

# Requirements Modeling

---

with

Use Cases

# Outline

---

## Scenarios quick review

- Finding Scenarios
- Identifying actors

## Use Cases

- Finding Use Cases
- Flow of Events
- Use Case Associations
- Use Case Refinement
- Documenting Use Cases

## Requirements Document

# Scenario example from last lecture:

## Warehouse on Fire

- Bob, 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 tablet, 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 types of scenarios that cover 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
- Create a *use case* to describe all types of scenarios in more detail
  - Participating actors
  - Describe the entry condition
  - Describe the flow of events
  - Describe the exit condition
  - Describe exceptions
  - Describe nonfunctional requirements

# Use cases and scenarios

- A *use case* is a definition of a goal-oriented set of interactions between external actors and the system under consideration.
- An *actor* is a party outside the system that interacts with the system.
- UC is initiated by an actor with a particular goal in mind and completes successfully when that goal is satisfied.
- A *scenario* is an instance of a use case, and represents a single path through the use case.

Booch, G., I. Jacobson and J. Rumbaugh, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

# Actors

## Actors

- An actor may be a *class of users*, roles users can play, or other computer system (actors don't need to be human users).
- A *primary actor* is one having a goal requiring the assistance of the system and initiates a use case.
- A *secondary actor* is one from which the system needs assistance.

Actors can be used in user stories, as well.

# Use case description

- UC describes the *basic sequence* of interactions between actors and the system necessary to satisfy the goal
- UC may include *variants* of this sequence, i.e., alternative sequences that may also satisfy the goal
- UC may include *exceptions*, i.e. sequences that may lead to failure because of exceptional behavior, error handling, etc.
- The system is treated as a "black box", and the interactions with system, including system responses, are as perceived from outside the system



# How to find Use Cases

- Select a narrow vertical slice of the system (i.e. one scenario)
  - Discuss it in detail with the user to understand the user's preferred style of interaction
- Select a horizontal slice (i.e. many scenarios) to define the scope of the system.
  - Discuss the scope with the user
- Use illustrative prototypes (mock-ups) as visual support
- Find out what the user does
  - Task observation (Good)
  - Questionnaires (Bad)

# Use Case Example: Report Emergency

- Use case name: ReportEmergency
- Participating Actors:
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
- Exceptions:
  - The Field Officer is notified immediately if the connection between terminal and central is lost.
  - The Dispatcher is notified immediately if the connection between a FieldOfficer and central is lost.
- Flow of Events
- Special Requirements:
  - The FieldOfficer's report is acknowledged within 30 seconds. The selected response arrives no later than 30 seconds after it is sent by the Dispatcher.

# Use Case Example: ReportEmergency

## Flow of Events

1. The **FieldOfficer** activates the “Report Emergency” function of her terminal. FRIEND responds by presenting a form to the officer.
2. The FieldOfficer fills the form, by selecting the emergency level, type, location, and brief description of the situation. The FieldOfficer also describes a response to the emergency situation. Once the form is completed, the FieldOfficer submits the form, and the **Dispatcher** is notified.
3. The Dispatcher creates an Incident in the database by invoking the OpenIncident use case. He selects a response and acknowledges the report.
4. The FieldOfficer receives the acknowledgment and the selected response.

## Another Example: Allocate a Resource

- Actors:
  - **Field Supervisor:** This is the official at the emergency site.
  - **Resource Allocator:** The Resource Allocator is responsible for the commitment and decommitment of the Resources managed by the FRIEND system.
  - **Dispatcher:** A Dispatcher enters, updates, and removes Emergency Incidents, Actions, and Requests in the system. The Dispatcher also closes Emergency Incidents.
  - **Field Officer:** Reports accidents from the Field

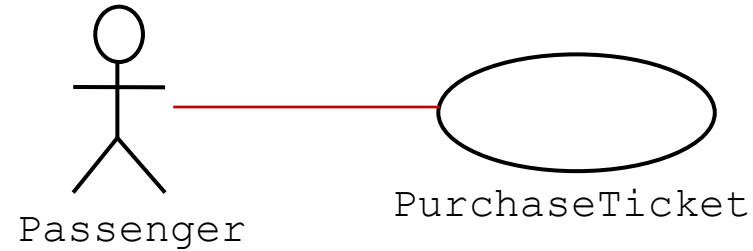
## Allocate a Resource (cont'd)

- Use case name: AllocateResources
- Participating Actors:
  - Field Officer (Bob and Alice in the Scenario)
  - Dispatcher (John in the Scenario)
  - Resource Allocator and Field Supervisor
- Entry Condition:
  - The Resource Allocator has selected an available resource
- Flow of Events:
  1. The Resource Allocator selects an Emergency Incident
  2. The Resource is committed to the Emergency Incident
- Exit Condition:
  - The use case terminates when the resource is committed
  - The selected Resource is unavailable to other Requests.
- Special Requirements:
  - The Field Supervisor is responsible for managing Resources

# Order of steps when formulating use cases

- First step: Name the use case
  - Use case name: ReportEmergency
- Second step: Find the actors
  - Generalize the concrete names (“Bob”) to participating actors (“Field officer”)
  - Participating Actors:
    - Field Officer (Bob and Alice in the Scenario)
    - Dispatcher (John in the Scenario)
- Third step: Concentrate on the flow of events
  - Use informal natural language

# Textual Use Case Description Example



*1. Name:* Purchase ticket

*2. Participating actor:* Passenger

*3. Entry condition:*

- Passenger stands in front of ticket machine
- Passenger has sufficient money to purchase a ticket

*4. Exit condition:*

- Passenger has ticket

*5. Flow of events:*

1. Passenger selects the destination station
2. Ticket Machine displays the amount due
3. Passenger inserts money, at least the amount due
4. Ticket Machine returns change
5. Ticket Machine issues ticket

