COMSATS University Islamabad

Abbottabad

Assignment 2

OF

Software Design Architecture

By

Muhammad Ijlal ATIF **FA21-BSE-120**

Hammad Ayub **FA21-BSE-053**

Mastering Component-Based Architecture: Everything You Need to Know

Front-end software development is progressing and evolving to meet users' needs and become as flexible as possible. That's why many developers are adopting a component-based architecture. Learn more about this approach, its advantages, features, types, and examples.

What is Component-Based Architecture in Software Development?

Component-Based Architecture (CBA) is a design and development approach that focuses on turning elements into separate reusable components. For example, a mobile app has a search bar, a header, and images. In a component-based architecture, these components are independent. Companies like Uber, Spotify, and PayPal have utilized component-based architecture when building their systems. Recently, component-based architecture has been replacing monolithic architecture, which is too unwieldy and doesn't permit reuse because the elements are all together in one block.

The primary objective of component-based architecture is to ensure component reusability. A component encapsulates functionality and behaviors of a software element into a reusable and self-deployable binary unit.

What is a Component?

A component is a reusable element that provides extra functionalities while quickening an app's deployment and development. As flexible and modular as they are, they are reusable in several projects. You can use components consistently across multiple interfaces and modules without jeopardizing the security of your code or the user experience.

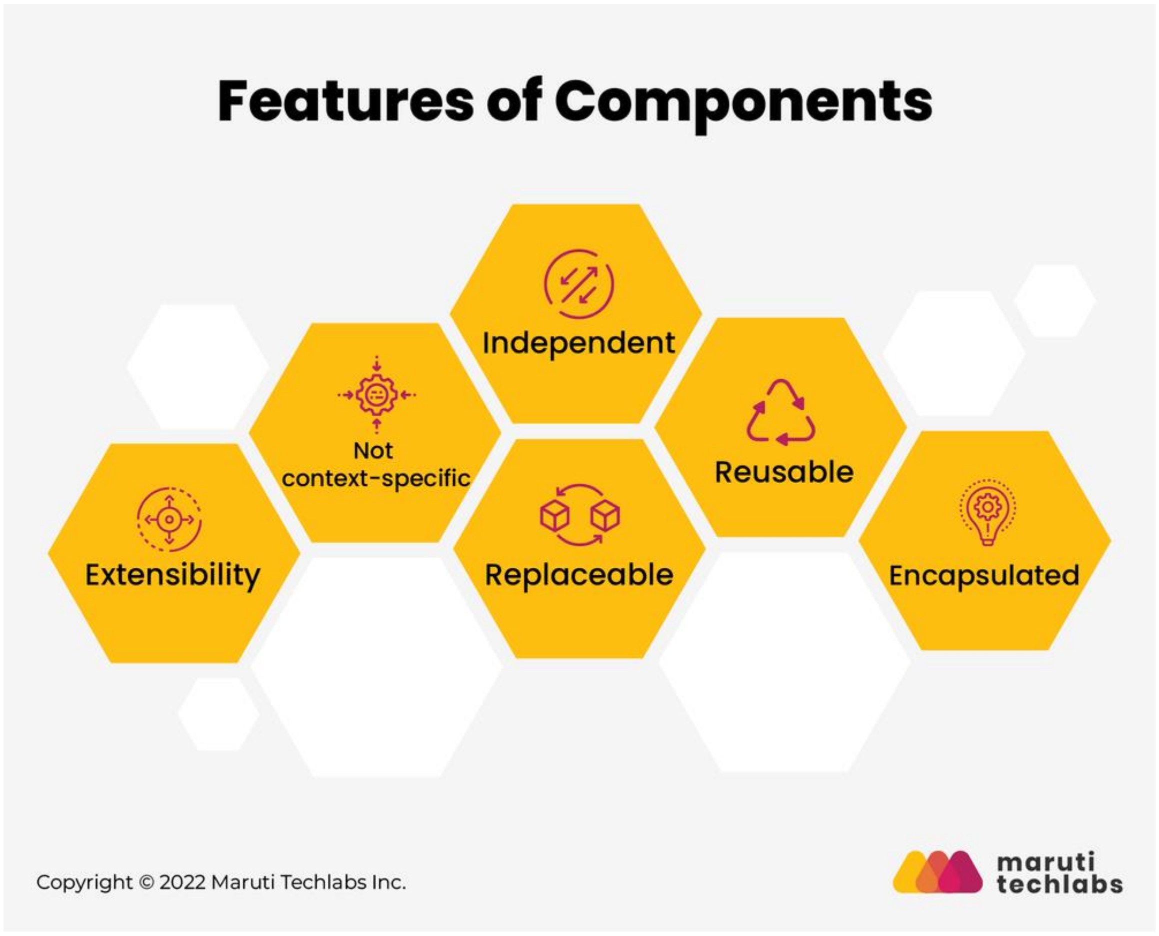
Why Do You Need Components?

