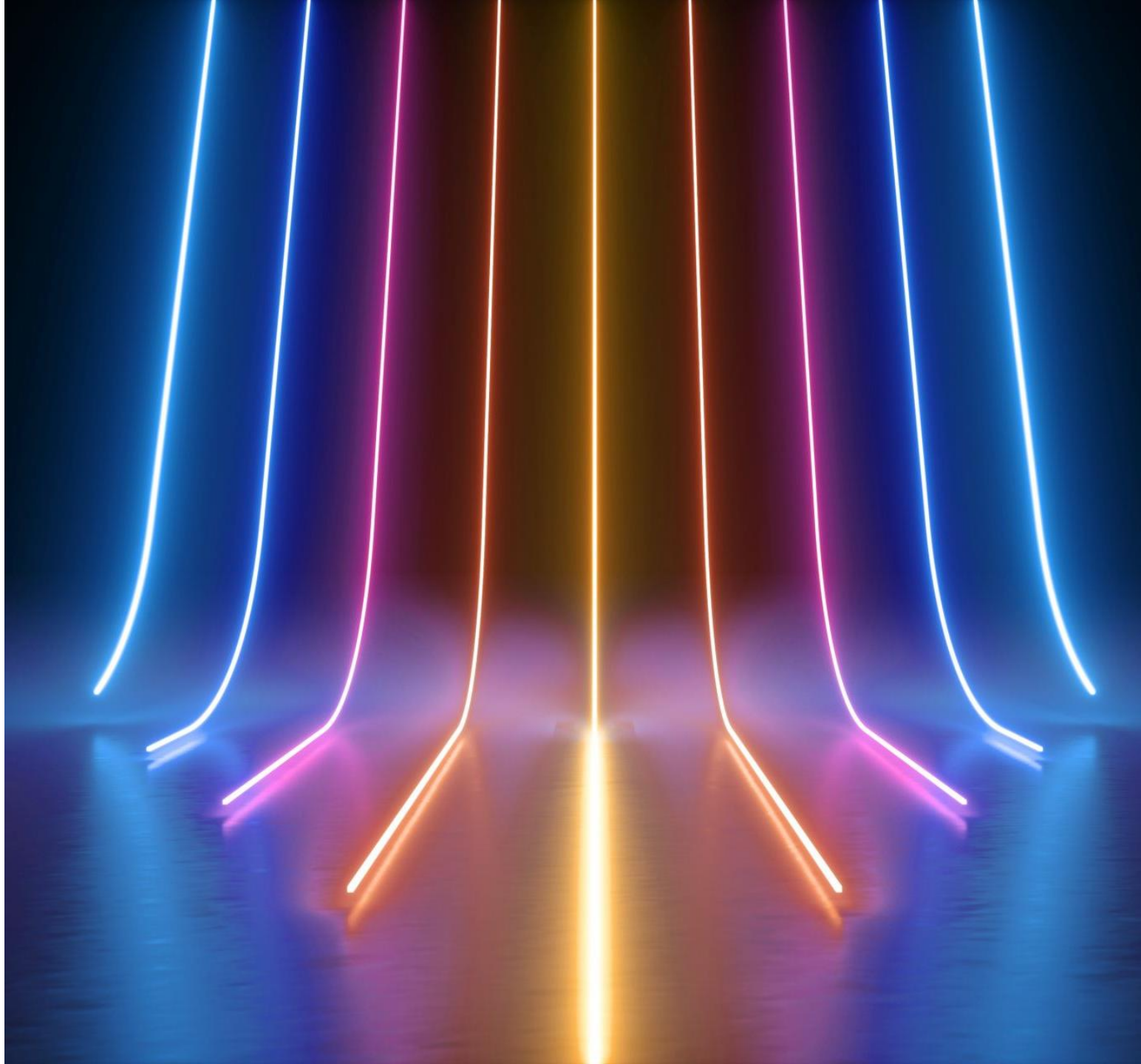
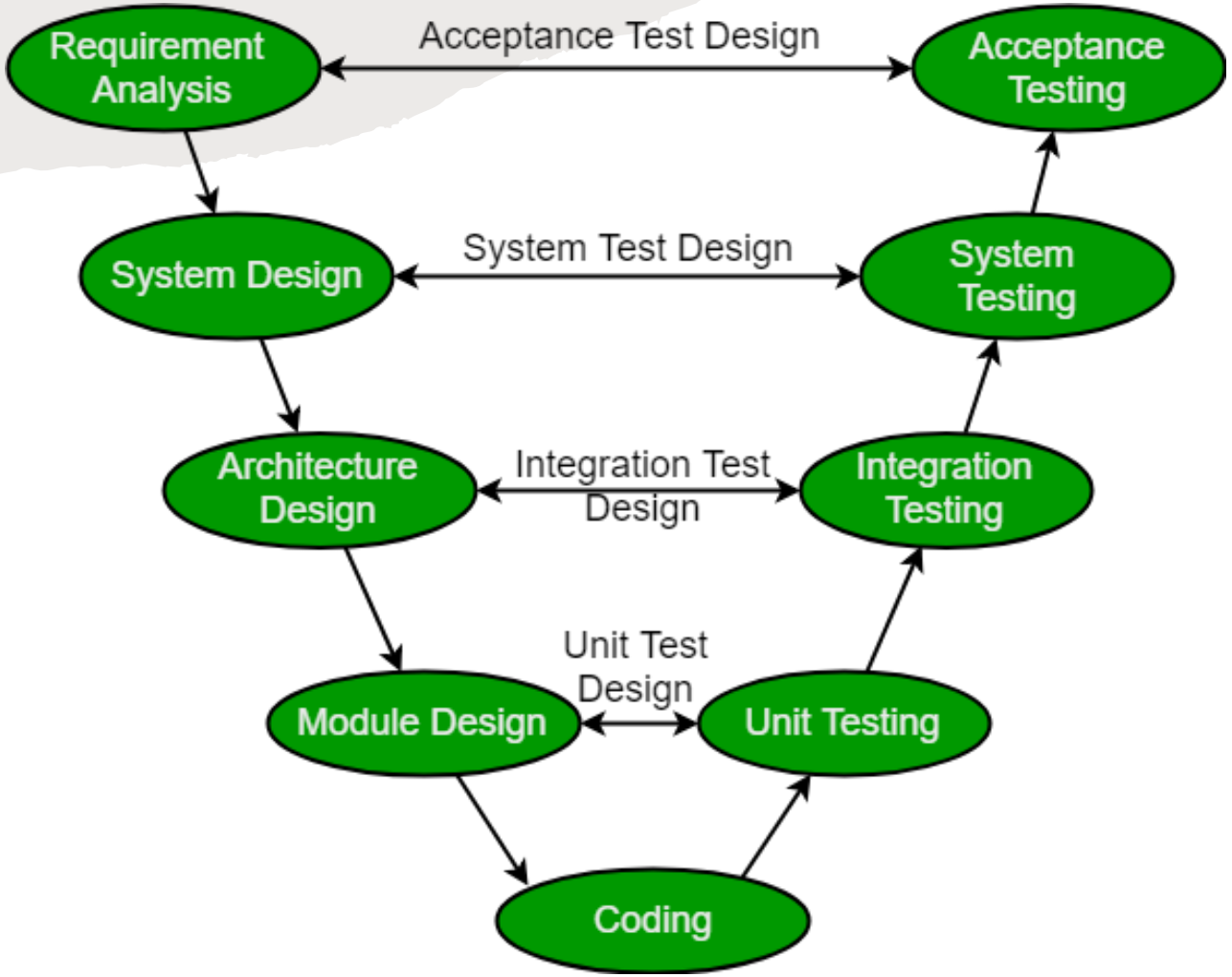
content:

V-model
DevOps tools use
UML diagrams types
Architecture pattern



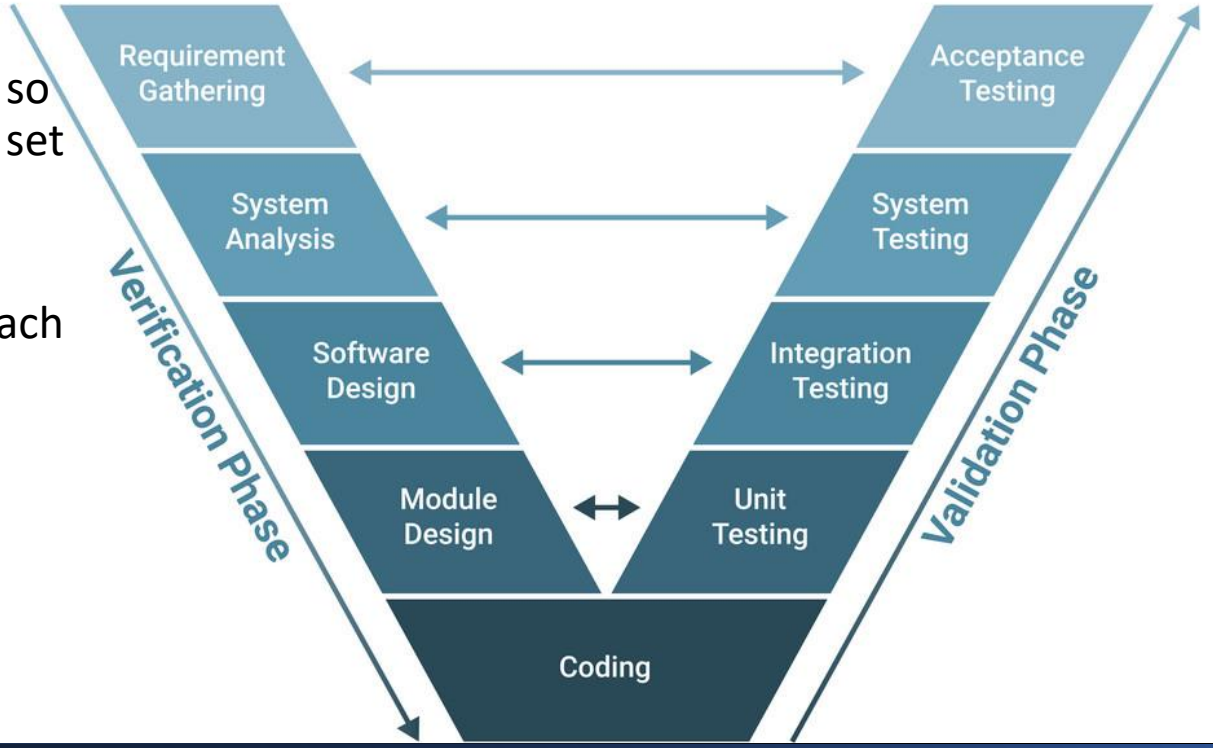
1) The V-model is a type of SDLC model where process executes in a sequential manner in V-shape. It is also known as Verification and Validation model. It is based on the association of a testing phase for each corresponding development stage. Development of each step directly associated with the testing phase. The next phase starts only after completion of the previous phase

2) V-model integrates the test process throughout the development processes, implementing the principle of early testing



- Provides guidance that testing needs to begin as early as possible in life cycle.
- Shows that testing is not only an execution-based activity.
- There are a variety of activities that needs to be preformed before the end of the coding phase.
- Testing activities should be carried out in parallel with development activities.

- Tester need to work with developers and business analysts so they can preform theses activities and tasks and produce a set of test deliverables.
- A v-model ia a model that illustrate hoe testing activities(verification & validation) can be integrated into each phase of the life cycle.



The V-Model is a linear and sequential model that consists of the following phases:

1. **Requirements Gathering and Analysis:** The first phase of the V-Model is the requirements gathering and analysis phase, where the customer's requirements for the software are gathered and analyzed to determine the scope of the project.
2. **Design:** In the design phase, the software architecture and design are developed, including the high-level design and detailed design.
3. **Implementation:** In the implementation phase, the software is actually built based on the design.
4. **Testing:** In the testing phase, the software is tested to ensure that it meets the customer's requirements and is of high quality.
5. **Deployment:** In the deployment phase, the software is deployed and put into use.
6. **Maintenance:** In the maintenance phase, the software is maintained to ensure that it continues to meet the customer's needs and expectations.

