Object-Oriented Software Engineering
Practical Software Development using UML and Java

Focusing on Users and Their Tasks

7.1 User Centred Design

Software development should focus on the needs of users

- Understand your users
- Design software based on an understanding of the users' tasks
- Ensure users are involved in decision making processes
- Design the user interface following guidelines for good usability
- Have users work with and give their feedback about prototypes, on-line help and draft user manuals

The importance of focusing on users

- Reduced training and support costs
- Reduced time to learn the system
- Greater efficiency of use
- Reduced costs by only developing features that are needed
- Reduced costs associated with changing the system later
- Better prioritizing of work for iterative development
- Greater attractiveness of the system, so users will be more willing to buy and use it

7.2 Characteristics of Users

Software engineers must develop an understanding of the users

- Goals for using the system
- Potential patterns of use
- Demographics
- Knowledge of the domain and of computers
- Physical ability
- Psychological traits and emotional feelings

What is a Use Case

- Use Cases are the main tasks performed by the users of the system.
- Use Cases describe the behavioral aspects of the system.
- Use Cases are used to identify how the system will be used.
- Use Cases are a convenient way to document the functions that the system must support.
- Use Cases are used to identify the components (classes) of the system.

Development of a Use Case Diagram

1. **Identify all of the actors who will use the system.**
2. **Interview these actors to identify the functions that they need to perform. (use cases)**
3. **Identify scenarios (sequence of steps to accomplish a use case).**
4. **Identify common steps within the different scenarios. Separate them into different use cases so that they can easily be included in other scenarios.**
5. **Identify relationships between use cases.**

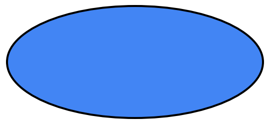
Use cases [1]

- **Excellent technique for improving the understanding of requirements**
- **Narrative in nature**
- **Use cases are dependent on having some understanding of the requirements (expressed in functional specifications document).**

Use Cases [2]

Use case - narration of the sequence of events of an actor using a system

UML icon for use case



Actors [1]

- **Actor - an entity external to the system that in some way participates in the use case. Actor represents a role that a user can play.**
- **An actor typically stimulates the system with input events or receives outputs from the system or does both.**
- **Actors are treated like classes and can be generalized.**

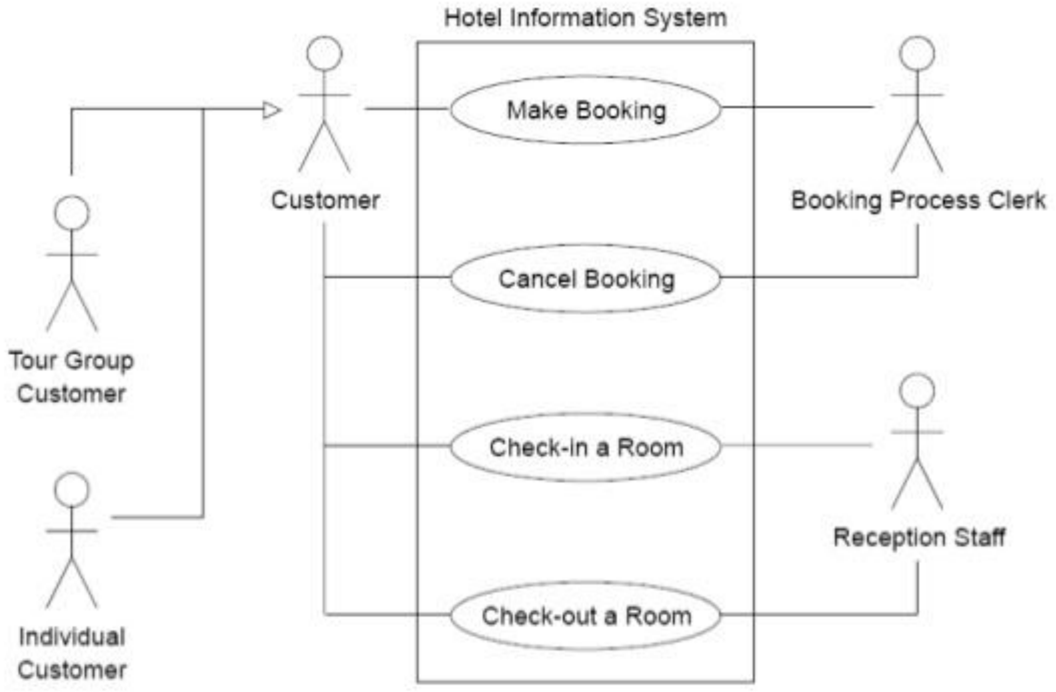
Primary actor: Use the system to achieve a goal. (found in left side of the use case diagram)

Secondary actor: plays a supporting role to facilitate the primary actor to achieve their goal. (right side)

Use-Case Diagram: Example

- Let us consider a simple hotel information system for *two* types of customers:
  - **Tour Group** customers and **Individual customers**
  - Tour Group customers are those who have made reservations through a tour operator in advance, while Individual customers make their reservations directly with the hotel
  - Both types of customers can **book**, **cancel**, **check-in** and **check-out** of a room by **phone** or **via the Internet**

Use-Case Diagram: Example



Identification of Use Cases

Method 1 - Actor based

- Identify the actors related to the system
- Identify the processes these actors initiate or participate in

Method 2 - Event based

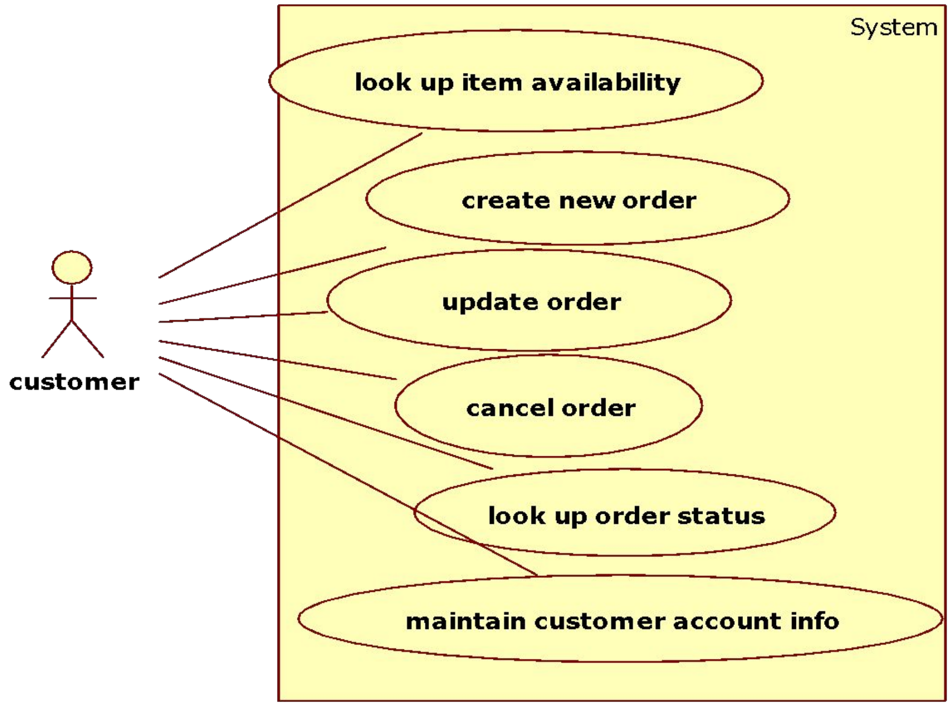
- Identify the external events that a system must respond to
- Relate the events to actors and use cases

Method 3 – Goal based

- [Actors have goals.]
- Find user goals. [Prepare actor-goal list.]
- Define a use case for each goal.

(***try to use several approaches for identifying use cases, and then cross-check to be sure no use cases have been overlooked***)

Actor based use cases



Event based use cases

- **External event**: Occurs outside the system, initiated by an external agent.

