A CASE STUDY OF SOFTWARE ARCHITECTURES IN PRESENT TIMES FOR A CHATBOT USE-CASE

Amit Singh[1] Dr. Sanjeev Kumar Sharma[2]

[1]School of Engineering & Information Technology, Sanskriti University, Mathura
[2]School of Engineering & Information Technology, Sanskriti University, Mathura
Received 17 November, 2025; Revised 11 January, 2026; Accepted 20 January, 2026
Available online 20 January, 2026 at www.atlas-journal.org, doi: 10.22545/2026/00281

Abstract: Software architecture (SA) for any system is vital, in which architects make decisions in designing an application system. This is to meet business requirements, and non-functional requirements (NFR) such as performance, scalability, reliability, maintenance and security. This article examines software architecture and design in current times as it evolved over the decades, including limitations and expectations. This article recommends best possible architecture which can cater to a simple chatbot system to a complex chatbot system.

Keywords: *Software architecture, Software design, layered architecture, Chatbot architecture, IaaS, PaaS, SaaS*

1. Introduction

Software architecture refers to the high-level design and structure of a software system. It involves making important decisions about how various components of a software application will interact with each other to achieve the desired functionality and performance. Software architecture provides a blueprint for the system's construction and helps ensure that it meets its intended goals and requirements.

Software architecture is typically documented using diagrams and other visual representations. This helps to communicate the system's design to other stakeholders, such as developers, testers, and users.

Here is an example of a simple software architecture for a web application:

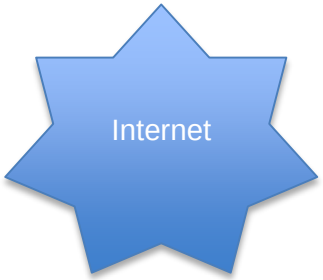


Figure 1: A typical Software Architecture

This architecture shows that the web application is made up of three main components: a web server, a client, and a database. The web server handles incoming requests from clients and returns responses. The client is the user-facing interface of the application, and it communicates with the web server to fetch and display data. The database stores the application's data.
The client sends HTTP requests to the web server, and the web server sends HTTP responses back to the client. The web server also communicates with the database to retrieve and store data.

The components in this architecture communicate with each other using HTTP requests and responses.
This is a simple example of a software architecture. More complex systems may have many more components and interactions.

2. Benefits of a software Architecture

Here are some of the benefits of having a well-designed software architecture:

1. **Improved software quality**: A good architecture can help to improve the overall quality of the software system by making it more reliable, scalable, and secure.
2. **Reduced Maintenance costs:** A good architecture can help to reduce the cost of developing and maintaining the software system by making it easier to understand and modify.
3. **Increased flexibility:** A good architecture can make the software system more flexible and adaptable to change. This is important because software requirements often change over time.
4. **Improved communication:** A good architecture can help to improve communication between stakeholders by providing a common understanding of the system's design. ?? how this helps in communication? Write about upstream and downstream the format and how they handshake.

Overall, software architecture is an important part of the software development process. By making careful architectural decisions, software architects can help to create high-quality software systems that meet the needs of users and businesses. This paper examines SA from Chatbot system perspective.

3. Key aspects of software architecture include:

Components: Identifying the major building blocks or components of the software system. These components can include user interfaces, databases, application logic, and external services.

Interactions: Defining how these components interact with each other, including the flow of data and control between them. This often involves specifying APIs (Application Programming Interfaces) and communication protocols.

Quality Attributes: Considering non-functional requirements such as scalability, performance, security, reliability, and maintainability. These attributes are essential for ensuring that the software system meets its operational and performance goals.

Patterns and Styles: Utilizing established design patterns and architectural styles (e.g., client-server, microservices, layered architecture) to solve common problems and improve the maintainability of the system.

Trade-offs: Making decisions that involve trade-offs, such as choosing between performance and maintainability or flexibility and security.

Decomposition: Breaking down the system into smaller, manageable subsystems or modules to simplify development and testing.

Documentation: Creating documentation that describes the architecture, its rationale, and how it should be implemented. This documentation is crucial for guiding development teams and ensuring consistency.

Evolution: Recognizing that software architecture is not static and may need to evolve over time to accommodate changing requirements or technologies.

The software architecture serves as a foundation for the development process. It helps ensure that the various teams working on a project have a clear understanding of how their components fit into the larger system and how they should interact with other parts of the system. A well-designed architecture can lead to more maintainable, scalable, and robust software systems.

Software architecture (SA) is a set of structures needed to reason about a software system. The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment. We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In **Architecture**, nonfunctional decisions are cast and separated by the functional requirements. In **Design**, functional requirements are accomplished.

