ICS 3105
OBJECT ORIENTED SOFTWARE ENGINEERING

Chapter 5.1

Use Cases and Use Case Diagrams

Learning Outcomes

- By the end of this chapter, the learner should be able to:
  - Describe how the user will use the system
  - Draw use case diagrams from narratives.

Use Cases

- Use case analysis is a systematic approach to working out what users should be able to do with the software you are developing.
- It is one of the key activities in requirements analysis.

Use Case Analysis

- The first step in use case analysis is to determine the types of users or other systems that will use the facilities of this system.
- These are called actors.
- An actor is a role that a user or some other system plays when interacting with your system; each actor will need to interact with the system in different ways.

Actors

- Most of the actors will be users; a given user may be considered as several different actors if they play different roles from time to time – that is, if they have different job functions.
- Other actors will be systems that automatically exchange information with your system.

Actors

- If you performed domain analysis, you will have already listed the different types of users.
- E.g. a system for managing the processes of a small town public library, the actors for this system might include Librarian, Checkout Clerk, Borrower.

Use Case

- The second step in use case analysis is to determine the tasks that each actor will need to do with the system.
- Each task is called a use case because it represents one particular way the system will be used.

Use Case

- A use case is a typical sequence of actions that an actor performs in order to complete a given task.
- When listing use cases, make sure you respect the system scope i.e. only list use cases that actors will need to do when they are using the system to solve the customer’s problem.

Actors and Use cases

- Borrower:
  - Search for items by title.
  - by author.
  - by subject.
  - Place a book on hold if it is on loan to somebody else.
  - Check the borrower’s personal information and list of books currently borrowed.

Actors and Use cases

- Checkout Clerk:
  - All the Borrower use cases, plus
  - Check out an item for a borrower.
  - Check in an item that has been returned.
  - Renew an item.
  - Record that a fine has been paid.
  - Add a new borrower.
  - Update a borrower's personal information (address, telephone number etc.).

Actors and Use cases

- Librarian:
  - All of the Borrower and Checkout Clerk use cases, plus
  - Add a new item to the collection.
  - Delete an item from the collection.
  - Change the information the system has recorded about an item.

Actors and Use cases

- Accounting System (acting autonomously):
  - Obtain the amount of overdue fines paid by borrowers.

Building a use case model

- To build a complete use case model, you now need to describe the use cases in more detail.
- A use case model consists of a set of use cases, and optional descriptions or diagrams indicating how they are related.

How to describe a single use case

- The following are suggestions on how you can describe a complete use case:
  - Name, Actors , Goals, Pre conditions.
  - Summary, Post conditions, Related use cases.
  - Steps

Name

- Give a short, descriptive name to the use case.
- This should be a verb phrase describing the action the user will do with the system.
- It is also useful to include a number as a unique identifier for each use case.

Actors

- List the actor or actors who can perform this use case.
- For example, in a library system both a borrower and a librarian can check out a book.

Goals

- Explain what the actor or actors are trying to achieve.
- For example, in a library system, the goal of checking out a book would be to borrow the book in order to read it.

Preconditions

- Describe the state of the system before the use case occurs by listing any conditions that must be true before an actor can initiate this use case.
- For example, to be able to check out a book, the book must be available and the client must not have any overdue fines.

Summary

- Summarize what occurs as the actor or actors perform the use case.

Related use cases

- List use cases that may be generalizations, specializations, extensions or inclusions of this one.
- Use case analysis is similar to task analysis.
- Task analysis has been traditionally used to understand and improve the efficiency of the tasks that users perform, whether or not they are using a computer.
- User interface designers sometimes continue to use the term task analysis.

Steps

- Describe each step of the use case using a two-column format, with the left column showing the actions taken by the actor, and the right column showing the system's responses.

Post-conditions

- What state is the system in following the completion of this use case.

Use Cases

- Only the name and steps are essential – you may choose to provide a simplified use case description that omits the other components.
- In general, a use case should cover the full sequence of steps from the beginning of a task until the end.

Use Cases

- A use case should describe the user's interaction with the system, not the computations the system performs.
- For example in a use case for withdrawing money from an automated teller machine, you would describe the fact that the user inserts his or her card, responds to various prompts by pressing some buttons, and then removes his or her card and money.