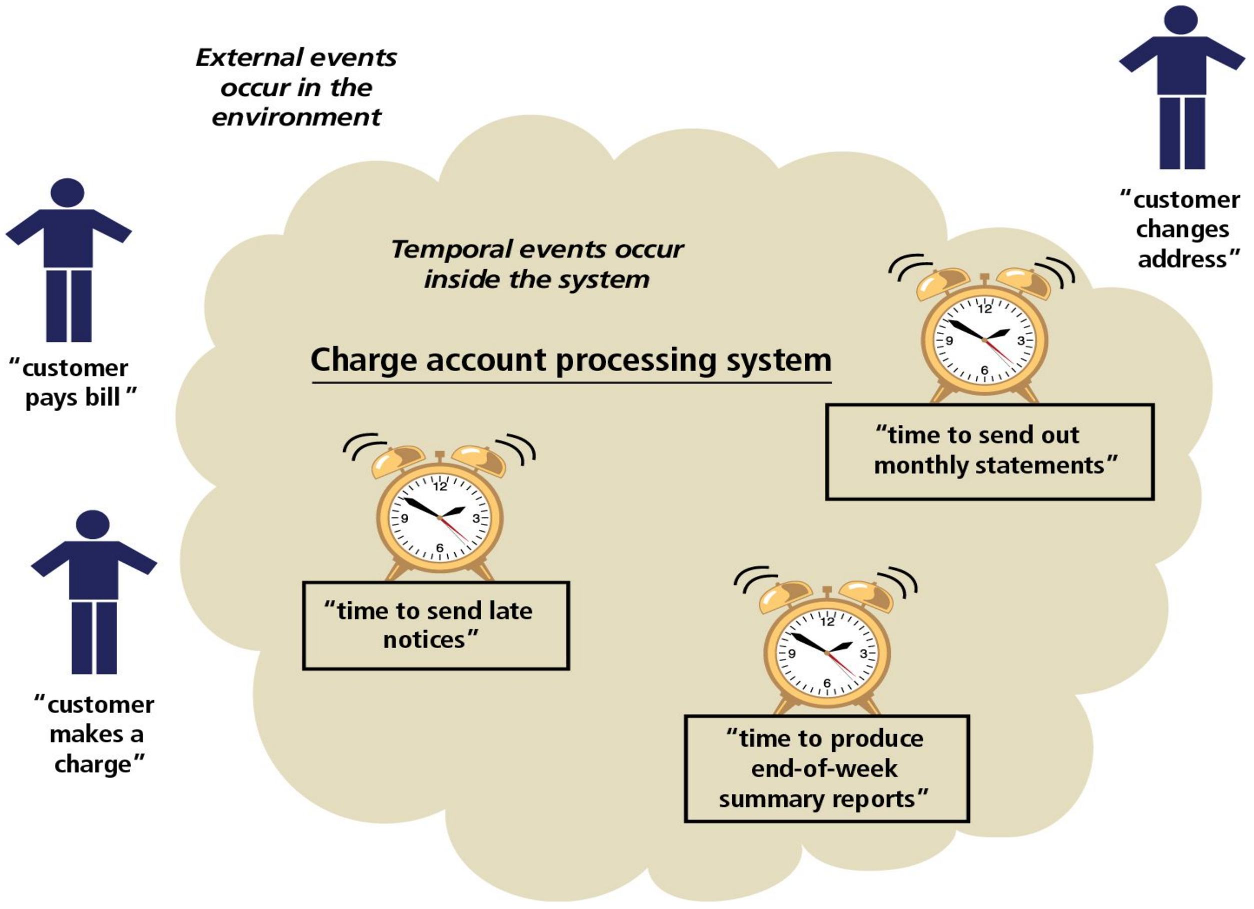
Example: "customer pays bill", "customer changes address", "management wants some information".

- **Temporal event**: occurs as a result of reaching a point in time. It occurs inside the system. System produces some output automatically without being told to do so.
  No external agent is making demands, but the system is supposed to generate needed information.

Example: "time to send late notice", "time to send monthly statements", "time to produce weekly summary report".

- **State event:** occurs when something happens inside the system that triggers the need for processing.

Example: if the sales of a product results in an adjustment to an inventory record and the inventory in stock drops below a Reorder point reached.

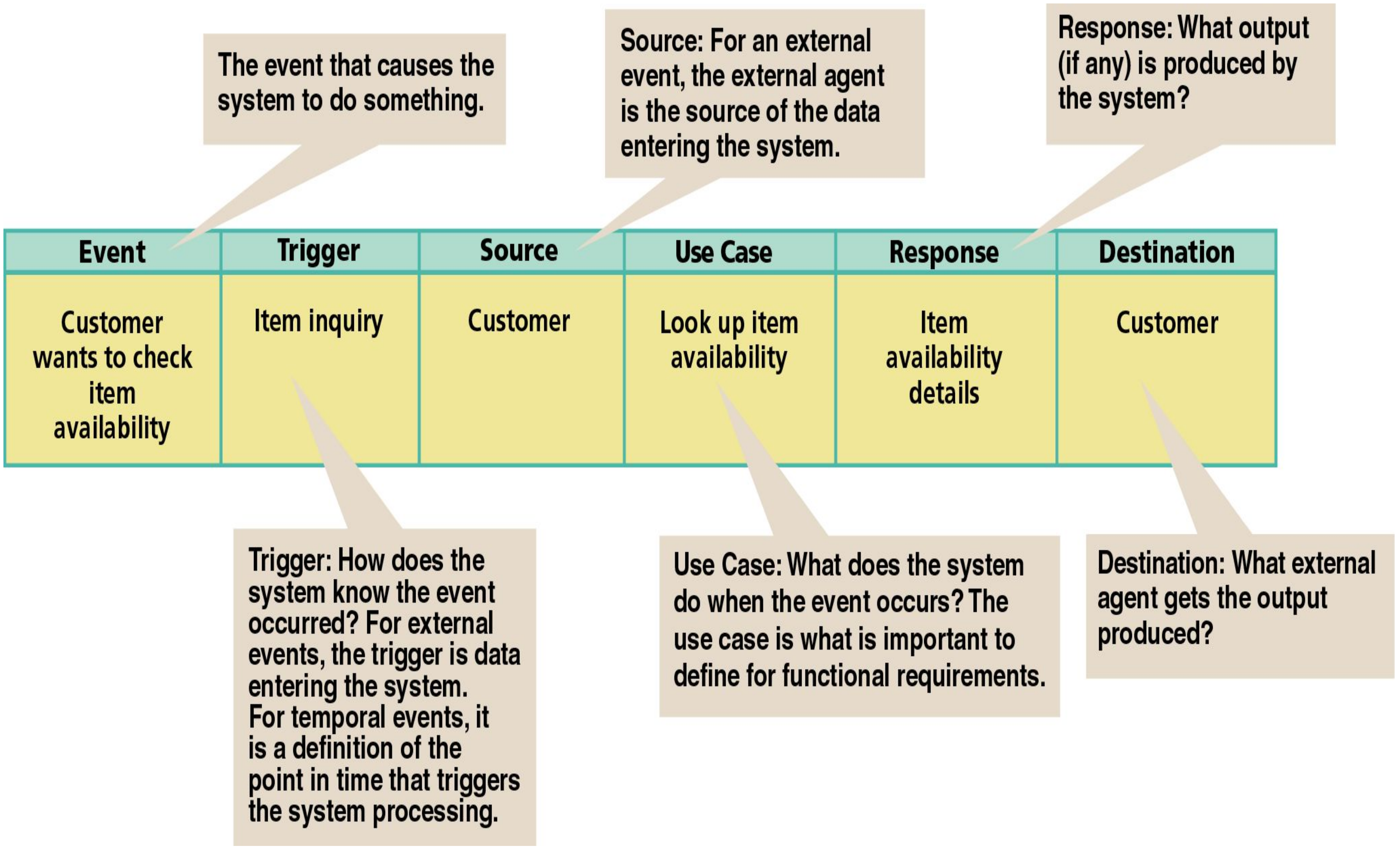

External events occur in the environment
"customer changes address"
"customer pays bill"
Temporal events occur inside the system
Charge account processing system
"time to send out monthly statements"
"time to send late notices"
"customer makes a charge"
"time to produce end-of-week summary reports"

Events Affecting a Charge Account Processing System that Lead to Use Cases

Goal-based

Identifying use cases by focusing on users and their goals

USER\ACTOR	USER GOAL
customer	1. Look up item availability 2. Create new order 3. Update order
Shipping clerk	Record order fulfillment Record back order



The event that causes the system to do something.

Source: For an external event, the external agent is the source of the data entering the system.

Response: What output (if any) is produced by the system?

Event	Trigger	Source	Use Case	Response	Destination
Customer wants to check item availability	Item inquiry	Customer	Look up item availability	Item availability details	Customer

Trigger: How does the system know the event occurred? For external events, the trigger is data entering the system. For temporal events, it is a definition of the point in time that triggers the system processing.