DevOps Tools Use

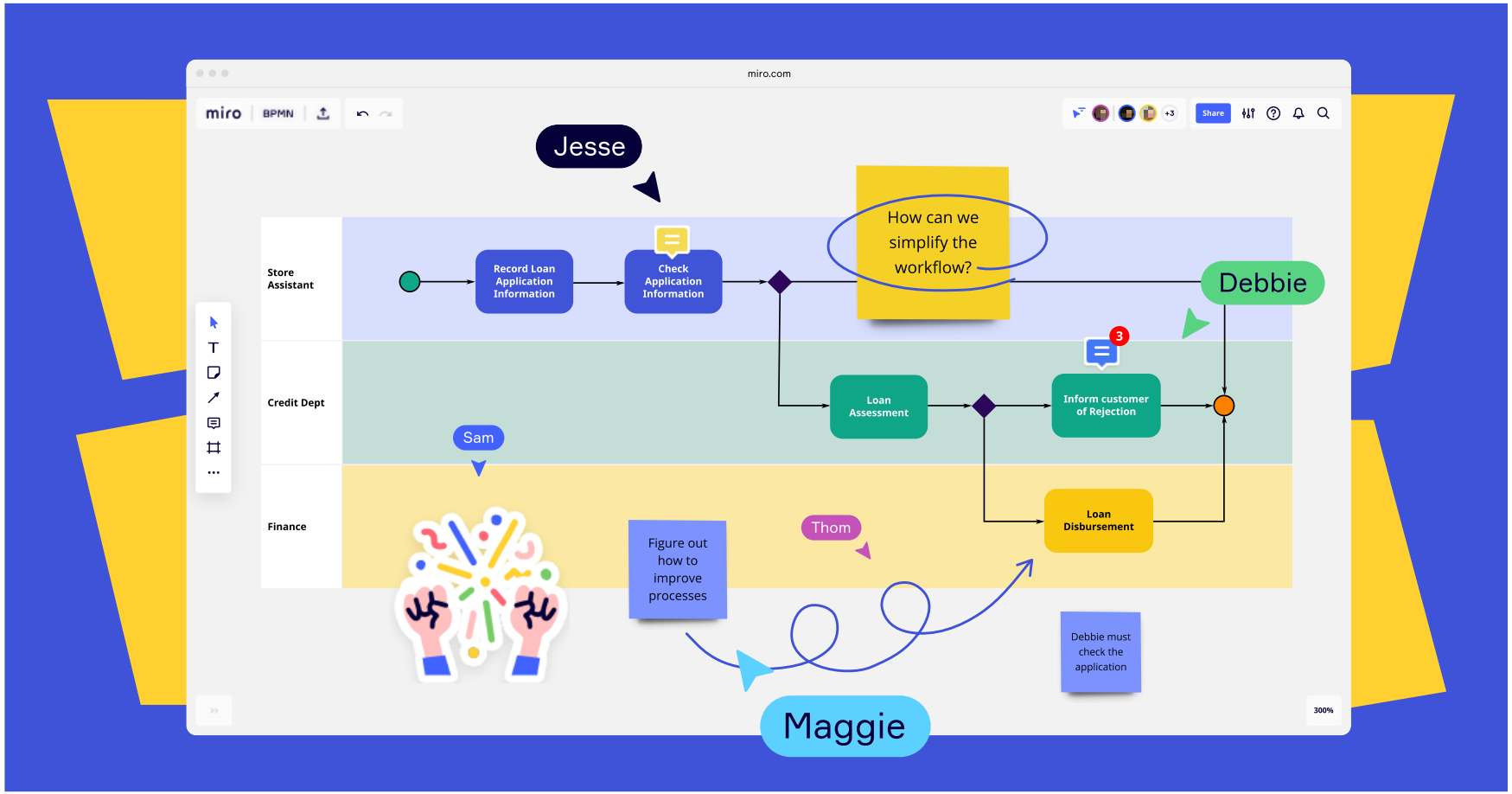
DevOps Model Defined

DevOps is the combination of cultural philosophies, practices, and tools that increases an organization's ability to deliver applications and services at high velocity: evolving and improving products at a faster pace than organizations using traditional software development and infrastructure management processes. This speed enables organizations to better serve their customers and compete more effectively in the market.

DevOps tools use:

help teams rapidly and reliably deploy and innovate for their customers. These tools automate manual tasks, help teams manage complex environments at scale, and keep engineers in control of the high velocity that is enabled by DevOps