Use Cases

- You would not describe how communication with the bank is established or how the system computes any fees it charges.
- The latter information is clearly important, but it belongs in a different part of the functional requirements.

Use Cases

- A use case should also be written so as to be as independent as possible from any particular user interface design.
- For example, instead of writing, ‘Push the “Open…” button’ as the first steps, we write ‘Choose the “Open…” command’.

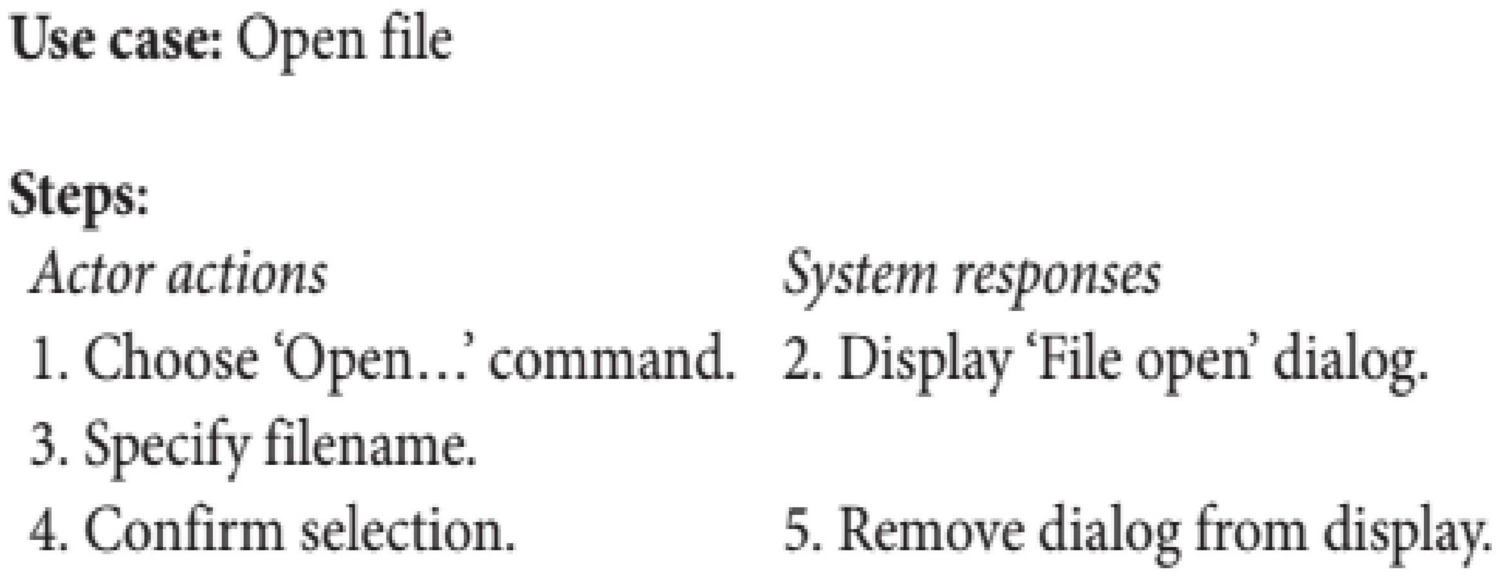
Use Cases

- The command could then be implemented as a button, a menu item, a keystroke or a voice command.
- Similarly, we have not specified whether the user types the file name or uses the mouse to select it from a list nor have we indicated what the 'File open' dialog looks like.

Example

- Description in a simplified format a use case for opening a file in an application.

Example



Use case: Open file

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 diagrams

- Use case diagrams are UML's notation for showing the relationships among a set of use cases and actors.
- They help a software engineer to convey a high-level picture of the functionality of a system.

Use Case Diagrams

- It is not necessary to create a use case diagram for every system or subsystem.
- For a small system, or a system with just one or two actors, a simple list of use cases will be adequate.