Use Case: What does the system do when the event occurs? The use case is what is important to define for functional requirements.

Destination: What external agent gets the output produced?

Information about each Event and the Resulting Use Case in an Event Table

Identification of Use Cases[2]

To identify use cases, focus on *elementary business processes (EBP)*.

An EBP is a task performed by one person in one place at one time, in response to a business event. This task adds measurable business value and leaves data in a consistent state.

Point of Sale - Actors

Actors:

- Cashier
- Customer
- Supervisor

Choosing actors:

- Identify system boundary
- Identify entities, human or otherwise, that will interact with the system, from outside the boundary.
- Example: A *temperature sensing device* is an actor for a temperature monitoring application.

High level vs. Low Level Use cases[1]

- **Consider the following use cases:**
  - Log out
  - Handle payment
  - Negotiate contract with a supplier

These use cases are at different levels. Are they all valid? To check, use the EBP definition.

Log out: a secondary goal; it is necessary to do something but not useful in itself.

Handle payment: A necessary EBP. Hence a primary goal.

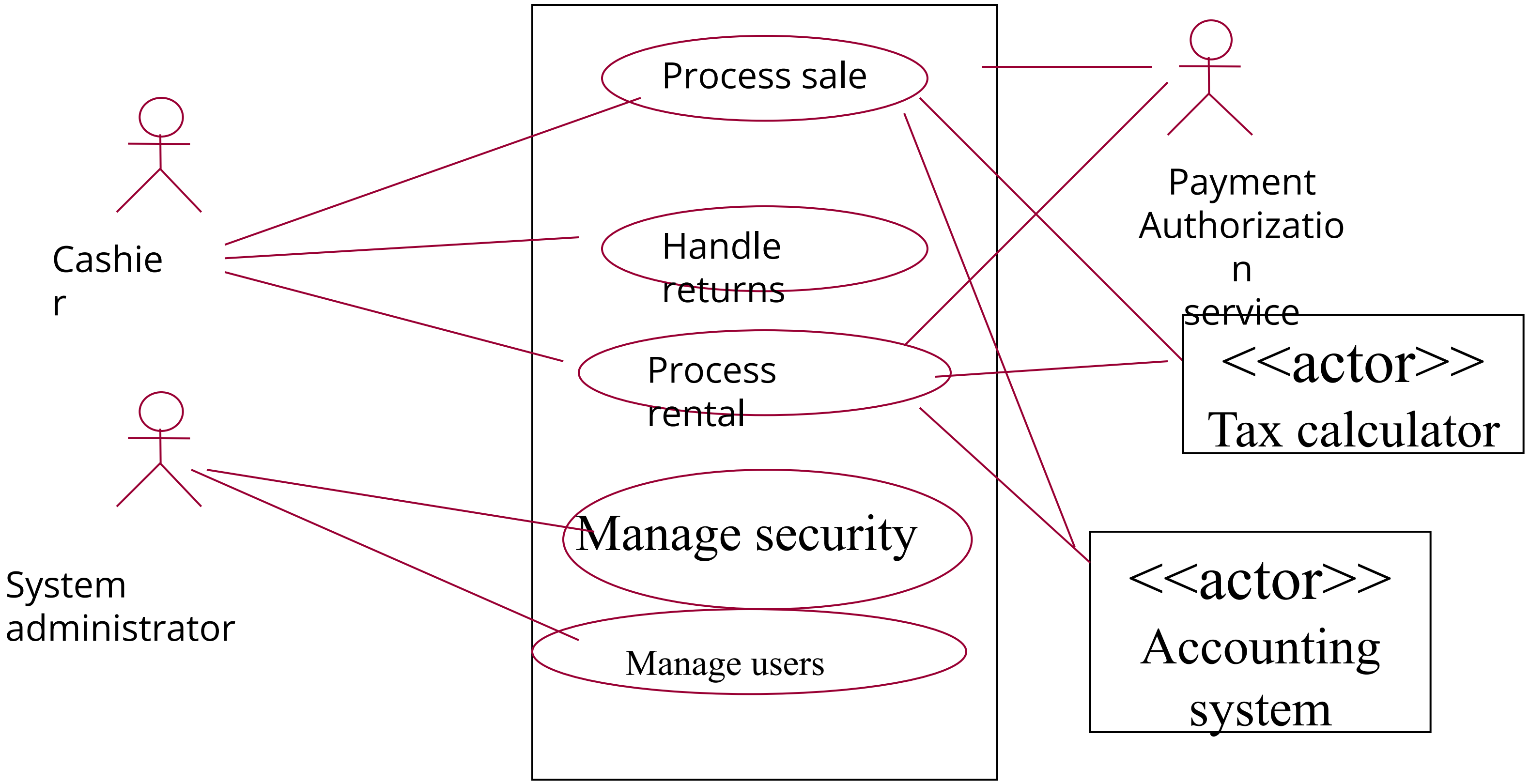
High level vs. Low Level Use cases [2]

Log out: a secondary goal; it is necessary to do something but not useful in itself.

Handle payment: A necessary EBP. Hence a primary goal.

Negotiate contract: Most likely this is too high a level. It is composed of several EBPs and hence must be broken down.

Use Case Diagram - Example

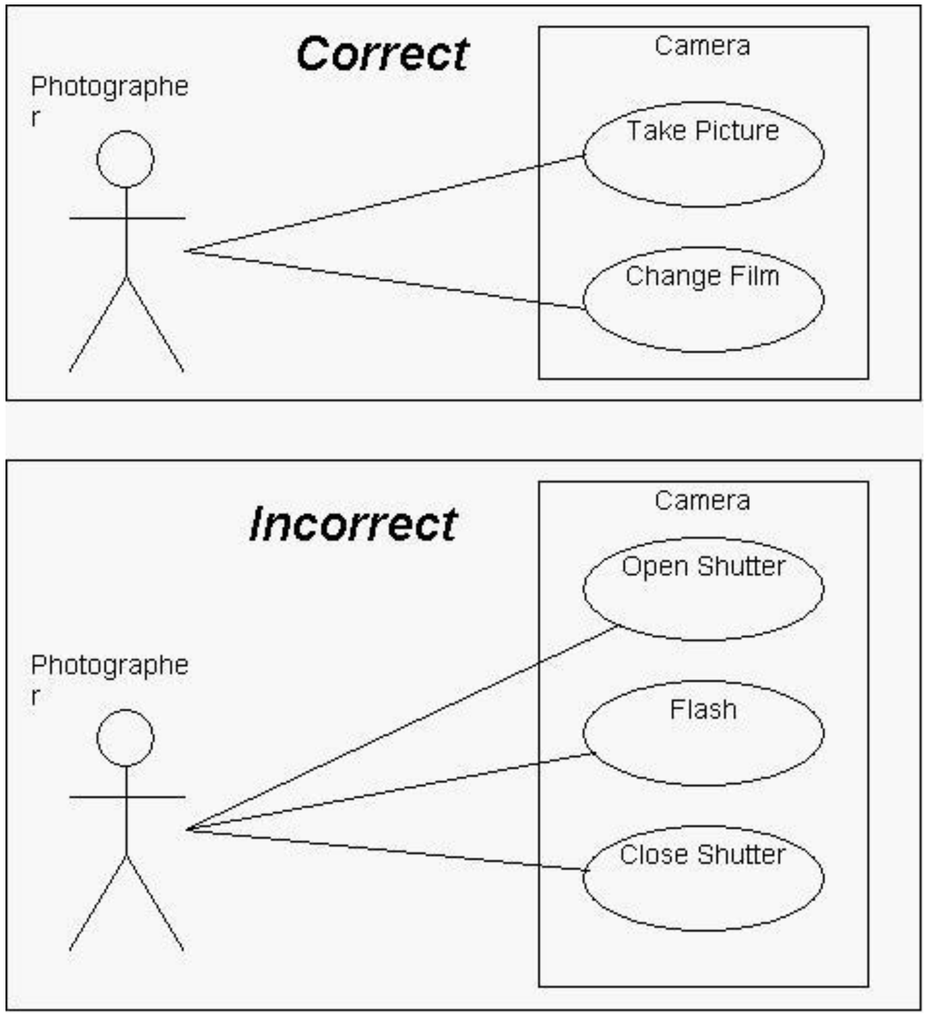


Use Case Diagram: illustrates a set of use cases for a system.