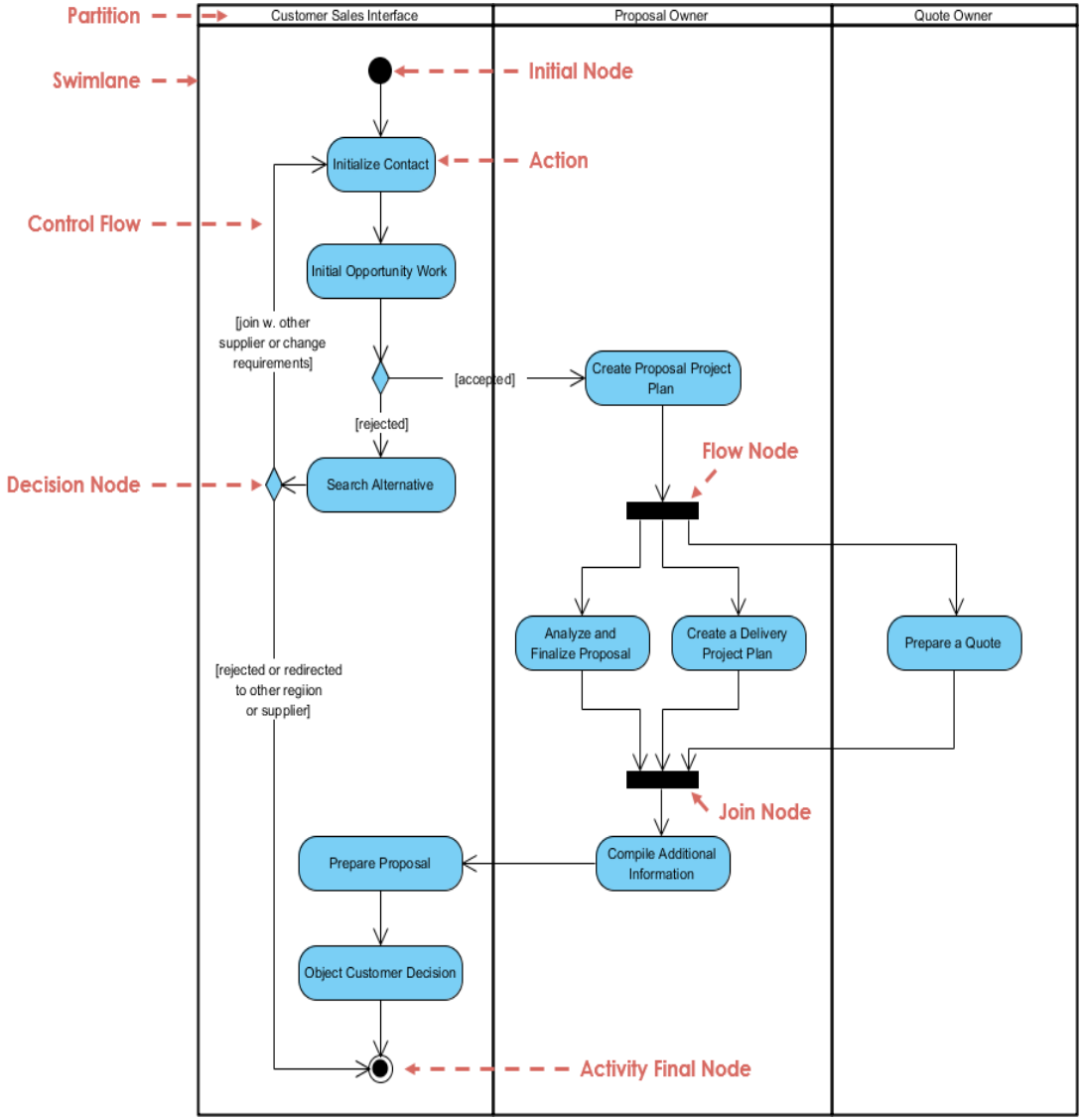
UML diagram Types



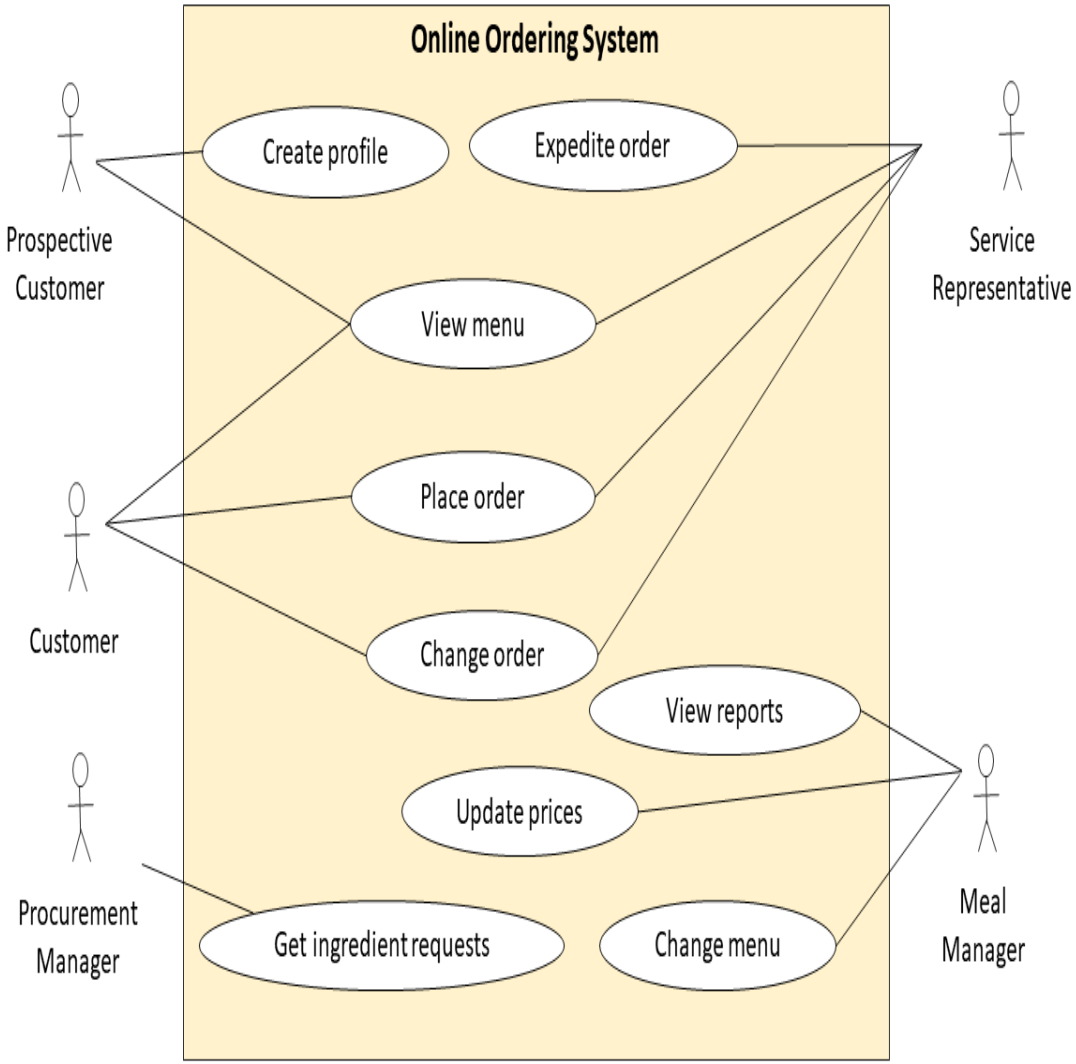
UML Diagram Types

- **Activity diagrams,** which show the activities involved in a process or in data processing.
- **Use case diagrams,** which show the interactions between a system and its environment.
- **Sequence diagrams,** which show interactions between actors and the system and between system components.
- **Class diagrams,** which show the object classes in the system and the associations between these classes.
- **State diagrams,** which show how the system reacts to internal and external events.