*6. Special requirements:*  
*None.*

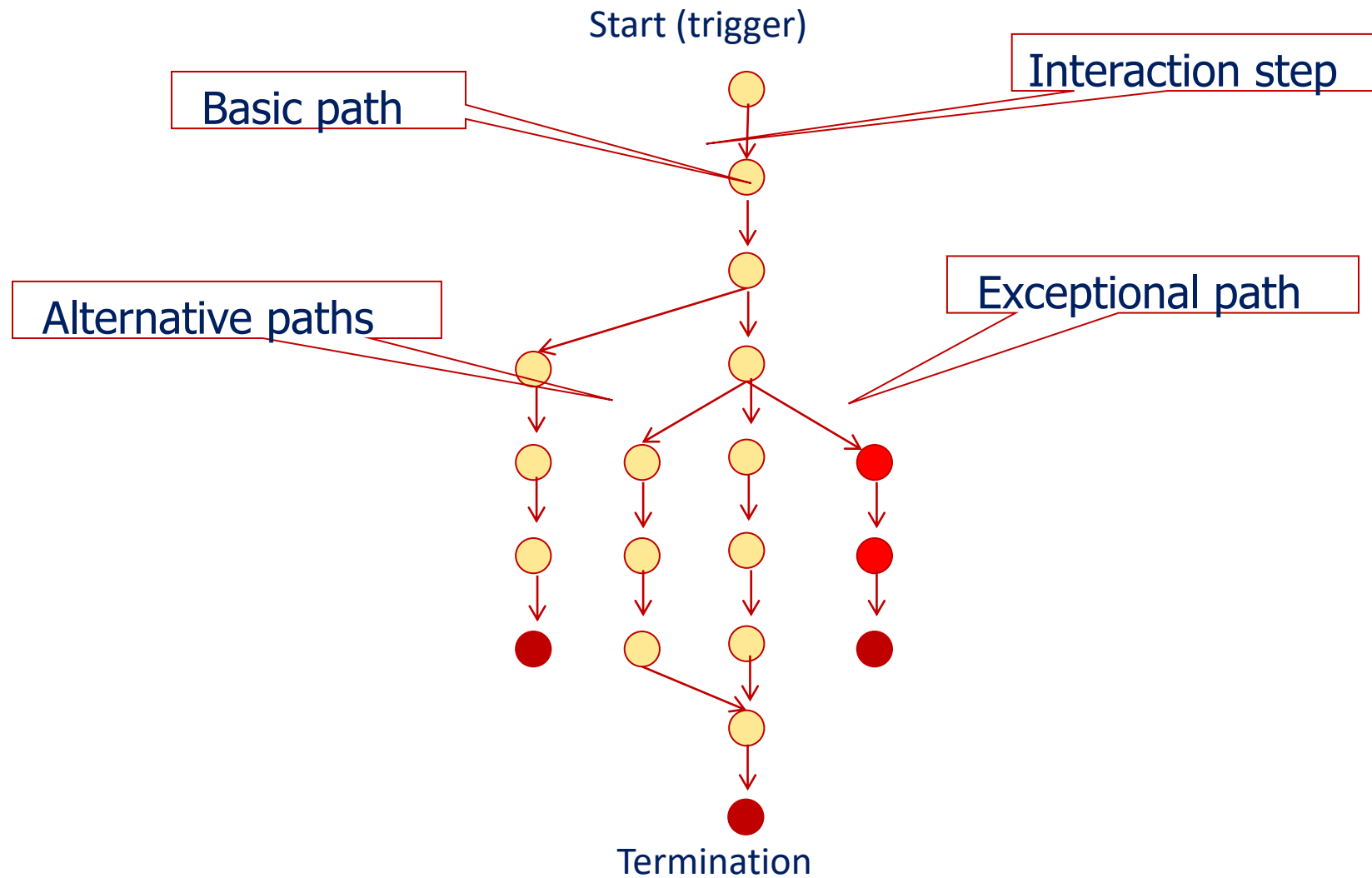
## Fully Developed Use Case Description

### Use case: *Create customer account*

<b>Use case name:</b>	Create customer account.	
<b>Scenario:</b>	Create online customer account.	
<b>Triggering event:</b>	New customer wants to set up account online.	
<b>Brief description:</b>	Online customer creates customer account by entering basic information and then following up with one or more addresses and a credit or debit card.	
<b>Actors:</b>	Customer.	
<b>Related use cases:</b>	Might be invoked by the <i>Check out shopping cart</i> use case.	
<b>Stakeholders:</b>	Accounting, Marketing, Sales.	
<b>Preconditions:</b>	Customer Account subsystem must be available. Credit/debit authorization services must be available.	
<b>Postconditions:</b>	Customer must be created and saved. One or more Addresses must be created and saved. Credit/debit card information must be validated. Account must be created and saved. Address and Account must be associated with Customer.	
<b>Flow of activities:</b>	Actor	System
	1. Customer indicates desire to create customer account and enters basic customer information.  2. Customer enters one or more addresses.  3. Customer enters credit/debit card information.	1.1 System creates a new customer. 1.2 System prompts for customer addresses.  2.1 System creates addresses. 2.2 System prompts for credit/debit card.  3.1 System creates account. 3.2 System verifies authorization for credit/debit card. 3.3 System associates customer, address, and account. 3.4 System returns valid customer account details.
<b>Exception conditions:</b>	1.1 Basic customer data are incomplete. 2.1 The address isn't valid. 3.2 Credit/debit information isn't valid.	

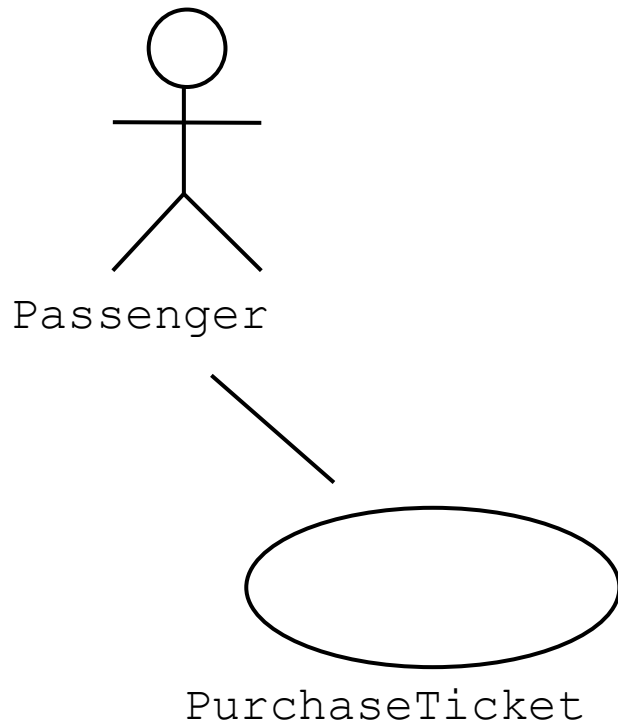


# Flows in a Use Case



# UML Use Case Diagrams

**Used during requirements elicitation and analysis to represent external behavior (“visible from the outside of the system”)**



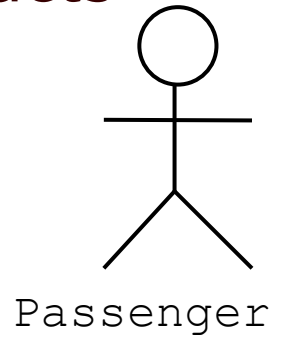
- An *actor* represents a role, that is, a type of user of the system
- A use case represents a class of functionality provided by the system

## *Use case model:*

The set of all use cases that completely describe the functionality of the system.

# Actors

- An actor is an abstraction of an external entity which interacts (communicates) with the system:
  - User
  - External system (another system)
  - Physical environment (e.g., weather)
- An actor has a unique name and an optional description
- Examples:
  - **Passenger**: A person on the train
  - **GPS satellite**: An external system that provides the system with GPS coordinates.

