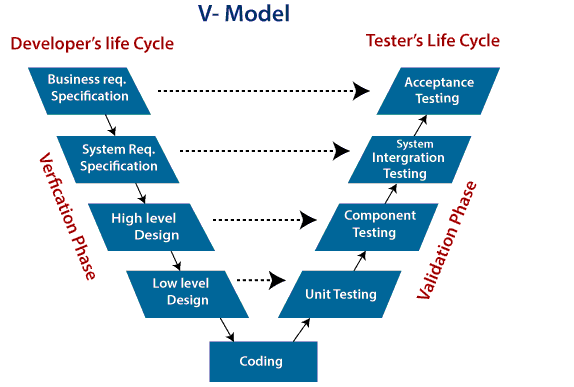
T1

V-Model

V-Model also referred to as the Verification and Validation Model. In this, each phase of SDLC must complete before the next phase starts. It follows a sequential design process same as the waterfall model. Testing of the device is planned in parallel with a corresponding stage of development.

**Verification:** It involves a static analysis method (review) done without executing code. It is the process of evaluation of the product development process to find whether specified requirements meet.

**Validation:** It involves dynamic analysis method (functional, non-functional), testing is done by executing code. Validation is the process to classify the software after the completion of the development process to determine whether the software meets the customer expectations and requirements.

So V-Model contains Verification phases on one side of the Validation phases on the other side. Verification and Validation process is joined by coding phase in V-shape. Thus it is known as V-Model.

**There are the various phases of Verification Phase of V-model:**

1. **Business requirement analysis:** This is the first step where product requirements understood from the customer's side. This phase contains detailed communication to understand customer's expectations and exact requirements.
2. **System Design:** In this stage system engineers analyze and interpret the business of the proposed system by studying the user requirements document.
3. **Architecture Design:** The baseline in selecting the architecture is that it should understand all which typically consists of the list of modules, brief functionality of each module, their interface relationships, dependencies, database tables, architecture diagrams, technology detail, etc. The integration testing model is carried out in a particular phase.
4. **Module Design:** In the module design phase, the system breaks down into small modules. The detailed design of the modules is specified, which is known as Low-Level Design
5. **Coding Phase:** After designing, the coding phase is started. Based on the requirements, a suitable programming language is decided. There are some guidelines and standards for coding. Before checking in the repository, the final build is optimized for better performance, and the code goes through many code reviews to check the performance.

**There are the various phases of Validation Phase of V-model:**

1. **Unit Testing:** In the V-Model, Unit Test Plans (UTPs) are developed during the module design phase. These UTPs are executed to eliminate errors at code level or unit level. A unit is the smallest entity which can independently exist, e.g., a program module. Unit testing verifies that the smallest entity can function correctly when isolated from the rest of the codes/ units.
2. **Integration Testing:** Integration Test Plans are developed during the Architectural Design Phase. These tests verify that groups created and tested independently can coexist and communicate among themselves.
3. **System Testing:** System Tests Plans are developed during System Design Phase. Unlike Unit and Integration Test Plans, System Tests Plans are composed by the client?s business team. System Test ensures that expectations from an application developer are met.
4. **Acceptance Testing:** Acceptance testing is related to the business requirement analysis part. It includes testing the software product in user atmosphere. Acceptance tests reveal the compatibility problems with the different systems, which is available within the user atmosphere. It conjointly discovers the non-functional problems like load and performance defects within the real user atmosphere.

Summary

How does the V model work?

When creating a V model, the project manager includes the key tasks and features that their team needs to work on throughout the project lifecycle to meet all of the client's requirements. The V model is similar to the waterfall model, which organizes these steps into a sequential life cycle. While the waterfall method uses a linear approach to list each task, the V model uses a V-graph.

On the left side of the V-shaped diagram, the project manager includes details for each verification stage. These stages describe what the client's requirements are and how the team plans to achieve them. The right side of the V represents the validation stages. This includes the specific tests that the team must perform to verify if the software meets all of the customer's requirements.