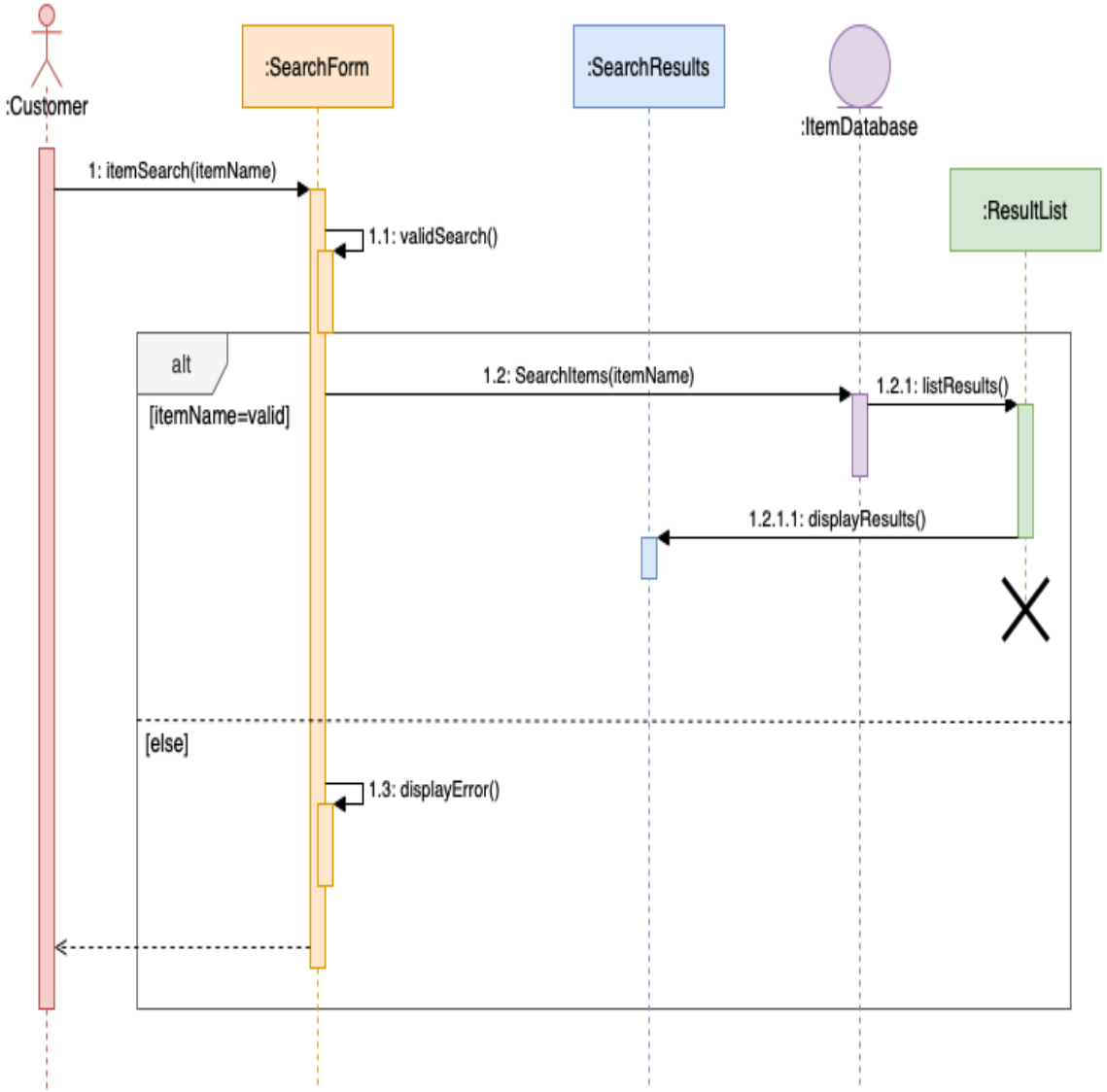
Activity diagrams



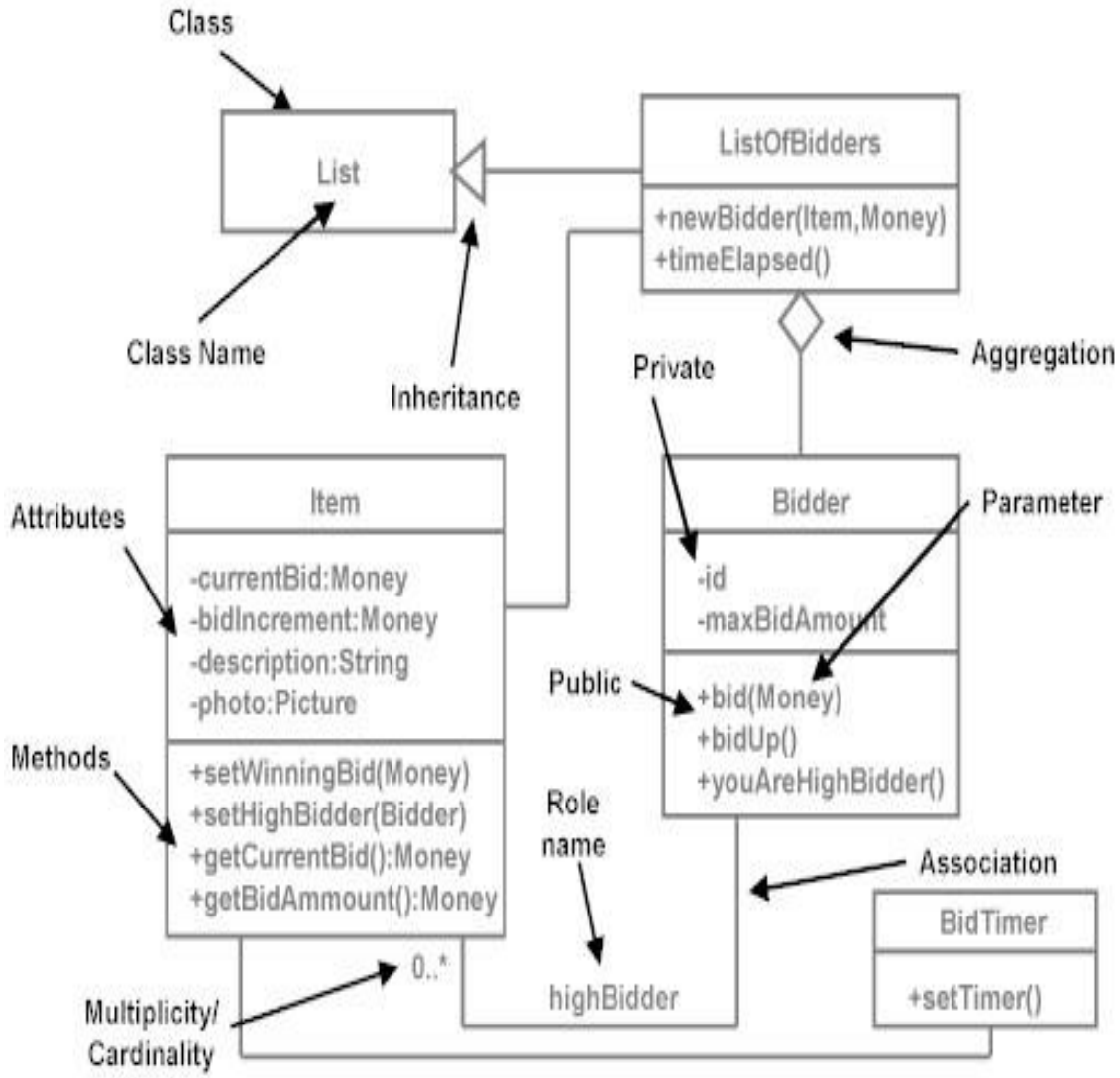
Use case diagrams



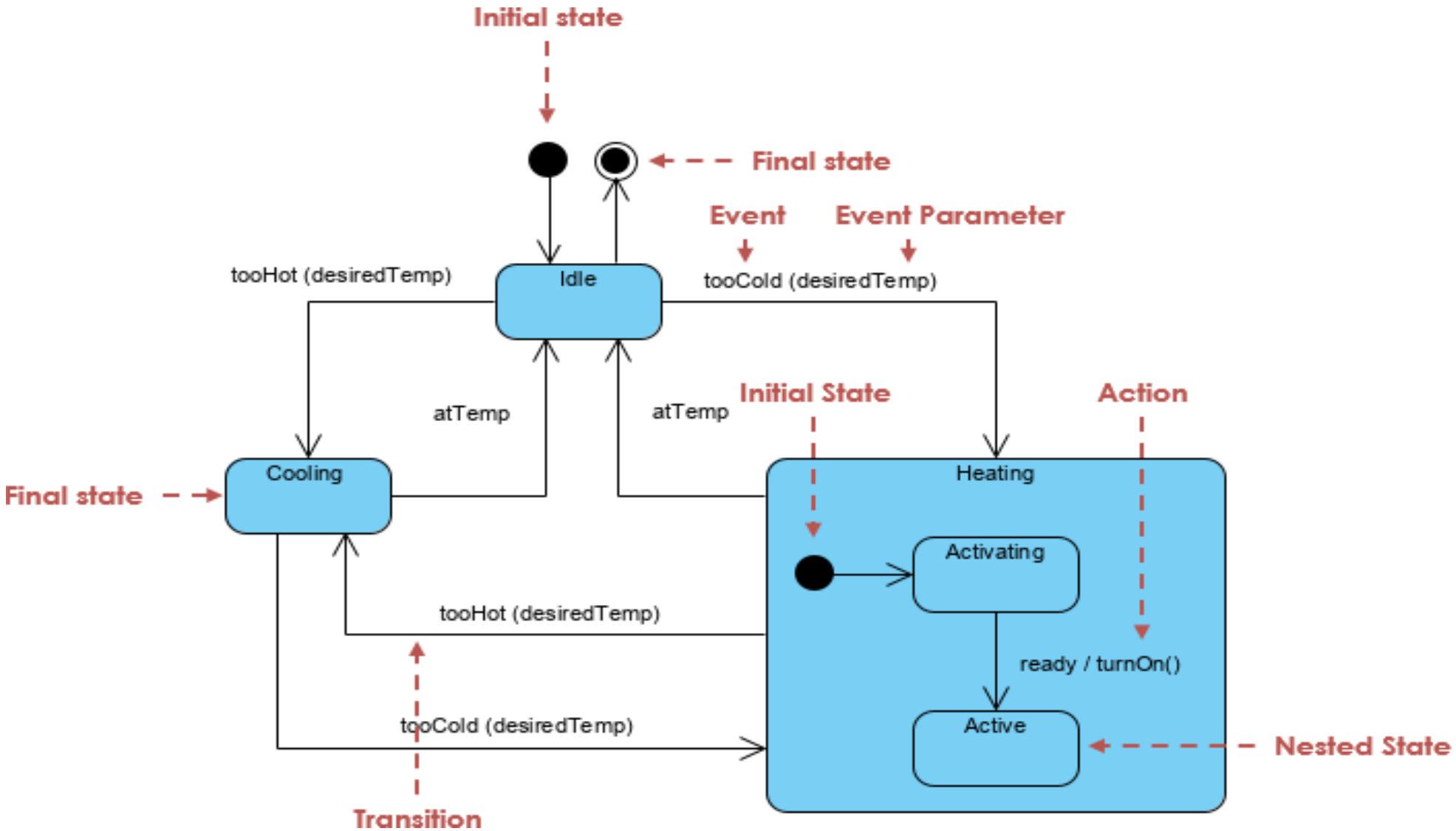
Sequence diagrams



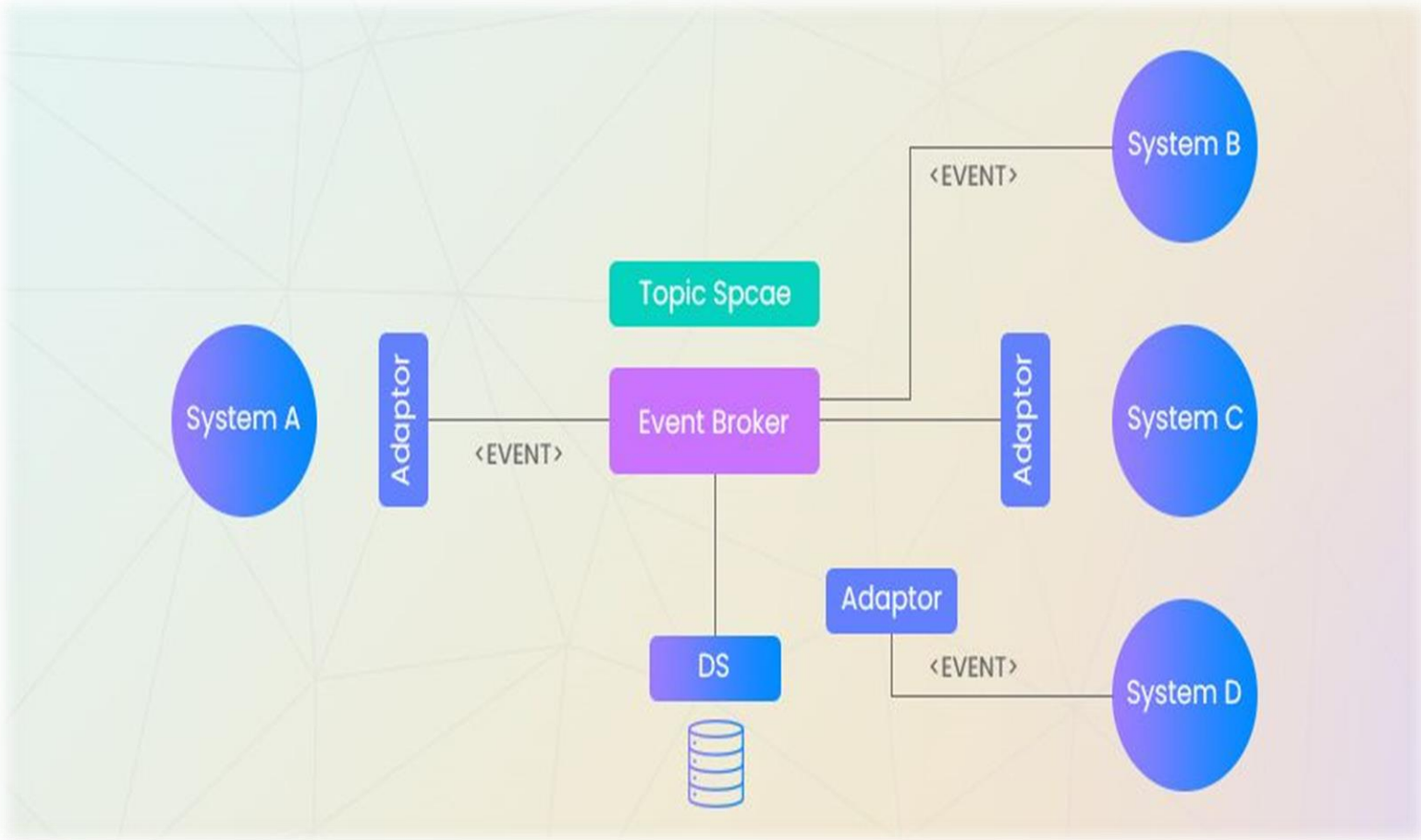
Class diagrams



State diagrams



Architectural Patterns



Architectural Patterns

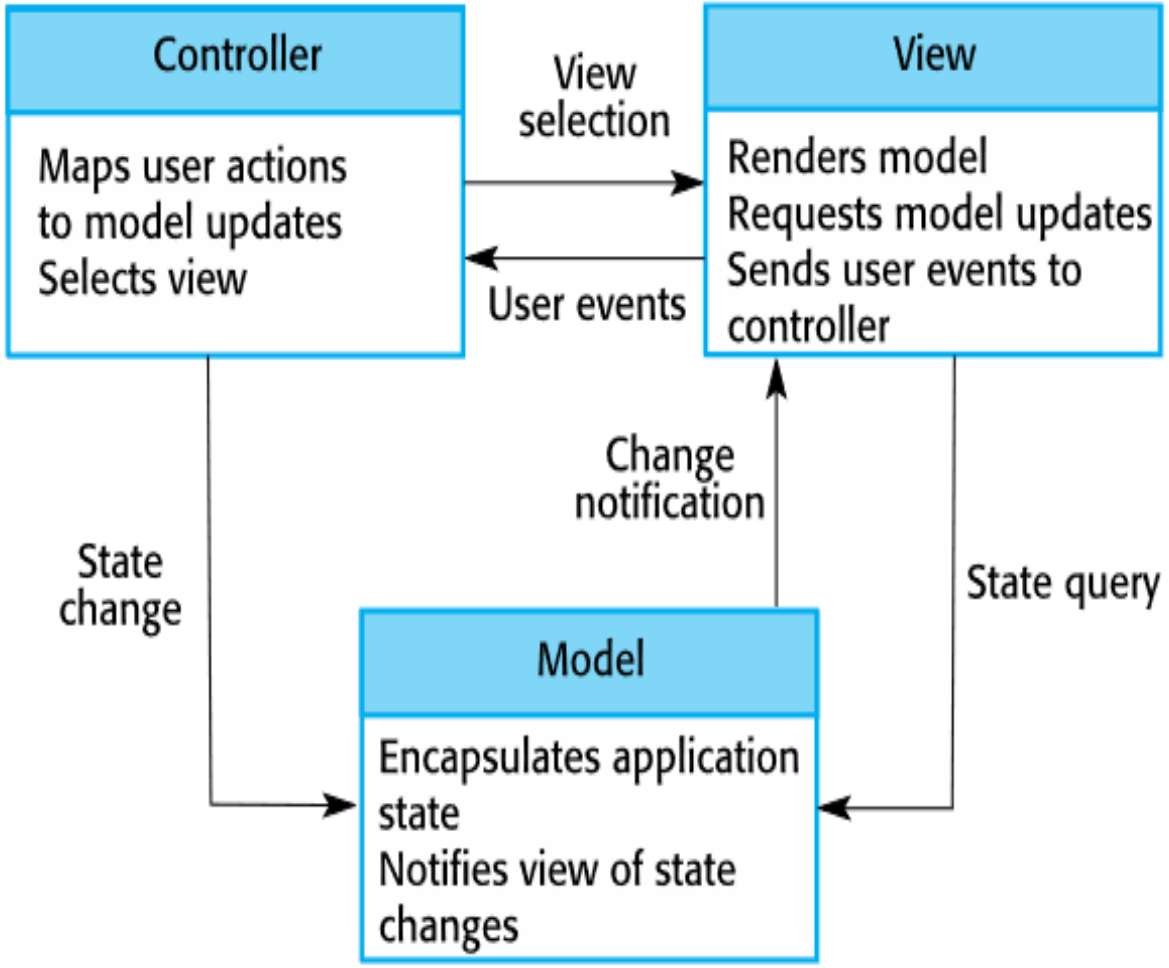
- Patterns are a means of representing, sharing and reusing knowledge.
- An architectural pattern is a stylized description of good design practice, which has been tried and tested in different environments.
- Patterns should include information about when they are and when the are not useful.
- Patterns may be represented using tabular and graphical descriptions.