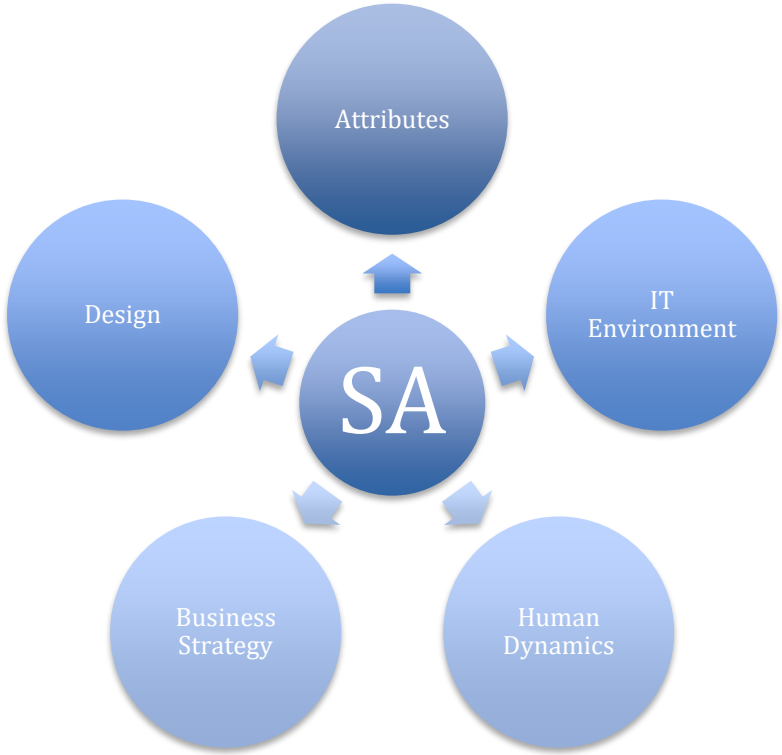


Figure 2: Key aspects of software architecture

4. Software Design

Software design provides a **design plan** that describes the elements of a system, how they fit, and work together to fulfill the requirement of the system. The objectives of having a design plan are as follows −

1. To negotiate system requirements, and to set expectations with customers, marketing, and management personnel.
2. Act as a blueprint during the development process.
3. Guide the implementation tasks, including detailed design, coding, integration, and testing.

It comes before the detailed design, coding, integration, and testing and after the domain analysis, requirements analysis, and risk analysis.