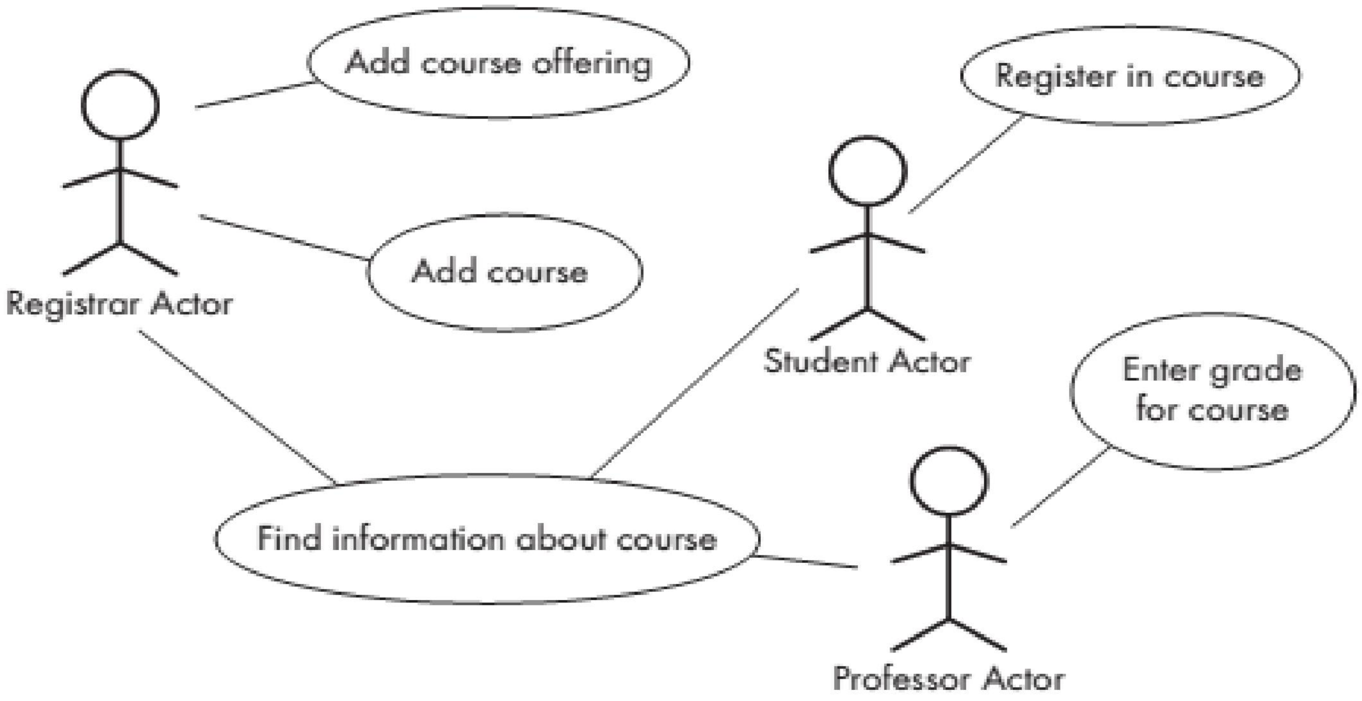
Use Case Diagrams

- There are two main symbols in use case diagrams:
  - An actor is shown as a stick person and
  - A use case is shown as an ellipse.

Use Case Diagrams

- Lines indicate which actors perform which use cases.
- You do not need to write the word ‘Actor’ in each actor’s name; however, it is useful to do this when it helps prevent confusion with classes of the same names.

Use Case Diagram: Example



Extension, generalization and inclusion of use cases

- You may want to develop a group of distinct but related use cases.
- For example, when an actor interacts with a system to achieve a particular goal, he or she may select different options, perform some action repetitively, provide different inputs, or answer too slowly causing a time-out error.

Extension, generalization and inclusion of use cases

- Each variant or repetitive pattern of interaction can be represented as a separate use case.
- Similarly, the system might have a special reaction when a file cannot be found.
- The use case modeler can use extensions, generalizations or inclusions to represent different types of relationships among use cases.

Extensions

- Extensions are used to make optional interactions explicit or to handle exceptional cases.
- A use case extension can, for example, describe what happens if an actor provides a wrong filename to access a given file or describes the extra interaction that occurs if the actor decides to browse in order to locate the required file instead of simply typing a filename.

