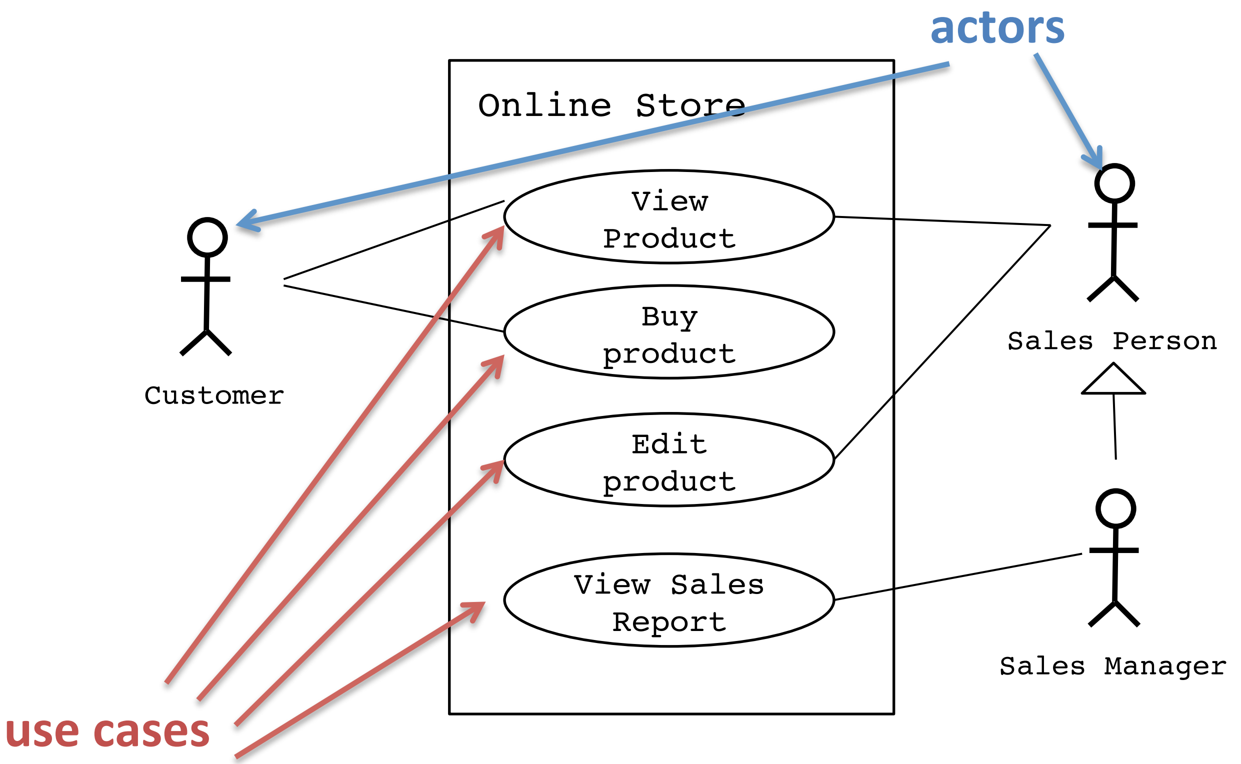
CRUD Example

- CRUD can be used to verify that the use cases to create, read/report, update and delete the data are there

Data Entity	**CRUD**	**Verified Use Case**
Product	Create	Add new product item
	Read/Report	Search for product Obtain product info
	Update	Update product info
	Delete	Delete product item

