

| **Title: Study of Umbrello Unified Modelling Language tool Or Lucidchart: flowcharts and diagram drawing tool** |
| --- |

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Aim:** To learn and understand the way of creating various UML diagrams for requirement analysis

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**CO:** Analyse the software requirements and Model the defined problem with the help of UML diagram.

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Books/ Journals/ Websites referred:**

1. Roger Pressman, “Software Engineering”, sixth edition, Tata McGraw Hill.
2. System Analysis & Design by Satzinger, Jackson and Burd, Cengage Learning, 2007
3. System Analysis and Design Methods by Jeffery l. Whitten, Lonnie D Bentley,McGraw Hill, 7th edition.
4. System Analysis and Design by Alan Dennis, Barbara H. Wixom, Roberta M. Roth,Wiley India 4th edition
5. <http://en.wikipedia.org/wiki/Software_requirements_specification>
6. <http://en.wikipedia.org/wiki/Use_case>

**\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Pre Lab/ Prior Concepts:**

In [software](http://en.wikipedia.org/wiki/Software_engineering) and [systems engineering](http://en.wikipedia.org/wiki/Systems_engineering), a **use case** is a list of steps, typically defining interactions between a role (known in [Unified Modeling Language](http://en.wikipedia.org/wiki/Unified_Modeling_Language) (UML) as an "[actor](http://en.wikipedia.org/wiki/Actor_(UML))") and a system, to achieve a goal. The actor can be a human or an external system.

In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or [stakeholder](http://en.wikipedia.org/wiki/Project_stakeholder) goals. The detailed requirements may then be captured in [Systems Modeling Language](http://en.wikipedia.org/wiki/Systems_Modeling_Language) (SysML) or as contractual statements.

As an important requirement technique, use cases have been widely used in modern software engineering over the last two decades. Use case driven development is a key characteristic of process models and frameworks like [Unified Process](http://en.wikipedia.org/wiki/Unified_Process) (UP), [Rational Unified Process](http://en.wikipedia.org/wiki/Rational_Unified_Process) (RUP), [Oracle Unified Method](http://en.wikipedia.org/wiki/Oracle_Unified_Method) (OUM), etc. With its iterative and evolutionary nature, use case is also a good fit for [agile development](http://en.wikipedia.org/wiki/Agile_software_development).

A sequence diagram is a graphical view of a scenario that shows object interaction in a time-based sequence of what happens first, what happens next.

Sequence diagrams establish the roles of objects and help provide essential information to determine class responsibilities and interfaces.

In UML, class diagrams are one of six types of structural diagram. Class diagrams are fundamental to the object modelling process and model the static structure of a system. Depending on the complexity of a system, you can use a single class diagram to model an entire system, or you can use several class diagrams to model the components of a system.

Class diagrams are the blueprints of your system or subsystem. You can use class diagrams to model the objects that make up the system, to display the relationships between the objects, and to describe what those objects do and the services that they provide.

In its basic form, an activity diagram is a simple and intuitive illustration of what happens in a workflow, what activities can be done in parallel, and whether there are alternative paths through the workflow. Activity diagrams as defined in the Unified Modeling Language are derived from various techniques to visually illustrate workflows. Activity diagrams are used to visualize the workflow of a business use case. A complete workflow description will have a basic flow, and one or several alternative flows. This workflow has a structure that we can define textually, using informal if, if-then-else, or does-until statements of various kinds. For a simple workflow with a simple structure such textual definitions may be quite sufficient, but in the case of more complex structures, activity diagrams help to clarify and make more apparent what the workflow is. Historically, activity diagramming techniques have mostly been used in the business process modeling domain, but this article will also briefly discuss how you can use it in the system modeling domain.

**Requirement Modeling:**

*Significance of every diagram is to be written*

1. **Use Case**

**Purpose:** Use Case Diagrams capture the functional requirements of a system by illustrating the interactions between actors (users or external systems) and the system itself. They define the system's behavior in response to external stimuli, helping to ensure that all stakeholder requirements are identified and addressed.

