

Research on the Application of Layered Architecture in Computer Software Development

Zhenan Tu

Biaoliang Technology (Shanghai) Co., Ltd.; Shanghai 200441, China

Abstract: The objective of this study is to examine the application of layered technologies in computer software development. As software scale and complexity continue to increase, the adoption of appropriate software architecture becomes crucial. Layered architecture, a commonly employed software design pattern, enables effective organization and management of software systems, enhancing their maintainability, scalability, and reusability. This paper explores the fundamental principles and advantages of layered architecture and presents its specific applications in computer software development, such as client-server architecture, MVC architecture, and microservices architecture. Through analysis and comparison of relevant case studies, the suitability and effectiveness of layered technologies in different contexts are evaluated. Lastly, the study concludes with an overview of the application prospects of layered technologies and suggests future research directions.

Keywords: Layered architecture; Software development; Software architecture; Maintainability; Scalability; Reusability; Client-server architecture; MVC architecture; Microservices architecture.

1. Introduction

With the continuous growth of computer software scale and complexity, the adoption of appropriate software architecture has become a key factor in ensuring the maintainability, scalability, and reusability of software systems. Layered architecture, as a commonly used software design pattern, has gained considerable attention in computer software development due to its powerful capabilities and advantages. This paper aims to investigate the application of layered technologies in computer software development and explore their potential in enhancing software system quality and development efficiency. By dividing the software system into multiple independent layers, layered technologies achieve modularity and decoupling of the software system, enabling individual components to be developed, tested, and maintained independently, while also providing better scalability and reusability. This paper will introduce the fundamental principles, design principles, and advantages of layered architecture, and delve into its specific applications in computer software development, such as client-server architecture, MVC architecture, and microservices architecture. Through case analysis and comparison, the suitability and effectiveness of layered technologies in different environments will be evaluated, providing valuable guidance and recommendations for software developers. Finally, we will summarize the application prospects of layered technologies and propose future research directions to drive the quality and sustainable development of software systems.

2. Basic Principles of Layered Architecture

2.1. Overview of Layered Architecture

Layered architecture is a commonly used software design pattern that aims to divide a complex software system into multiple layers, with each layer assuming specific functions and responsibilities. Each layer is built upon the foundation of the layer below it, forming a logical hierarchy. Typically,

layered architecture consists of three main layers: the presentation layer, the business logic layer, and the data access layer. The presentation layer is responsible for interacting with users and forwarding their requests to the business logic layer. It includes components related to the user interface (UI) and user input processing. By providing a user-friendly interface and interactive functionality, the presentation layer enables users to intuitively operate the software system. The business logic layer is the core of the software system, containing components that handle business logic and rules. In this layer, data is passed from the presentation layer to the data access layer for appropriate processing and calculations, resulting in outcomes. The business logic layer coordinates the interaction between various components to ensure that the software system operates as expected. The data access layer is responsible for interacting with data storage, such as databases, file systems, or other data storage methods. It provides operations for data retrieval, writing, updating, as well as handling data validation and security protection. The data access layer interacts with the business logic layer, passing data for processing and returning the processed results to the presentation layer or data storage.

2.2. Design Principles of Layered Architecture

The design of layered architecture follows several important principles to ensure the maintainability, scalability, and reusability of the system. The principle of separation of concerns requires different functional modules to be divided into different layers, with each layer focusing on specific tasks. This reduces the complexity of the system and allows each layer to be developed, tested, and maintained independently. The principle of dependency inversion emphasizes relying on abstractions rather than concrete implementations. By using interfaces or abstract classes to define agreements between layers, coupling between layers is reduced, enhancing system flexibility and substitutability. The principle of single responsibility requires that each layer and component should have a clear and specific responsibility and function. Each layer should focus on solving specific

problems, avoiding functional overlap and coupling, thereby improving code readability and maintainability [1].

