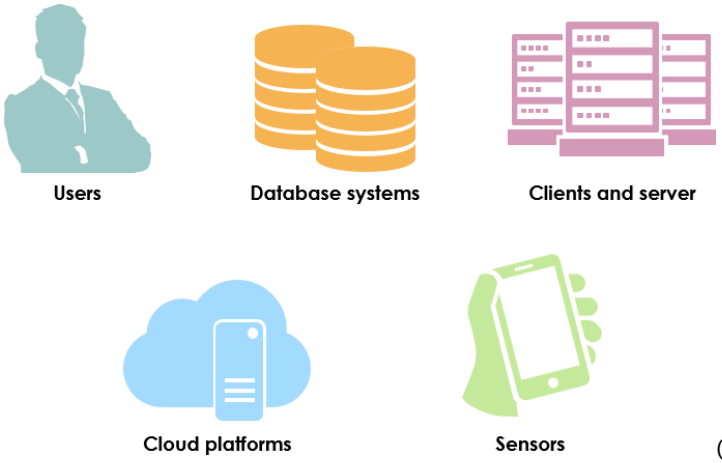
1. **Explore how actors interact with software regarding the functional requirements of a piece of software.**

According to Miles & Hamilton (2006), actors are external entities that interact with software systems. They could be physical objects, various software systems, or humans. A piece of software's functional requirements specifies what it must be able to do to meet the needs of its users and those involved. To attain these operational goals, actors engage with the programme. Actors are an essential element of a use case.

For example, a software system's operating environment consists of the users, devices, and programmes with which the system interacts (Visual Paradigm, N.D.). These are known as actors, and they have the following characteristics:

- In use case modelling, an actor is a user or any other system interacting with the subject.
- An Actor represents a role played by an entity that interacts with the subject but is not part of it (for example, through exchanging signals and data).
- Actors can perform the roles of human users, external hardware, or other subjects.
- Actors are not necessarily actual entities but specified aspects (i.e., "roles") of some commodities essential to the specification of their related use cases.
- A single physical instance can portray several different actors, while a single actor can play multiple roles.

Users, database systems, clients and servers, the cloud, platforms, and gadgets are examples of actors.



(Visual Paradigm, N.D.).

A use case is a strategy for identifying, clarifying, and organising system needs in system analysis (Brush, 2022). Furthermore, a use case is a collection of potential interactions between systems and users in a specific environment tied to a particular purpose. The method generates a document describing the user's steps to finish an activity.

Malan & Bredemeyer (2001) explain that each use case specifies a series of interactions between external actors and the system under examination. Actors are outside parties who engage with the system. An actor can be a user class, a role that users can perform, or other systems.

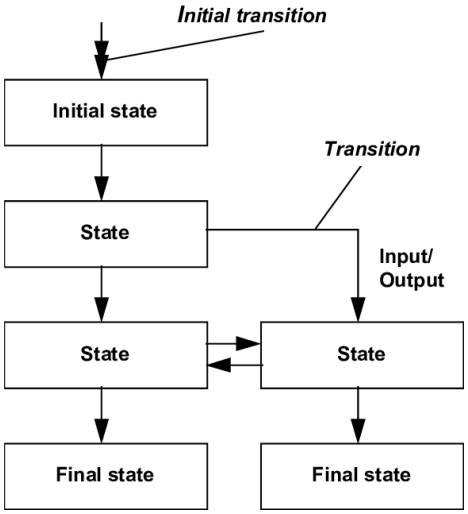
Identifying the actors is often the easiest way to begin the requirements elicitation process (Visual Paradigm, 2019). For instance, the following questions can help us identify system actors:

- Who uses the system?
- Who is in charge of installing the system?
- Who activates the system? Who looks after the system?
- Who is in charge of shutting down the system?
- Which other systems make use of this system?
- Who receives data from this system?
- Who supplies data to the system?
- Is anything happening automatically right now?