**Key Components:**

* **Actors:** Entities that interact with the system. Actors can be users or other systems.
* **Use Cases:** Functionalities or services provided by the system. Each use case represents a goal or task that an actor wants to accomplish.
* **System Boundary:** Defines the scope of the system, distinguishing what is inside the system from what is outside.
* **Relationships:** Include associations between actors and use cases, and relationships between use cases (e.g., include, extend).

**Applications:**

* **Requirements Gathering:** Helps in identifying and documenting what the system should do from a user’s perspective.
* **Scope Definition:** Delineates the boundaries of the system and its interactions with external entities.
* **Stakeholder Communication:** Provides a visual representation that facilitates discussions with stakeholders to ensure their needs are understood.

1. **Activity Diagram**

**Purpose:** Activity Diagrams represent workflows or processes within a system. They illustrate the sequence of activities, decisions, and parallel processing, helping to visualize the flow of control and identify potential improvements in a process.

**Key Components:**

* **Activities:** Actions or tasks performed in the workflow.
* **Transitions:** Arrows showing the flow from one activity to another.
* **Decision Nodes:** Points where the workflow branches based on conditions.
* **Parallel Activities:** Represented by fork and join nodes, showing simultaneous execution paths.
* **Start and End Nodes:** Indicate the beginning and termination of the workflow.

**Applications:**

* **Workflow Visualization:** Helps in understanding complex processes and identifying inefficiencies.
* **Process Improvement:** Aids in analyzing and optimizing workflows for better performance.
* **System Modeling:** Useful in detailing how different parts of a system interact and handle processes.

1. **Class Diagram**

**Purpose:** Class Diagrams model the static structure of a system. They define the system’s classes, including their attributes, methods, and relationships, providing a blueprint for the system's architecture.

**Key Components:**

* **Classes:** Define objects with attributes (properties) and methods (functions).
* **Attributes:** Characteristics or data associated with a class.
* **Methods:** Functions or operations provided by a class.
* **Relationships:** Include associations (links between classes), inheritance (class hierarchies), and dependencies.

**Applications:**

* **System Design:** Provides a detailed view of the system’s structure, aiding in object-oriented design.
* **Architecture Documentation:** Serves as a reference for system developers and stakeholders.
* **Code Generation:** Often used in conjunction with tools that generate code from class diagrams.

1. **Sequence**

**Purpose:** Sequence Diagrams depict the interactions between objects over time, showing the order in which messages are exchanged to accomplish a particular scenario or use case.