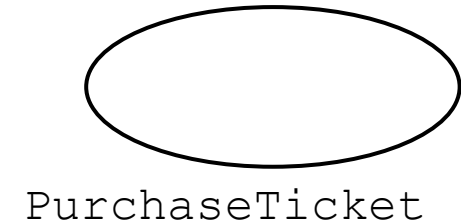


Optional  
Description

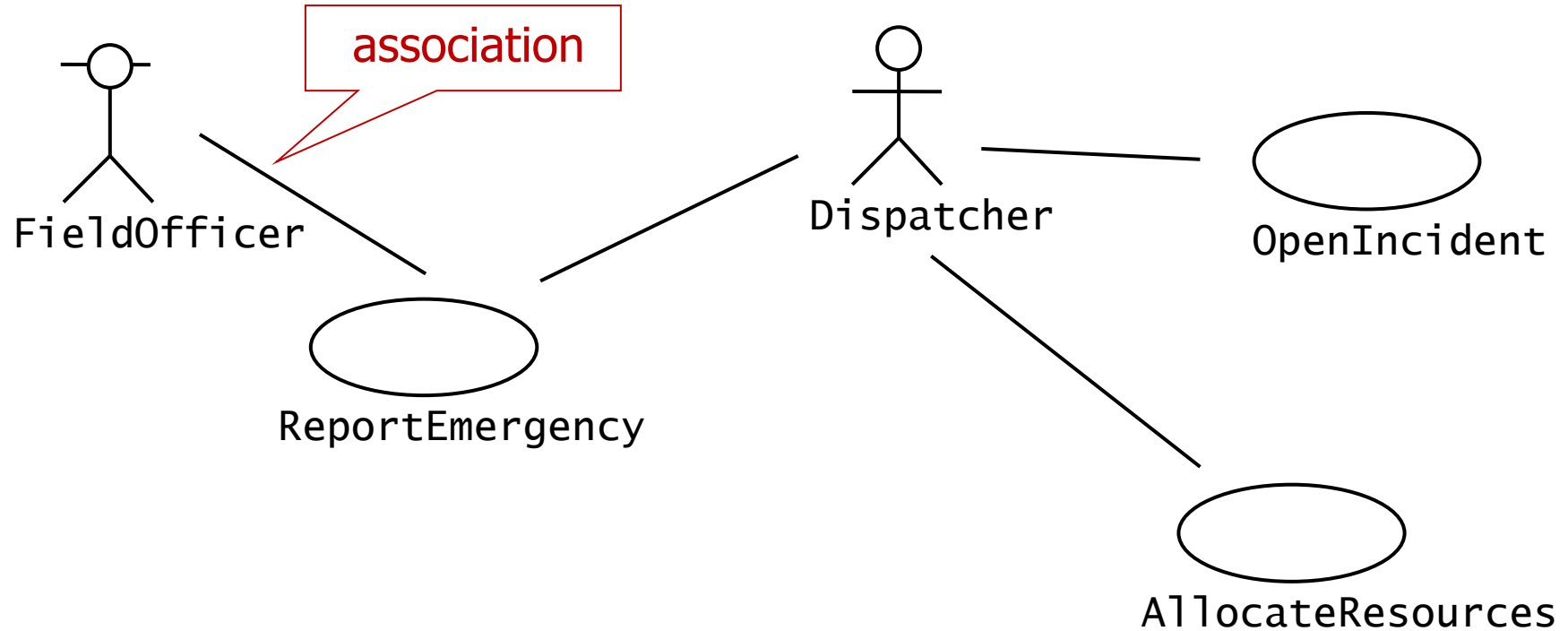
Name

# Use Case

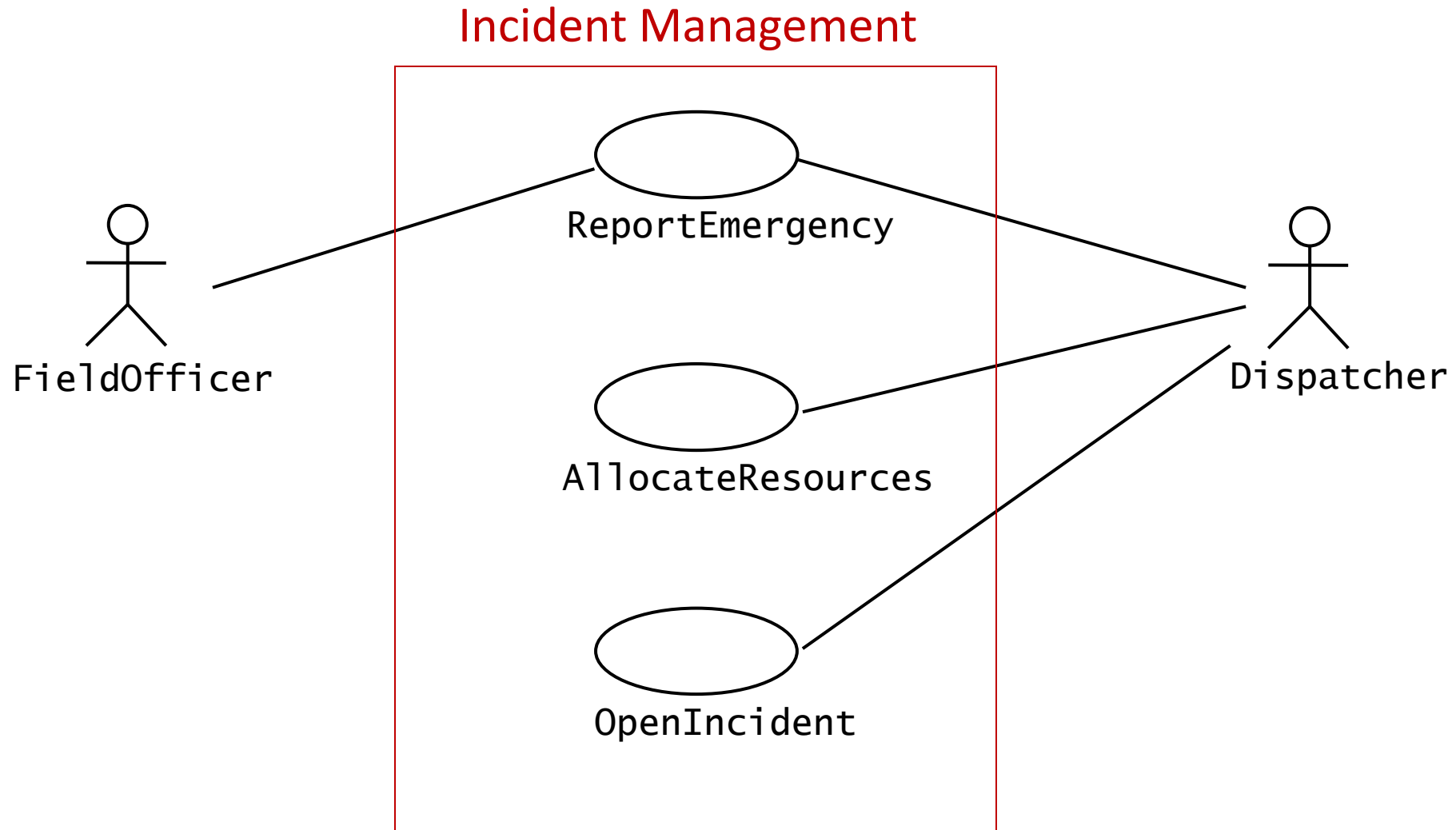
- A use case represents a class of functionality provided by the system
- Use cases can be described textually, with a close focus on the event flow between actor and system
- The textual use case description consists of (more details later):
  1. Unique name
  2. Participating actors
  3. Entry conditions
  4. Exit conditions
  5. Flow of events
  6. Special requirements.



# Use Case Model for Incident Management



# Use Case Model for Incident Management



# Use Case Associations

- Some use cases may be interdependent
- Dependencies between use cases are represented with

## **use case associations**

- Associations are used to reduce complexity
  - Decompose a long use case into shorter ones
  - Separate alternate flows of events
  - Refine abstract use cases
- Types of use case associations
  - Includes
  - Extends
  - Generalization

# Uses Cases can be related

- Includes Relationship

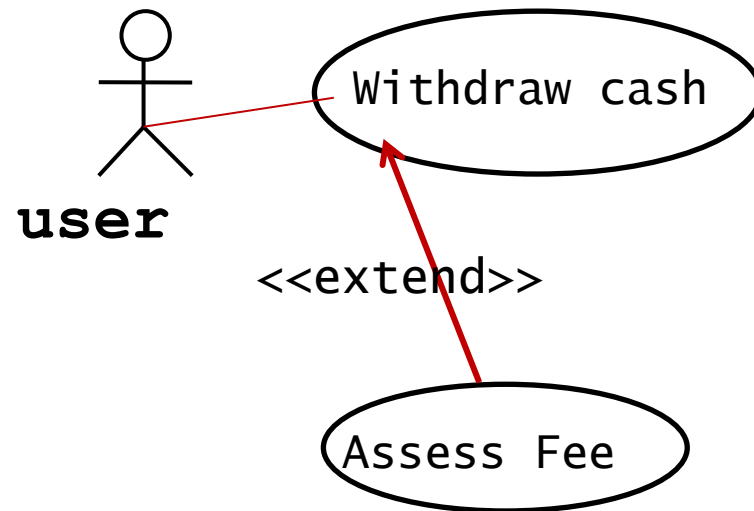
- To represent functional behavior common to more than one use case.

- Extends Relationship

- To represent seldom invoked use case fragments or exceptional functionality
- An extension is typically not a complete use case

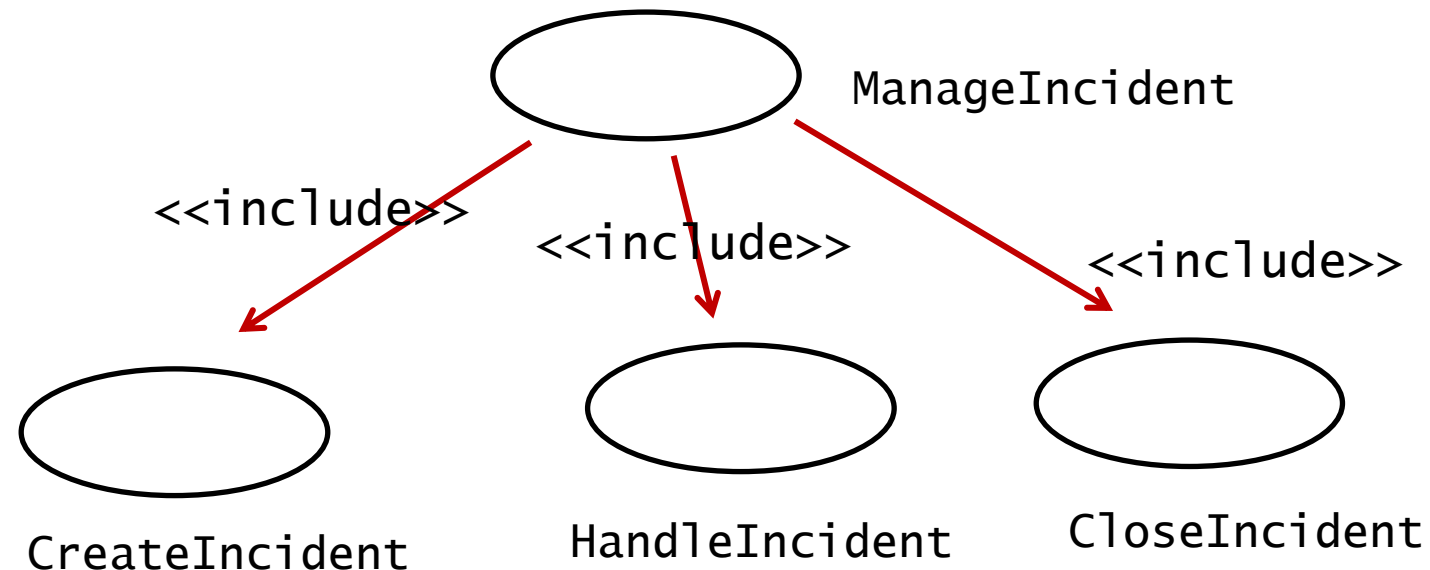


- If the ATM user doesn't bank at the ATM's owning institution, Fees must be calculated and deducted from the user account.



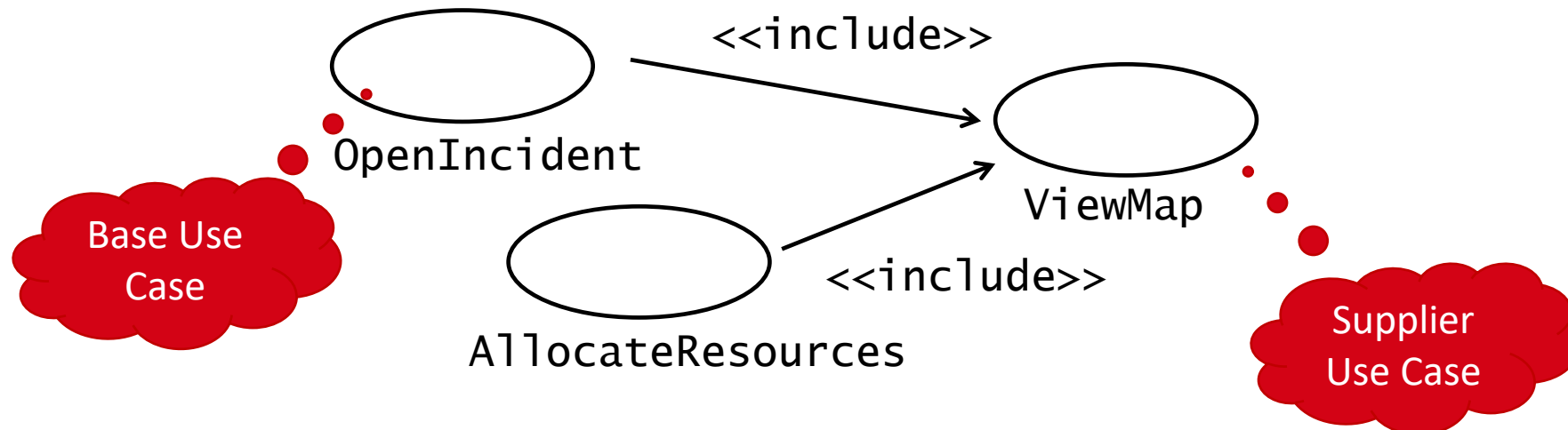
# <<include>>: Functional Decomposition

- Problem:
  - A function in the original problem statement is too complex
- Solution:
  - Describe the function as the aggregation of a set of simpler functions. The associated use case is decomposed into shorter use cases



# <<include>>: Reuse of Existing Functionality

- **Problem:** There may exist overlaps among use cases. How can we *reuse* flows of events instead of duplicating them?
- **Solution:** The *includes association* from use case A to use case B indicates that an instance of use case A performs all the behavior described in use case B (“A delegates to B”)
- **Example:** Use case “ViewMap” describes behavior that can be used by use case “OpenIncident” (“ViewMap” is factored out)



## Example: Draw a use case diagram

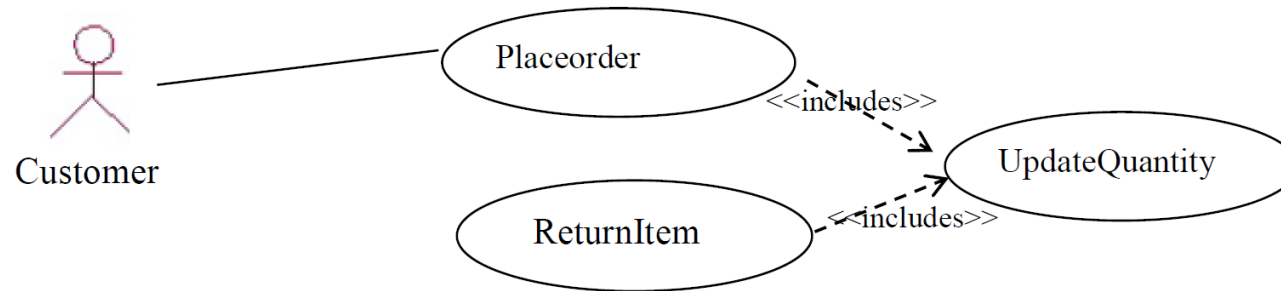
The Customer can access the Video Shop to place an order or to return an item. Both operations; placing an order or returning an item should update the quantity on hand for the item.