# T2

# DevOps Tools

DevOps is the next evolution of agile methodologies. A cultural shift that brings development and operations teams together. DevOps is a practice that involves a cultural change, new management principles, and technology tools that help to implement best practices.

When it comes to a DevOps toolchain, organizations should look for tools that improve collaboration, reduce context-switching, introduce automation, and leverage observability and monitoring to ship better software, faster.

There are two primary approaches to a DevOps toolchain: an all-in-one or open toolchain. An all-in-one DevOps solution provides a complete solution that usually doesn’t integrate with other third-party tools. An open toolchain can be customized for a team’s needs with different tools. Atlassian believes an open toolchain is the best approach since it can be customized with best-of-breed tools to the unique needs of an organization. Using this approach often leads to increased time efficiency and reduces time to market.

Regardless of the type of DevOps toolchain an organization uses, a DevOps process needs to use the right tools to  address the key phases of the DevOps lifecycle :

* Discover



Tools like Mural and Miro empower the entire software team to gather ideas and conduct research. Jira Product Discovery organizes this information into actionable inputs and prioritizes actions for development teams. As you’re prioritizing, you’ll also need to keep your backlog of user feedback in mind.

* Plan



we recommend tools that allow development and operations teams to break work down into smaller, manageable chunks for quicker deployments. This allows you to learn from users sooner and helps with optimizing a product based on the feedback. Look for tools that provide sprint planning, issue tracking, and allow collaboration, such as Jira. Test .

* Build

### **Production-identical environments for development**

A picture containing font, logo, text, graphics

Description automatically generated

Developers use open source tools like Kubernetes and Docker to provision individual development environments.

### **Source control and collaborative coding**



* Continuous Delivery

A close-up of a logo

Description automatically generated with low confidence

Continuous integration is the practice of checking in code to a shared repository several times a day, and testing it each time. That way, you automatically detect problems early, fix them when they’re easiest to fix, and roll out new features to your users as early as possible.

* Test

### **Automated testing**



* Continuous feedback



Customers are already telling you whether you’ve built the right thing – you just have to listen. Continuous feedback includes both the culture and processes to collect feedback regularly, and tools to drive insights from the feedback. Continuous feedback practices include collecting and reviewing NPS data, churn surveys, bug reports, support tickets, and even tweets. In a DevOps culture, everyone on the product team has access to user comments because they help guide everything from release planning to exploratory testing sessions.

# T3

# software architecture design patterns

**Software Architecture :**  
Software architecture is the blueprint of building software. It shows the overall structure of the software, the collection of components in it, and how they interact with one another while hiding the implementation.

This helps the software development team to clearly communicate how the software is going to be built as per the requirements of customers.

**Different Software Architecture Patterns :**

1. Layered Pattern
2. Client-Server Pattern
3. Event-Driven Pattern
4. Microkernel Pattern
5. Microservices Pattern

Let’s see one by one in detail.

**1. Layered Pattern :**  
As the name suggests, components(code) in this pattern are separated into layers of subtasks and they are arranged one above another.

Each layer has unique tasks to do and all the layers are independent of one another. Since each layer is independent, one can modify the code inside a layer without affecting others.

It is the most commonly used pattern for designing the majority of software. This layer is also known as ‘N-tier architecture’. Basically, this pattern has 4 layers.

1. Presentation layer (The user interface layer where we see and enter data into an application.)
2. Business layer (this layer is responsible for executing business logic as per the request.)
3. Application layer (this layer acts as a medium for communication between the ‘presentation layer’ and ‘data layer’.
4. Data layer (this layer has a database for managing data.)

Ideal for:

E-commerce web applications development like Amazon.

**2. Client-Server Pattern :**  
The client-server pattern has two major entities. They are a server and multiple clients.

Here the server has resources(data, files or services) and a client requests the server for a particular resource. Then the server processes the request and responds back accordingly.

Examples of software developed in this pattern:

* Email.
* WWW.
* File sharing apps.
* Banking, etc…

So this pattern is suitable for developing the kind of software listed in the examples.

**3. Event-Driven Pattern :**  
Event-Driven Architecture is an agile approach in which services (operations) of the software are triggered by events.

