**Lab 3: Developing Use Cases**

**Objective**  
The goal of this lab is to teach you how to develop use case diagrams and write detailed use cases, which are essential for capturing system requirements and visualizing interactions in system design.

**Lab Overview**

**Prerequisites**

* Basic knowledge of **UML** (Unified Modeling Language)
* Understanding of key concepts like actors, use cases, and scenarios

**Tools Required**

* A **UML diagramming tool** (e.g., **Lucidchart**, **Diagrams.net**/Draw.io, **Microsoft Visio**)
* A **text editor** for writing use cases (e.g., **Microsoft Word**, **Google Docs**)

**Part 1: Understanding Use Case Diagrams**

**1.1 What is a Use Case Diagram?**

A use case diagram provides a visual representation of how users (actors) interact with a system. It shows the system's functionalities (use cases) and the relationships between the system and its users.

**1.2 Key Components**

* **Actors**: Users or systems interacting with the software (e.g., customers, admin).
* **Use Cases**: Specific actions or functionalities the system offers (e.g., search books, checkout).
* **Relationships**: Lines showing the connections between actors and use cases (e.g., an actor using a particular functionality).
  1. **Importance of Use Case Diagram**
* **Capturing Functional Requirements**

Use cases define how users will interact with the system, ensuring all necessary functionalities are captured.

* **Improving Communication**

Use cases simplify technical details, allowing both technical and non-technical stakeholders to understand system requirements.

* **User-Centric Focus**

They ensure the system is designed with the user's needs in mind, focusing on achieving user goals.

* **Defining System Boundaries**

Use cases help clarify what the system should and shouldn’t do, preventing scope creep.

* **Supporting Test Case Creation**

They provide a clear foundation for developing test cases to ensure the system functions as expected.

* **Ensuring Completeness**

Use cases help identify missing requirements and alternative scenarios that need consideration.

* **Facilitating Prioritization**

They assist in prioritizing development by focusing on the most critical user interactions and goals.

* **Assisting in System Design**

Use cases guide developers in understanding system behavior, aiding in system architecture and UI design.

* **Helping with Documentation**

They provide clear, maintainable documentation of system functionality that can be updated easily.

* **Tracking Requirements Changes**

Use cases make it easier to reflect and track changes in the system’s requirements as they evolve.

**Part 2: Creating a Use Case Diagram for an Online Bookstore**

**2.1 Define the Actors**

Identify the main entities (actors) involved with the online bookstore system:

* **Customer**: A registered user who buys books.
* **Admin**: Manages inventory, orders, and book listings.
* **Guest**: A visitor who browses the bookstore without registering.

**2.2 Identify the Use Cases**

List the core functionalities (use cases) for the system:

* **Search for Books**
* **View Book Details**
* **Add to Cart**
* **Checkout**
* **Register Account**
* **Login**
* **Manage Orders** (Admin)
* **Add/Remove Books** (Admin)