## Example: Draw a use case diagram

The Customer can access the Video Shop to place an order or to return an item. Both operations; placing an order or returning an item should update the quantity on hand for the item.

## Example: Draw a use case diagram

The Customer can access the Video Shop to place an order or to return an item. Both operations; placing an order or returning an item should update the quantity on hand for the item.

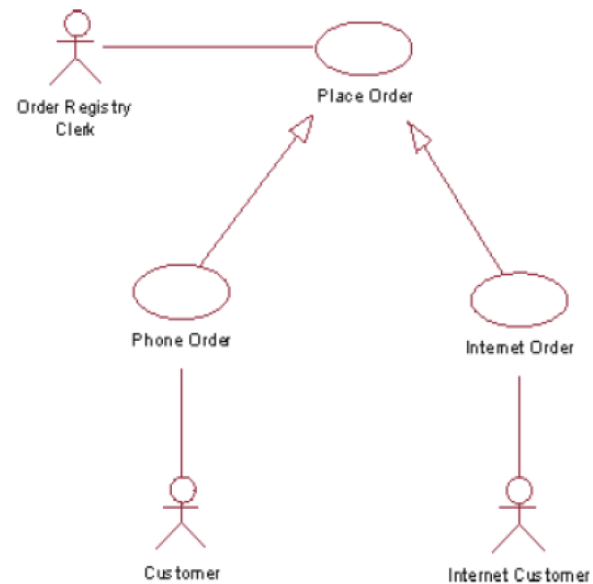


## Example: Draw a use case diagram

The order Registry Clerk can invoke the general use case *Place Order*. *Place Order* can also be specialized by the use cases *Phone Order* or *Internet Order* that are performed by the actors: customer and internet customer, respectively.

## Example: Draw a use case diagram

The order Registry Clerk can invoke the general use case *Place Order*. *Place Order* can also be specialized by the use cases *Phone Order* or *Internet Order* that are performed by the actors: customer and internet customer, respectively.



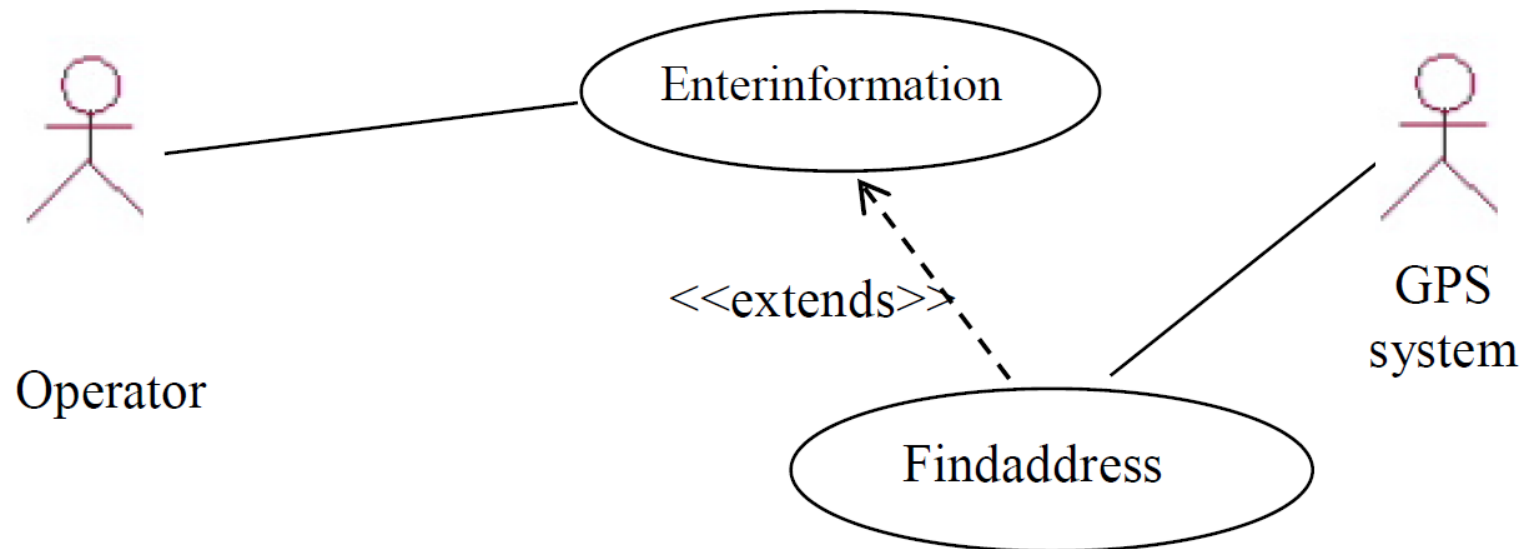


## Draw a use case diagram

In an ambulance dispatch system, when the request for ambulance comes to the operator, he takes information about the incident from the caller. This information is entered into the ambulance dispatch system. This information includes caller phone number and address. If the caller does not know the exact address of the patient, the operator can find the address using the GPS system.

## Draw a use case diagram

In an ambulance dispatch system, when the request for ambulance comes to the operator, he takes information about the incident from the caller. This information is entered into the ambulance dispatch system. This information includes caller phone number and address. If the caller does not know the exact address of the patient, the operator can find the address using the GPS system.



**Example:** Part of the use case description, draw a use case diagram

Use case	Checkout	
Flow of events	Customer	System
	<ol style="list-style-type: none"><li>1. User shows desire to check out</li><li>2. The user confirms the address</li><li>3. The user confirms the card</li><li>4. .....</li></ol>	<ol style="list-style-type: none"><li>1.1 The system displays the stored shipping address and asks the user to confirm the address.</li><li>3.1 The system displays stored card and asks the user to confirm the card</li></ol>
Alternative flows	<ol style="list-style-type: none"><li>1. At step 1 if the system did not find a stored address or the user requested to use another address, the system invokes the use case Enter Address.</li><li>2. At step 3 if the system did not find a stored payment cards or the user requested to use another card, the system invokes the use case Enter Payment Info.</li></ol>	

## Example: Draw a use case diagram

Use case	Checkout	
Flow of events	Customer	System
	1. User shows desire to check out 2. The user confirms the address 3. The user confirms the card 4. ....	1.1 The system displays the stored shipping address and asks the user to confirm the address. 3.1 The system displays stored card and asks the user to confirm the card
Alternative flows	1. At step 1 if the system did not find a stored address or the user requested to use another address, the system invokes the use case Enter Address. 2. At step 3 if the system did not find a stored payment cards or the user requested to use another card, the system invokes the use case Enter Payment Info.	


# RMO Checkout Page

- RMO shopping cart checkout
  - Simple, easy to read

www.rmo.biz - / - Windows Internet Explorer

http://www.rmo.biz/

Welcome to RMO.biz (Log in or Register) 0 items [Checkout](#)

