The commonly used methodology which is appropriate in software engineering is agile methodology.

**Agile Methodologies**

* Agile methodologies, such as Scrum and Kanban, promote adaptive planning, iterative development, and continuous feedback.
* They are well-suited for projects where requirements are likely to change or evolve over time.
* Agile methodologies emphasize collaboration, flexibility, and delivering working software in short iterations.

Interaction models used in software engineering

1. **Use Case Diagrams:** Use case diagrams depict the interactions between actors (users or external systems) and the system under development. They illustrate the various use cases (actions or operations) that the system supports and show how actors interact with the system to achieve specific goals or tasks.
2. **Sequence Diagrams:** Sequence diagrams illustrate the chronological flow of messages and interactions between objects or components within the system. They focus on the temporal ordering of interactions and the exchange of messages between objects over time, helping to visualize the dynamic behavior of the system.
3. **Communication Diagrams (formerly Collaboration Diagrams):** Communication diagrams emphasize the structural organization of objects or components and the interactions between them. They show the relationships and dependencies between objects and the messages exchanged between them, providing a holistic view of the system's components and their collaboration.
4. **State Machine Diagrams:** State machine diagrams represent the behavior of an individual object or component within the system. They depict the different states that an object can be in and the transitions between these states based on specific events or conditions. State machine diagrams are useful for modeling complex behavior and defining the lifecycle of objects.
5. **Activity Diagrams:** Activity diagrams illustrate the flow of activities or processes within the system. They depict the sequence of actions, decisions, and control flows, showing how activities are performed and how control passes between different parts of the system. Activity diagrams are particularly useful for modeling business processes or complex workflows.

6. **Interaction Overview Diagrams:** Interaction overview diagrams provide an overview of the interactions and control flows between various components in a high-level and abstract manner. They combine elements of activity diagrams and sequence diagrams to show the flow of control between different interaction fragments.

Interaction models

* Interaction models aid in system understanding, requirements analysis, system design,and communication among stakeholders.
* They help to visualize the interactions, behavior, and flow of data within the system, enabling effective system development, testing, and documentation.
* By using these models, software engineers can identify potential issues, ensure proper system functionality, and facilitate collaboration among development teams.

context models

* Context models, also known as system context diagrams or environment models, are visual representations that depict the interactions between a system under development and its external entities or components.
* They provide a high-level overview of the system's boundaries, its relationships with external entities, and the flow of information between them.
* Context models help in understanding the system's position within its larger environment and provide a foundation for further analysis and design.

Components and concepts of context models

1. System: The central focus of the context model is the system itself. It represents the software or hardware being developed and includes the functionalities, processes, and components that make up the system.
2. External Entities: External entities are entities outside the system that interact with it. These entities can be people, other systems, hardware devices, databases, or any other external component that has a relationship with the system. External entities are depicted as boxes surrounding the system.
3. Interactions: Interactions represent the flow of information or data between the system and the external entities. These interactions can be in the form of inputs to the system, outputs from the system, or both. Arrows or lines connecting the system and external entities indicate the direction of information flow.
4. Boundaries: The boundaries of the system define its scope and demarcate what is included within the system and what lies outside of it. They help in clearly identifying the system's limits and what is considered part of the system's responsibility.

Simplification: Context models simplify the complexities of the system by focusing on the most significant external entities and interactions. They provide a high-level view and do not delve into the internal workings of the system or its internal components.

Purposes served by context models in software development

* Communication: They provide a clear visual representation that facilitates communication between stakeholders, developers, and other project participants, helping them understand the system's relationships and its environment.
* Requirement Analysis: Context models aid in identifying and defining the system's functional and non-functional requirements by highlighting external entities and their interactions with the system.
* Scope Definition: They help establish the boundaries of the system, enabling a clear understanding of what falls within the system's responsibility and what lies outside of it.

Design Decisions: Context models provide a foundation for making design decisions by identifying the key external entities and their interactions that need to be supported by the system.