Well, what does an event mean?

When a user takes action in the application built using the EDA approach, a state change happens and a reaction is generated that is called an event.

**Eg:** A new user fills the signup form and clicks the signup button on Facebook and then a FB account is created for him, which is an event.

Ideal for:

Building websites with JavaScript and e-commerce websites in general.

**4. Microkernel Pattern :**  
Microkernel pattern has two major components. They are a core system and plug-in modules.

* The core system handles the fundamental and minimal operations of the application.
* The plug-in modules handle the extended functionalities (like extra features) and customized processing.

**Let’s imagine**, you have successfully built a chat application. And the basic functionality of the app is that you can text with people across the world without an internet connection. After some time, you would like to add a voice messaging feature to the application, then you are adding the feature successfully. You can add that feature to the already developed application because the microkernel pattern facilitates you to add features as plug-ins.

Microkernel pattern is ideal for:

Product-based applications and scheduling applications. We love new features that keep giving dopamine boost to our brain. Such as Instagram reels, YouTube Shorts and a lot more that feasts us digitally. So this pattern is mostly preferred for app development.

**5. Microservices Pattern :**  
The collection of small services that are combined to form the actual application is the concept of microservices pattern. Instead of building a bigger application, small programs are built for every service (function) of an application independently. And those small programs are bundled together to be a full-fledged application.

So adding new features and modifying existing microservices without affecting other microservices are no longer a challenge when an application is built in a microservices pattern.

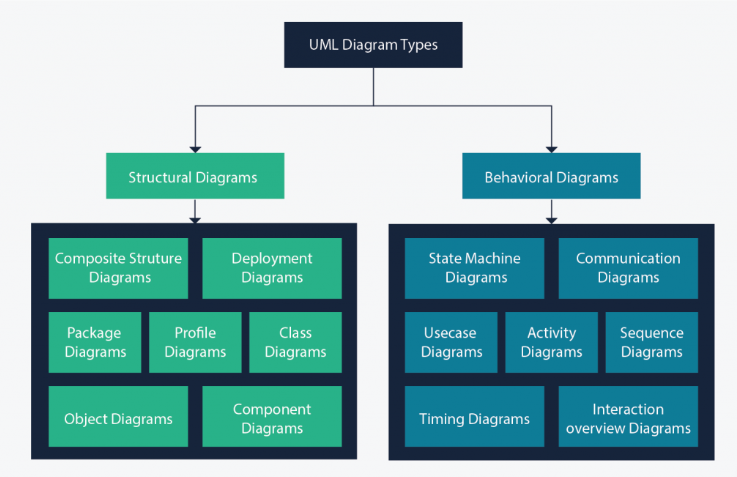
Modules in the application of microservices patterns are loosely coupled. So they are easily understandable, modifiable and scalable.

**Example** Netflix is one of the most popular examples of software built-in microservices architecture. This pattern is most suitable for websites and web apps having small components.

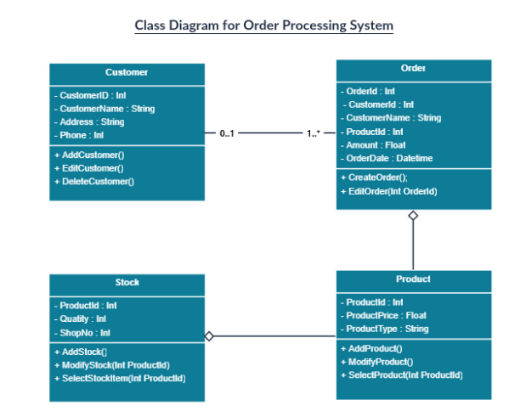
**That’s all!**

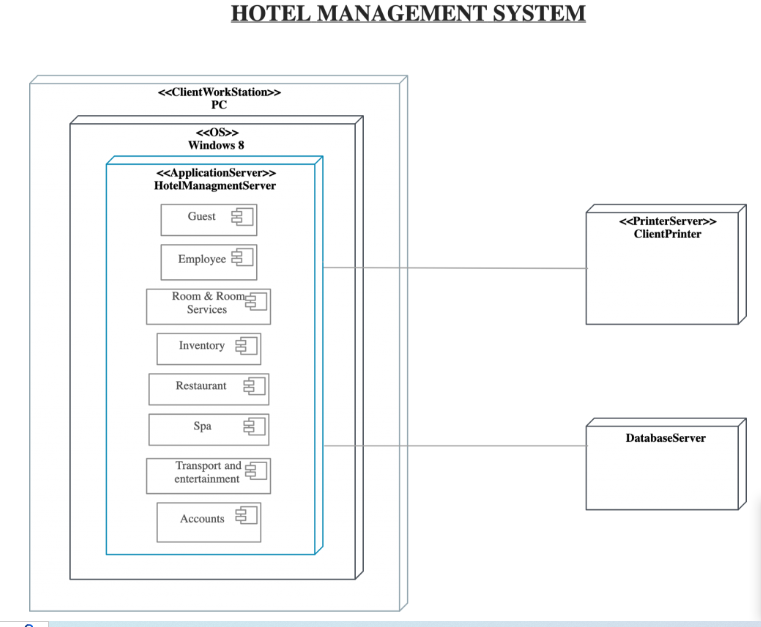
Architecture patterns determine the destiny of the software about to be built. And there is no one-stop solution to build any kind of software. So choose what suits you the most!

### T4 UML Diagram Types

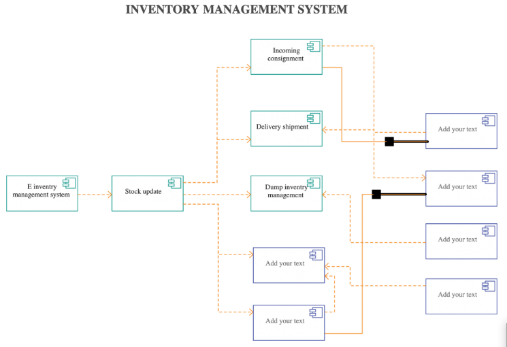


Class Diagram Deployment Diagram

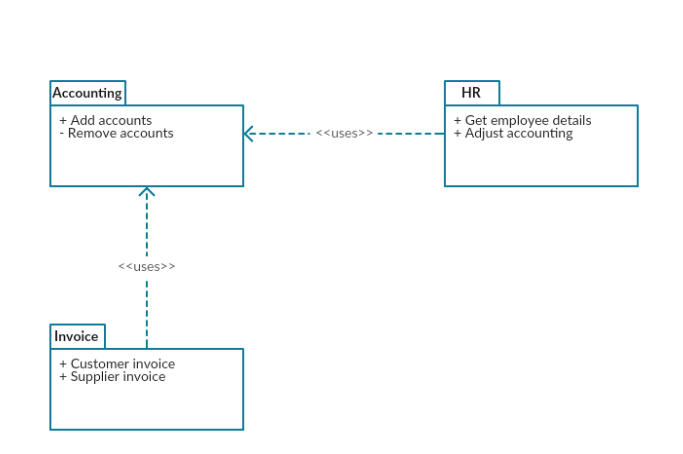


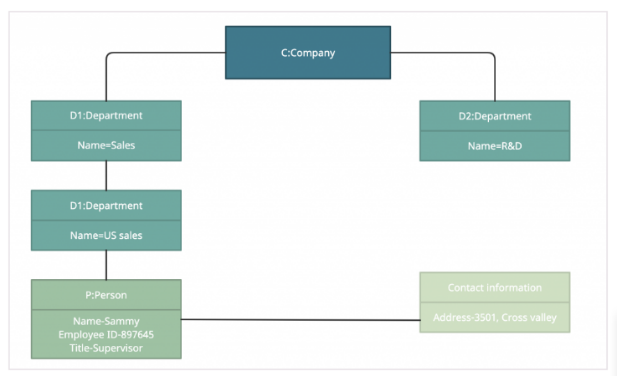


Component Diagram



Object Diagram Package Diagram

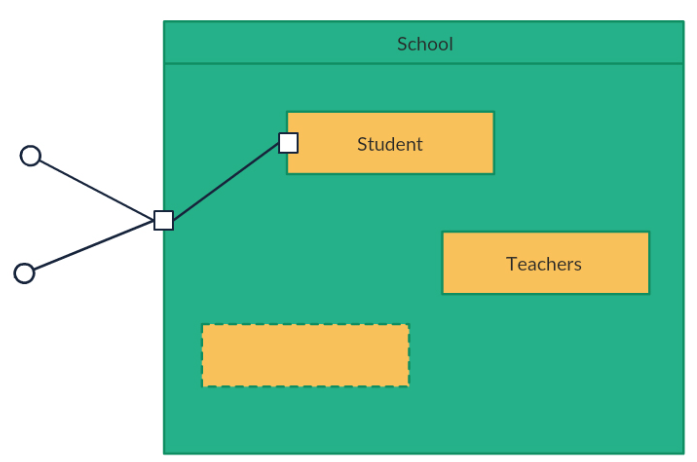




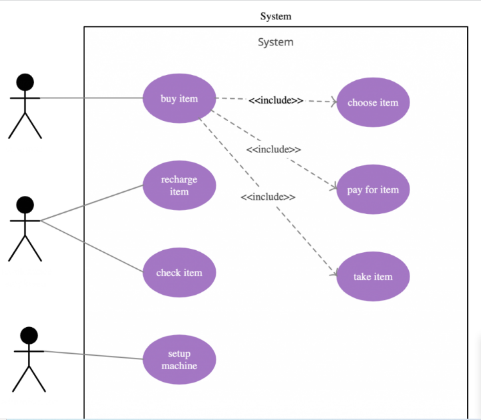
Profile Diagram Composite Structure Diagram