  All Departments

**FREE SHIPPING**  
on orders of \$100 or more

Account Information	Shipping Information	Payment Information	Order Confirmation																														
<b>Product Summary</b>																																	
<table border="1"><thead><tr><th>Description</th><th>Size</th><th>Color</th><th>Price</th><th>Quantity</th><th>Total</th></tr></thead><tbody><tr><td><a href="#">Ladies parka</a></td><td>Medium (10-12)</td><td>Blue</td><td>\$72.95</td><td>1</td><td>\$72.95</td></tr><tr><td><a href="#">Toddler parka</a></td><td>Medium</td><td>Red</td><td>\$44.95</td><td>1</td><td>\$44.95</td></tr></tbody></table>				Description	Size	Color	Price	Quantity	Total	<a href="#">Ladies parka</a>	Medium (10-12)	Blue	\$72.95	1	\$72.95	<a href="#">Toddler parka</a>	Medium	Red	\$44.95	1	\$44.95												
Description	Size	Color	Price	Quantity	Total																												
<a href="#">Ladies parka</a>	Medium (10-12)	Blue	\$72.95	1	\$72.95																												
<a href="#">Toddler parka</a>	Medium	Red	\$44.95	1	\$44.95																												
<table border="1"><thead><tr><th colspan="3">Billing Summary</th><th colspan="2">Order Summary</th></tr></thead><tbody><tr><td>Customer</td><td>Nancy Wells</td><td><a href="#">Change</a></td><td>Subtotal</td><td>\$117.90</td></tr><tr><td>Billing address</td><td>1122 Silicon Avenue 87989</td><td><a href="#">Change</a></td><td>Shipping</td><td>\$0.00</td></tr><tr><td>Delivery address</td><td>1612 Jefferson Street NE 87123</td><td><a href="#">Change</a></td><td>Sales Tax</td><td>\$7.66</td></tr><tr><td>Payment type</td><td>Visa, XXXX-XXXX-XXXX-0089</td><td><a href="#">Change</a></td><td>Credits</td><td>\$0.00</td></tr><tr><td>Delivery method</td><td>UPS - Ground (3-5 days)</td><td><a href="#">Change</a></td><td>Order Total</td><td>\$125.56</td></tr></tbody></table>				Billing Summary			Order Summary		Customer	Nancy Wells	<a href="#">Change</a>	Subtotal	\$117.90	Billing address	1122 Silicon Avenue 87989	<a href="#">Change</a>	Shipping	\$0.00	Delivery address	1612 Jefferson Street NE 87123	<a href="#">Change</a>	Sales Tax	\$7.66	Payment type	Visa, XXXX-XXXX-XXXX-0089	<a href="#">Change</a>	Credits	\$0.00	Delivery method	UPS - Ground (3-5 days)	<a href="#">Change</a>	Order Total	\$125.56
Billing Summary			Order Summary																														
Customer	Nancy Wells	<a href="#">Change</a>	Subtotal	\$117.90																													
Billing address	1122 Silicon Avenue 87989	<a href="#">Change</a>	Shipping	\$0.00																													
Delivery address	1612 Jefferson Street NE 87123	<a href="#">Change</a>	Sales Tax	\$7.66																													
Payment type	Visa, XXXX-XXXX-XXXX-0089	<a href="#">Change</a>	Credits	\$0.00																													
Delivery method	UPS - Ground (3-5 days)	<a href="#">Change</a>	Order Total	\$125.56																													
Click Accept to confirm and process you order -----> <a href="#">Accept</a>																																	

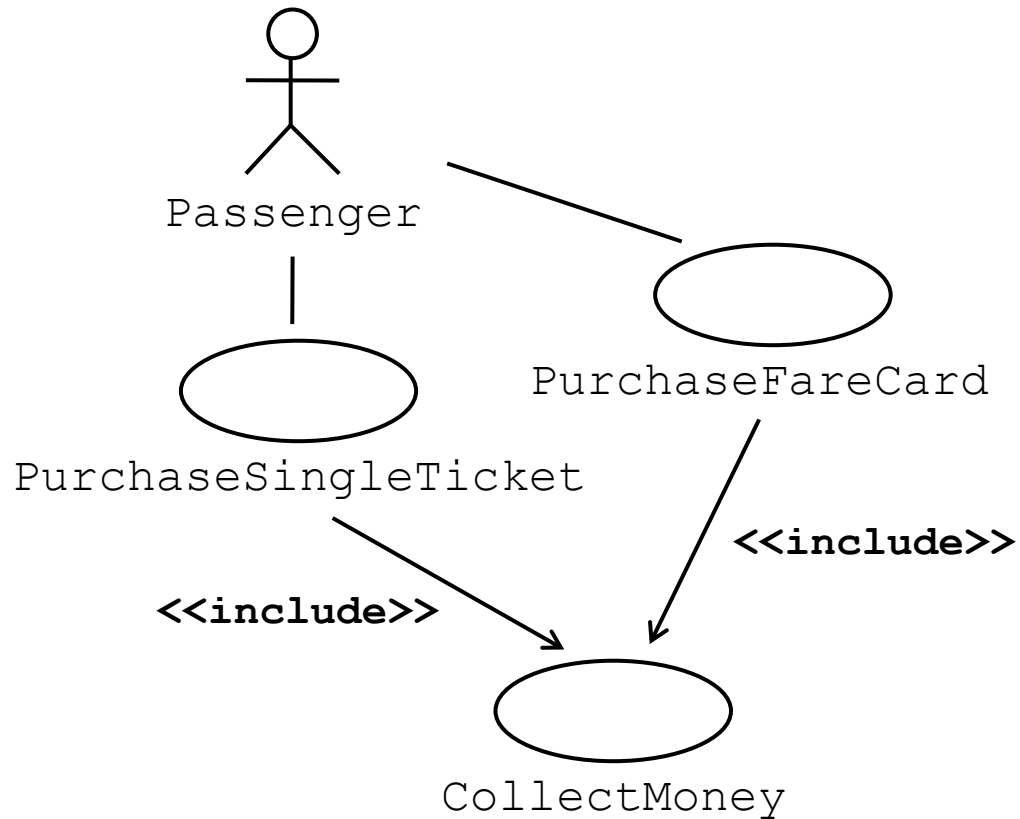
Done Internet | Protected Mode: Off 100%

## Example:

Part of the use case description, draw a use case diagram

Use case	Checkout	
Flow of events	<ol style="list-style-type: none"><li>1. User shows desire to check out</li><li>2. The user confirms the address</li><li>3. The user confirms the card</li><li>4. ....</li></ol>	<ol style="list-style-type: none"><li>1.1 The system invokes the use case process address</li><li>3.1 The system displays stored card and asks the user to confirm the card</li></ol>
Alternative flows	<ol style="list-style-type: none"><li>1. At step 1 if the system did not find a stored address or the user requested to use another address, the system invokes the use case Enter Address.</li><li>2. At step 3 if the system did not find a stored payment cards or the user requested to use another card, the system invokes the use case Enter Payment Info.</li></ol>	

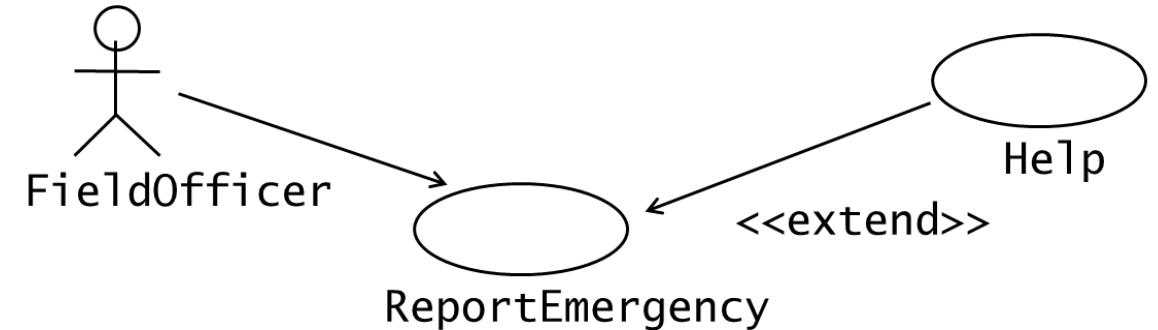
# The <<include>> Relationship



- <<include>> relationship represents common functionality needed in more than one use case
- <<include>> behavior is factored out for reuse, not because it is an exception
- The direction of a <<include>> relationship is to the using use case.

# <<extend>> Association for Use Cases

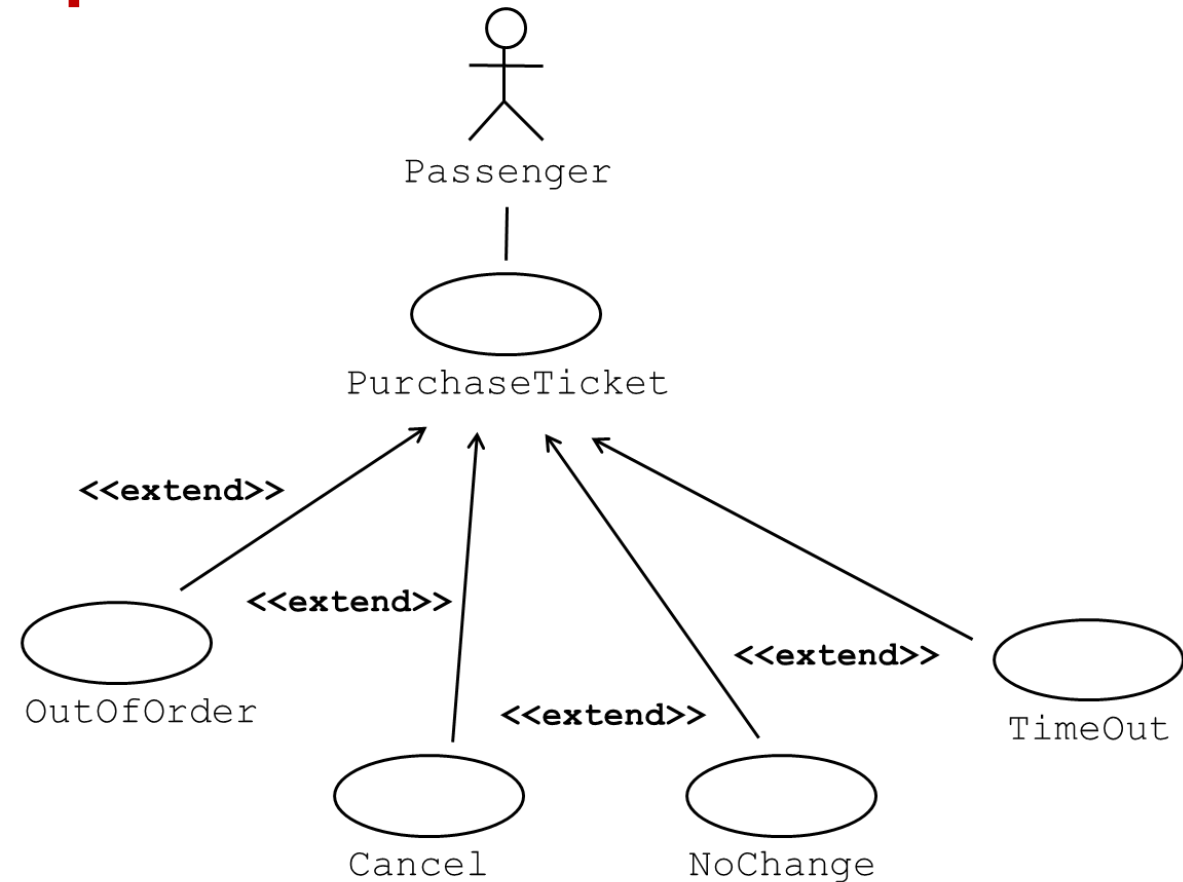
- **Problem:** The functionality in the original problem statement needs to be extended.
- **Solution:** An *extend association* from use case A to use case B
- **Example:** “ReportEmergency” is complete by itself, but can be extended by use case “Help” for a scenario in which the user requires help