Extensions

- By creating separate extension use cases, the description of the basic use case remains simple.
- In the extension, you should indicate which step is the extension point– the point at which the extension changes the basic sequence.

Generalizations

- Generalizations work the same way as in a class diagram and use the same triangle symbol: several similar use cases can be shown along with a common generalized use case.
- The general ‘open file’ use case has two sub use cases: ‘open file by typing name’, and ‘open file by browsing’.

Inclusions

- Inclusions allow you to express a part of a use case so that you can capture commonality between several different use cases.
- Even very different use cases can share a sequence of actions.

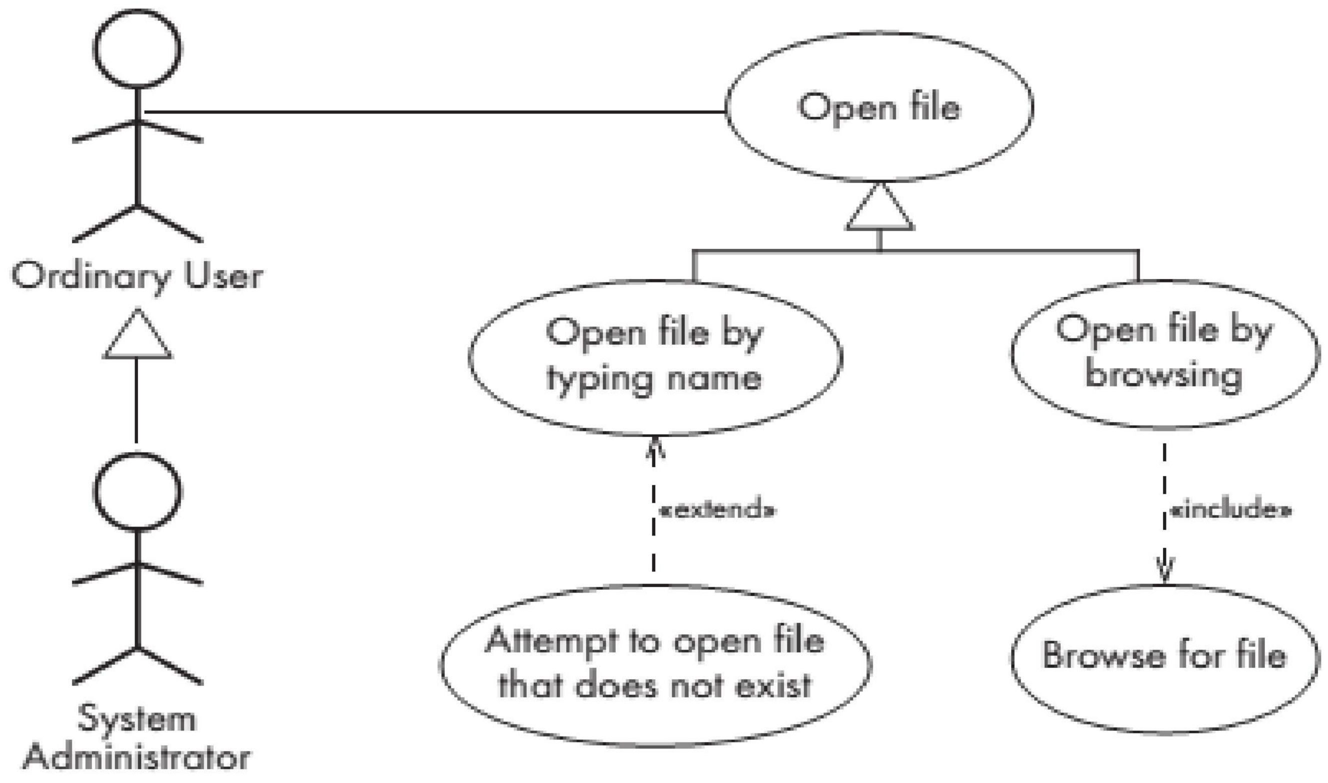
Inclusions

- For example, many different use cases might require an actor to specify a password, to browse through a list of items, or to open a file.

Inclusions

- Rather than repeating the details of such common interactions in multiple use cases, you can create a special use case that will be included in other use cases.
- Such a use case represents the performing of a lower-level task with a lower-level goal.

