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

The Unified Modelling Language (UML) is a comprehensive modelling 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 behaviour and structure of systems.

It enables developers and business analysts to sketch ideas and concepts and think through complex systems in a structured way to build a bridge between requirements analysis, design and actual implementation.

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

Benefits of UML include:

1. Simplify system analysis and design by depicting complicated system components in clear diagrams.

2. Support in the requirements engineering phase by using UML diagrams to visually record and communicate requirements.

3. Promote cross-team communication and mutual understanding as UML diagrams can serve as universally understandable, visual documentation.

4. 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. Even so, UML provides several diagram types that can be more suited into different sections of the Software Development Lifecycle (SDLC) provided in the table below.

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.

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:

States

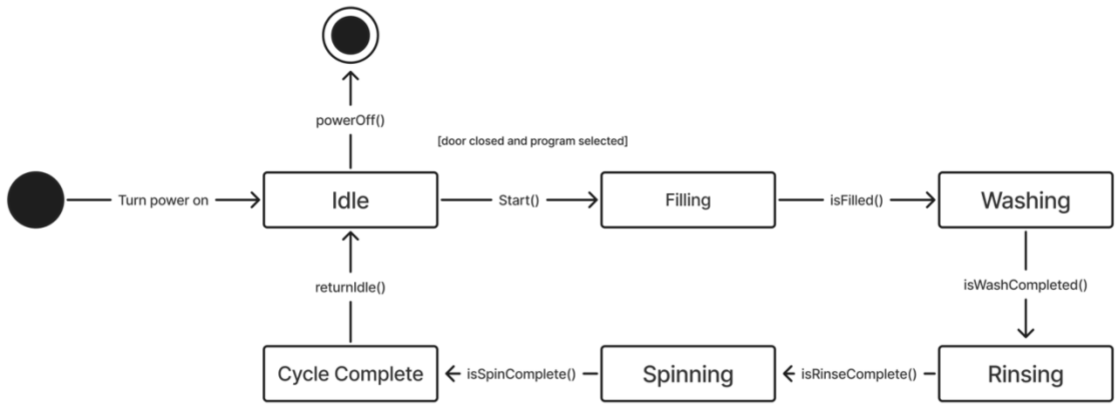
1. **Off State:** The washing machine is turned off.
2. **Idle State:** The washing machine is powered on but not running any cycle.
3. **Filling State:** The washing machine is filling with water.
4. **Washing State:** The washing machine is agitating to wash the clothes.
5. **Rinsing State:** The washing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

Transitions

- **Initial to Idle:** The process begins when the user presses the power button (Turn power on), moving the machine into the Idle state.
- **Idle to Filling:** A transition to the Filling state occurs when the Start() button is pressed. This transition is protected by a **guard condition** and will only happen if [door is closed and a program is selected].
- **Filling to Washing:** The machine automatically transitions to the Washing state once the water level sensor indicates it is full (isFilled()).
- **Washing to Rinsing:** After the wash timer is complete (isWashCompleted()), the machine transitions to the Rinsing state.

- **Rinsing to Spinning:** Once the rinsing process is finished (isRinseCompleted()), the machine transitions to the Spinning state.
- **Spinning to Cycle Complete:** When the final spin is done (isSpinComplete()), the machine enters the Cycle Complete state.
- **Cycle Complete to Idle (The Loop):** The machine returns to the Idle state (returnIdle()) after the cycle is finished, making it ready for the next wash. This models the machine's reusability.
- **Idle to Final State (Power Off):** From the Idle state, if the user presses the power button again (powerOff()), the machine transitions to a final, powered-off state, and the process terminates.

The drawn diagram was made through Figma and can be viewed below.



3 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 lifecycle models’, *American Journal of Computer Science and Technology* [Preprint]. doi:10.11648/j.ajcst.20230601.11.