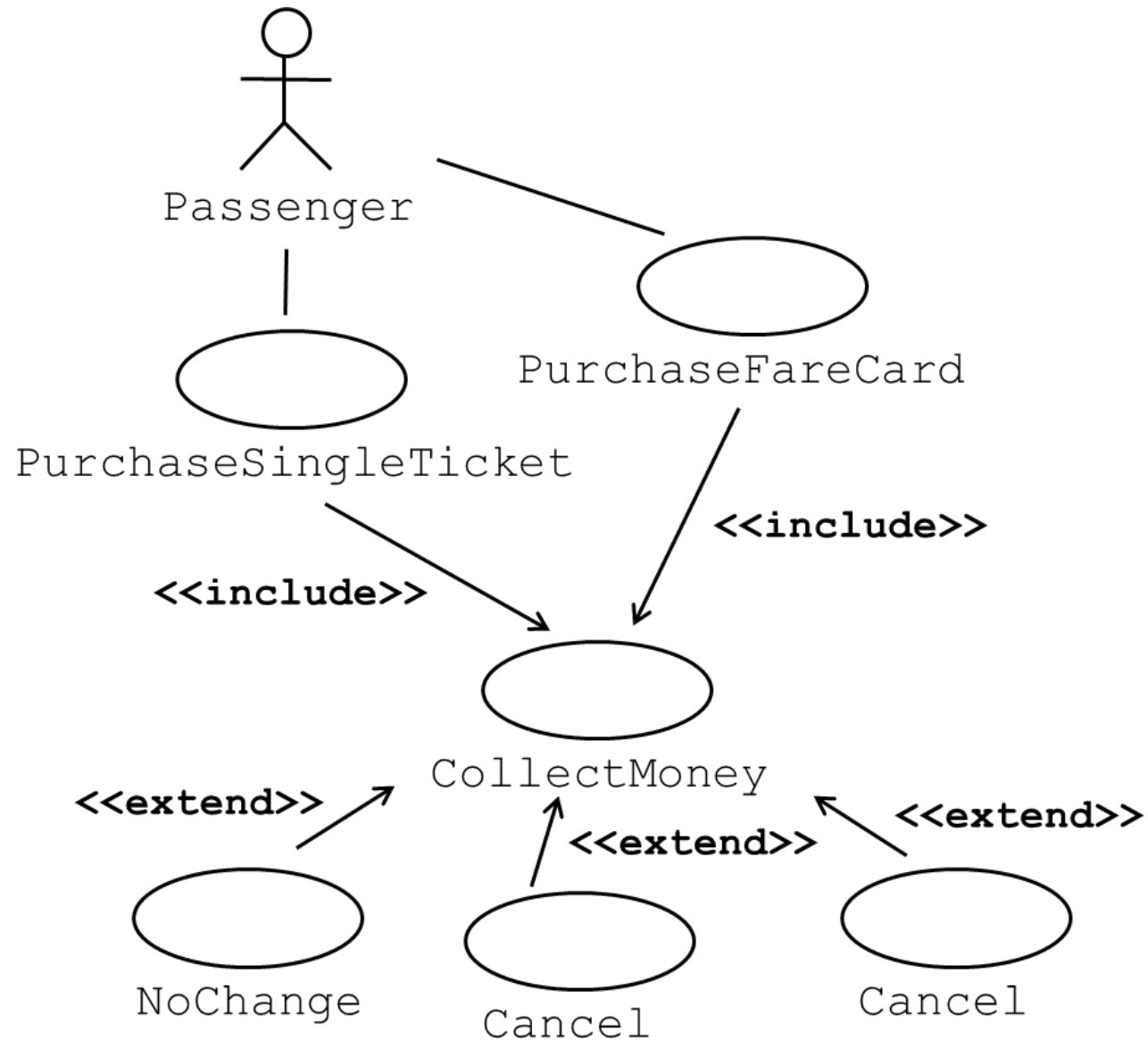


# The <<extend>> Relationship

- <<extend>> relationships model exceptional or seldom invoked cases
- The exceptional event flows are factored out of the main event flow for clarity
- The direction of an <<extend>> relationship is to the extended use case, not the extension
- Use cases representing exceptional flows can extend more than one use case.

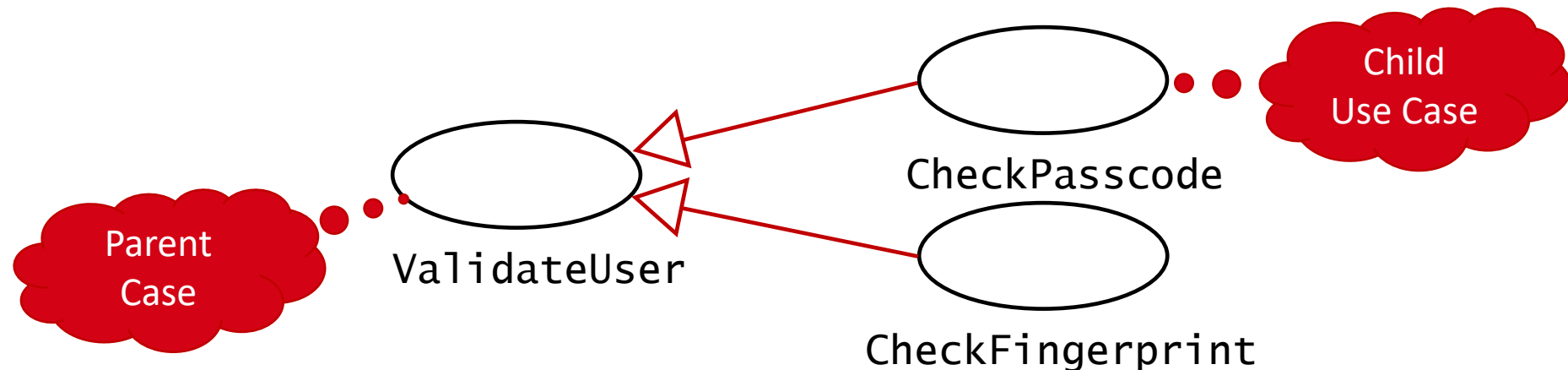


## Both relationships used in a UC



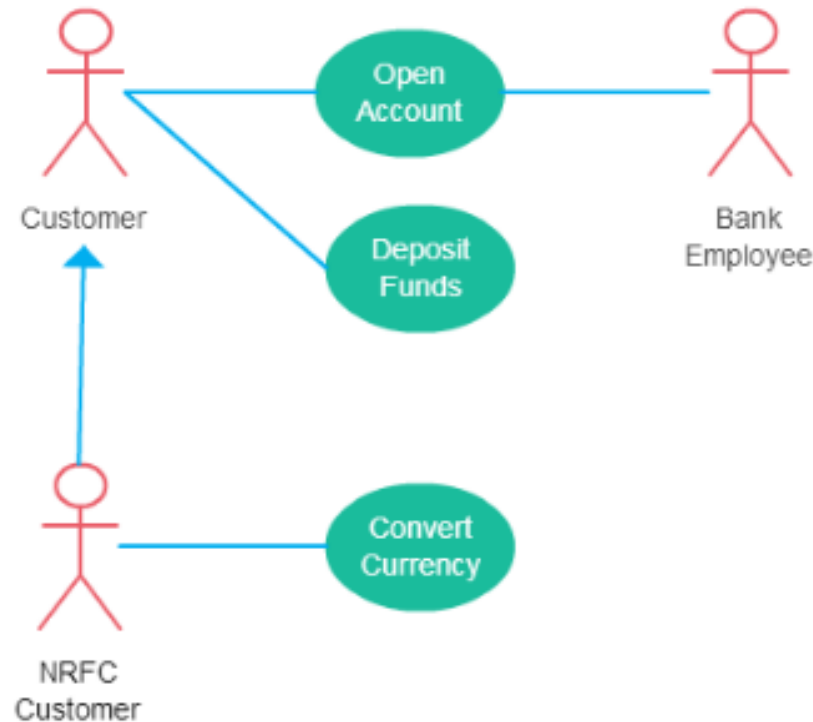
# Generalization in Use Cases

- **Problem:** We want to factor out common (but not identical) behavior.
- **Solution:** The child use cases inherit the behavior and meaning of the parent use case and add or override some behavior.
- **Example:** “ValidateUser” is responsible for verifying the identity of the user. The customer might require two realizations: “CheckPasscode” and “CheckFingerprint”



# Generalization between actors

- Generalization of an actor means that one actor can inherit the role of the other actor.
- The descendant inherits all the use cases of the ancestor.
- The descendant has one or more use cases that are specific to that role.



# Another Use Case Example

Actor **Bank Customer**

- Person who owns one or more Accounts in the Bank.