Extension, generalization and inclusion of use cases



Extension, generalization and inclusion of use cases

- The open triangle points to a generalization.
- The «extend» and «include» stereotypes show the other relationships between use cases.

Extension, generalization and inclusion of use cases

- Note that actors can also be arranged in a generalization hierarchy.
- ‘System Administrator’ is a sub-actor of ‘Ordinary User’.
- This means that all System Administrators can also act as ordinary users, and do such things as open files.

The modeling processes: choosing use cases on which to focus

- You should identify all the use cases associated with the software product, but you have to put varying emphasis on them when actually developing the system.
- Often one use case (or a very small number) can be identified as central to the system.

The modeling processes: choosing use cases on which to focus

- For example, in an airline reservation system, the central use case will be ‘Reserve a seat on a flight’.
- Once identified, the system can be built around this particular use case.

Other reasons for focusing on particular use cases

1. You may identify some use cases as high risk because you expect to have problems implementing them.
   – These use cases will require exploratory prototypes.

Other reasons for focusing on particular use cases

2. Some use cases will have high political or commercial value.
   - For example, in an online stock exchange system, a use case in which the user can see, in real time, the evolution of the value of a given stock would not be essential – the users could do their jobs without it.

Other reasons for focusing on particular use cases

- Nevertheless, it might be given high priority because of its appeal when presenting the product prototype to potential clients.

The benefits of basing software development on use cases

- Use case analysis is an intuitive way to understand and organize what the system should do, since it is based on user tasks and expresses the tasks in natural language.
- It can also be used to drive the development process.

The benefits of basing software development on use cases

1) Use cases can help to define the scope of the system – that is, what the system must do and does not have to do.

2) Use cases are often used to plan the development process. The number of identified use cases is a good indicator of a project's size. Development progress can be measured in terms of the percentage of use cases that have been completed.

The benefits of basing software development on use cases

3) Are used to both develop and validate the requirements. If some piece of proposed functionality does not support any use case, then it can be eliminated. Users and customers can also understand requirements better if they are expressed in terms of use cases. The use cases therefore can serve as part of the contract between the customers and the developer.

The benefits of basing software development on use cases

4) Use cases can form the basis for the definition of test cases.

5) Use cases can be used to structure user manuals.

Three things to watch out for when using use cases

1. The use cases themselves must be validated, using the requirements validation methods. For example, it is important to make sure that the set of use cases is complete and that they are expressed consistently and unambiguously.

Three things to watch out for when using use cases

2. There are some aspects of functional requirements that are not covered by use case analysis. For example, only activities triggered by an actor will appear as use cases. An example of something not shown as a use case is the automatic cleaning of a database to remove outdated information. Although it is not a use case, this is still a functional requirement since it is important to the user that the outdated information be periodically purged.

Three things to watch out for when using use cases

3. You should be aware that when software requirements are derived from use cases, the software tends simply to mirror the way users worked before the software was developed. In other words, innovative solutions may not be considered.

Scenarios

- A scenario is an instance of a use case that expresses a specific occurrence of the use case with a specific actor operating at a specific time and using specific data.
- It can help to clarify the associated use case.
- It is also often simply called a use case instance.

Techniques for gathering requirements

- Observation.
- Interviewing.
- Brainstorming.
- Prototyping.

Managing changing requirements

- One of the most important things to realize about requirements is that they change.
- Just because you have written a requirements document, and have obtained approval of it by all the stakeholders, does not mean that you can confidently design and implement the system as specified.

Managing changing requirements

- By the time the system is delivered, the users' and customers' needs will likely have evolved so that the requirements as documented no longer completely solve the customers' problem.

Changes to anticipate

- Business process changes.
- Technology changes.
- Better understanding of the problem.

Business process changes

- Businesses regularly adjust the way they do things in order to better compete in the market or merely because they gain experience and decide that an alternative approach is better.
- Changes to business processes can also be prompted by such things as changes in laws, as well as growth or rearrangement of the company.

Technology changes

- A new release of the operating system, or some other system with which your system interacts, may force you to reassess the requirements.

Better understanding of the problem

- Even though everybody might be confident about the requirements when they are first approved, various stakeholders may discover problems when looking at them again several months later.

End