2.3. Advantages of Layered Architecture

Layered architecture offers several advantages, making it a widely adopted software design pattern. Layered architecture provides excellent modularity and decoupling features. By dividing the system into multiple layers, each layer focuses only on specific functions and responsibilities, allowing different parts of the system to be independently developed and maintained. This modular design makes the code easier to understand, test, and reuse. Layered architecture enhances system maintainability. Due to the explicit separation between different layers, when modifications or extensions are needed, relevant layers can be located and addressed more easily without affecting other layers. This allows developers to work more efficiently in development and maintenance tasks. Layered architecture provides good scalability. The loose coupling between layers allows the introduction of new functionalities or adjustments to the system structure by adding new layers or modifying existing ones, without the need for extensive modifications to the entire system. Layered architecture promotes code reuse. With clear responsibilities and functions for each layer, it becomes easier to extract a specific layer or component for reuse in other projects or modules. This reduces the effort of writing redundant code and improves development efficiency. As a commonly used software design pattern, layered architecture holds an important position and finds widespread application in computer software development. Through its explicit layer structure, modularity and decoupling features, as well as the advantages of maintainability, scalability, and reusability, layered architecture provides an effective design approach for software developers to construct high-quality and sustainable software systems.

3. Application of MVC Architecture

3.1. Overview of MVC Architecture

MVC (Model-View-Controller) architecture is a common and widely adopted software design pattern used to build maintainable and scalable applications. It divides the functionality of an application into three core components: the model, the view, and the controller. The model is responsible for handling data logic and business rules. It represents the data and state of the application and provides operations such as data access, updates, and validation. The model typically includes functionalities related to data storage, database connections, data manipulation, and business logic. The view is the presentation part of the user interface. It is responsible for presenting the data from the model to the user, often through a graphical interface or other forms of display. The view is passive and updates itself based on changes in the model and passes user input to the controller. The controller acts as a coordinator between the model and the view. It receives user input and invokes appropriate methods in the model for data manipulation and business logic processing. The controller is also responsible for passing updated data to the view for display and adjusting the application's state based on user feedback. The core idea behind MVC architecture is separation of concerns. By dividing the different functionalities of an application into the model, view, and controller, MVC architecture provides a structured approach to developing and maintaining complex software systems [2].

3.2. Applications of MVC Architecture in Software Development

MVC architecture has widespread applications in software development. Here are some common use cases:

1). Web Development: MVC architecture is widely used in web application development. By dividing the application's logic into the model, view, and controller, developers can better organize and manage the code. The model handles data and business logic, the view presents the user interface, and the controller handles user requests and coordinates the interaction between the model and the view. This layered structure makes web applications easier to scale and maintain.

2). Desktop Applications: MVC architecture is also applicable to desktop application development. By dividing the application's functionality into the model, view, and controller, developers can achieve decoupling of logic and interface. The model handles data and business logic, the view presents the user interface, and the controller handles user input and coordinates the interaction between the model and the view. This structure makes desktop applications easier to test, maintain, and update.

3). Mobile Applications: MVC architecture is widely used in mobile application development as well. Similar to web and desktop applications, the model handles data and business logic, the view presents the user interface, and the controller handles user input and coordinates the interaction between the model and the view. By adopting MVC architecture, developers can better organize the code and achieve maintainability and scalability of the application.

3.3. Case Study: Developing an Enterprise-Level Application Using MVC Architecture

Let's take an example of developing an enterprise-level application using MVC architecture for a case study. MVC architecture also provides better code reusability. The model can be reused in different applications, while the view and controller can be customized and extended as per requirements. This layered structure also promotes team collaboration, where different developers can independently work on and test their respective components. With MVC architecture, an enterprise-level CRM system can effectively handle large amounts of data and complex business logic. The model provides structured management of data and consistency validation, the view provides a convenient user interface, and the controller coordinates the interaction between the model and the view. This layered architecture makes the system highly maintainable, scalable, and flexible, providing powerful customer relationship management tools for businesses. Developing enterprise-level applications like CRM systems using MVC architecture enables modular code development, maintainability, and performance optimization. The layered structure of this architecture makes system development more flexible, efficient, and promotes team collaboration. However, careful definition of interfaces and interactions between the model, view, and controller is required during the design phase. Proper code organization and architectural design require experience and technical expertise. The team should plan and undergo sufficient training to ensure successful implementation of the project. MVC architecture is a powerful software design pattern suitable for developing various types of applications. It achieves separation of concerns, provides a good code

organization approach, and offers flexibility. In enterprise-level application development, MVC architecture can help build reliable, maintainable, and scalable systems. By properly dividing modules and responsibilities, teams can collaborate more effectively, improve development efficiency, and enhance system quality [3].