**Key Components:**

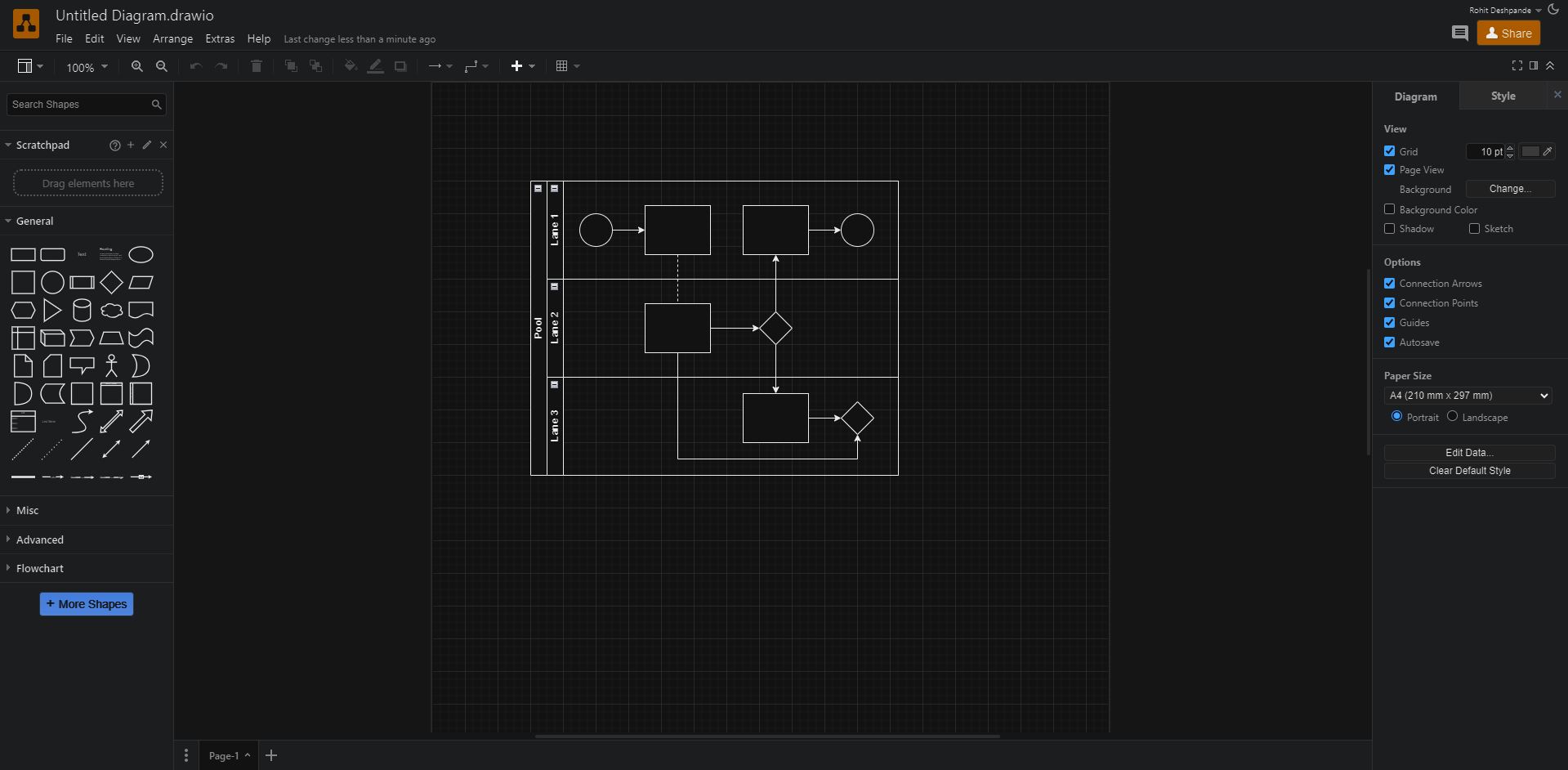
* **Objects:** Entities that interact with each other in the diagram.
* **Lifelines:** Represent the existence of objects over time.
* **Messages:** Arrows indicating communication between objects, including synchronous and asynchronous messages.
* **Activation Bars:** Represent the period during which an object is active in a process.