More on Use Cases

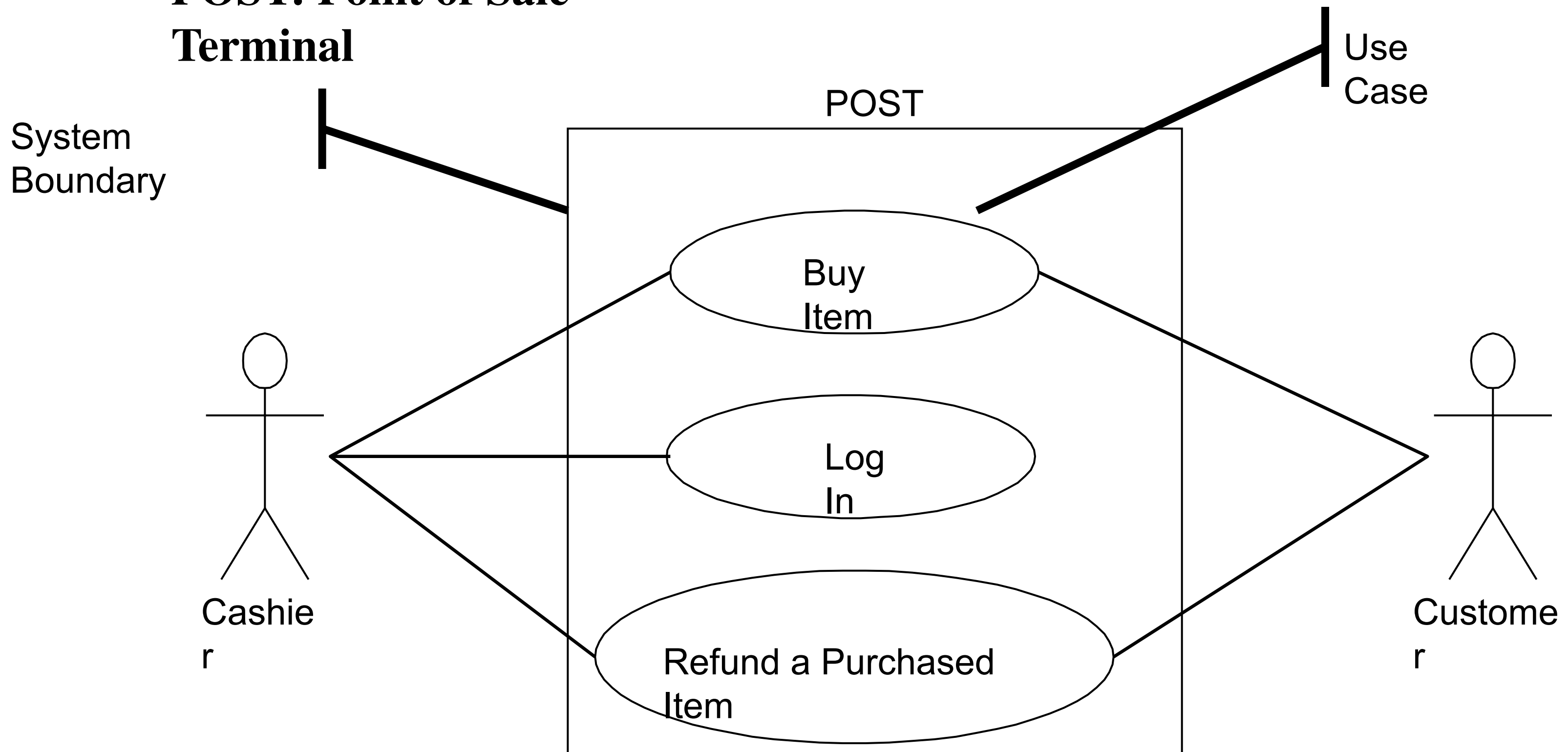
- **Try to describe use cases *independent* of implementation**
- **Be as narrative as possible**
- **State success scenarios (how do you measure the success of an use case)**
- **A use case can have many scenarios (threads of execution)**
- **Agree on a “format style” for use case description**
- **Name a use case starting with a verb in order to emphasize that it is a process (Buy Items, Enter an order, Reduce inventory)**

Identify high level functions



Use-Case Diagrams (POST)

POST: Point of Sale Terminal

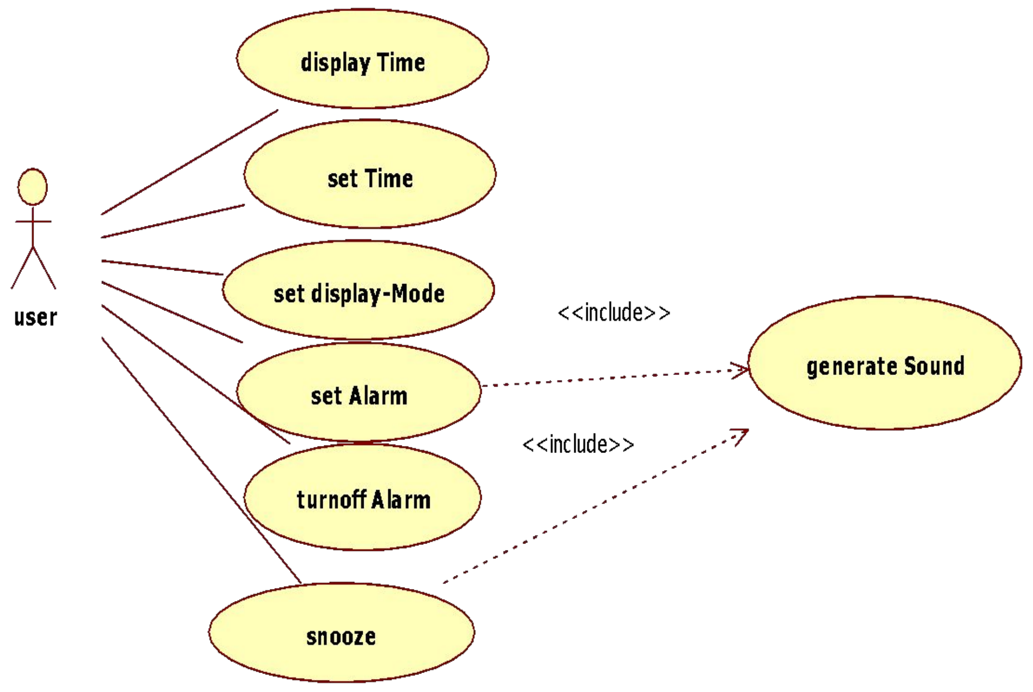


Question answer

Suppose we want to develop software for an alarm clock. The clock shows the time of day. Using buttons, the user can set the hours and minutes fields individually, and choose between 12 and 24-hour display. It is possible to set one or two alarms. When an alarm fires, it will sound some noise. The user can turn it off, or choose to ’snooze’. If the user does not respond at all, the alarm will turn off itself after 2 minutes. ’Snoozing’ means to turn off the sound, but the alarm will fire again after some minutes of delay. This ’snoozing time’ is pre-adjustable.

Identify the top-level functional requirement for the clock, and model it with a use case diagram.

Answer


display Time
set Time
set display-Mode
set Alarm
turnoff Alarm
snooze
user
<<include>>
<<include>>
generate Sound

Assume that 'snooze' would be one of your use cases. Work it out to the fully dressed level.

.

Use case: Snooze

Primary actor: User

Secondary actors: -

Pre-condition: An alarm is firing.