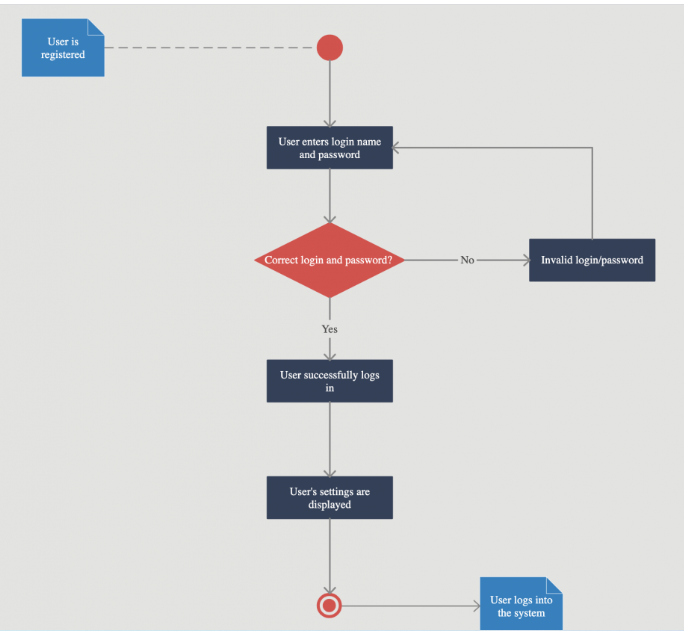
A screenshot of a computer

Description automatically generated with low confidence

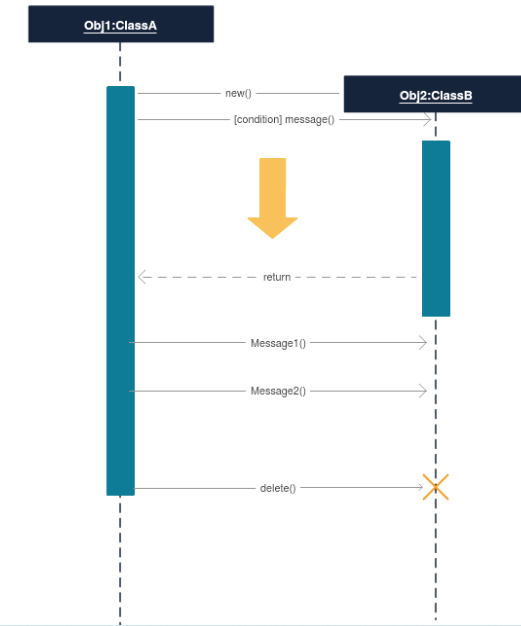


Use Case Diagram Activity Diagram





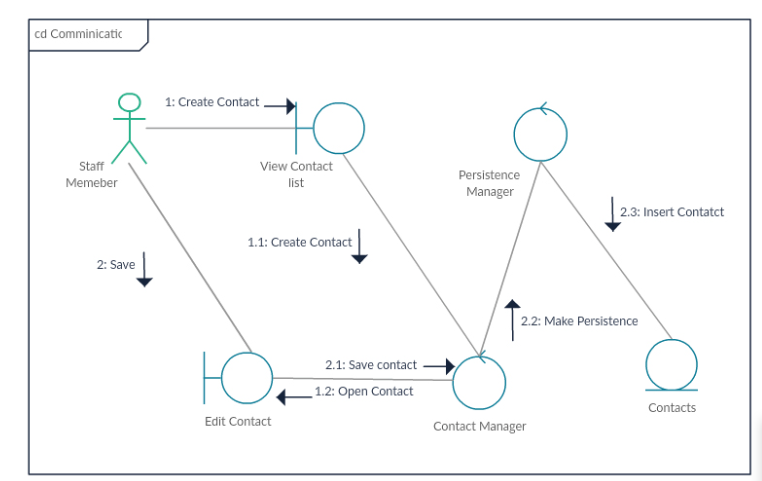
State Machine Diagram Sequence Diagram



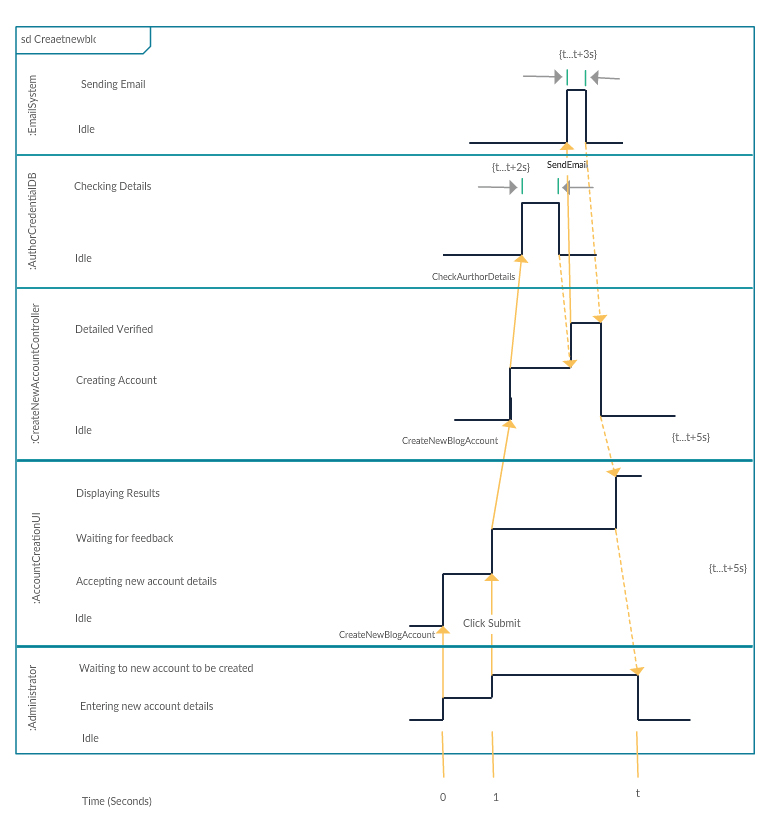
A picture containing text, diagram, line, screenshot

Description automatically generated

Communication Diagram



Timing Diagram



Interaction Overview Diagram

A picture containing text, screenshot, diagram, plan

Description automatically generated