2. Using a state machine diagram, Identify the interactions and behaviours invoked within the software once active.

A state machine diagram in UML is a graph that explains and models an object's lifecycle (state life cycle) in response to various events (Doungsa-ard et al. 2007). Moreover, state machine diagrams in UML are an object-oriented variation of traditional state charts. It is a graph that describes and models the lifetime of an object

(state life cycle) in response to various occurrences. State chart diagrams illustrate the state machines by emphasising potential states and their transitions.



(Salt, 1999).

State diagram elements are the graphical components used to illustrate a finite state machine's states, transitions, inputs, and outputs (vpadmin, 2023). For instance, here is a quick rundown of each of these components:

1. **States** indicate the various contexts or situations a system can be in at any time. In a state diagram, they are represented by circles or ovals. Each state should have a name or description that clarifies its depiction.

2. **Transitions** are the shifts from one state to another that occur in response to an input. In a state diagram, they are represented by arrows or lines. Each transition should be tagged with the input or event that causes it.

3. **Inputs** are events or situations that cause a transition from one state to another. Labels on arrows or lines in a state diagram can represent them.

4. **Outputs** are the acts or outcomes that occur when a transition is made. They are not always included in a state diagram, but labels on the arrows and lines can represent them or the states themselves.

5. **The initial state** is where the system begins before receiving inputs. An arrow pointing to the initial state circle or oval represents it.
6. **The final state** is the state the system returns to when it has completed its work. A double circle or oval is used to represent it.

I may represent a system's dynamic activity in a state diagram using these properties in a clear, straightforward, and understandable way. Software developers and other professionals who must model and analyse systems with finite states and state transitions might benefit significantly from state diagrams.

As explained by (Miles & Hamilton, 2006), state machine diagrams are widely used in specialised software and hardware systems such as the following:

- Mission-critical/real-time systems, such as heart-monitoring software.
- Dedicated devices, such as ATMs, whose behaviour is defined in terms of state.
- First-person shooter games like Doom and Half-Life.

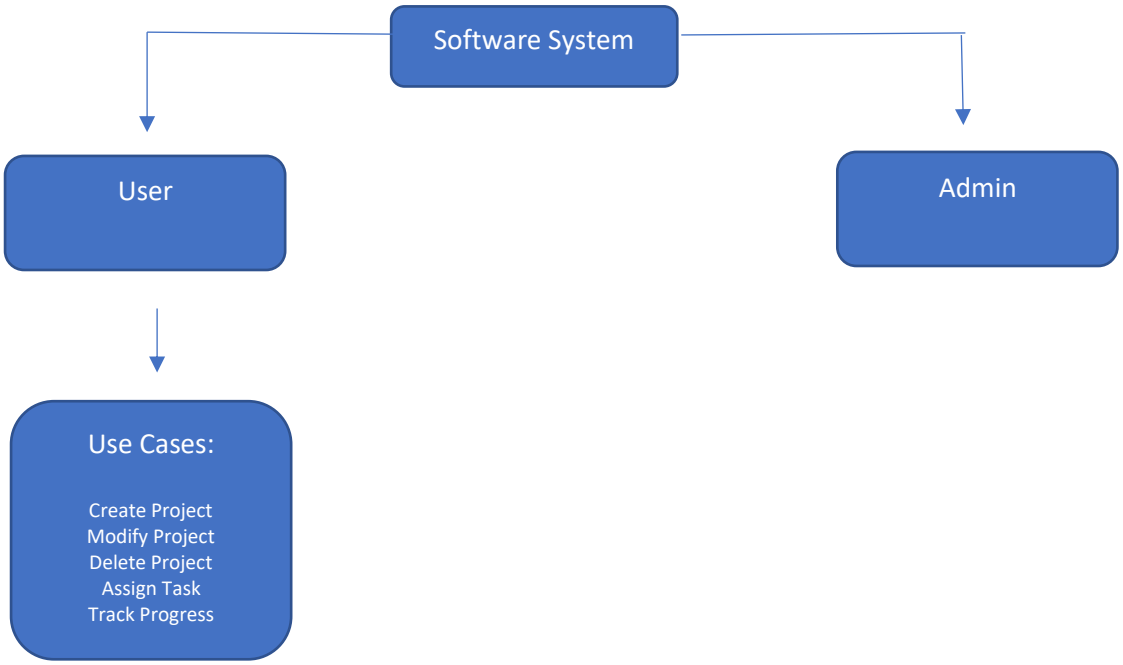
3. Design a use case diagram for a software development scenario.

The primary representation of system/software requirements for an unfinished new software programme is a UML use case diagram. Use cases outline the desired behaviour (what), not the precise process to carry it out (how). Once defined, use cases can be represented textually and visually (using a use case diagram) (Visual Paradigm, 2019).

The desired behaviour of the system (or subsystem, class, or interface) I am building is encapsulated in a use case without any specifics of how the intended activity will be carried out. However, the use case diagram example below demonstrates how a common use case diagram is defined in the UML:

There are various processes involved in creating a use case diagram for a software development scenario.

Here is a use case diagram for a software scenario:



In this case, the two actors are the "User" and "Admin." The "User" interacts with the system to create, edit, remove, assign project tasks, and monitor progress. Although the "Admin" may have particular use cases or extra capabilities, they still interact with the system.

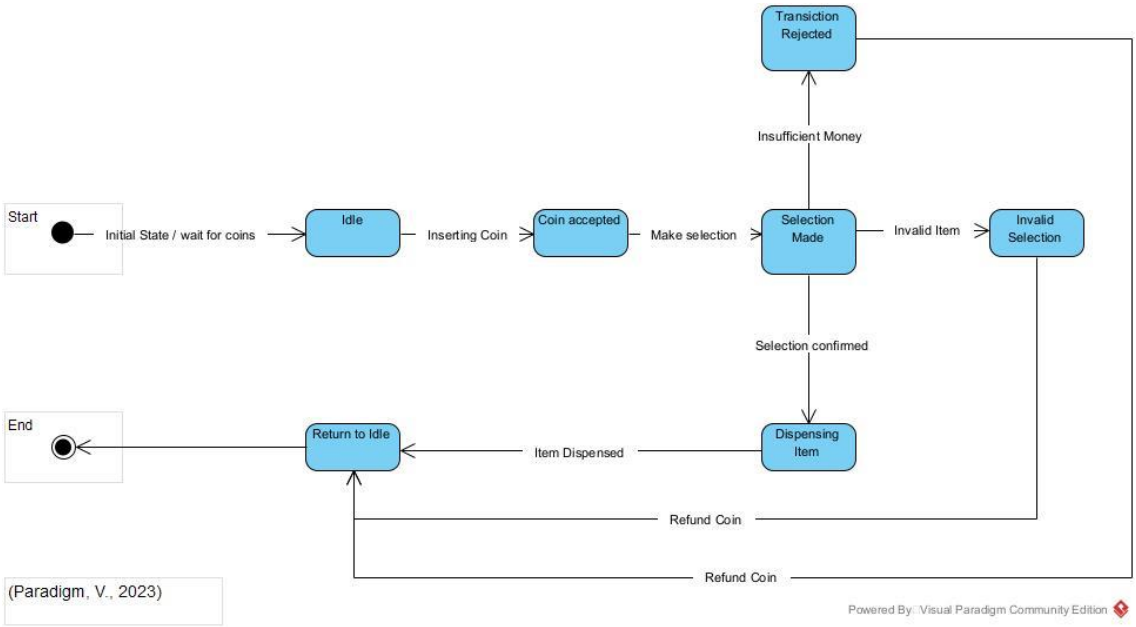
The "Software System" is a representation of the software that is being created. The use cases, which reflect the many functionalities offered by the software, are shown as ovals within the system border. The interactions between actors and use cases are indicated by arrows.

This is a simple example, and I may have to add extra players, use cases, or relationships to illustrate based on the software I plan to develop. Use case diagrams can be modified to meet the unique needs and objectives of your software project.

4. **Create a state machine diagram to examine the behaviours and interactions within a system.**

The series of events that an object experiences throughout its existence in reaction to events are specified in a state machine diagram, which mimics the behaviour of a single object (sparxsystems, N.D.).