Architectural Patterns:

- The Model-View-Controller (MVC) Pattern
- Layered Architecture Patter
- The Repository Patter
- The Client–Server Pattern
- Pipe and Filter Architecture

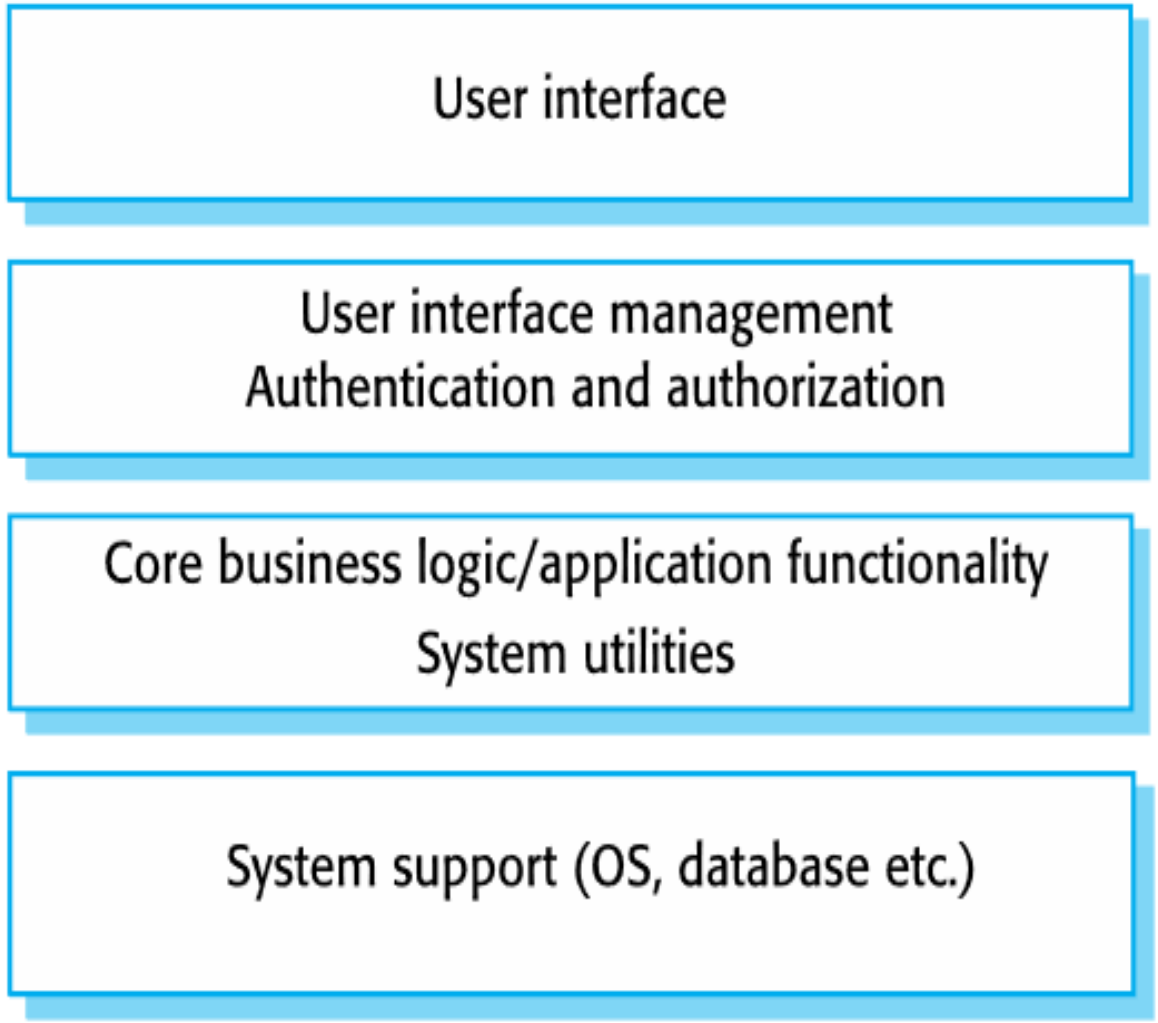
The Model-View-Controller (MVC) Pattern

Name	MVC (Model-View-Controller)
Description	Separates presentation and interaction from the system data. The system is structured into three logical components that interact with each other. The Model component manages the system data and associated operations on that data. The View component defines and manages how the data is presented to the user. The Controller component manages user interaction (e.g., key presses, mouse clicks, etc.) and passes these interactions to the View and the Model. See Figure 6.3.
Example	Figure 6.4 shows the architecture of a web-based application system organized using the MVC pattern.
When used	Used when there are multiple ways to view and interact with data. Also used when the future requirements for interaction and presentation of data are unknown.
Advantages	Allows the data to change independently of its representation and vice versa. Supports presentation of the same data in different ways with changes made in one representation shown in all of them.
Disadvantages	Can involve additional code and code complexity when the data model and interactions are simple.