**Applications:**

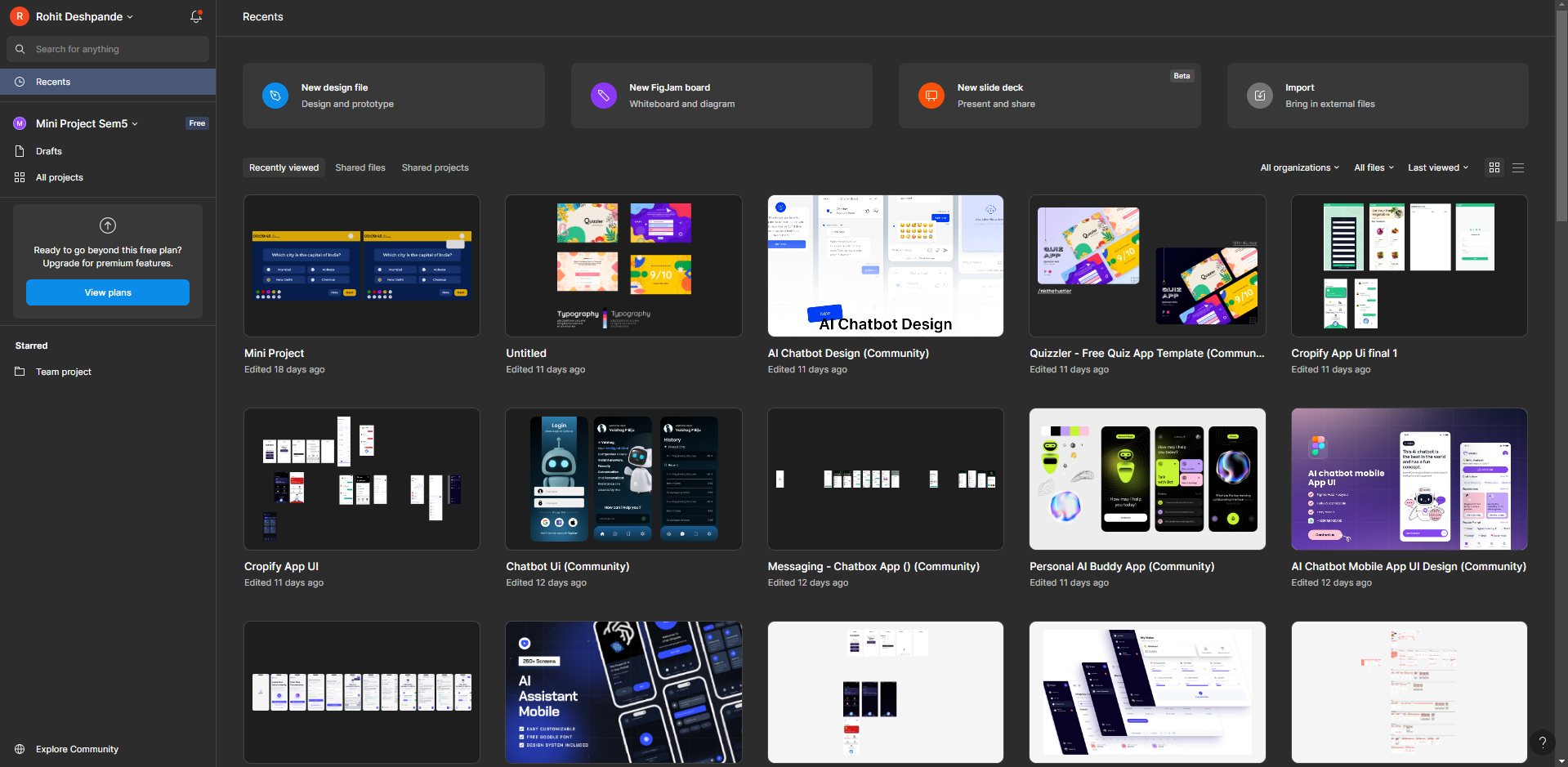
* **Interaction Analysis:** Helps in understanding how objects collaborate and communicate to achieve specific goals.
* **Behavior Documentation:** Provides a detailed view of the dynamic behavior of the system.
* **Scenario Modeling:** Useful for modeling and validating different scenarios within a system.