Components are critical aspects of any front end technology; following the foundation of AJAX requests, calls to the server can be made directly from the client side to update the DOM and display content without causing a page refresh. A component's interface can request its business logic, updating its interface without forcing other component to refresh or modifying their UI, as components are independent.

It is ideal for tasks that might otherwise unnecessarily cause other components or the entire page to reload (which would be a drain on performance). Each component has specific features that can be overridden or isolated depending on how an application uses it.

Features of Components

- **Extendable:** Components can extend to work with other components to create new functions.
- **Replaceable**: Developers can easily replace components with other components.
- **Encapsulated:** Components appear as interfaces but don't show the internal processes.
- **Independent_context:** Components are independent and non-context-specific, making them suitable for various scenarios.
- **Reusable:** Businesses and developers can reuse components without modifying or adjusting them.
- **Independent:** Components have few dependencies on other components and may function in various situations and scenarios independently.
- **Not Context-Specific:** Components are built to work in various situations and scenarios. State data, for example, should be supplied to the component rather than being contained in or retrieved



Advantages of Component-Based Architecture

1. **Faster Development**: The approach helps teams build applications faster by 60% compared to other methodologies. Developers can pick components directly from libraries without worrying about these components affecting security, usability, or performance. In a monolithic system, they would build the elements from scratch.
2. **Simpler Maintenance:** Maintenance is simpler with this approach. Instead of writing code to adjust single components, you update one component only once and then add it to the model when releasing a software update.
3. **Autonomous Teams:** Teams can work autonomously and independently, creating components without external assistance or interference. This allows for a smooth workflow with freedom, flexibility, and accountability.
4. **Re-usability**: The approach offers re-usability, which has numerous benefits. Instead of spending time writing the same code lines repeatedly, developers can focus on core functionality. Moreover, the components can be applied in various ways for different purposes on multiple platforms.
5. **Consistent UI**: All the components created with the same design document would have a consistent UI.

6. **Scalability:** Scalability is crucial for any new app. Using component-based architecture, scaling elements is like putting together puzzle pieces that work together seamlessly.
7. **Robust and Complex Apps**: This architecture facilitates the creation of robust, complex apps. Developers can add many blocks and components that have already been tested without needing to write many lines of code, leaving room for error.

Drawbacks of Component-Based Architecture

1. **Integration Complexity:** Integrating multiple components can be complex and may require significant effort to ensure they work together seamlessly.
2. **Customization Limits**: Reusing components in different applications makes them less customizable.
3. **High Maintenance:** The component-based approach can be high-maintenance. Finding a component to meet an application's needs can be challenging. Additionally, maintaining component libraries requires frequent monitoring.
4. **Readability Issues**: Using too many components in an app or website can compromise readability, making it hard for the reader to understand the text.

Views of a Component

A component can have three different views − object-oriented view, conventional view, and process-related view.

Object-oriented view

A component is viewed as a set of one or more cooperating classes. Each problem domain class (analysis) and infrastructure class (design) are explained to identify all attributes and operations that apply to its implementation. It also involves defining the interfaces that enable classes to communicate and cooperate.

Conventional view

It is viewed as a functional element or a module of a program that integrates the processing logic, the internal data structures that are required to implement the processing logic and an interface that enables the component to be invoked and data to be passed to it.

Process-related view

In this view, instead of creating each component from scratch, the system is building from existing components maintained in a library. As the software architecture is formulated, components are selected from the library and used to populate the architecture.

Example of Component-Based Architecture

Consider an e-commerce application that consists of the following components:

- **User Interface Component:** Handles the presentation logic and user interactions.
- Product Catalog Component: Manages product information, including listing, searching, and filtering products.
- **Shopping Cart Component:** Manages the user's shopping cart, including adding, removing, and updating items.
- **Order Processing Componen**t: Handles order placement, payment processing, and order tracking.
- **User Management Component:** Manages user accounts, authentication, and authorization.

Advantages in this Example:

- **Reusability:** The User Interface Component can be reused across different applications with minor modifications.
- **Maintainability**: If a new payment gateway needs to be integrated, only the Order Processing Component needs to be updated.
- **Scalability**: If the product catalog grows, the Product Catalog Component can be scaled independently without affecting other components.

Disadvantages in this Example:

- **Integration Complexity:** Integrating the Shopping Cart Component with the Order Processing Component requires careful handling of data and interactions.
- **Performance Overhead:** Communication between the User Interface Component and the Product Catalog Component may introduce latency, especially if they are deployed on different servers.

Conclusion

Component-Based Architecture offers numerous benefits, especially in terms of reusability, maintainability, and scalability. However, it also presents challenges, particularly in integration and performance. Careful planning and design are essential to maximize the advantages and minimize the disadvantages of this architectural approach.

REFREENCE

MarutiTechlabs.com

tutorialspoint

Medium

nandbox.com

Chatgpt