Use Case Diagram

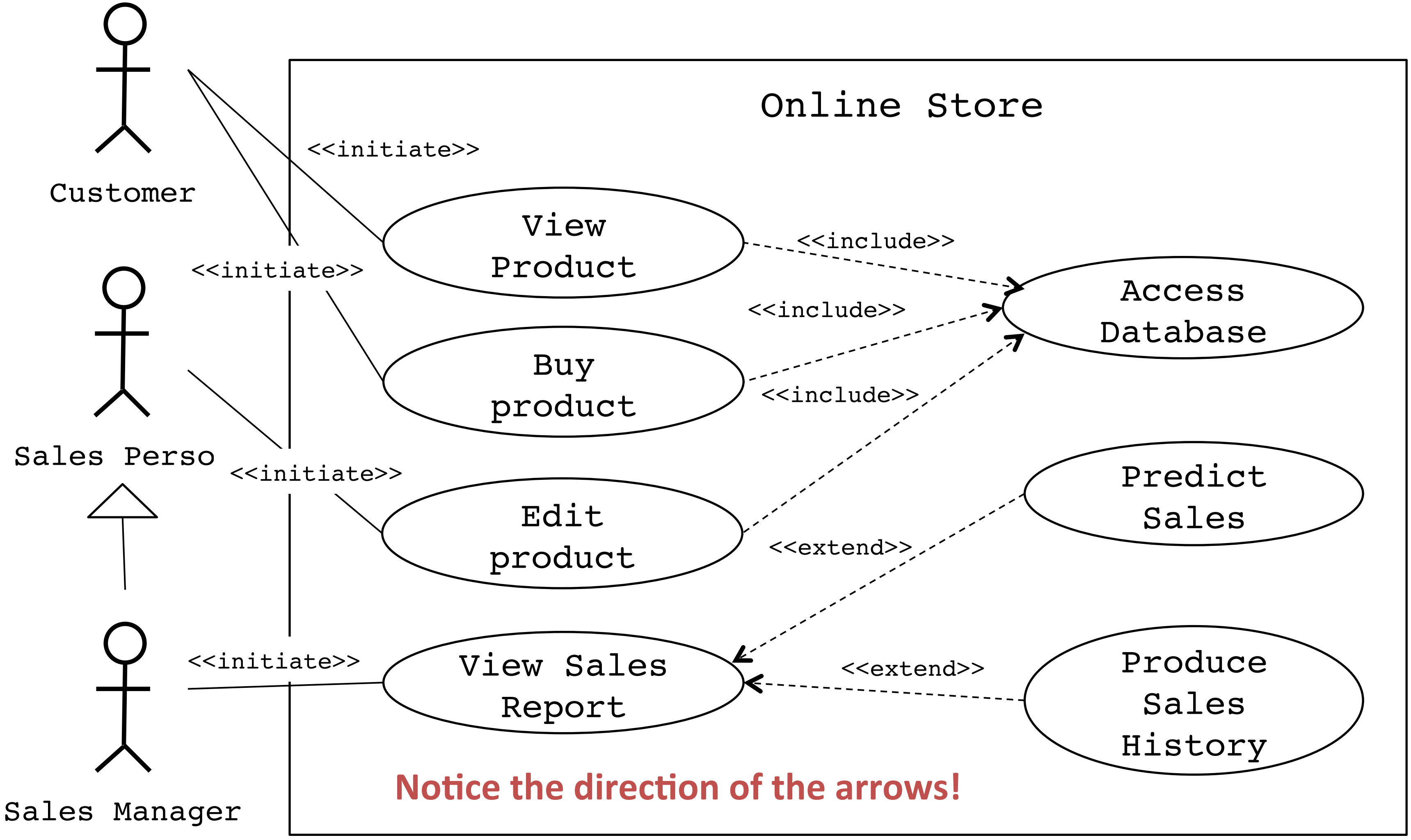
- UML defines a diagram to describe set of use cases: **Use Case Diagram**
  - Illustrate the name of the use cases
  - Relationship among use cases
  - Displays behaviors of the systems and its actors



Use Case Relationships

- Use cases may have **relationships** among each other
- The relationships are denoted by **stereotypes**
- Relationships with actors may have stereotypes too
- Use case stereotypes
  - **Include**: The base use case depends on and use the included use case
    - Included use case is a fully functional use case
    - Included use case may be shared by many base cases.
  - **Exclude**: The extending use case depends on the base case. It extends the functionality of the base case.
    - The base use case is fully functional without extending case
    - Extending case is an optional/extended functionality to the base case
  - In short: **include = reusing functionalities**,
    extend = adding functionalities

Use Case Diagram



Notice the direction of the arrows!

© Isara Anantavrasilp