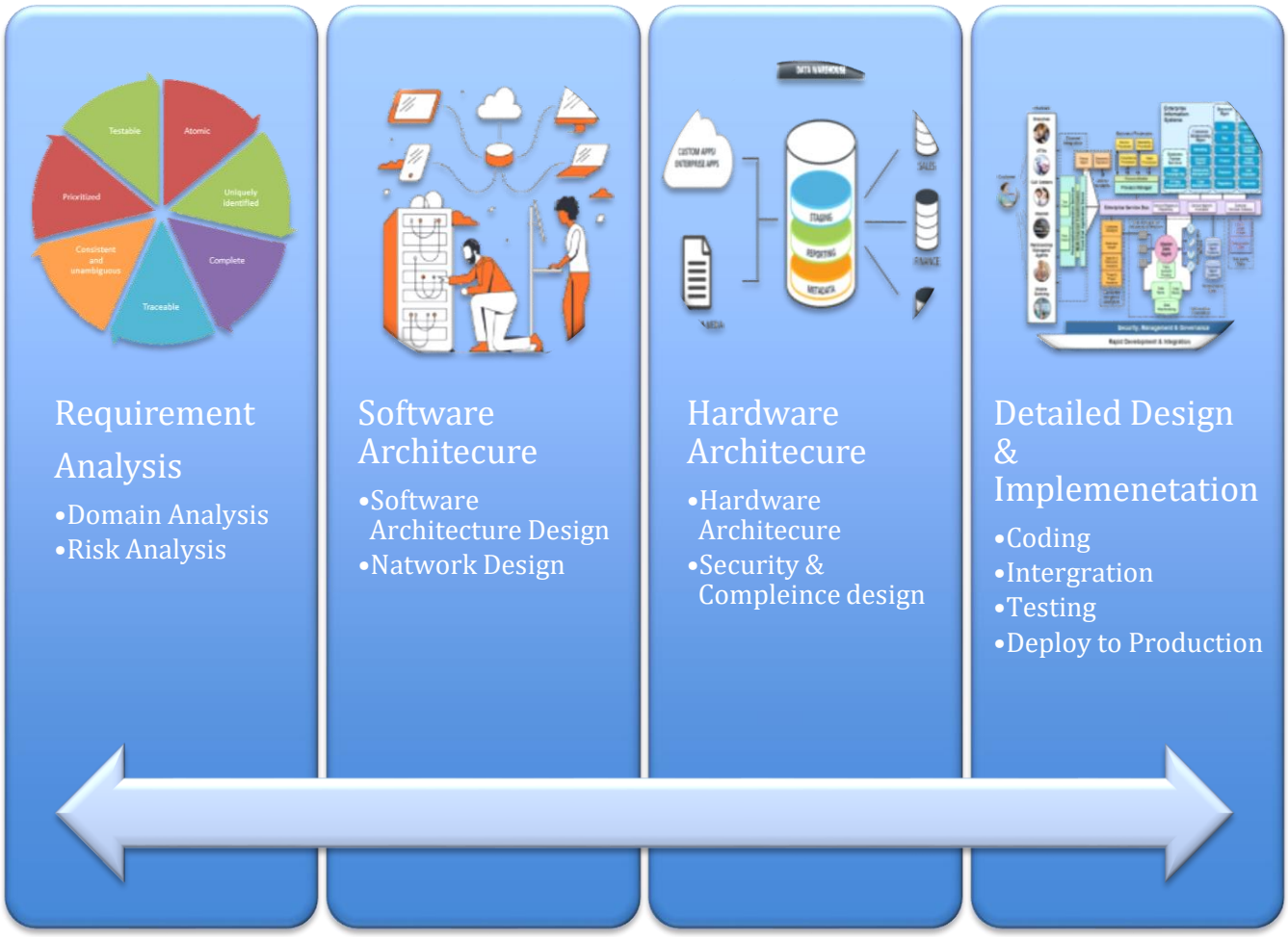


Figure 1 : Architecture Design with Major Phases

5. Architecture Design with Major Phases

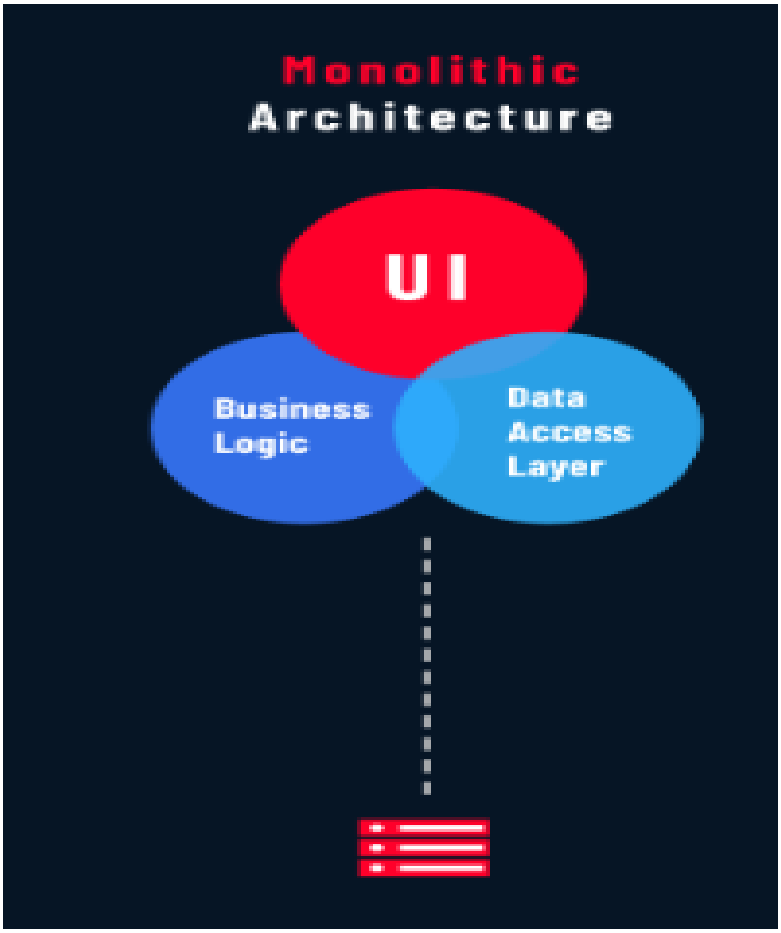
Software design is the process of creating a blueprint for a software system. It involves defining the system's architecture, components, and interfaces. Software design is important because it helps to ensure that the system is well-structured, efficient, and maintainable.

