|  |  |
| --- | --- |
| Software engineering  YouTube UML | Tanjir Ahmed Bhuban  10616591  CA1 |

System software is a set of generalized programs that manage the resources of the computer, such as the processing unit, communication links, and peripheral devices. System software has three components:

1. The operating system

2. Language translator

3. Utility program.

Analysing requirements is crucial to understanding what the software needs to achieve, and UML (Unified Modelling Language) is a powerful tool for designing and modelling software systems.

For analysing requirements, start by gathering information from stakeholders. Understand their needs, expectations, and any constraints. Use techniques like interviews, surveys, or workshops to collect detailed requirements. Create a Requirements Document to document and prioritize these requirements.

Next, for designing and modelling using UML, follow these steps:

1. **Use Case Diagrams**: Identify and define the interactions between the system and external entities. This helps in understanding the functional requirements.

2. **Activity Diagrams**: Visualize the workflow and activities within the system. This helps in understanding the flow of control within the software.

3. **Class Diagrams**: Identify and define the key classes, their attributes, and relationships. This provides a blueprint for the data structure in the system.

4. **Sequence Diagrams**: Illustrate the interactions between different components or objects over time. This is especially useful for understanding the dynamic behaviour of the system.

5. **State Machine Diagrams**: If your system has complex states, represent them using state machine diagrams. This is helpful for systems with intricate state transitions.

6. **Component Diagrams**: Identify the components of the system and how they interact. This helps in understanding the modular structure of the software.

7. **Deployment Diagrams**: Visualize how the software will be deployed across hardware components. This is crucial for systems that involve distributed computing.

At this point I’d like to discuss about some of the UML.

Start with creating a Use Case Diagram for software system. A Use Case Diagram illustrates how a system interacts with external entities, known as actors, to accomplish specific tasks. Here's a step-by-step guide:

**Use Case Diagram**

1. Identify Actors: List all external entities that interact with the system. These could be users, other systems, or even hardware devices.

2. Identify Use Cases: Define the high-level functionalities or tasks that the system needs to perform. Each task represents a use case.

3. Draw the Diagram: Place the actors on the left side and use ovals to represent use cases in the middle or right side. Connect actors to use cases with lines to show the interactions.

4. Include Relationships: Use associations (lines) to show relationships between actors and use cases. You can also use generalization (extends or includes) to represent relationships between use cases.

5. Add Multiplicity: If necessary, specify how many instances of an actor are involved in a use case.

6. Document the Use Cases: Provide brief descriptions or notes for each use case to capture essential details.

Example Use Case Diagram:

Software System

Actor 1

(Use Case 1)

Actor 2

(Use Case 2)

System

**Class Diagram**

1. Identify Classes: Identify the main entities or objects in your system that encapsulate data and behaviour. These will become classes.

2. Determine Attributes: For each class, define the attributes (properties or characteristics) that represent the data associated with that class.

3. Define Methods: Specify the methods (functions or operations) that each class can perform. These represent the behaviour of the class.

4. Identify Relationships: Determine the relationships between classes. Common relationships include association, aggregation, and composition.

5. Draw the Diagram: Place classes in the diagram and connect them with lines to represent relationships. Use multiplicity notation to indicate the number of instances of one class associated with an instance of another class.

6. Add Multiplicity: If a class has a specific relationship with another class, indicate the multiplicity (e.g., 1, 0..1, etc.).

7. Document Attributes and Methods: Add a brief description or notes for each class, including its attributes and methods.

Software System

Class 1

Attribute1

Attribute2

Method1

Method2

Class 2

Attribute3

Attribute4

Method3

Method4

Association

Relationship

**Sequence Diagram**

Sequence Diagrams illustrate the interactions between different components or objects over time, showing the order in which messages are exchanged. Here's a step-by-step guide:

1. Identify Actors and Objects: Identify the actors (external entities) and objects (classes/components) involved in the sequence.

2. Determine Lifelines: Create a vertical dashed line for each actor or object. These lines represent the lifelines of the entities.

3. Sequence of Messages: Identify the sequence of messages exchanged between the actors and objects. Messages can be method calls, data exchanges, or signals.

4. Draw the Diagram: Place the lifelines (actor and object lines) vertically on the diagram. Use arrows to represent the flow of messages between lifelines.