**Popular Drawing Tools:**

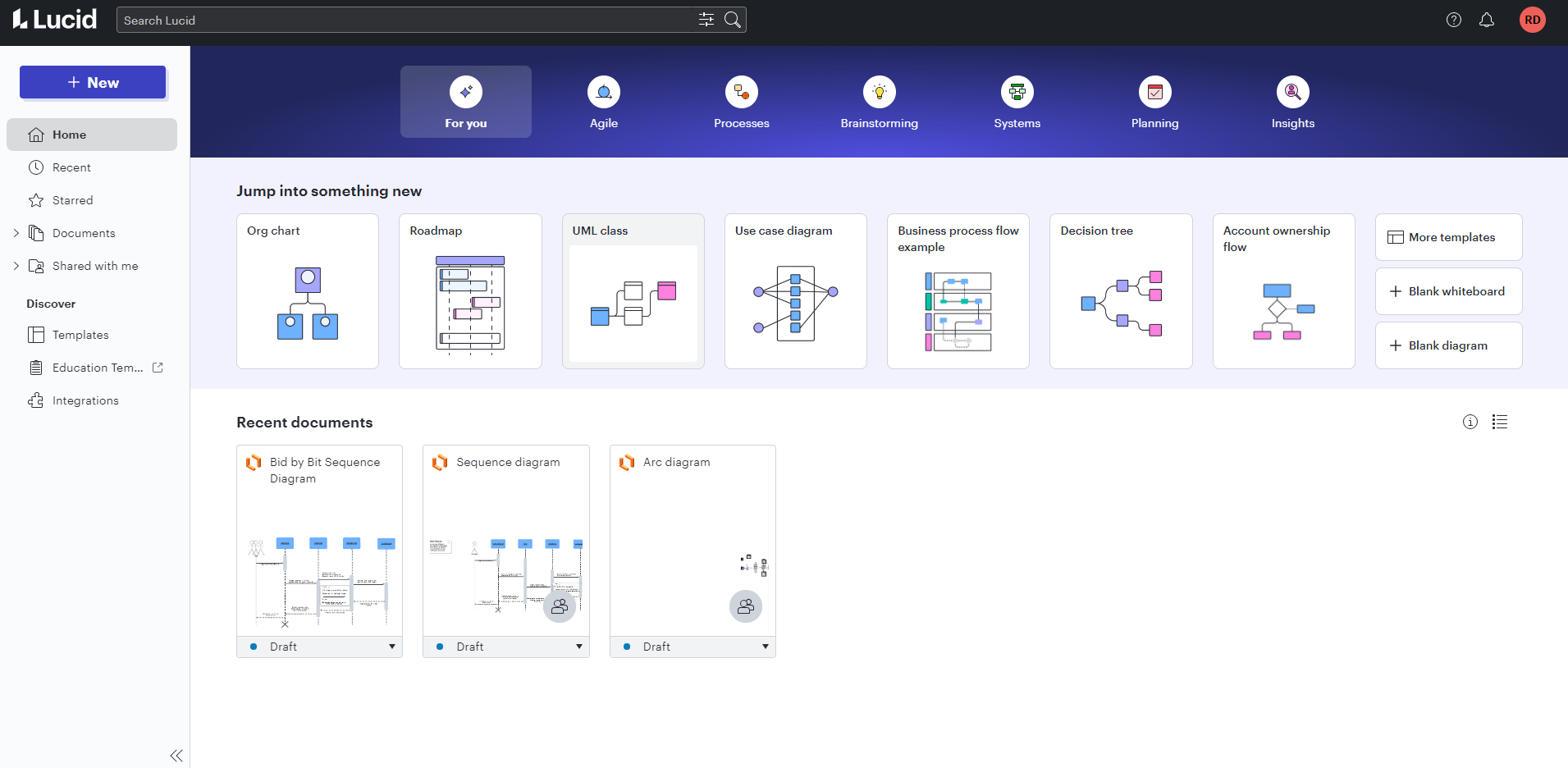
* **Draw.io:** An online diagramming tool that supports creating use case diagrams with a user-friendly interface. Provides templates and shapes for creating detailed activity diagrams. Supports the creation of class diagrams with various UML shapes and styles. Offers features for creating detailed sequence diagrams with various UML elements.
* **Figma:** A versatile design tool that can be used to create use case diagrams collaboratively in real-time. Allows for the creation and sharing of activity diagrams with a collaborative approach. Useful for collaborative class diagram design with real-time editing capabilities. Provides collaborative capabilities for designing sequence diagrams and integrating them with other design elements.
* **Lucidchart:** A web-based diagramming tool with extensive templates and shapes for use case modeling. Offers robust features for designing and annotating activity diagrams. Offers a wide range of UML templates and customization options for class diagrams. Includes extensive support for sequence diagram creation with interactive features.