The software design process typically involves the following steps:

1. **Requirements analysis:** This step involves understanding the needs of the users and the business. The software designer will work with stakeholders to gather and document the system requirements. [9]
2. **Architectural design:** This step involves designing the high-level structure of the system. The software designer will decide what components the system will be made up of and how they will interact with each other.
3. **Detailed design:** This step involves designing the individual components of the system in more detail. The software designer will define the components' interfaces, data structures, and algorithms.

4. **Implementation:** This step involves writing the code for the system. The software designer will work with developers to implement the design.
5. **Testing:** This step involves testing the system to ensure that it meets the requirements and works as expected. The software designer will work with testers to identify and fix any defects.

Software design is a complex and challenging task, but it is essential for creating high-quality software systems. By following the steps above, software designers can create systems that are efficient, maintainable, and meet the needs of the users and the business. Overall, software design is an important part of the software development process. By following sound design principles and practices, software designers can help to create high-quality software systems that meet the needs of users and businesses.



6. Popular type Software Architecture in Software Development:
There are various types of software architecture, and the choice of architecture depends on the specific requirements and goals of a software project. Here is an overview of some common software architectural styles and approaches that are widely used at this time. Please note that software development evolves rapidly, and new architectural styles may have emerged at rapid rate. Here are some common types of software architecture:

Monolithic Architecture: In a monolithic architecture, the entire software application is developed as a single, self-contained unit. All components and functions are tightly integrated. While this architecture is straightforward, it can become challenging to maintain and scale as the application grows.
Chatbot system use case: Using this, Chatbot architecture can be used to package the entire application in in a single unit. This means that all of the application's code, libraries, and resources are packaged together into a single executable or deployable. Such system can be difficult to scale and maintain as the application grows. Other use cases could be websites, mobile apps, Internal business applications etc. Monolithic architecture is a good choice for applications that are relatively small and simple. It is also a good choice for applications that need to be highly performant or cost-effective.
Overall, monolithic architecture is a good choice for applications that are relatively small and simple.

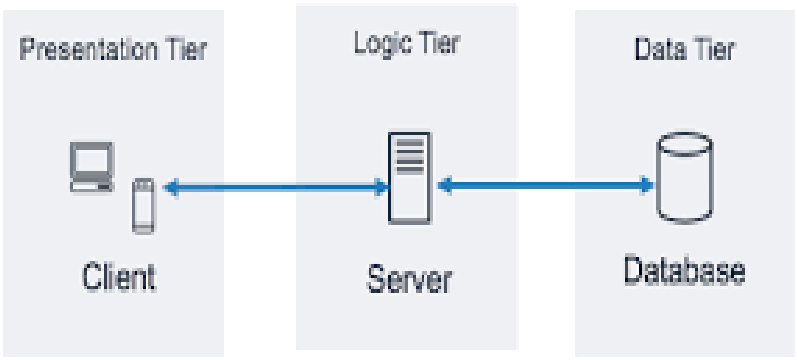


Figure 4: Client-Server Architecture

Client-Server Architecture: This architecture divides the software into two major parts: the client, which handles the user interface and presentation logic, and the server, which manages the application's core functionality and data. Communication between the client and server is typically done over a network. [2][3]
Chatbot system use case: This type of architecture suits to most of the application systems including chatbots. Chatbot system utilize an integrated UI on website while much of processing is delegated to backend system, which is often a server.

Event-Driven Architecture (EDA): EDA is an architectural style where components communicate by emitting and consuming events. This approach is often used in systems that need to handle asynchronous and real-time data processing.