5.Include Activation Bars: Draw activation bars to represent the duration of time during which an object is active and processing a message.

6. Specify Message Details: Label each message with the method name or description and include any parameters or return values.

Example Sequence Diagram:

Actor 1

Object A

Method 1

Return Value

Actor 2

Object B

Method 2

Return Value

**State Machine Diagram**

State Machine Diagrams are useful for modelling the dynamic behaviour of a system by illustrating the states an object or system can be in and the transitions between those states. Here's a step-by-step guide:

1. Identify States: Identify the different states that the object or system can be in. These states represent different modes or conditions.

2. Determine Transitions: Define the transitions between states. These transitions are triggered by events and conditions.

3. Draw the Diagram: Create a box for each state and connect them with arrows to represent transitions. Label the arrows with the events or conditions that trigger the transitions.

4. Specify Actions: Optionally, you can include actions or activities that occur when a transition takes place.

Example State Machine Diagram:

Object

State 1

Action A

Transection Event

State 2

Action B

Transection Event

Analysing the requirements for a YouTube-like system involves understanding both functional and non-functional aspects. Functionally, the system must support video uploading, playback, user authentication, content recommendation, comments, and user interaction tracking. Non-functional requirements include scalability to handle a large user base, high availability, data security for user-generated content, and responsive user interfaces. Stakeholders include content creators, viewers, administrators for content management, advertisers, and regulatory bodies. Constraints may revolve around copyright regulations, data privacy laws, and the need for a user-friendly interface.

Research and analysis have led to the adoption of an Agile development life cycle model for this dynamic and rapidly evolving system. Agile allows for iterative development, accommodating changing user needs and technological advancements.

The Initial System Architecture/Design Specification incorporates six key classes: User, Video, Comment, Playlist, Admin, and Recommendation Engine. The Class Diagram illustrates the relationships and attributes of these classes. The Use Case Diagram outlines interactions, covering actions like user registration, video upload, content recommendation, and administrative functions. The Sequence Diagram depicts the flow of operations during video playback, emphasizing real-time data retrieval and user interactions. An Activity Diagram showcases dynamic processes such as content recommendation algorithms and user engagement tracking. Finally, a State Diagram captures the different states a video can be in, from uploading to public availability.

This technical documentation forms a solid foundation for the development of a sophisticated and user-centric YouTube-like system, meeting the specified requirements and adapting to the evolving landscape of online video platforms.

A diagram of a company

Description automatically generated

\*\*Relationships:\*\*

- Users can upload, view, and comment on videos.

- Users can create playlists and add videos to them.

- Admins can monitor content and review reports.

- The Recommendation Engine provides personalized video recommendations to users.

This simple class diagram captures the essential components of a YouTube-like system, making it easy to understand the main classes, their attributes, and how they relate to each other. It's a concise representation of the core functionalities in a video-sharing platform.

Certainly! Let's create a simplified sequence diagram for a user playing a video on YouTube:

1. \*\*Actor: User\*\*

2. \*\*Object: YouTube System\*\*

\*\*Sequence:\*\*

1. \*\*A user sends a request to play a video:\*\*

- User -> YouTube System: Play Video Request

2. \*\*YouTube System processes the request:\*\*

- YouTube System -> Video Class: Retrieve Video Information

- YouTube System -> User Class: Deduct View Count

3. \*\*YouTube System sends video details to the User:\*\*

- YouTube System -> User: Video Details (Title, Views, Likes)

4. \*\*User interacts with the video:\*\*

- User -> Video: Clicks Play

5. \*\*YouTube System updates video statistics:\*\*

- YouTube System -> Video: Update View Count

6. \*\*User interacts with the video (likes or comments):\*\*

- User -> Video: Like/Comment

7. \*\*YouTube System updates video statistics and user interactions:\*\*

- YouTube System -> Video: Update Like/Comment Count

- YouTube System -> Comment Class: Save Comment

8. \*\*YouTube System sends updated video details to the User:\*\*

- YouTube System -> User: Updated Video Details (Views, Likes, Comments)

9. \*\*Sequence ends.\*\*

This sequence diagram captures the flow of interactions between a User and the YouTube System when playing a video. It emphasizes the key steps, such as retrieving video information, updating statistics, and handling user interactions. Keep in mind that this is a simplified representation, and actual YouTube system interactions would involve more complexity and details.