Post-condition: -

Main flow:

1. The use-case is activated when the user hits the snooze button.
2. The alarm is turned off.
3. Wait for snooze time.
4. Include use case 'Make sound'

Use case: Make sound

Primary actor: System

Secondary actors: -

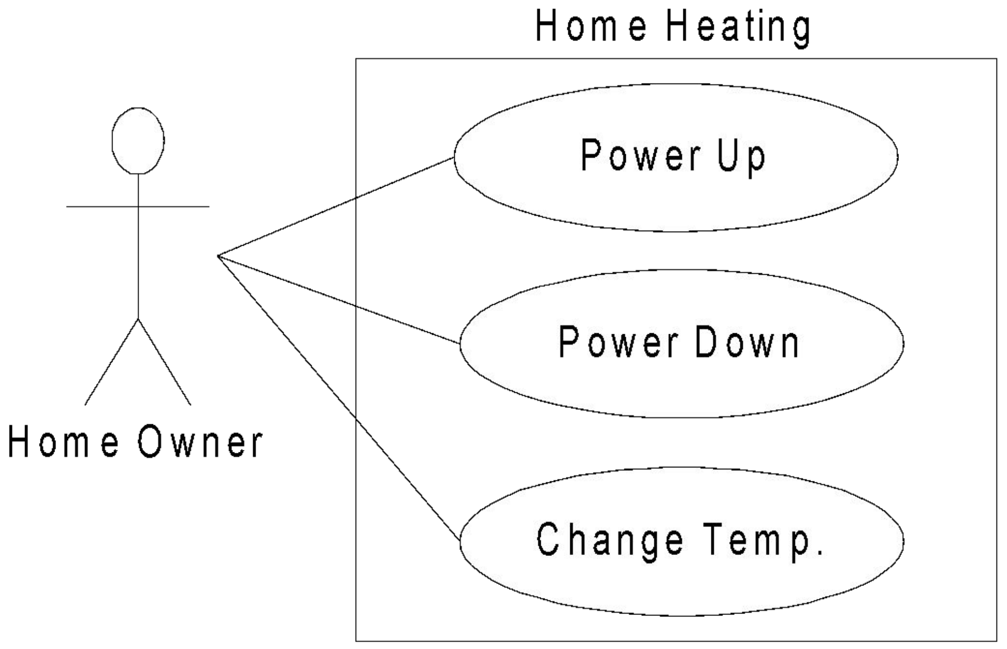
Pre-condition: -

Post-condition: -

Main flow:

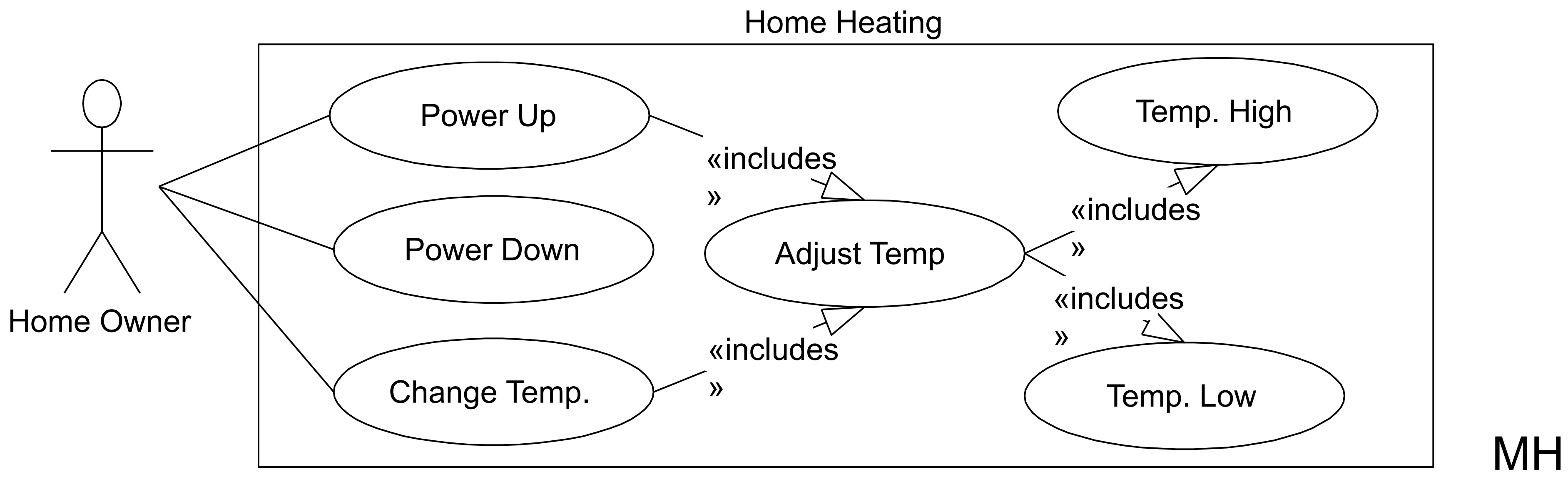
The use case starts when it is called. What it does is to just make some noisy sound.

Home Heating Use-Case Diagram



M H

Modified Home Heating



Modified: Home Heating Use-Cases

*

Use case: Power Up
Actors: Home Owner (initiator)
Type: Primary and essential
Description: The Home Owner turns the power on.
<u>Perform Adjust Temp</u>. If the temperature
in all rooms is
above the desired temperature, no actions are taken.
Cross Ref.: Requirements XX, YY, and ZZ
Use-Cases: Perform Adjust Temp

Modified: Home Heating Use-Cases

*

Use case:	**Adjust Temp**
Actors:	System (initiator)
Type:	Secondary and essential
Description:	Check the temperature in each room. For each room:
	Below target: **<u>Perform Temp Low</u>**
	Above target: **<u>Perform Temp High</u>**
Cross Ref.:	Requirements XX, YY, and ZZ
Use-Cases:	Temp Low, Temp High

Emphasize Goals

Investigating goals rather than tasks and procedures improves information gathering by focusing on the essence of requirements—the intent behind them.

Seeing requirements as identifying tasks to be done has a strong bias toward reproducing the existing system, even when it is being replaced because it is seriously defective.

Why Use Cases?

Simple and familiar storytelling makes it easier, especially for customers, to contribute and review goals.

Use cases keep it simple (KISS)

They emphasize goals and the user perspective.

New use case writers tend to take them too seriously.

Actors or Use Case First?

Because you have to understand each part of Use Cases, the parts are presented separately. But those who create use cases switch back and forth. The text describes use cases substantially before paying attention to actors. Typically, both actors and use cases are identified early and then examined to see if more use cases can be found from the actors, or more actors found by examining the use cases.

Identify Use Cases

Capture the specific ways of using the system as dialogues between an actor and the system.

Use cases are used to

- Capture system requirements
- Communicate with end users and Subject Matter Experts
- Test the system

Use Cases are not Diagrams

- **A use case is an activity the system carries out, in response to a request by a user.**
- **Techniques for identifying use cases:**
  1. List all users and think through what they need the system to do for their jobs.
  2. Start with the existing system and list all system functions that are currently included, adding any new functionality required by the user.
- **Use Cases may have a diagram associated with them, and a use case diagram is an easy way for an analyst to discuss a process with a subject matter expert.**

  But use cases are primarily text. The text is important. The diagram is optional.

Specifying Use Cases

Create a written document for each Use Case

- Clearly define intent of the Use Case
- Define Main Success Scenario (Happy Path)
- Define any alternate action paths
- Use format of Stimulus: Response
- Each specification must be testable
- Write from actor's perspective, in actor's vocabulary

www.usecases.org Template

Name
Primary Actor
Scope
Level
Stakeholders and Interests
Minimal Guarantee
Success Guarantee
Main Success Scenario
Extensions

This is the basic format used in the text and in Alistair Cockburn's *Writing Effective Use Cases* (Addison Wesley, 2000, ISBN 0201702258).

I prefer to modify it slightly to use the actor actions and system response in tabular form. Larman calls this the Two-Column Variation.

Identify Actors

We cannot understand a system until we know who will use it

- Direct users
- Users responsible to operate and maintain it
- External systems used by the system
- External systems that interact with the system

Specifying Actors

Actors are external to the system

Actors are non-deterministic

What interacts with the system?

Actors may be different roles that one individual user interacts with the system

Actors may be other systems

Don’t assume that Actor = Individual

Types of Actors

Primary Actor

- Has goals to be fulfilled by system

Supporting Actor

- Provides service to the system

Offstage Actor

- Interested in the behavior, but no contribution

In diagrams, Primary actors go on the left and others on the right.

Define Actors

Actors should not be analyzed or described in detail unless the application domain demands it.

Template for definition:

- Name
- Definition

Example for an ATM application:

Customer: Owner of an account who manages account by depositing and withdrawing funds

Identifying Actors

Primary Actor

- Emphasis is on the primary actor for the use case.

Stakeholders and Interests

- Other actors are listed as stakeholders.
- The interests of each key actor should be described.

Working with Use Cases

Determine the actors that will interact with the system
Examine the actors and document their needs
For each separate need, create a use case
During Analysis, extend use cases with interaction diagrams

Preconditions

Anything that must always be true before beginning a scenario is a precondition.

Preconditions are assumed to be true, not tested within the Use Case itself.

Ignore obvious preconditions such as the power being turned on. Only document items necessary to understand the Use Case.

Success Guarantees

Success Guarantees (or Postconditions) state what must be true if the Use Case is completed successfully. This may include the main success scenario and some alternative paths. For example, if the happy path is a cash sale, a credit sale might also be regarded a success.

Stakeholders should agree on the guarantee.