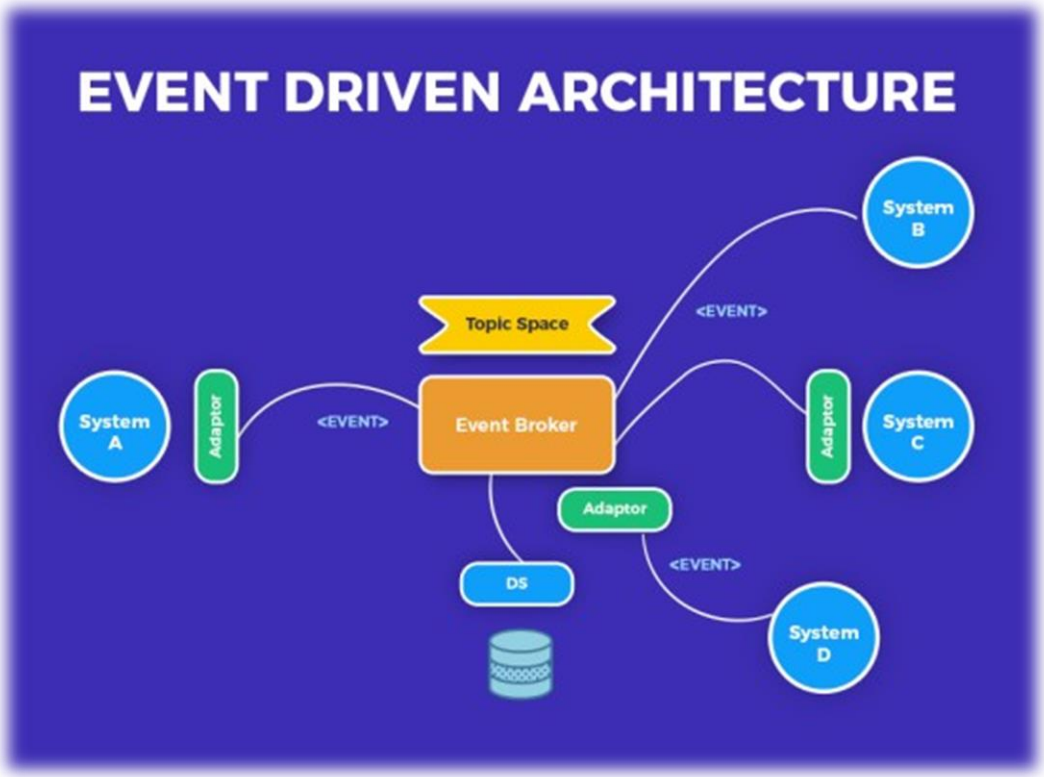


Figure 5: Event-Driven Architecture (EDA)

This architecture is based on the concept of events. When an event occurs, the system responds by executing a set of predefined actions. For example, an event-driven architecture might be used to implement a real-time chat application.
Chatbot system use case: If we design a dedicated and advanced chatbot system, this type of architecture can be utilized by decoupling the different components and enabling them to communicate and react to events independently. Integrated chatbot dialog can wait for user input and each input can trigger a series of processes and present user with an answer.

Service-Oriented Architecture (SOA): SOA is an architectural style that focuses on organizing software functionality as a collection of services that can be reused across various applications. Services in SOA are typically designed to be loosely coupled and communicate through standardized protocols.
SOA is employed in scenarios such as enterprise integration, microservices, web services, and legacy system modernization. It supports B2B integration, composite applications, scalability, load balancing, and resource optimization.
Chabot system use case: Chatbot system using SOA optimizes resource allocation, promotes adaptability, and strengthens security via loosely coupled services, making it suitable for dynamic and complex environments.

Layered Architecture: In a layered architecture, software is organized into distinct layers, each responsible for a specific aspect of the application (e.g., presentation, business logic, data access). This separation helps with modularity and maintainability.
This architecture separates the system into two parts: a client and a server. The client is responsible for interacting with the user and displaying the user interface. The server is responsible for processing user requests and storing data.
This architecture divides the system into layers, with each layer having a specific responsibility. For example, a web application might have a presentation layer (responsible for the user interface), a business logic layer (responsible for processing user requests), and a data access layer (responsible for interacting with the database).
Chatbot system use case: A prevalent Chatbot application of a layered architecture is in the development of web-based systems. In this model, the Chatbot system can be designed to be divided into distinct tiers, each with specific responsibilities. The presentation layer manages the user interface and its visual presentation, the business logic layer governs the core functionality, and the data layer is responsible for database operations. This separation simplifies the development process, enhances system maintainability, and supports scalability

Component-Based Architecture: Component-based architecture involves building an application from reusable software components or modules. These components can be combined to create more complex systems.
Chatbot system use case: This SA can be used in complex and largescale Chatbot system which each component designed with much of independence. These components would seamlessly interact, allowing for the replacement or enhancement of one component without necessitating a full software rebuild. Like buttons on Chat window or fine tune response; this enables to construct more extensive applications, resulting in efficiency gains in both development and upkeep.

N-Tier Architecture: N-tier architecture divides the application into multiple tiers or layers, often including presentation, application logic, and data storage. It is an extension of the layered architecture and can provide better scalability and maintainability.
Chatbot system use case: N-tier architecture simplifies Chatbot web app development and maintenance, akin to assembling a sandwich with layers, each serving a specific function

Container-Based Architecture: Containerization technologies like Docker have given rise to container-based architectures, where applications are packaged in containers for easy deployment and management. It enhances scalability by adjusting the number of containers, optimizes resource utilization, and expedites software development through swift testing and deployment.
Chatbot system use case: In designing Chatbot SA, this method involves encapsulating applications, along with all required components, in isolated containers. It offers consistency, ensuring uniform operation, and enables easy movement across diverse environments.

Microservices Architecture: Microservices architecture breaks down an application into a collection of loosely coupled, independently deployable services. Each service is responsible for a specific business capability and communicates with others through APIs. Microservices are known for their scalability and flexibility.
This architecture breaks down the system into a collection of small, independent services. Each service has a single responsibility and can be deployed and scaled independently. Microservices architectures are often used to build large, complex systems.
Chatbot system use case: This type of Chatbot system would be a modular, compartmentalized approach to software development, akin to constructing a system from distinct, specialized components. Each "microservice" serves a unique role, such as user authentication or response processing, while operating autonomously.

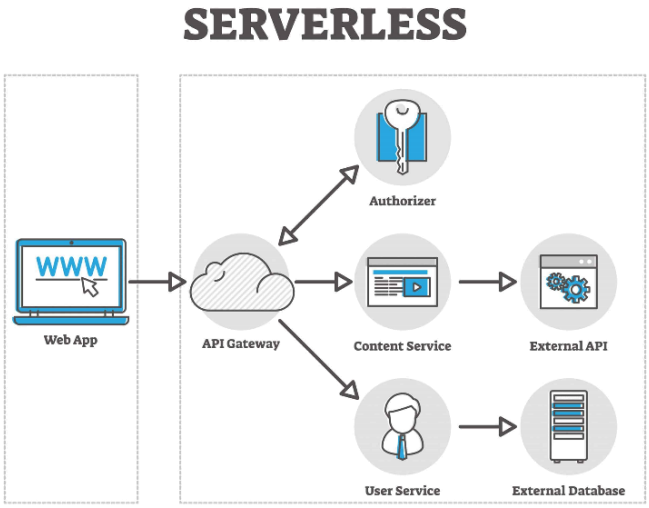
Event-Driven Microservices: This combines the principles of microservices with event-driven architecture. Each microservice communicates through events, promoting decoupling and scalability.
Chatbot system use case: Event-driven Chatbots SA are similar to responsive, interconnected pieces in a dynamic system. They act in real-time, triggered by specific events, like a new order or a user login, without the need for constant requests.

Blockchain Architecture: In blockchain-based systems, the architecture is centered around distributed ledgers and smart contracts, which ensure transparency and security in various applications.
Chatbot system use case: Chatbot SA using Blockchain architecture can serves as an immutable digital ledger across a decentralized network of nodes. Securely verifying identities, from passports to digital IDs, in a range of applications. This architecture guarantees data integrity and trustworthiness in these use cases.

Cloud Architecture: Cloud architecture is the framework that enables the delivery of scalable and flexible computing resources over the internet. It's a multi-layered framework that encompasses: Physical infrastructure, Virtualization, Resource orchestration.
Cloud architecture is a combination of both SOA (Service Oriented Architecture) and EDA (Event Driven Architecture).
The five key pillars that support effective cloud architecture are: Cost optimization, Reliability, Operational excellence, Performance efficiency, Security.

There are a number of different cloud architecture patterns, such as: Infrastructure as a Service (IaaS), Platform as a Service (PaaS), Software as a Service (SaaS) [8]
Chatbot system use case: This type of SA for Chatbot can bring lots of benefits in design and often shine in heavy demand like situations. This Chatbot SA offers scalability, flexibility, cost efficiency, global reach, collaboration, disaster recovery etc.

Serverless Architecture: Serverless architecture allows developers to build and run applications without managing servers. It relies on cloud providers to automatically scale and manage the underlying infrastructure.



/users
GET: Get all users
POST: Create a new user
DELETE: Delete a user
/users/:id
GET: Get a specific user
PUT: Update a specific user
DELETE: Delete a specific user

Figure 6: Server less SA

This architecture is a type of cloud computing architecture in which the cloud provider manages the server infrastructure. The developer only needs to provide the code for their application. Serverless architectures can be very cost-effective for applications that experience variable demand.
Chatbot system use case: Chatbot Serverless architecture would be essentially a cloud computing model, allows developers to focus on code, abstracting infrastructure management. This SA offers same benefits as Cloud SA and more such as microservices, IOT Integration etc.

7. RPA: Software Architecture using URIpath
Robotic process automation (RPA) is a software technology that allows businesses to automate repetitive tasks that are typically performed by humans. RPA bots can be programmed to interact with any application or system, and they can be used to automate a wide range of tasks[11]
RPA bots can be used to automate tasks in any industry, and they can be a valuable tool for businesses of all sizes.

Here are some of the benefits of using RPA:

1. Increased efficiency: RPA bots can automate tasks that are typically performed by humans, which can free up employees to focus on more strategic work.
2. Reduced errors: RPA bots are more likely to perform tasks accurately than humans, which can help to reduce errors and improve quality.
3. Improved compliance: RPA bots can be used to automate tasks that are required by regulations, which can help businesses to comply with all applicable laws and regulations.
4. Reduced costs: RPA can help businesses to reduce costs by automating tasks that would otherwise be performed by humans.

RPA is a powerful tool that can help businesses to improve their efficiency, reduce errors, improve compliance, and reduce costs. RPA is a rapidly growing field, and there are a number of different RPA tools available on the market. Businesses of all sizes can benefit from using RPA to automate their workflows. There are a number of different RPA tools available on the market. Some popular RPA tools include UiPath, Automation Anywhere, and Blue Prism. This paper will look into most popular tool called URIPath as an example on how this technology can be explored to create a chatbot system.

URIPath can be used to create software architectures in a number of ways. One common approach is to use URIPath to define the different resources that a system exposes. For example, a REST API might use URIPath to define resources such as /users, /posts, and /comments. Each of these resources would be mapped to a specific handler that is responsible for handling requests to that resource.

Another way to use URIPath in software architecture is to define the different layers of a system. For example, a layered architecture might use URIPath to define the different endpoints that are exposed by each layer. For example, the presentation layer might expose endpoints such as /home and /about, while the data access layer might expose endpoints such as /users/get and /posts/create.

Finally, URIPath can also be used to define the different components of a system and how they interact with each other. For example, a microservices architecture might use URIPath to define the different endpoints that each microservice exposes. For example, the user service might expose an endpoint such as /users/get, while the post service might expose an endpoint such as /posts/create.

Here is an example of how URIPath can be used to define the software architecture of a simple REST API:

In this example, the /users resource exposes endpoints for getting all users, creating a new user, and deleting a user. The /users/:id resource exposes endpoints for getting, updating, and deleting a specific user.

URIPath can be used to define the software architecture of systems of all sizes and complexity. It is a flexible and powerful tool that can be used to create architectures that are easy to understand, maintain, and scale.

Here are some additional benefits of using URIPath in software architecture:

1. Consistency: URIPath provides a consistent way to define the different resources, layers, and components of a system. This can make it easier to understand and maintain the system's architecture.
2. Flexibility: URIPath is a flexible tool that can be used to define a variety of different software architectures. This makes it a good choice for systems of all sizes and complexity.
3. Scalability: URIPath-based architectures are typically easy to scale. This is because URIPath provides a clear separation of concerns between the different parts of the system.

Overall, URIPath is a powerful tool that can be used to create software architectures that are consistent, flexible, and scalable.

Start Get user input Extract keywords from user input Search knowledge base for articles that match keywords If articles are found Summarize articles and send summary to user Else Send message to user indicating that no articles were found End
An example of a simple chatbot workflow that can be developed in URIPath

Chatbot Use case: RPA chatbots can be used to automate a wide range of tasks, and they can be a valuable tool for businesses of all sizes. RPA is a good option to consider. RPA chatbots are flexible, scalable, and cost-effective.
Design the chatbot's workflow as to how will the chatbot interact with users and how will it complete tasks. URIPath provides a visual editor that can be used to create the chatbot's workflow. RPI can be used to create more complex chatbot workflows using URIPath. URIPath provides a variety of activities and features that can be used to develop chatbots that can handle a wide range of tasks.
Once you have developed the chatbot's workflow, we need to train it to understand the different types of questions and requests that it will receive from users. Once the chatbot is trained, we can deploy it to production so that it can start interacting with users.
Keep in mind that the choice of *chatbot architecture* depends on factors like project requirements, scalability needs, team expertise, and available technology stack. New architectural patterns and styles may have emerged in recent times, so it's essential to stay up-to-date with current industry trends and best practices in software architecture.
In addition to these common architectures, there are many other specialized architectures that can be used for specific types of systems. For example, there are architectures for real-time systems, distributed systems, and embedded systems.
The best type of architecture for a particular system will depend on the specific requirements of that system. However, by understanding the different types of architectures that are available, software architects can choose the right architecture for the job.

8. Limitations
Software architecture is important for designing and building high-quality software systems, but it also has some limitations. Here are a few examples:
Complexity: Software architecture can be complex, especially for large and complex systems. This can make it difficult to understand and maintain the system over time.
Rigidity: A well-designed software architecture should be flexible enough to adapt to change, but it is important to strike a balance between flexibility and rigidity. Too much flexibility can lead to a poorly structured system, while too much rigidity can make it difficult to make changes.
Cost: Software architecture can be expensive, especially for large and complex systems. The cost of software architecture includes the cost of hiring and training qualified software architects, as well as the cost of developing and maintaining the system's architecture documentation.
Tools and standardization: There is a lack of standardized tools and methods for representing and analyzing software architectures. This can make it difficult to communicate and evaluate software architectures.

Despite these limitations, software architecture is an essential part of the software development process. By understanding the limitations of software architecture, software architects can make informed decisions about how to design and build high-quality software systems.

Here are some tips for mitigating the limitations of software architecture:

Use a modular approach: Break down the system into smaller, more manageable modules. This will make the system easier to understand and maintain.
Use well-known design patterns: Design patterns are reusable solutions to common software design problems. Using design patterns can help to reduce the complexity of the system's architecture.
Document the architecture: Document the system's architecture in a clear and concise way. This will help to communicate the architecture to other stakeholders and make it easier to maintain the system over time.
Use architecture analysis tools: There are a number of tools available to help software architects analyze their architectures. These tools can help to identify potential problems with the architecture and make recommendations for improvement.

By following these tips, software architects can mitigate the limitations of software architecture and build high-quality software systems that meet the needs of users and businesses.

9. Role of Software Architect
A Software Architect provides a solution that the technical team can create and design for the entire application. A software architect should have expertise in the following areas −
Software architects need a broad range of expertise in order to be successful. Some of the most important areas of expertise include:

Technical knowledge: Software architects need to have a deep understanding of software development technologies, such as programming languages, frameworks, and databases. They also need to be familiar with software design patterns and best practices.
Problem-solving skills: Software architects need to be able to identify and solve complex problems. They also need to be able to think creatively and come up with innovative solutions.
Communication skills: Software architects need to be able to communicate effectively with a variety of stakeholders, including developers, testers, and business users. They need to be able to explain complex technical concepts in a clear and concise way.
Leadership skills: Software architects often lead teams of developers. They need to be able to motivate and inspire their team members, as well as delegate tasks and manage conflict.

In addition to these core areas of expertise, software architects may also need to have expertise in specific areas, such as cloud computing, security, or big data. The specific expertise required will vary depending on the type of systems that the software architect is working on.

Here are some tips for developing the expertise needed to be a successful software architect:
Get a strong education in computer science: A degree in computer science will give you a foundation in the technical knowledge and problem-solving skills that you need to be a successful software architect.
Gain experience in software development: The best way to learn about software architecture is to gain experience in software development. This will give you a chance to learn about different technologies, design patterns, and best practices.
Get certified: There are a number of certifications available for software architects. Getting certified can help you to demonstrate your expertise and make you more marketable to potential employers.
Read books and articles about software architecture: There are a number of books and articles available about software architecture. Reading these resources can help you to learn more about the field and develop your expertise.
Attend meetups and conferences: Attending meetups and conferences is a great way to network with other software architects and learn from their experiences.

By developing the expertise needed to be a successful software architect, one can position oneself for a rewarding and challenging career.
From simple to complex, chatbot system can be made using these techniques and methodologies. Eventually SA will support business decision, asking what all features and complexities we want in designing such a system.

Authors' Contribution: Amit Singh: Conceptualization, Proposal Methodological, Research, Formal Analysis, Original Draft Writing. Dr. Sanjeev Kumar Sharma: Supervision, Validation, Drafting (Review and Editing).

Funding Statement: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Conflicts of Interest: The authors declare no conflict of interest.

Acknowledgments: The authors would like to express their gratitude to the School of Engineering & Information Technology, Sanskriti University, Mathura for the facilities provided and institutional support for the development of this research.

Copyright ©2026 by the author(s). This is an open access article distributed under the Creative Commons Attribution License (CC BY-NC International,https://creativecommons.org/licenses/by/4.0/), which allow others to share, make adaptations, tweak, and build upon your work non-commercially, provided the original work is properly cited. The authors can reuse their work commercially

References:

https://en.wikipedia.org/wiki/Software_architecture
https://en.wikipedia.org/wiki/Client%E2%80%93server_model
https://www.simplilearn.com/what-is-client-server-architecture-article
https://www.tutorialspoint.com/software_architecture_design/introduction.htm
https://www.geeksforgeeks.org/fundamentals-of-software-architecture/
https://blog.sparkfabrik.com/en/microservices-and-cloud-native-applications-vs-monolithic-applications/
https://www.linkedin.com/pulse/anti-patterns-event-driven-architecture-arpit-jain/
https://www.vmware.com/in/topics/glossary/content/cloud-architecture.html
https://reqtest.com/en/knowledgebase/requirements-analysis/
https://www.kofi-group.com/serverless-architecture-explained-in-10-minutes/
https://www.uipath.com/rpa/robotic-process-automation