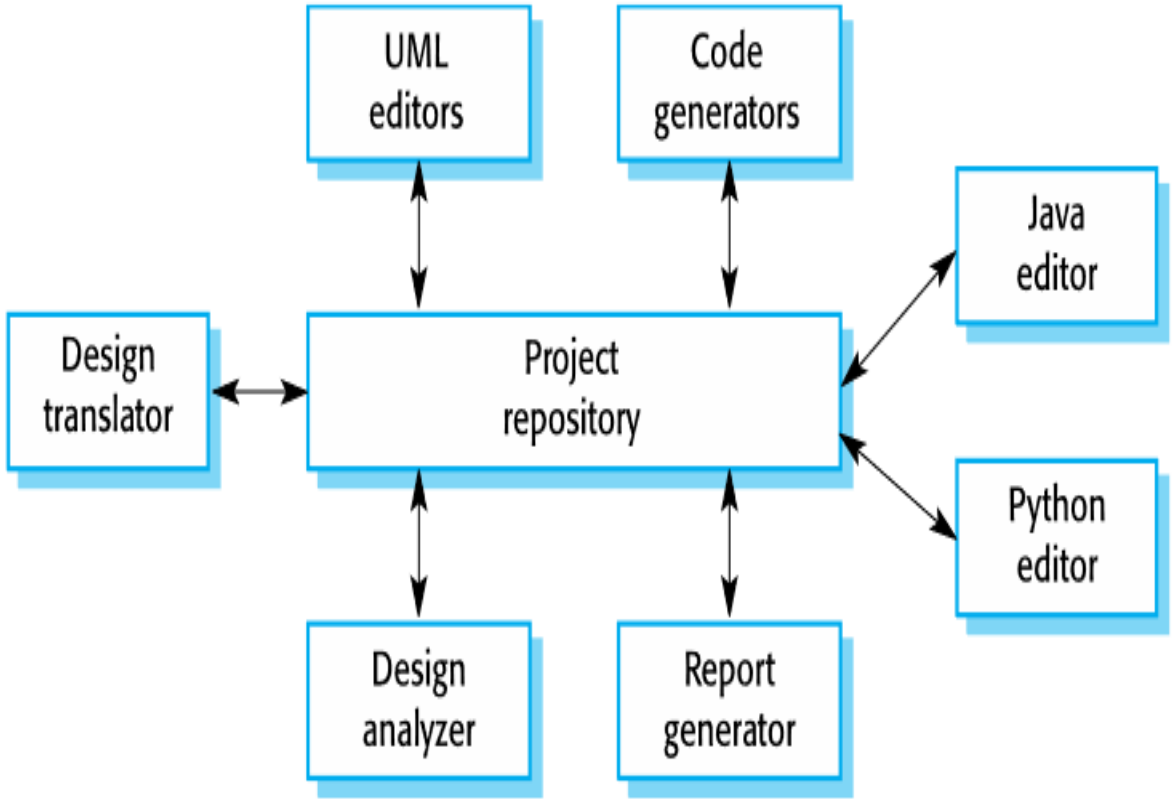
Layered Architecture Patter

Name	Layered architecture
Description	Organizes the system into layers with related functionality associated with each layer. A layer provides services to the layer above it so the lowest-level layers represent core services that are likely to be used throughout the system. See Figure 6.6.
Example	A layered model of a system for sharing copyright documents held in different libraries, as shown in Figure 6.7.
When used	Used when building new facilities on top of existing systems; when the development is spread across several teams with each team responsibility for a layer of functionality; when there is a requirement for multi-level security.
Advantages	Allows replacement of entire layers so long as the interface is maintained. Redundant facilities (e.g., authentication) can be provided in each layer to increase the dependability of the system.
Disadvantages	In practice, providing a clean separation between layers is often difficult and a high-level layer may have to interact directly with lower-level layers rather than through the layer immediately below it. Performance can be a problem because of multiple levels of interpretation of a service request as it is processed at each layer.



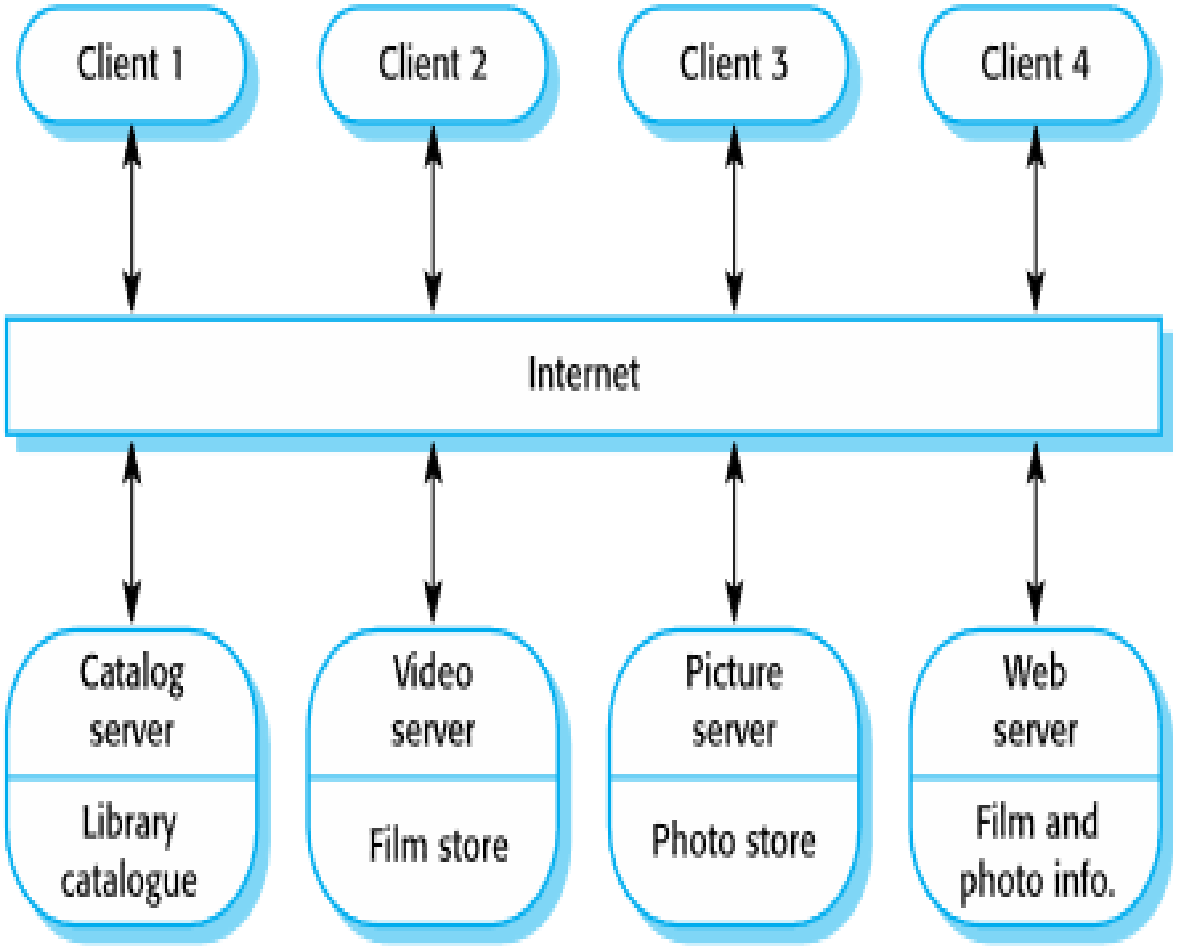
The Repository Patter

Name	Repository
Description	All data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.
Example	Figure 6.9 is an example of an IDE where the components use a repository of system design information. Each software tool generates information which is then available for use by other tools.
When used	You should use this pattern when you have a system in which large volumes of information are generated that has to be stored for a long time. You may also use it in data-driven systems where the inclusion of data in the repository triggers an action or tool.
Advantages	Components can be independent-they do not need to know of the existence of other components. Changes made by one component can be propagated to all components. All data can be managed consistently (e.g., backups done at the same time) as it is all in one place.
Disadvantages	The repository is a single point of failure so problems in the repository affect the whole system. May be inefficiencies in organizing all communication through the repository. Distributing the repository across several computers may be difficult.



The Client–Server Pattern

Name	Client-server
Description	In a client–server architecture, the functionality of the system is organized into services, with each service delivered from a separate server. Clients are users of these services and access servers to make use of them.
Example	Figure 6.11 is an example of a film and video/DVD library organized as a client–server system.
When used	Used when data in a shared database has to be accessed from a range of locations. Because servers can be replicated, may also be used when the load on a system is variable.
Advantages	The principal advantage of this model is that servers can be distributed across a network. General functionality (e.g., a printing service) can be available to all clients and does not need to be implemented by all services.
Disadvantages	Each service is a single point of failure so susceptible to denial of service attacks or server failure. Performance may be unpredictable because it depends on the network as well as the system. May be management problems if servers are owned by different organizations.



Pipe and Filter Pattern

Name	Pipe and filter
Description	The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.
Example	Figure 6.13 is an example of a pipe and filter system used for processing invoices.
When used	Commonly used in data processing applications (both batch- and transaction-based) where inputs are processed in separate stages to generate related outputs.
Advantages	Easy to understand and supports transformation reuse. Workflow style matches the structure of many business processes. Evolution by adding transformations is straightforward. Can be implemented as either a sequential or concurrent system.
Disadvantages	The format for data transfer has to be agreed upon between communicating transformations. Each transformation must parse its input and unparse its output to the agreed form. This increases system overhead and may mean that it is impossible to reuse functional transformations that use incompatible data structures.