Scenarios

The Main Success Scenario, or “happy path” is the expected primary use of the system, without problems or exceptions.

Alternative Scenarios or Extensions are used to document other common paths through the system and error handling or exceptions.

Documenting the Happy Path

The Success Scenario (or basic course) gives the best understanding of the use case

Each step contains the activities and inputs of the actor and the system response

If there are three or more items, create a list

Label steps for configuration management and requirements traceability

Use present tense and active voice

Remember that User Interface designers will use this specification

Note: Do not use the term “happy path” in formal documents.

Extensions (Alternative Flows)

Extensions or Alternative Flow Use Cases allow the specification of

- Different ways of handling transactions
- Error processes

Sections are convenient way to handle alternative courses of action

- Sections are a segment of a use case executed out of sequence

Two Parts for Extensions

Condition

- Describe the reason for the alternative flow as a condition that the user can detect

Handling

- Describe the flow of processing in the same manner as the happy path, using a numbering system consistent with the original section.

Expanded Essential Use Cases

How to make one:

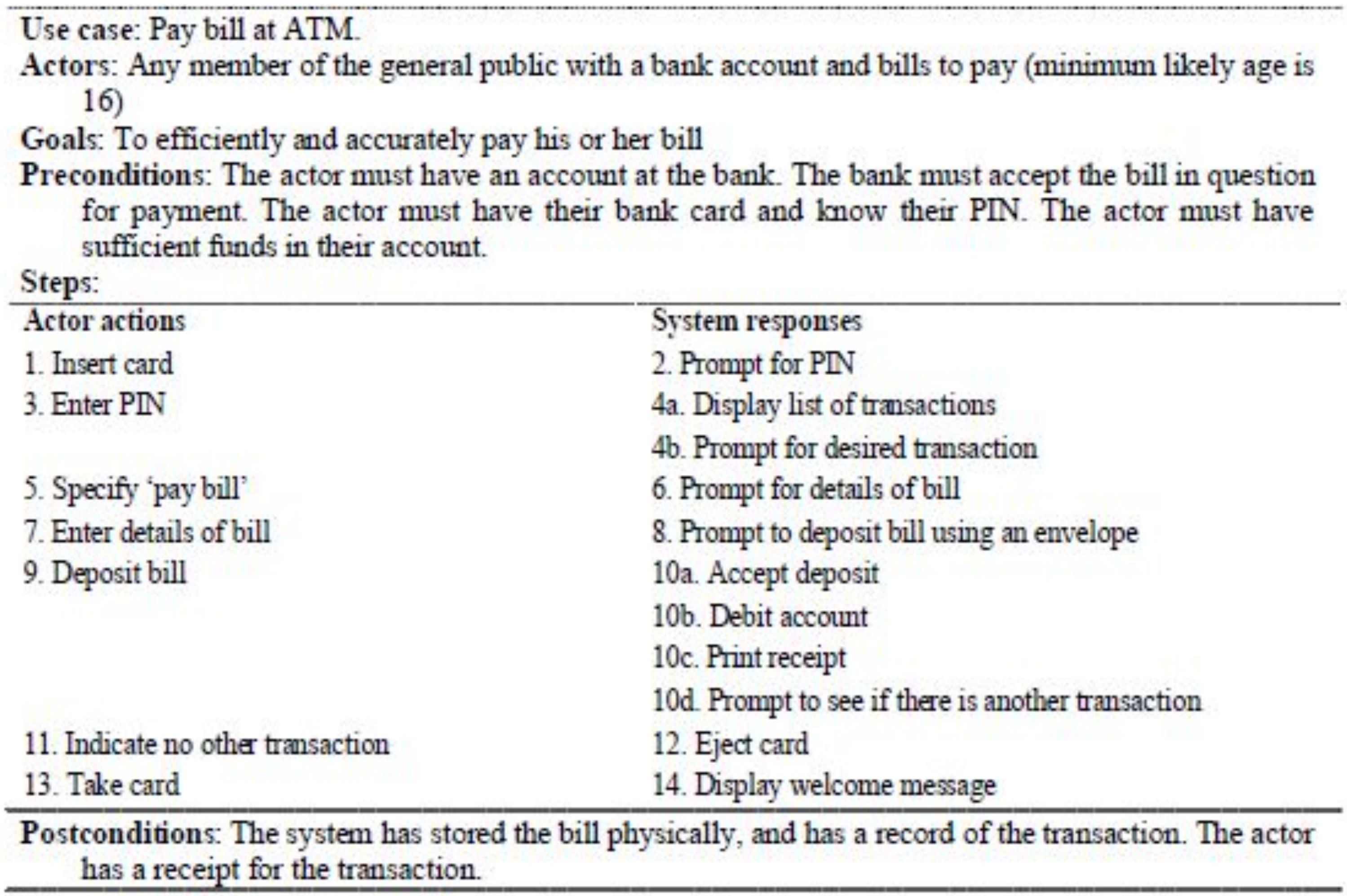
- Step 1: Name the Use Case (system function, e.g. “enter timesheet information”).
- Step 2: Identify the Actor(s) involved.
- Step 3: Describe the Intent of the Use Case in language the client will understand.
- Step 4: Identify the Assumptions and Limitations relevant to this Use Case and other Use Cases which the current one might extend or build upon.
- Step 5: Specify the ideal flow of actions using two columns labeled “Actor Actions” and “System Responses.” Number each step. This constitutes the Happy Path for this Use Case.
- Step 6: Identify opportunities for user error and create an Alternative Path to handle each.

More on Use Cases
Document exception handling or branching

- when a “Buy Item” fails, what is expected of the system
- when a “credit card” approval fails, what is expected of the system

Use Case ID:			
Use Case Name:			
Created By:		Last Updated By:	
Date Created:		Last Revision Date:	

Actors:	
Description:	
Trigger:	
Preconditions:	
Postconditions:	
Normal Flow:	
Alternative Flows:	
Exceptions:	
Includes:	
Frequency of Use:	
Special Requirements:	
Assumptions:	
Notes and Issues:	



Use case: Pay bill at ATM.

Actors: Any member of the general public with a bank account and bills to pay (minimum likely age is 16)

Goals: To efficiently and accurately pay his or her bill

Preconditions: The actor must have an account at the bank. The bank must accept the bill in question for payment. The actor must have their bank card and know their PIN. The actor must have sufficient funds in their account.

Steps:

Actor actions	**System responses**
1. Insert card	2. Prompt for PIN
3. Enter PIN	4a. Display list of transactions
	4b. Prompt for desired transaction
5. Specify 'pay bill'	6. Prompt for details of bill
7. Enter details of bill	8. Prompt to deposit bill using an envelope
9. Deposit bill	10a. Accept deposit
	10b. Debit account
	10c. Print receipt
	10d. Prompt to see if there is another transaction
11. Indicate no other transaction	12. Eject card
13. Take card	14. Display welcome message

Postconditions: The system has stored the bill physically, and has a record of the transaction. The actor has a receipt for the transaction.