4. Application of Microservices Architecture

4.1. Overview of Microservices Architecture

Microservices Architecture is a software architectural style used to build complex applications. It divides the application into a set of small, independent services, with each service focusing on performing specific business functions. These services interact with each other through lightweight communication mechanisms and can be deployed, scaled, and maintained independently. In a Microservices Architecture, each service has its own codebase and database. Each service can use different programming languages, technology stacks, and databases to best meet its specific requirements. This decomposition of the application helps improve team autonomy and flexibility, enabling teams to develop, test, and deploy services independently as needed.

4.2. Application of Microservices Architecture in Software Development

Microservices Architecture has a wide range of applications in software development. Here are some common use cases:

1). Monolithic Application Decomposition: Microservices Architecture is suitable for splitting large monolithic applications into smaller, more manageable services. By dividing the functionality into independent services, developers can assign development tasks to different teams and achieve parallel development. This decomposition also makes the application easier to scale and maintain, as only the services requiring more resources need to be scaled, rather than the entire application.

2). Resilience and Scalability: Microservices Architecture makes applications more resilient and scalable. Since each service is independently deployed, specific services can be horizontally scaled as needed to handle high loads and traffic demands. This flexibility allows the system to better adapt to changing requirements and traffic spikes, improving application performance and availability.

3). Technology Diversity: Microservices Architecture allows different services to use different technology stacks and programming languages. This enables development teams to choose tools and technologies that best suit their specific needs and technical backgrounds. For example, some services can be developed using Java, while others can use Python or Go. This flexibility allows development teams to leverage existing technologies and skills more effectively, enhancing development efficiency [4].

4.3. Case Study: Cloud-Native Application Development with Microservices Architecture

Let's analyze a case study of cloud-native application development based on Microservices Architecture. Consider an online e-commerce platform that needs to handle a large number of user requests, inventory management, order

processing, and payment functionalities. To achieve high availability, resilience, and scalability, the development team decides to build this application using Microservices Architecture. In cloud-native application development based on Microservices Architecture, different business functionalities are divided into independent services. For example, the user management service is responsible for handling user registration, login, and personal information management; the inventory management service is responsible for managing product inventory and inventory changes; the order service is responsible for handling order creation, processing, and payment workflows. Each service has its own database and codebase and uses technology stacks suitable for its requirements. These services interact with each other through lightweight communication mechanisms. For example, when a user places an order, the order service calls the inventory management service to check the availability of the products. If the inventory is sufficient, the order service creates the order and notifies the payment service for payment. This loosely coupled communication mechanism allows services to scale and evolve independently while providing better reliability and fault tolerance. By using cloud-native technologies, this Microservices Architecture-based application can leverage the advantages of cloud computing platforms. The development team can package each service as an independent container using containerization technologies like Docker to achieve lightweight, portable, and scalable deployment. Additionally, using container orchestration tools like Kubernetes, services can be automatically managed, scaled, and load-balanced to handle evolving requirements and traffic. Cloud-native application development based on Microservices Architecture also brings higher observability and monitoring capabilities. By using tools like logging, metrics collection, and distributed tracing, development teams can better understand and monitor the performance, health, and failure situations of the application. This observability enables teams to quickly identify and resolve issues, improving application stability and reliability. Microservices Architecture-based cloud-native application development provides a flexible, scalable, and reliable approach to building complex applications. By decomposing the application into independent services and deploying and managing them using cloud-native technologies, development teams can better meet evolving requirements and deliver exceptional user experiences. However, teams need to be mindful of challenges such as defining service boundaries, managing communication between services, and ensuring data consistency to ensure overall system stability and consistency.

5. Application Scenarios Comparison and Evaluation

5.1. Applicability of Layering Techniques in Different Scenarios

Layering techniques are widely applied in various scenarios in the field of computer science, including network communication, operating systems, database management systems, and more. In different application scenarios, layering techniques exhibit different applicability and advantages. Let's first consider the field of network communication. In network communication, layering techniques are commonly used to implement network protocol stacks. The protocol stack divides network functionality into multiple layers, with

each layer responsible for specific tasks, such as data transmission, routing, and error detection. This layered design makes network communication more modular and scalable, as different layers' functionalities can be implemented and upgraded independently, while providing better flexibility and interoperability. In the field of operating systems, layering techniques are widely used in operating system design and implementation. Operating systems typically consist of multiple layers, including the hardware abstraction layer, kernel layer, system call layer, and application layer. Each layer is responsible for different functionalities, such as the hardware abstraction layer handling low-level hardware access, the kernel layer providing core operating system functionality, the system call layer offering an interface between user programs and the kernel, and the application layer containing user applications. This layered design makes operating systems more modular and maintainable, as different layers' functionalities can be developed and tested independently, while providing better security and resource management [5].