The state machine in the diagram below represents behaviours and interactions inside a simple vending machine system during its operations and shows the process:



The following activities and occurrences accompany each state:

The **start** is the initial state of the vending machine.

Idle is the vending machine waiting for user input.

Coin Accepted is the machine recognises the amount inserted and waits for the user to select an item.

Selection Made is the user selecting an item by pressing a button.

Dispensing Item is the machine dispensing the selected item.

Transaction Rejected is the machine rejecting the user's option due to insufficient money.

Invalid Selection is the machine rejecting the user's choice because the item is not available in the machine.

Return to Idle is the machine returning to the idle state.

The **end** is the final state, where the transaction is complete.

The events and actions associated with each state are as follows:

- From Start to Idle – no event is required to transition from Start to Idle. The vending machine is in an idle state by default.
- Insert Coin – the user's action to insert a coin into the machine.
- Coin Inserted – the machine detects that a coin has been inserted successfully.
- Make Selection – the user's action to make a selection by pressing a button for a specific item.
- Selection Made – the machine receives the user's selection.
- Confirm Selection – the user's action to confirm the selection made.
- Selection Confirmed – the machine acknowledges the user's selection.
- Dispense Item – the machine dispenses the selected item.
- Item Dispensed – the machine successfully dispenses the item.
- Refund Coin – the user's action to request a coin refund.
- Invalid Selection – the user has made an invalid selection.
- Transaction rejected – the user has selected an item with a different value.
- Return to Idle: The machine returns to the Idle state.

This state diagram depicts the primary states and transitions involved in the vending machine's behaviour, as well as a detailed state diagram with additional scenarios, such as invalid selection or transaction rejection, emphasising the importance of taking an incremental approach to problem-solve, beginning with a simple version of the problem and gradually refining it as requirements and constraints become clearer (Paradigm V., 2023).

References:

Miles, R. & Hamilton, K. (2006). *Learning UML 2.0: a pragmatic introduction to UML*. " O'Reilly Media, Inc.".

Visual Paradigm (N.D.). *Use Case Analysis: How to Identify Actors?* [online] Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/how-to-identify-actors/.

Brush, K. (2022). *What is a Use Case?* [online] SearchSoftwareQuality. Available at: https://www.techtarget.com/searchsoftwarequality/definition/use-case.

Malan, R. & Bredemeyer, D. (2001). Functional requirements and use cases. *Bredemeyer Consulting*.

Visual Paradigm (2019). *What is Use Case Diagram?* [online] Visual-paradigm.com. Available at: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-use-case-diagram/.

Doungsa-ard, C., Dahal, K., Hossain, A. & Suwannasart, T. (2007). Test data generation from UML state machine diagrams using gas. In *International Conference on Software Engineering Advances (ICSEA 2007)* (pp. 47-47). IEEE.

Salt, J.D. (1999). *The specification of interactive behaviour patterns in object-oriented discrete-event simulation modelling* (Doctoral dissertation, Brunel University).

vpadmin (2023). *Introduction to State Diagrams: A Comprehensive Guide for Software Engineering*. [online] Visual Paradigm Guides. Available at: https://guides.visual-paradigm.com/introduction-to-state-diagrams-a-comprehensive-guide-for-software-engineering/ [Accessed 19 May 2023].

smartdraw (2019). *Use Case Diagrams - Use Case Diagrams Online, Examples, and Tools*. [online] Smartdraw.com. Available at: https://www.smartdraw.com/use-case-diagram/.

sparxsystems (N.D.). *State Machine Diagram - UML 2 Tutorial | Sparx Systems*. [online] Available at: https://sparxsystems.com/resources/tutorials/uml2/state-diagram.html.

Paradigm, V. (2023). *Visualizing System Behavior: A Practical Guide to State Diagrams with Examples*. [online] Visual Paradigm Guides. Available at: https://guides.visual-paradigm.com/visualizing-system-behavior-a-practical-guide-to-state-diagrams-with-examples/.