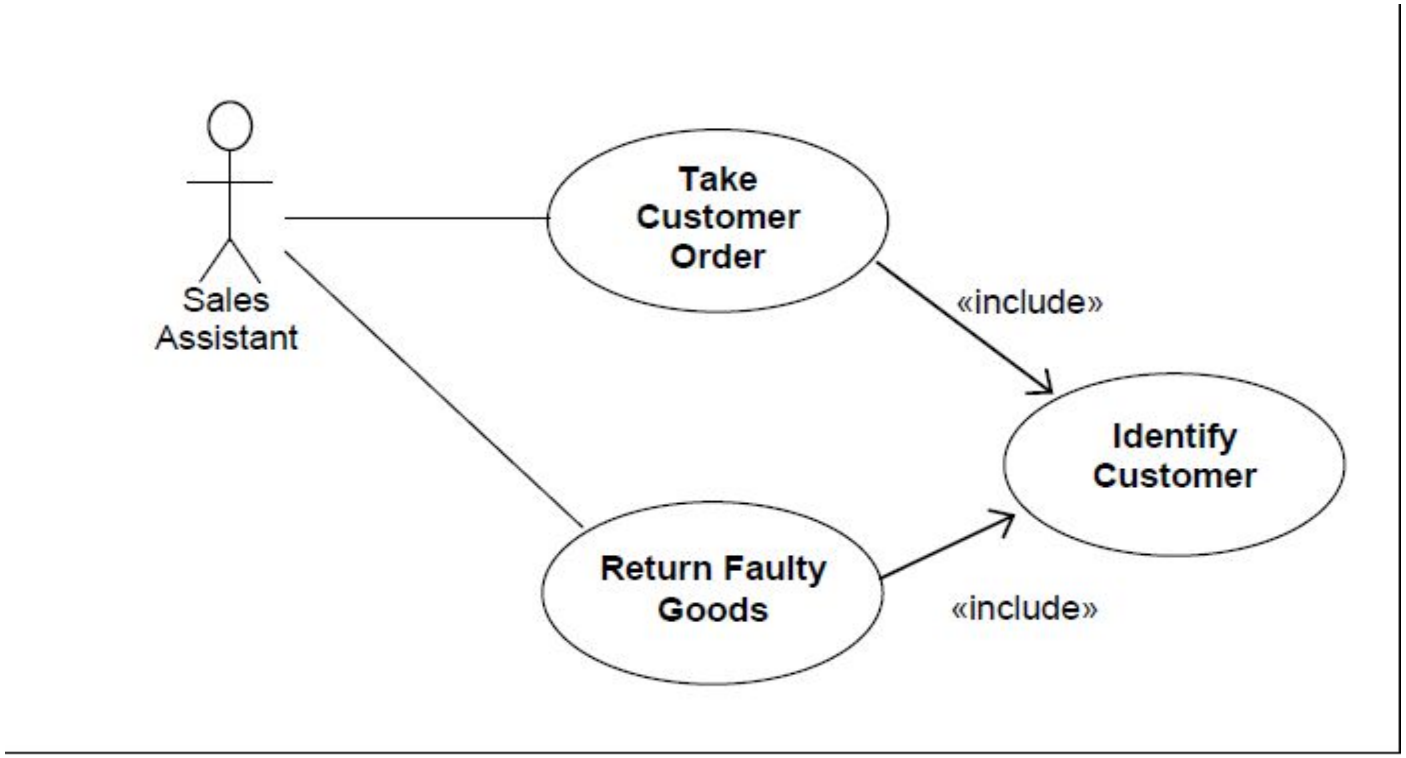
The «include» and «extend» Relationships in Use Case Models

UML defines three stereotypes of association between Use Cases

- **«include»**
- **«extend»**
- **generalisation**

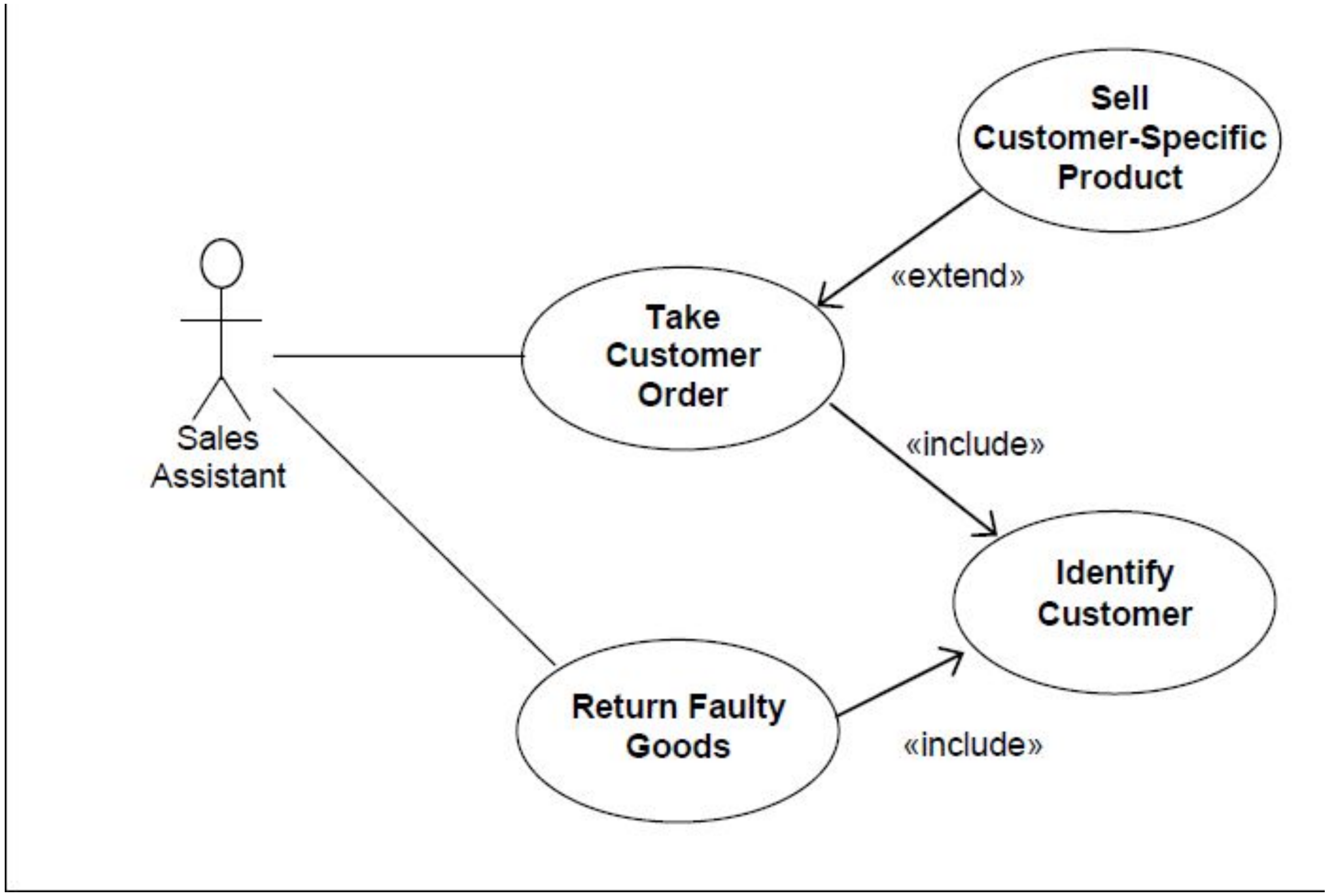
<<include>>

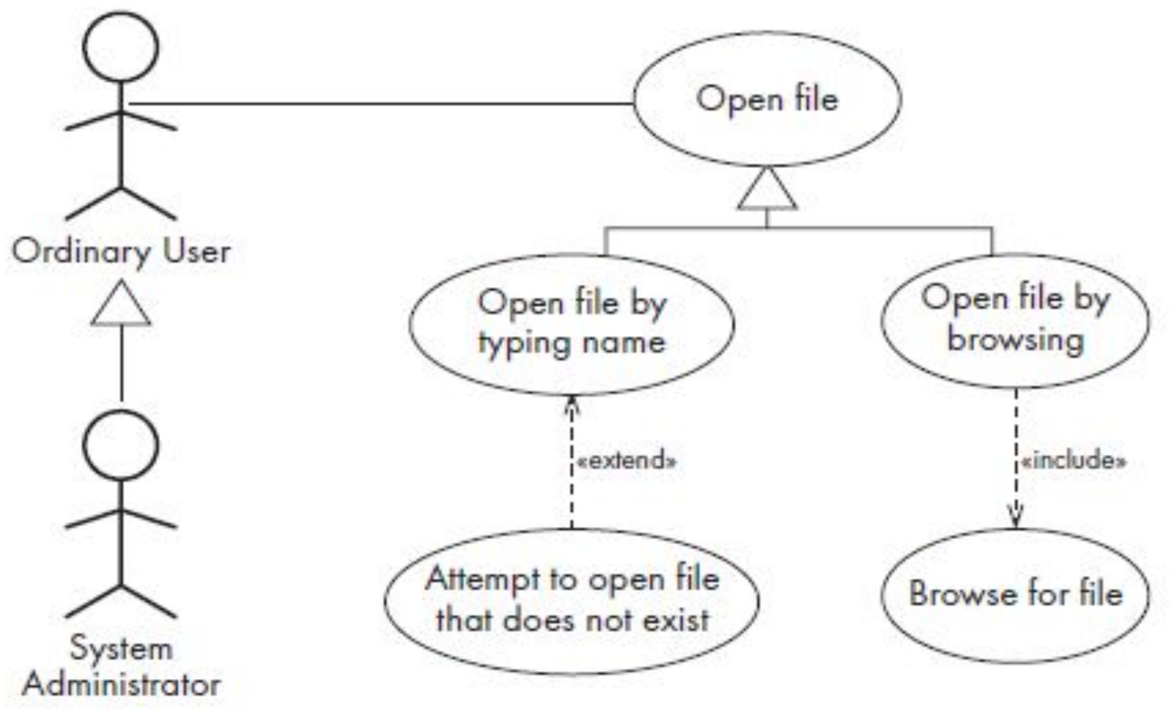
The «include» relationship allows us to include the steps from one Use Case into another.



«extend»

- The «extend» relationship allows us to modify the behavior of the base Use Case.
- Suppose we want to sell products that are made to order and require a degree of customer specification. For these products we will need to record the customer’s additional requirements, such as size and colour. In this case we are adding something extra to the flow of the base Use Case.


Sell Customer-Specific Product
«extend»
Take Customer Order
Sales Assistant
«include»
Identify Customer
Return Faulty Goods
«include»



Extension, generalization and inclusion in a use case diagram

Use case: Open file

Related use cases:

Generalization of:

- ❑ open file by typing name
- ❑ open file by browsing

Steps:

Actor actions	*System responses*
1. Choose 'Open…' command.	2. Display 'File open' dialog.
3. Specify filename.	
4. Confirm selection.	5. Remove dialog from display.

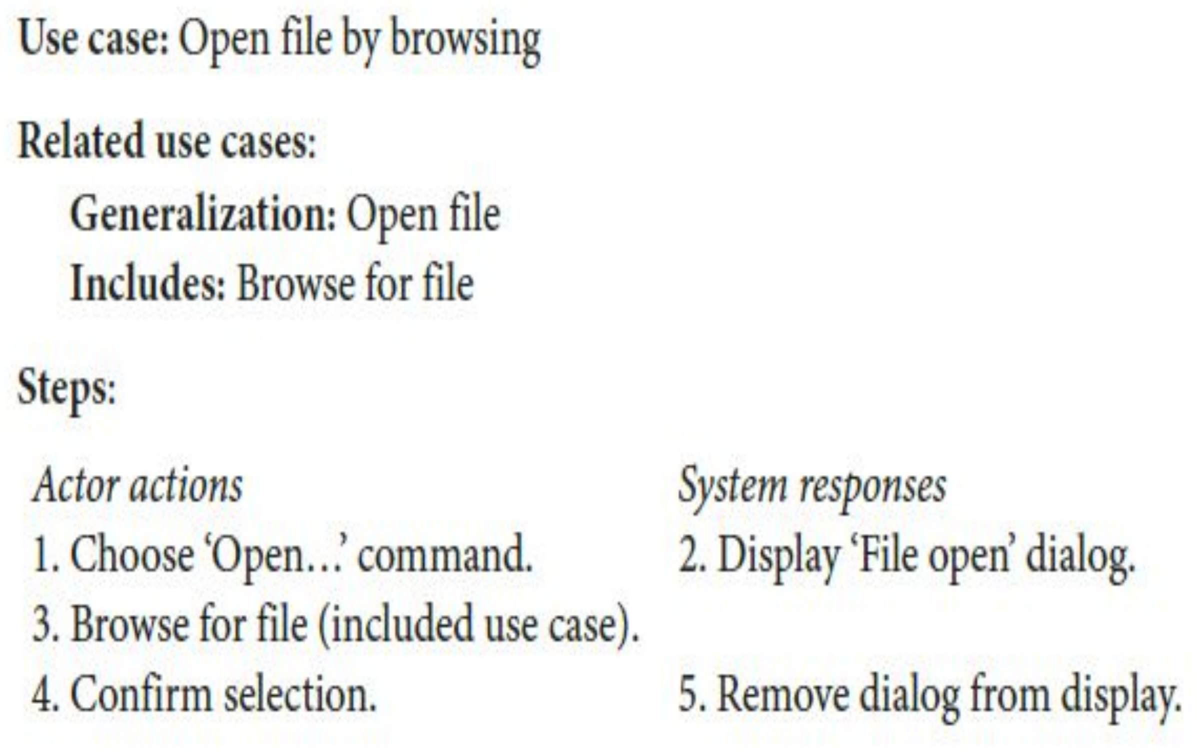
Use case: Open file by typing name

Related use cases:

Generalization: Open file

Steps:

Actor actions	*System responses*
1. Choose 'Open…' command.	2. Display 'File open' dialog.
3a. Select text field.	
3b. Type file name.	
4. Click 'Open'.	5. Remove dialog from display.



Use case: Open file by browsing

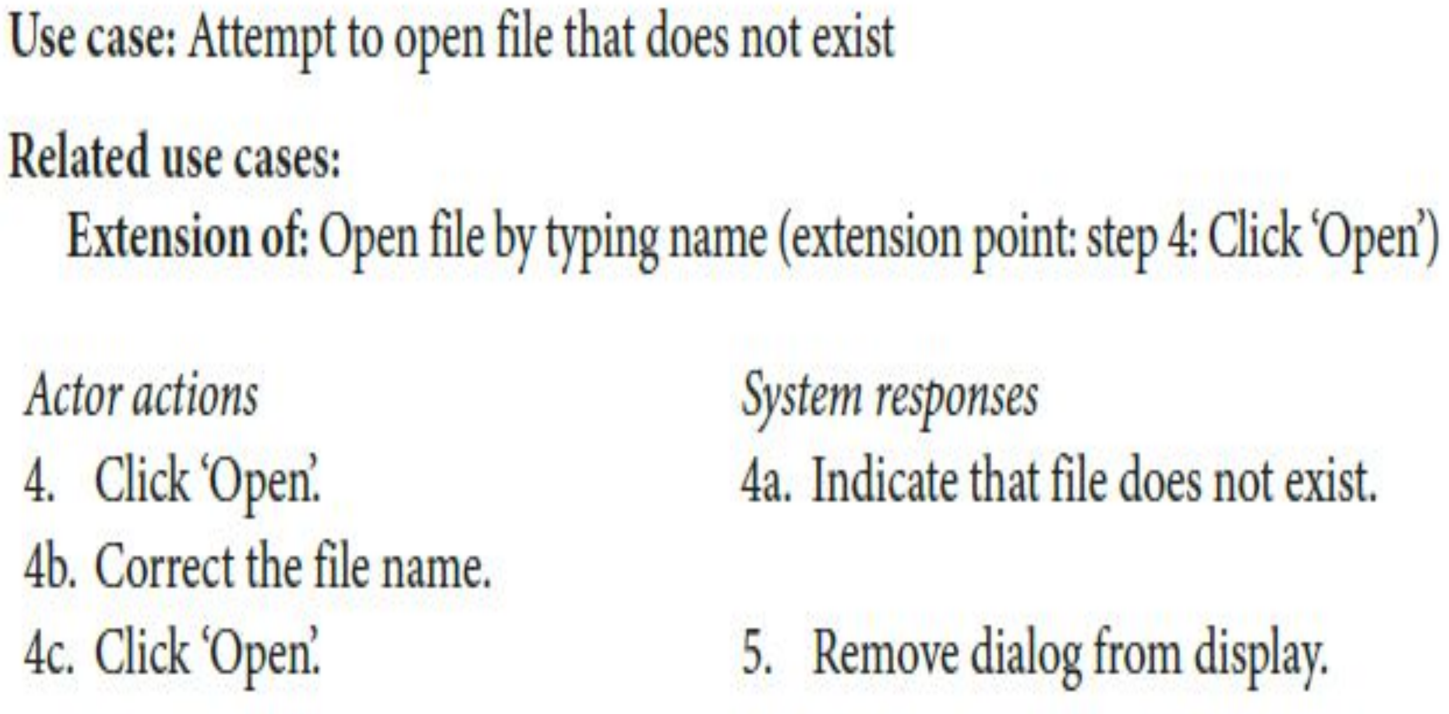
Related use cases:

  Generalization: Open file

  Includes: Browse for file

Steps:

Actor actions	*System responses*
1. Choose 'Open...' command.	2. Display 'File open' dialog.
3. Browse for file (included use case).	
4. Confirm selection.	5. Remove dialog from display.



Use case: Attempt to open file that does not exist

Related use cases:

Extension of: Open file by typing name (extension point: step 4: Click 'Open')

Actor actions	*System responses*
4. Click 'Open'.	4a. Indicate that file does not exist.
4b. Correct the file name.	
4c. Click 'Open'.	5. Remove dialog from display.

Use case: Browse for file (inclusion)

Steps:

Actor actions	*System responses*
1. If the desired file is not displayed, select a directory.	2. Display directory.
3. Repeat step 1 until the desired file is displayed.	
4. Select a file.	