## **Withdraw Money**

- The Bank Customer specifies an Account and provides credentials to the Bank proving that s/he is authorized to access the Account.
- The Bank Customer specifies the amount of money s/he wishes to withdraw.
- The Bank checks if the amount is consistent with the rules of the Bank and the state of the Bank Customer's account. If that is the case, the Bank Customer receives the money in cash.

# Use Case Attributes

## Use Case **Withdraw Money Using ATM**

Initiating actor:

- Bank Customer

Preconditions:

- Bank Customer has opened a Bank Account with the Bank **and**
- Bank Customer has received an ATM Card and PIN

Postconditions:

- Bank Customer has the requested cash **or**
- Bank Customer receives an explanation from the ATM about why the cash could not be dispensed

# Use Case Flow of Events

## Actor steps

**1.The Bank Customer inputs the card into the ATM.**

**3.The Bank Customer types in PIN.**

**5. The Bank Customer selects an account.**

**7. The Bank Customer inputs an amount.**

## System steps

**2.The ATM requests the input of a four-digit PIN.**

**4. If several accounts are recorded on the card, the ATM offers a choice of the account numbers for selection by the Bank Customer**

**6.If only one account is recorded on the card or after the selection, the ATM requests the amount to be withdrawn.**

**8.The ATM outputs the money and a receipt and stops the interaction.**

# Use Case Exceptions

## Actor steps

1. The Bank Customer inputs her card into the ATM. **[Invalid card]**
3. The Bank Customer types in PIN. **[Invalid PIN]**
5. The Bank Customer selects an account .
7. The Bank Customer inputs an amount. **[Amount over limit]**

**[Invalid card]**

The ATM outputs the card and stops the interaction.

**[Invalid PIN]**

The ATM announces the failure and offers a 2nd try as well as canceling the whole use case. After 3 failures, it announces the possible retention of the card. After the 4th failure it keeps the card and stops the interaction.

**[Amount over limit]**

The ATM announces the failure and the available limit and offers a second try as well as canceling the whole use case.



# Guidelines for Formulation of Use Cases (1)

- Name

- Use a verb phrase to name the use case.
- The name should indicate what the user is trying to accomplish.
- Examples:
  - “Request Meeting”, “Schedule Meeting”, “Propose Alternate Date”

- Length

- A use case description should not exceed 1-2 pages. If longer, use include relationships.
- A use case should describe a complete set of interactions.

# Guidelines for Formulation of Use Cases (2)

Flow of events:

- Use the active voice. Steps should start either with “The Actor” or “The System ...”.
- The causal relationship between the steps should be clear.
- All flow of events should be described (not only the main flow of event).
- The boundaries of the system should be clear.  
Components external to the system should be described as such.
- Define important terms in the glossary.

## Example of a badly written Use Case

“The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through.”

What is wrong with this use case?

# Example of a badly written Use Case

“The driver arrives at the parking gate, the driver receives a ticket from the distributor, the gate is opened, the driver drives through.”

## Problems:

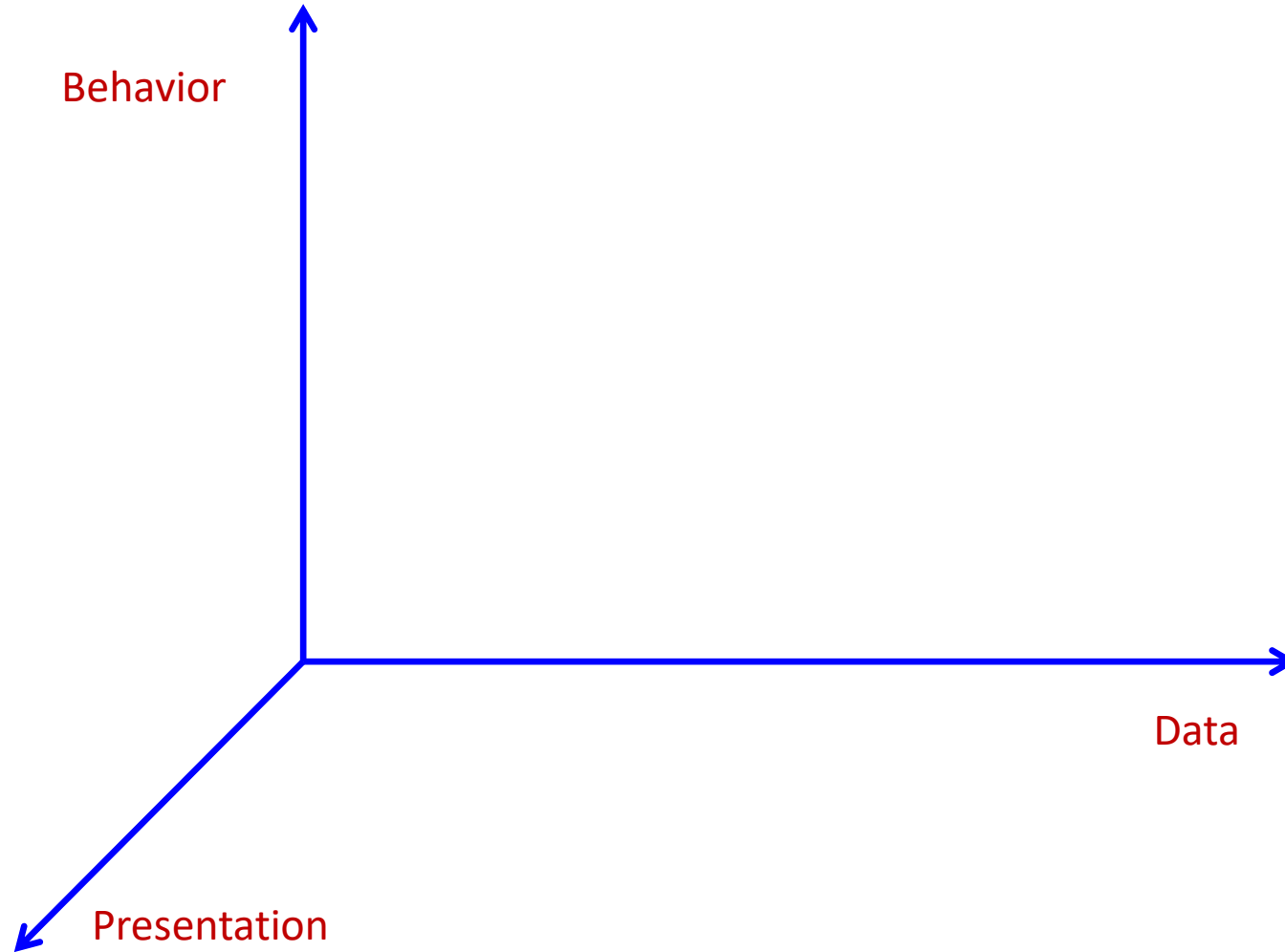
- It is not clear which action triggers the ticket being issued.
- Because of the passive form, it is not clear who opens the gate (The driver? The distributor? An attendant?)
- It is not a complete transaction. A complete transaction would also describe the driver paying for parking and driving out of the parking lot.

## Use case description Missing details ...

Software is specified in terms of:

- Behavior
  - Specified as a set of use cases
- Data
  - Specified as a domain object model
- Presentation (user interface)
  - Specified as a set of screens (interface mockups)

# Behavior, Data, and Presentation



# Domain Object Model

Domain Object Model captures domain objects in the context of the system.

- Business objects -- entities manipulated by the system (student record, bank account, result of an experiment, etc.)
- Real life objects (pressure sensor, keypad on a gas pump, keypad on an ATM machine)
- Events (button pressed, nozzle handle lifted, message received)

**Goal: do not specify the internals of the system!**

# Domain Object Model

## Domain Objects

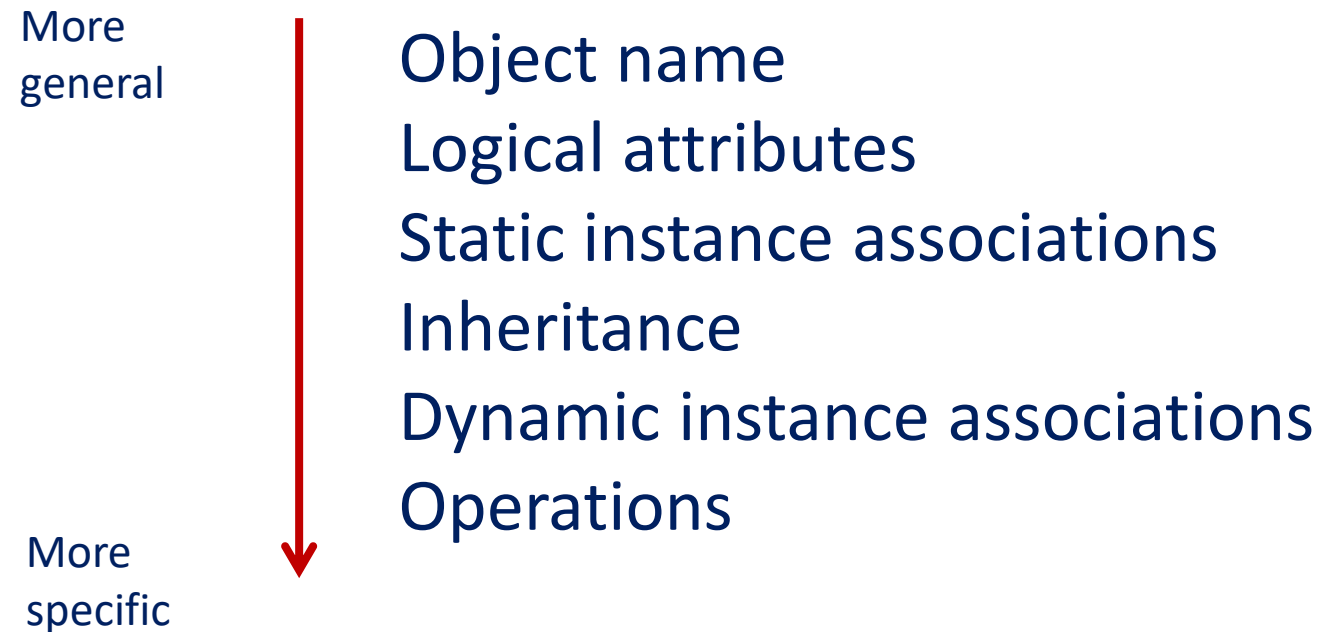
- Are a good tool to communicate with customer and users about the system
- Customer and users will recognize the concepts
- Many are identified during the use scenario and case drafting; additional identified later

**Goal: define a common base of understanding.**



# Problem Domain Objects

## Levels of definition refinement (Jacobson '92)



**Goal: stop as soon as sufficient details have been specified.**

## Problem Domain Objects

Identifying problem domain objects (heuristics):

- terms used by users or developers to clarify the understanding of use cases
- recurring nouns
- real world entities
- data sources and sinks (scanner, printer)
- interface artifacts (slider, button)
- real world processes maintained by the system.

