



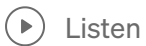
# Basics of Software Architecture: A Guide for Developers

Mastering Software Architecture: A Beginners Guide to Principles and Patterns With Examples



Learn With Whiteboard · [Follow](#)

8 min read · Mar 27



Credit — [Seth Eckert for Omni](#)

Software architecture is the process of designing and organizing the software components, their relationships, and how they interact with each other to achieve the desired functionality. It involves the identification of the main components of a system, their properties, and the way they interact with each other to deliver the required functionality.

Software architecture is an important process that ensures the quality of software products and helps businesses resolve complexity in the long and short term. Let's understand it by taking a software architecture example.

### **Example**

One example of software architecture is the design of an e-commerce website. The website consists of various components, such as the user interface, the database, and the payment gateway. The architecture of the website defines the way these components interact with each other, and the overall structure of the website. The architecture also includes the choice of technologies, tools, and frameworks used to build the website.

The importance of software architecture cannot be overstated. It plays a vital role in resolving complexity in the long and short term.

In the short term, software architecture helps in reducing development time and costs.

By defining the structure and relationships of software components, the architecture provides a roadmap for developers to follow. This makes it easier to design, develop, and test software components, resulting in faster development times and reduced costs.

In the long term, software architecture helps in maintaining the software system.

As the system evolves, it may become more complex, making it difficult to maintain and update. A well-designed software architecture provides a framework for adding new features and functionality without impacting the existing system. This ensures that the system remains flexible, scalable, and maintainable over time.

A good example of the importance of software architecture in business is the case of Amazon. Amazon is one of the largest e-commerce platforms in the world, with millions of customers and billions of transactions. The success of Amazon can be attributed in part to its software architecture. Amazon's architecture is designed to be highly scalable, flexible, and resilient. It is built on a microservices architecture, where the various components of the system are broken down into smaller, more manageable services. This makes it easier to develop, test, and deploy new features and also makes it easier to maintain and scale the system.

Source — Zenbusiness

Amazon's architecture also includes various technologies and tools that enable it to handle high volumes of traffic and transactions. **For instance**, Amazon's use of Amazon Web Services (AWS) provides the platform with the ability to scale up and down quickly, depending on the demand. This has allowed Amazon to handle peak traffic during the holiday season without any downtime or performance issues.

**Software Architecture Principles: S.O.L.I.D.**

Now that we know the basics of software architecture, let's understand what are SOLID principles. Well, S.O.L.I.D is a set of principles that help software developers design and build software systems that are easy to maintain, extend, and test. These principles were introduced by Robert C. Martin, also known as "Uncle Bob," and have become a standard for software development. Each letter in the acronym S.O.L.I.D stands for a principle that should be followed in software architecture design:

### 1. Single Responsibility Principle (SRP)

The Single Responsibility Principle states that a class should have only one responsibility or reason to change. A class that has more than one responsibility becomes difficult to maintain and test. By keeping each class focused on a single responsibility, it becomes easier to understand, change, and extend the code.

**For example,** consider a class that handles user authentication and also sends emails. In this case, if there is a change in the email sending functionality, it could impact the authentication functionality. Instead, separate classes should be created for user authentication and email sending, each with a single responsibility.

### 2. Open-Closed Principle (OCP)

The Open-Closed Principle states that a software entity should be open for extension but closed for modification. This means that the behavior of a software entity, such as a class, should be extendable without modifying its source code. The entity should be designed in a way that allows new functionality to be added without changing the existing code.

**For example,** consider a class that calculates the total cost of an order. If there is a new requirement to apply a discount to the order, the existing class should not be modified. Instead, a new class should be created that extends the functionality of the existing class.

### 3. Liskov Substitution Principle (LSP)

The Liskov Substitution Principle states that a derived class should be substitutable for its base class. This means that the derived class should be able to replace its base class without affecting the correctness of the program.

**For example,** consider a program that uses a base class called Shape to calculate the area of a shape. The Liskov Substitution Principle states that any derived class, such as Rectangle or Circle, should be able to replace the Shape class without breaking the program.

#### 4. Interface Segregation Principle (ISP)

The Interface Segregation Principle states that a class should not be forced to implement interfaces it does not use. This means that interfaces should be designed to be specific to the needs of a class, rather than forcing a class to implement methods that it does not need.

**For example**, consider an interface called Customer that includes methods for both creating a new customer and updating an existing customer. A class that only needs to create new customers should not be forced to implement the update method. Instead, the interface should be divided into two separate interfaces, one for creating customers and one for updating customers.

#### 5. Dependency Inversion Principle (DIP)

The Dependency Inversion Principle states that high-level modules should not depend on low-level modules. Instead, both should depend on abstractions. This means that the implementation details of a module should not be exposed to other modules.

**For example**, consider a program that uses a logging library. If the program directly depends on the logging library, it becomes difficult to replace the library with a different one. Instead, the program should depend on an abstraction, such as an interface, that can be implemented by any logging library. This allows the logging library to be changed without affecting the program's functionality.

Source: [Devopedia](#)

## Good Software Architecture Characteristics

Good software architecture characteristics are essential for creating software systems that are efficient, scalable, maintainable, and extensible. Here are some of the most important characteristics of good software architecture:

1. **Modularity:** The software system should be divided into modules, where each module performs a specific function. This makes it easier to develop, test, and maintain the system.

