Unit 3: Unified Modelling Language (UML)

e-Portfolio Activities:

1. Discuss which UML models are most applicable at different stages of the Software Development Life Cycle

The Unified Modeling Language (UML) is a comprehensive modeling language used in software development to visually represent software designs, architectures and system structures. It provides a range of diagram types that can be used to depict both the behavior and structure of systems.

UML enables developers to sketch ideas and concepts, think through complex systems in a structured manner and thus build a bridge between requirements analysis, design and actual implementation. In addition, UML also serves for communication within the development team and with stakeholders who are not directly involved in the development process.

UML plays a central role in modern software development as it provides a common language for the conception and design of software projects. By using UML, developers and stakeholders can develop a deeper and clearer understanding of the system architecture, data structures and business processes behind a software application.

Some of the primary benefits of UML include:

- Simplify system analysis and design by depicting complicated system components in clear diagrams.
- Support in the requirements engineering phase by using UML diagrams to visually record and communicate requirements.
- Promote cross-team communication and mutual understanding as UML diagrams can serve as universally understandable, visual documentation.
- Facilitation of the implementation phase by automatically generating parts of the code from the UML models or at least deriving structured specifications for the implementation.

The typical software development process consists of several phases: requirements-gathering, high-level design, low-level design, coding and unit testing, integration testing, and deployment (Pargaonkar, 2023; Yuge & Badarch, 2023; Crawford & Kaplan, 2003). Different methodologies divide these areas into different categories and subdivisions.

UML provides several diagram types that fit into each section of the Software Development Lifecycle (SDLC) provided in table 1.

Table 1. Rule based fraud detection

Requirements Analysis Stage	**Design Stage**	**Implementation Stage**	**Testing Stage**	**Maintenance Stage:**
- Use Case Diagrams: These depict the interactions between users (actors) and the system, helping to identify system requirements and functional specifications. **- Activity Diagrams:** Show workflows and processes within the system, aiding in understanding the sequence of actions needed to achieve a goal.	**- Class Diagrams:** Represent the static structure of the system, showing classes, their attributes, methods, and relationships. They form the foundation for object-oriented design. **- Sequence Diagrams:** Illustrate how objects interact over time to accomplish a specific task or use case, helping to design the dynamic behavior of the system. **- State Machine Diagrams:** Useful for modeling the behavior of individual entities or system components over their lifecycle.	**- Component Diagrams:** Illustrate the high-level organization of components and their interrelationships within the system, aiding in implementation planning. **- Deployment Diagrams:** Show the physical deployment of software components across hardware nodes, guiding the system's deployment architecture.	**- Sequence Diagrams:** Still useful during testing to understand and validate the dynamic behavior of the system as specified. **- Collaboration Diagrams:** Similar to sequence diagrams but focus more on the structural organization of objects and their interactions.	**- Package Diagrams:** Useful for organizing and managing large systems, showing the dependencies between packages/modules/components. **- Component Diagrams:** Remain helpful during the maintenance phase to understand the system's architecture and make modifications.
General Purpose				
- Object Diagrams: Provide a snapshot of instances of classes in a system at a particular point in time, aiding in understanding relationships and instances. **- Composite Structure Diagrams:** Show the internal structure of a class or collaboration, useful for detailed design and implementation.				

2. Making reference to 'The Unified Modeling Language Reference Manual Second Edition', use the State Machine Diagram in Figure 3-7 to design a similar model for a washing machine.

Creating a state diagram for a washing machine involves defining the different states the machine can be in and the transitions between these states based on certain events or conditions. Here's a conceptual state diagram for a typical washing machine:

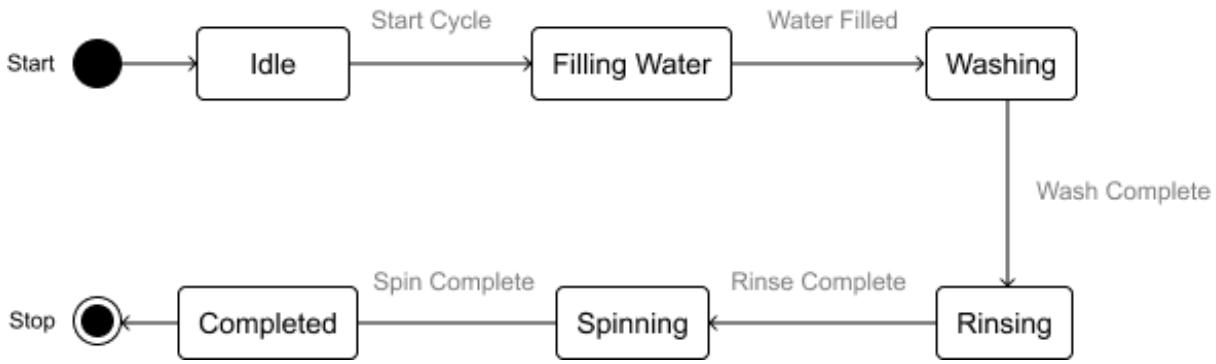
- **Off State:** The washing machine is turned off.
- **Idle State:** The washing machine is powered on but not running any cycle.
- **Filling State:** The washing machine is filling with water.
- **Washing State:** The washing machine is agitating to wash the clothes.
- **Rinsing State:** The washing machine is rinsing the clothes.
- **Spinning State:** The washing machine is spinning to remove excess water.
- **Completed State:** The washing cycle is complete.

Transitions

- **Off State** to **Idle State**: When the power button is pressed.
- **Idle State** to **Filling State**: When the start button is pressed and the door is closed.
- **Filling State** to **Washing State**: When the desired water level is reached.
- **Washing State** to **Rinsing State**: After the washing time is completed.
- **Rinsing State** to **Spinning State**: After the rinsing process is completed.
- **Spinning State** to **Completed State**: When the spinning cycle is completed.

Diagram

Here is a visual representation:


Start
Idle
Start Cycle
Filling Water
Water Filled
Washing
Wash Complete
Rinsing
Rinse Complete
Spinning
Spin Complete
Completed
Stop

References:

Crawford, W. & Kaplan, J.M. (2003) *J2EE Design Patterns*. Sebastopol, CA: O'Reilly.

Pargaonkar, S. (2023) 'A comprehensive research analysis of Software Development Life Cycle (SDLC) Agile & Waterfall Model Advantages, disadvantages, and application suitability in software quality engineering', International Journal of Scientific and Research Publications, 13(8), pp. 120–124. doi:10.29322/ijsrp.13.08.2023.p14015.

Rumbaugh, J., Jacobson, I. & Booch, G. (2004) The Unified Modeling Language Reference Manual. 2nd ed. Addison-Wesley.

Yuge, L. & Badarch, T. (2023) 'Research on contemporary software development life cycle models', *American Journal of Computer Science and Technology* [Preprint]. doi:10.11648/j.ajcst.20230601.11.