**Goal: use application domain terms.**

## Interface Description

Interface description should include sketches of what the actors will see when performing use cases.

Essential to have users involved – the user interface sketches must reflect the user's logical view of the system.

Critical to maintain the consistency between the actor's conceptual picture of the system and the system's actual behavior and presentation.

**Goal: Stress not appearance, but functionality.**

## Interface Description

When creating a *logical user interface design*, use typical components of UI for interaction (windows, dialogs, lists, text entry fields, icons, buttons, etc.)

### Focus on:

- what an actor is going to need to interact with the use case.
- what information an actor needs to supply to the system?
- what information the system needs to supply to an actor?

# Interface Screen Mock-up

File Edit View Tools Reports Service Help

Transfer Funds

From Account

FromAccount

To Account

ToAccount

Transfer Amount

Amount

Cancel

OK

Screen: SC-Transfer-1

# Interface Screen Mock-up

File Edit View Tools Reports Service Help

**Transfer Funds**

From Account

To Account

Transfer Amount

Cancel OK

This is the important part

Extraneous parts parts, including Graphics & color

Screen: SC-Transfer-1

## Describing Interaction Flows

- Interaction steps should be alternatively an actor's input and a corresponding system's response.
- When needed, interaction steps should refer to domain objects, preferably to their logical attributes and relationships.
- When needed, interaction steps should refer to interface sketches, preferably to their specific controls (text areas, labels, buttons, etc.).

# Describing Interaction Flows

For example:

1. The *Bank Customer* selects one of the listed *Accounts*.
2. The system displays the *Account* information, including the *Account.balance* and the last 10 *Transactions* (see *Screen-234*).



## Describing Interaction Flows

- Alternative paths should start with a conditional branch from the basic path.
- An alternative path may merge back into the basic path or terminate the use case on its own.
- Similarly, exceptional paths branch conditionally from the basic path and may terminate the use case or merge back into the basic path.

## Describing Interaction Flows

For example:

1. If in step 4 the *Loan.requestedAmount* is greater than \$300,000, the system invokes the *NotifyLoanManager* use case.
2. ...

# Specifying Use Cases: Content

Name	
ID, Version	Unique identifier and version number
Author	
Date	
Summary	Brief description of the use case
Assumption	General assumptions about the use case
Triggers	What triggers (starts) the use case
Pre-conditions	
Post-conditions	
Basic Path	Most common sequence of interactions
Alternative Paths	Conditional branches from the basic path
Exception Paths	Conditional exceptions (failures) from the basic path
Extension Points	Conditional triggers for extensions

# Documenting Use Cases

- Use case templates are on eLC (RequisitrPro template)

Example:

Cafeteria Ordering System

The word document is on eLC in the case studies folder

# Requirements Specification Document

1. Introduction
2. Current system (skip for greenfield engineering, and replace with a literature review of similar syaytem)
3. Proposed system
  - 3.1 Overview
  - 3.2 Functional requirements (fill in product features and use case description for all use cases)
  - 3.3 Nonfunctional requirements
  - 3.4 Constraints (“Pseudo requirements”)
  - 3.5 System models
    - 3.5.1 Scenarios
    - 3.5.2 Use case model
    - 3.5.3 Domain object model
      - 3.5.3.1 Data dictionary (domain objects)
      - 3.5.3.2 Class diagrams (not needed now)
    - 3.5.4 Dynamic models (not needed now)
    - 3.5.5 User interface (screen sketches)
4. Glossary

# Requirements Specification Document

## 1. Introduction

- Describe the the overall enterprise
- Provide an overview of the system to be developed, highlighting the role it will fulfill in the enterprise (or within the context it be used
- This section should be concise and to the point. The details will be provided later

## 2. Current system (empty for greenfield engineering)

- Provide an overview of the current system, what role it fulfills in the enterprise (or within the context it is used
- Describe why a new system will be built and highlight the differences between the old and new systems
- This section should be concise and to the point

# Requirements Specification Document

## 3. Proposed system

### 3.1 Overview

### 3.2 Functional requirements

- Include a comprehensive list of the functional requirements. Just list the requirements providing a brief explanation for each requirement (not their descriptions)
- Subdivide the requirements to form logical groups. Hint: start with each actor, and then split into smaller sets, if needed, that make sense
- All functional requirements should have unique identifiers, at least numbers
- This is a “birds-eye view” of all functional requirements

# Requirements Specification Document

## **3.3 Nonfunctional requirements**

- 3.3.1 User interface and human factors
- 3.3.2 Documentation
- 3.3.3 Hardware considerations
- 3.3.4 Performance characteristics
- 3.3.5 Error handling and extreme conditions
- 3.3.6 System interfacing
- 3.3.7 Quality issues
- 3.3.8 System modifications
- 3.3.9 Physical environment
- 3.3.10 Security issues
- 3.3.11 Resources and management issues



# Requirements Specification Document

## 3.4 Constraints (Pseudo-requirements)

Describe any requirements concerning:

- deployment platform (e.g., RedHat Enterprise Linux)
- implementation languages (e.g., C++ and Python)
- design and implementation tools (must be cross-compiled using GCC from a Linux platform)
- middleware components (must use Hibernate)
- type of the user interface (must be Web-based)
- packaging (e.g. delivered as RPM packages)
- legal and licensing (must have Apache license)

# Requirements Specification Document

## 3.5 System models

### 3.5.1 Scenarios

### 3.5.2 Use case model

Include your use case descriptions here

### 3.5.3 Domain object model

#### 3.5.3.1 Data dictionary

This section should include all domain objects:

- describe each of the domain objects
  - name
  - brief description
  - list all logical attributes (include types, when known, but avoid programming language types)
  - list and briefly describe all static relationships (e.g. a BankCustomer owns one or more Accounts)
- use 1 page per object for clarity and ease of incremental updates

# Requirements Specification Document

3.5.3.2 Class diagrams (not needed now)

3.5.4 Dynamic models (not needed now)

3.5.5 User interface (screen sketches)

Include screen sketches (mock-ups).

Screens should have unique identifiers for easy referencing in UC descriptions in section 3.5.2.

You may use an HTML editor to quickly create screens/forms.

## 4. Glossary

Include definitions and explanations of some uncommon but important concepts.

DO not include common terms and concepts, “just to have something here”.

# Requirements Specification Document

Important points:

- Use a common UC template
- Use the same word processor/editor
- Make the document as cohesive and uniform in appearance
- Number all pages
- Include a table of contents, listing all use cases, and other individual items and providing their page numbers
- Get a 3-ring binder and put together your final submission
- **The document should be prepared incrementally as you are submitting deliverables. Should be submitted to eLC on you're your final project demonstration day.**

# Tools

- You can start with setting your plan (project schedule) using Microsoft Planner.

<http://www.projectviewercentral.com/download/>

- **UML Modeling tools:**

- Modelio:

**An open source modeling environment ( windows, Mac OS and linux)**

<https://www.modelio.org/>

- MicroSoft Visio: <https://products.office.com/en-us/visio/visio-professional-free-trial-flowchart-software>
- **StarUML:** <http://staruml.io/> or at <https://sourceforge.net/projects/staruml/>
- **ArgoUML:** <https://argouml.en.softonic.com/>
- **UMLDesigner:** <http://www.uml designer.org/>
- **Rational Software Architect Designer**

<http://www.ibm.com/developerworks/downloads/r/architect/index.html>

# Additional Readings

- *Scenario-Based Design: Envisioning Work and Technology in System Development*, by John M. Carroll, John Wiley, 1995.
- *Requirements Analysis and System Design (3rd Ed.)*, by Leszek A. Maciaszek, Addison Wesley, 2007.
- *Use Case Driven Object Modeling with UML. Theory and Practice*, by Doug Rosenberg and Matt Stephens, Apress, 2007.
- *Use Cases: Patterns and Blueprints*, by Gunnar Overgaard and Karin Palmkvist, Addison-Wesley, 2004.
- *Writing Effective Use Cases*, by Alistair Cockburn, Addison-Wesley, 2000.
- *Advanced Use Case Modeling: Software Systems*, by Frank Armour, Addison-Wesley, 2000.
- *Use Cases: Requirements in Context (2nd Ed.)*, by Daryl Kulak and Eamonn Guiney, Addison-Wesley, 2003.