2. **Scalability:** The software system should be designed to handle changes in requirements, data volume, and user load. This means that the system should be able to scale up or down as needed.
3. **Flexibility:** The software system should be flexible and adaptable to change. It should be designed in a way that allows for modifications and extensions without requiring significant changes to the existing code.
4. **Maintainability:** The software system should be easy to maintain and update. This means that the code should be well-organized, easy to understand, and easy to modify.
5. **Testability:** The software system should be designed in a way that allows for easy testing. This means that the code should be written in a way that makes it easy to test each module individually.
6. **Reusability:** The software system should be designed in a way that allows for code reuse. This means that modules should be designed to be used in multiple contexts and applications.
7. **Performance:** The software system should be designed to be efficient and performant. This means that it should be able to handle a large number of users and transactions without experiencing significant slowdowns.

By incorporating these characteristics into software architecture design, developers can create software systems that are efficient, maintainable, and scalable.

## Popular Software Architectural Patterns

Software architecture patterns, also known as software design patterns, are reusable solutions to common problems in software architecture. These patterns provide a framework for solving specific architectural problems and can be used to improve the quality of software design.

There are various software architecture patterns available, and each pattern provides a solution to a specific set of problems. Here are some of the most common software architecture patterns:

1. **Model-View-Controller (MVC):** This pattern separates the user interface, business logic, and data storage into three distinct components: Model, View,

and Controller. This separation improves the maintainability, scalability, and reusability of the code.

2. **Layered Architecture:** This pattern divides the software system into logical layers, where each layer performs a specific set of functions. This separation of concerns makes it easier to develop, test, and maintain the system.
3. **Microservices:** This pattern involves breaking down a large software system into smaller, independent services that communicate with each other through APIs. This separation improves scalability, flexibility, and fault tolerance.
4. **Event-Driven Architecture (EDA):** This pattern involves creating systems that respond to events, such as user actions, system events, or external events. This pattern improves scalability, flexibility, and responsiveness.
5. **Domain-Driven Design (DDD):** This pattern involves creating a software design that reflects the domain of the problem being solved. This pattern improves the maintainability, scalability, and reusability of the code.
6. **Repository Pattern:** This pattern involves separating the data access logic from the rest of the application code. This pattern improves the testability, maintainability, and extensibility of the code.

By using software architecture patterns, developers can create software systems that are more efficient, scalable, and maintainable. These patterns provide a framework for solving specific architectural problems, and can be used to improve the quality of software design. If you want to quickly learn about all major software architecture patterns used in major industries, here's a video that could help.



## Conclusion

In conclusion, we should acknowledge software architecture as a critical process that ensures the quality and maintainability of software products. It plays a vital role in resolving complexity in the long and short term, by providing a framework for designing, developing, testing, and maintaining software systems.

The example of Amazon demonstrates the importance of software architecture in business, by providing a scalable, flexible, and resilient platform for e-commerce transactions. A well-designed software architecture is essential for businesses looking to develop and maintain high-quality software products that meet the demands of today's customers.

Lastly, as a developer, it's crucial to stay updated with the latest technological advancements. Here are the [top software architecture trends](#) to keep an eye on in 2023 and beyond.

*You may also like,*

### All Major Software Architecture Patterns Explained

Meaning, Advantages, Disadvantages & Applications

medium.com

## 6 Software Architecture Trends for 2023 & Beyond!

Software architecture is the foundation upon which software systems are built. It is the structural design that defines...

medium.com

## 7 Stages of Data Science Project Life Cycle Explained

Understanding the Step by Step Approach of Data Science Lifecycle

medium.com

Software Architecture

Software Development

Software Engineering

Software Testing

Programming



Follow



## Written by Learn With Whiteboard

1.1K Followers

Get byte-size whiteboard lessons to help you increase your tech and non tech vocabulary.

### More from Learn With Whiteboard



Learn With Whiteboard

## Difference Between Proxy vs Reverse Proxy vs Load Balancer Explained

Understanding Proxies, Reverse Proxies, and Load Balancers in Web Infrastructure

6 min read · Jul 16



--



1



Learn With Whiteboard

# Understanding What is API Gateway and How it Works (With Examples)

API Gateway Explained in Simple Terms with Examples

7 min read · Sep 30



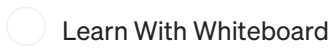
 Learn With Whiteboard

## All Major Software Architecture Patterns Explained

Meaning, Advantages, Disadvantages & Applications

18 min read · Feb 22





## What is Blockchain Layer 0, 1, 2, 3 Explained

Layers of Blockchain Architecture Explained

4 min read · Dec 23, 2022




---

See all from Learn With Whiteboard

---

### Recommended from Medium

 English For IT

## 10 YouTube Channels Every Software Developer Should Follow

One way to get good at something is to watch others do it. This applies to building your communication skills as well. And there is no...

5 min read · Sep 22

 --  8

---

 Carlos Arguelles

## My favorite coding question to give candidates

## A coding question, from the viewpoint of an Google/Amazon/Microsoft interviewer

11 min read · Nov 12



### Lists

#### General Coding Knowledge

20 stories · 617 saves

#### Stories to Help You Grow as a Software Developer


19 stories · 589 saves

#### Coding & Development

11 stories · 289 saves

#### Leadership

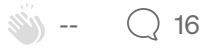
39 stories · 157 saves

 Kostya Stepanov in UX Planet

## Back-End & Web Development Trends For 2024

By Mary Moore, copywriter at Shakuro

9 min read · Oct 17



 Mohammad Faisal in Level Up Coding

## Awesome Terminal Applications

Sharpen the axe before you cut the wood.

★ · 4 min read · Nov 13







Suresh Podeti

## System design: Google Docs

Introduction

8 min read · Oct 26



--



2



Daniel Foo

## Software Architecture and Design Trend 2023

2023 is almost coming to the end. It's always a good idea to reflect back on what has been the popular topic on Software Architecture and...

7 min read · Oct 1



--



5

See more recommendations