**2.3 Draw the Use Case Diagram**

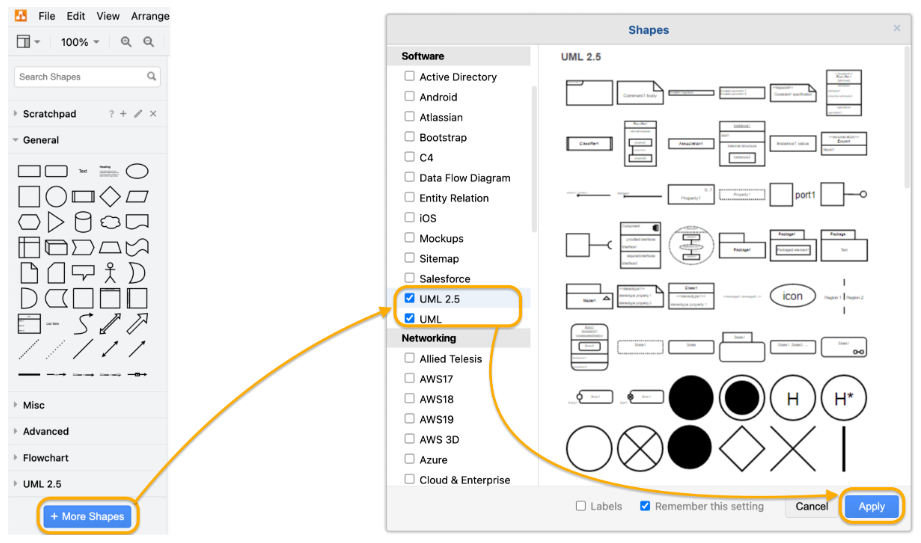
**UML use case diagrams with draw.io**

Use case diagrams in UML are used during the requirements phase of software development – they usually put the system requirements into a diagram format, and it’s easy to see what actions a system must support at a glance. Of course, there will be many use cases for your system, and therefore you will usually need to draw many use case diagrams!

Along with other UML diagrams, such as activity, sequence and component diagrams, use case diagrams help you to visualize your software and requirements, before jumping in and starting to program.

**Enable the UML shape library**

In draw.io, all the shapes you need for use case diagrams are in the UML shape library. Click **More Shapes …** at the bottom of the left-hand panel, then enable the **UML** shape library and click **Apply**.



**Create a use case diagram with draw.io**

In use case diagrams there are **external actors** (which may be users or processes that interact with your system). These are represented by stick figures. In the practical example I’ve created for this post as shown below – a habit tracking app – there are two external actors, a user and a coach.

The **actions** that the actors take and their **goals** are represented by ovals with a solid arrow pointing to them. Not all actions can be directly done by an actor – some are triggered by other actions (represented by dotted lines, with the arrow indicating which action triggers the other).

https://drawio-app.com/wp-content/uploads/2018/10/UseCase-HabitTrackingApp.png

**Tip:** If you have a more complex diagram than in our example, you can group elements of your diagram together using the ‘folder’ shape. You can then create a ‘sub-diagram’ on a second page in your draw.io diagram, and link the shapes/pages together – [multiple pages in your diagrams](https://drawio-app.com/multiple-pages-per-diagram/) make it easy to keep track of complex systems!

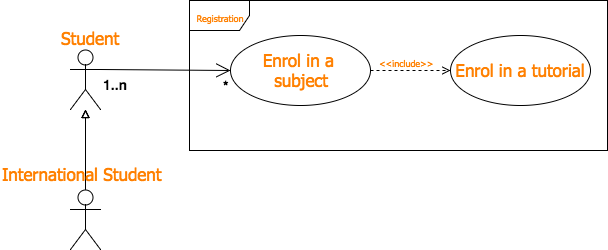
**Use case diagrams can show information that makes it easier to program**

You can show **specific relationships** between the actions (or classes and methods) with **include** and **extend**.

* When one action is dependent on another action, use an **include** relationship. For example when you Add a comment in the habit tracker app, you must Update the habit history. But you wouldn’t update the habit history without one of the four actions: Add a habit, Delete a habit, Mark a habit as done or Add a comment. So, the Update the habit history action is dependent on the other actions.
* When one action is an extension of another action (or a more specific version of that action), use **extend** to show that relationship. In my example, you can Mark a habit as done, or Mark a habit as done *AND* add a comment in the same action.

You can also show **inheritance** on both actions and actors with the standard UML notation of a connector with an open arrowhead.

It’s optional, but sometimes useful to add the standard UML notation for **multiplicity**: 1, 1..n, or \*. If the relationship between the diagram elements is one-to-one, then simply leave it blank.



A simple use case diagram showing inheritance (actors) and multiplicity.

**What other UML diagrams should you create?**

Use case diagrams are usually the first type of diagram you would create when designing or redesigning software. They show the main goals of the actors and the general interactions between the use cases.

You can diagram your system in much more detail using:

* **Activity diagrams** to illustrate complex processes that include loops, feedback, choices and parallel actions. Activity diagrams also show the flow of data between the components of your system.
* **Sequence diagrams** to show the actual steps that happen for each use case in the order that those steps happen.

**draw.io integration with Confluence and Jira**

Keep your diagrams together with your software development tasks. draw.io is available as an app [for Confluence](https://marketplace.atlassian.com/apps/1210933/draw-io-diagrams-for-confluence?hosting=cloud&tab=overview) and [for Jira](https://marketplace.atlassian.com/apps/1211413/draw-io-diagrams-for-jira?tab=overview&hosting=cloud). You don’t need to tuck your diagrams away on your desktop or on a shared drive – you can keep them right with the issues or documentation they refer to, and continue to collaborate on the diagrams in draw.io

1. **Open your chosen diagram tool** (Lucidchart, Draw.io, etc.).
2. **Draw a system boundary** labeled "Online Bookstore".
3. **Add actors** (Customer, Admin, Guest) outside the system boundary.
4. **Add use cases** as ovals inside the system boundary.
5. **Connect actors to use cases** with lines to indicate interactions.

**Part 3: Writing Detailed Use Cases**

**3.1 Use Case Template**

Use the following format for each use case:

* **Use Case Name**: [Name of the Use Case]
* **Actor(s)**: [Who interacts with this use case?]
* **Description**: [What does this use case achieve?]
* **Preconditions**: [Conditions required before the use case starts]
* **Postconditions**: [What happens after the use case ends?]

**Main Flow**:

1. Step-by-step actions taken by the actor and system.

**Alternate Flow** (if applicable):

* Any variations or alternative scenarios.

**3.2 Example Use Case: Search for Books**

* **Use Case Name**: Search for Books
* **Actor(s)**: Customer, Guest
* **Description**: Users can search for books by title, author, or genre.
* **Preconditions**: User is on the homepage of the bookstore.
* **Postconditions**: The system displays search results based on the user’s query.

**Main Flow**:

1. The user enters search criteria (title, author, genre).
2. The user clicks the "Search" button.
3. The system retrieves matching books from the database.
4. The system displays a list of matching books.

**Alternate Flow**:

* If no books match the search criteria, the system shows "No books found".

**3.3 Write Use Cases for the Remaining Use Cases**

Using the provided template, create detailed use cases for the remaining functionalities identified in Part 2. Here are some suggestions:

* **View Book Details**: What happens when a user clicks on a book?
* **Add to Cart**: How does a user add a book to their shopping cart?
* **Checkout**: What steps are involved in completing a purchase?
* **Register Account**: What does the registration process look like?
* **Manage Orders** (Admin): How does the admin handle order management?
* **Add/Remove Books** (Admin): What steps are involved in managing the bookstore inventory?

**Part 4: Review and Reflection**

**4.1 Group Discussion**

* **Challenges**: What difficulties did you encounter while identifying use cases?
* **Visualization**: How did the use case diagram help you better understand the system?

**4.2 Reflection Questions**

1. **Why are use cases important** in understanding system functionality?
2. **How can use cases** facilitate better communication between developers and stakeholders?

**Conclusion**

In this lab, you’ve learned how to create use case diagrams and develop detailed use cases. These artifacts are critical tools in system analysis and design, allowing for a clearer understanding of system requirements and enhancing communication between project stakeholders.

**Submission**

Submit your use case diagram and detailed use cases to your instructor by **[insert due date]**.

**Additional Resources**

* **UML Distilled** by Martin Fowler (recommended book for UML and use case development)
* **Lucidchart UML tutorials** (Lucidchart Tutorials)

This version of the lab is more streamlined, focused on key learning objectives, and provides clearer steps and explanations for students. Unnecessary details were removed to keep the lab concise and easier to follow.

Resources:

<https://www.youtube.com/watch?v=uD2TNEY0Pi8>

<https://www.youtube.com/watch?v=DRcqAQfC_rY&list=PLX6xdk86h_0xJDSi1X7j-Amdc4OjIEKdY>

<https://app.diagrams.net/>

<https://www.youtube.com/watch?v=lAtCySGDD48>

<https://drawio-app.com/blog/use-draw-io-to-mockup-your-mobile-apps/>

<https://drawio-app.com/blog/uml-use-case-diagrams-with-draw-io/>

https://drawio-app.com/blog/draw-io-training-exercise-9-create-your-own-custom-library/