In the field of database management systems, layering techniques are used to implement the architecture of database systems. Database systems typically consist of multiple layers, including the storage layer, query processing layer, and application layer. The storage layer handles the physical storage and access of data, the query processing layer handles query parsing, optimization, and execution, and the application layer contains user application programs. This layered design makes database systems more flexible and scalable, as different layers' functionalities can be independently optimized and adjusted, while providing better data access efficiency and data consistency. In addition to the above-mentioned application areas, layering techniques have widespread applications in many other scenarios. For example, in graphics and image processing, layering techniques can be used to implement image encoding and decoding algorithms. In software development, layering techniques can be used to implement software architecture and modular design. In machine learning and deep learning, layering techniques can be used to implement the hierarchical structure of neural networks.

5.2. Evaluation of Layering Techniques' Effectiveness

When evaluating layering techniques, we can consider the following aspects: Performance evaluation: We can evaluate the effectiveness by comparing the performance metrics of systems using layering techniques versus systems that do not use layering techniques. For example, in network communication, we can compare the transmission speed and latency of systems using layered network protocol stacks with those directly using network protocols. In database management systems, we can compare the query response time and concurrency performance of systems using layered architecture versus traditional database architectures. Scalability evaluation: Layering techniques should support system scalability, meaning they can effectively handle increased loads and concurrent requests. We can evaluate the scalability of layering techniques by simulating system behavior under different load conditions. For example, in network communication, we can simulate high concurrent requests and observe the system's throughput and response time. Maintainability evaluation: Layering techniques should make systems easier to maintain and debug. We can evaluate

the impact of layering techniques on system maintainability. For example, in operating systems, we can evaluate the impact of modular design at different layers on system maintainability. We can observe if it is easy to locate and fix errors and if different layers' functionalities can be modified and upgraded independently. In addition to the above evaluation methods, we can also use real-world applications and user feedback to assess the effectiveness of layering techniques. By applying layering techniques in real environments and collecting user feedback and experiences, we can understand how layering techniques perform in practical scenarios and make improvements and optimizations accordingly [6].

6. Conclusion

Layering techniques demonstrate wide applicability and advantages in different application scenarios. In fields such as network communication, operating systems, and database management systems, the application of layering techniques enables modular and scalable system designs, while providing improvements in flexibility, interoperability, security, and resource management. Through performance evaluations, we have observed that systems utilizing layering techniques outperform non-layered systems in terms of transmission speed, latency, query response time, and other metrics. Scalability evaluations indicate that layering techniques effectively handle high-concurrency requests and increased loads. Additionally, layering techniques enhance system maintainability, making systems easier to maintain, debug, and upgrade. While there are challenges and limitations, such as interface design and performance bottlenecks, by employing sound design principles and evaluations, we can fully leverage the advantages of layering techniques to enhance system performance and maintainability. Future research can further explore the application of layering techniques in other domains and refine existing methodologies. This study holds significant importance for the development and practice of computer science.

Author profile

Zhenan Tu (1985-), Male, Miao nationality, Native place: Guizhou Province, Research direction: Computer Software.

References

- [1] Yan Zhou, Xuerui Wang. Research on application and development trend of computer software technology[J]. Energy education science and technology, 2015, 33(4): 1785-1790.
- [2] . KU, R. FULCHER. Using Computer Software Packages to Enhance the Teaching in Engineering Management Science-Part 2: Programming Techniques[J]. Computer applications in engineering education, 2012, 20(1): p.114-123.
- [3] Catalanotto K.J. Computer software: federal policy for a critical technology[J]. IEEE Technology and Society Magazine, 1993, 12(4): P.7-12.
- [4] Nur Syazarin Natasha Abd Aziz, Salwani Mohd Daud, Rudzidatul Akmal Dziauddin, Mohamad Zulkefli Adam, Azizul Azizan. A Review on Computer Vision Technology for Monitoring Poultry Farm -- Application, Hardware, and Software[J]. Quality Control, Transactions, 2019, 9(1): 12431-12445.

- [5] Zhang Jinhui. Application analysis of layering technology in computer software [J]. Computer Programming Techniques and Maintenance, 2020, 000(12): 56-58.
- [6] Lu Jinrui. Application analysis of layered technology in computer software development [J]. Changjiang Information and Communication, 2019, 36(1): 108-110.