Structural models

* + Structural models refer to visual representations or diagrams that depict the structure, organization, and relationships among the various components or elements of a software system.
  + These models focus on representing the static aspects of the system, including its components, their attributes, relationships, and how they are organized.
  + Structural models are essential for understanding the architecture, design, and organization of a software system.
  + They help to communicate system structure to stakeholders, aid in system analysis and design, and facilitate collaboration among development teams.

Some commonly used structural models in software development

1. Class Diagrams: Class diagrams illustrate the static structure of a system by representing classes, their attributes, methods, and relationships. They show how classes are organized into packages or modules, inheritance relationships, associations, aggregations, and dependencies between classes.
2. Object Diagrams: Object diagrams provide a snapshot of the system at a specific point in time. They depict instances of classes (objects) and their relationships, showing how objects collaborate and interact with each other.
3. Component Diagrams: Component diagrams represent the physical or logical components of a system and their dependencies. They illustrate the relationships and interactions between components, interfaces, ports, and connectors. Component diagrams are particularly useful for visualizing the modular structure and system composition.
4. Package Diagrams: Package diagrams depict the organization of classes or components into logical groupings or packages. They show the dependencies and relationships between packages, facilitating system modularization and managing large-scale software systems.
5. Deployment Diagrams: Deployment diagrams illustrate the physical deployment of software components and their relationships to hardware or execution environments. They show how components are distributed across nodes, processors, or devices, and how they communicate and interact in a distributed system.
6. Composite Structure Diagrams: Composite structure diagrams provide a detailed view of the internal structure of a class or component. They depict the internal parts, ports, connectors, and relationships within a class or component, showcasing its internal organization and collaboration.

These structural models help in system analysis, design, and documentation. They assist in visualizing system components, their relationships, dependencies, and organization, allowing for better understanding and communication among stakeholders. Structural models serve as blueprints for system development, enabling developers to build systems that adhere to the desired architecture and design principles.

**Behavioral models**

* Behavioral models are visual representations or diagrams that depict the dynamic behavior and functionality of a software system.
* These models focus on illustrating how the system behaves, how its components interact, and how data flows and changes over time.
* Behavioral models help in understanding system functionality, requirements analysis, system design, and communication among stakeholders.

Some commonly used behavioral models in software engineering

1. Use Case Diagrams: Use case diagrams represent the interactions between actors (users or external systems) and the system. They illustrate the various use cases (actions or operations) that the system supports and show how actors interact with the system to achieve specific goals or tasks.
2. Sequence Diagrams: Sequence diagrams illustrate the chronological flow of messages and interactions between objects or components within the system. They focus on the temporal ordering of interactions and the exchange of messages between objects over time, helping to visualize the dynamic behavior of the system.
3. Activity Diagrams: Activity diagrams depict the flow of activities or processes within the system. They show the sequence of actions, decisions, and control flows, showing how activities are performed and how control passes between different parts of the system. Activity diagrams are particularly useful for modeling business processes or complex workflows.
4. State Machine Diagrams: State machine diagrams represent the behavior of an individual object or component within the system. They depict the different states that an object can be in and the transitions between these states based on specific events or conditions. State machine diagrams are useful for modeling complex behavior and defining the lifecycle of objects.
5. Communication Diagrams (formerly Collaboration Diagrams): Communication diagrams emphasize the structural organization of objects or components and the interactions between them. They show the relationships and dependencies between objects and the messages exchanged between them, providing a holistic view of the system's components and their collaboration.

6. Timing Diagrams: Timing diagrams focus on the timing constraints and ordering of events in a system. They depict the time intervals, delays, and synchronization between different components or processes, helping to ensure that the system behavior adheres to the specified timing requirements.

* + These behavioral models help to describe the dynamic aspects of a system, including how it behaves in response to user interactions, how components interact and exchange data, and how the system transitions between different states or activities.
  + By using these models, software engineers can validate system requirements, design appropriate system behavior, and communicate system functionality effectively to stakeholders.
  + Behavioral models play a crucial role in system development, testing, and documentation.