**Draw.io**

****

**Figma**

****

**Lucid**

**Conclusion:** This study effectively demonstrates the capabilities of drawing tools in creating various UML diagrams, highlighting their features for different use cases in requirement analysis.

**Post Lab Descriptive Questions:**

1. Where do use cases fit in the software development life cycle?

Use cases fit into the **requirements analysis** phase of the Software Development Life Cycle (SDLC). They help define the system's functional requirements by describing interactions between users (actors) and the system to achieve specific goals.

**Role in SDLC**:

* 1. In the early stages, use cases guide the **gathering and specification** of system requirements.
  2. In **design**, they help identify system operations and map out interactions between components.
  3. During **testing**, use cases are useful for creating test cases, as they focus on ensuring that the system performs the specified functions.
  4. In the **maintenance** phase, use cases provide insights into the functionality when modifications are needed.

2. Compare sequence diagram with collaboration diagram. Explain pros and cons of each.

**Sequence Diagram**:

* **Definition**: Illustrates the order in which messages are exchanged between objects over time to accomplish a specific use case. It focuses on the time sequence of message exchanges between objects.
* **Pros**:
  + Easy to understand the flow of control over time.
  + Provides a clear visualization of the interactions in a specific time sequence.
  + Useful for understanding the execution order of processes and the life cycle of objects.
* **Cons**:
  + Can become complex and difficult to manage if there are many interactions.
  + Focuses primarily on time-ordering, making it less useful for understanding structural relationships between objects.
* **Collaboration Diagram** (also known as Communication Diagram):
  + **Definition**: Emphasizes the structural organization of objects that interact with each other to perform a task. It shows how objects are linked and the sequence of messages exchanged between them, without a strict time sequence.
  + **Pros**:
    - Easier to understand the relationships between objects and the way they collaborate.
    - More compact than sequence diagrams for simpler cases, especially in representing how objects are connected.
    - Useful when you need to visualize the static structure of interactions.
  + **Cons**:
    - Harder to interpret the flow of control and the time order of messages.
    - Becomes difficult to track the sequence of interactions in complex systems as it focuses on relationships rather than time.

3. List different notations used in Class diagram with example

**Class**:

* Represented by a rectangle divided into three parts: the top part contains the class name, the middle part contains attributes, and the bottom part contains methods or operations.

**Attributes**:

* Represent the data fields of the class. They are written in the second compartment with visibility markers (+ for public, - for private, # for protected).
  + **Example**: -name: String, -age: int

**Methods/Operations**:

* Represent the behavior of the class and are written in the third compartment.
  + **Example**: +enrollCourse(), +getDetails()

**Associations**:

* Represent relationships between classes, shown as a line between two classes, often with labels and multiplicities (e.g., 1, 0..\*).
  + **Example**: Teacher --- Student (indicating that a teacher teaches students).

**Inheritance (Generalization)**:

* Shown as a solid line with a hollow arrow pointing from the child class to the parent class.
  + **Example**: Teacher <|-- Person (indicating that a Teacher is a type of Person).

**Aggregation**:

* Represents a whole-part relationship, shown as a line with a hollow diamond near the whole class.
  + **Example**: Classroom <>--- Chair (indicating that a classroom consists of chairs).

**Composition**:

* A stronger form of aggregation where the part cannot exist independently of the whole, represented by a filled diamond.
  + **Example**: House ◼--- Room (indicating that rooms are part of a house and cannot exist without the house).

**Multiplicity**:

* Specifies how many instances of one class are associated with instances of another class, written at the ends of association lines.
  + **Example**: 1..\* (meaning one to many).

Example of class diagram:

