12 DESIGN CONCEPTS

What is meant by design classes? List and explain four characteristics of a well-formed design class.

In the context of software design, transitioning from analysis to implementation involves creating design classes that refine and expand upon analysis classes. These design classes provide the detailed technical specifications necessary to build a software system. Here are the key points regarding design classes:

Types of Design Classes

Design classes fall into five different categories, each addressing a specific aspect of the system architecture:

1. **User Interface Classes**: These classes manage all the abstractions necessary for human-computer interaction, often implementing the interface within a specific metaphor or design language.
2. **Business Domain Classes**: These classes define the attributes and services (methods) needed to implement elements of the business domain identified by the analysis classes.
3. **Process Classes**: These classes implement lower-level business abstractions required to manage business domain classes effectively.
4. **Persistent Classes**: These represent data stores (such as databases) that need to persist beyond the software's execution.
5. **System Classes**: These classes handle software management and control functions, enabling the system to operate and communicate both internally and with the external environment.

Transition from Analysis to Design

As you transition from analysis to design:

- **Analysis Classes**: These classes describe the problem domain at a high level of abstraction, focusing on user-visible aspects.
- **Design Classes**: These classes provide detailed design information, guiding the implementation and constructing a software infrastructure that supports the business solution.

Characteristics of Well-Formed Design Classes

To ensure design classes are effective and efficient, Arlow and Neustadt suggest they should be:

1. **Complete and Sufficient**: The class should encapsulate all expected attributes and methods, ensuring it fully represents the intended functionality without superfluous features.
2. **Primitive**: Each method should accomplish one specific service for the class, with no redundant ways to achieve the same outcome.
3. **Highly Cohesive**: The class should have a focused set of responsibilities, with methods and attributes dedicated to achieving those responsibilities.
4. **Low Coupling**: Collaboration between classes should be minimal to reduce complexity, following the Law of Demeter, which advises that a method should only interact with methods in directly related (neighbouring) classes.

Benefits of Well-Formed Design Classes

- **Ease of Implementation**: Clear and focused classes make the development process more straightforward.
- **Testability**: Low coupling and high cohesion facilitate easier testing.
- **Maintainability**: Systems designed with well-formed classes are easier to maintain and extend over time.

Basic Design Principles

Four basic design principles are applicable to component-level design and are widely adopted in object-oriented software engineering. These principles help create designs that are more amenable to change and reduce the propagation of side effects when changes occur. Use these principles as a guide while developing each software component.

1. The Open-Closed Principle (OCP)

- **Definition:** "A module [component] should be open for extension but closed for modification." [Mar00]
- **Explanation:** This principle seems contradictory but is crucial for good component-level design. Simply put, a component should be designed in a way that allows it to be extended (within its functional domain) without requiring modifications to its internal code or logic. To achieve this,

create abstractions that serve as a buffer between the functionality likely to be extended and the design class itself.

- **Example:** Consider a Detector class in the SafeHome security function that needs to check various security sensor statuses. If internal processing logic is based on a sequence of if-then-else constructs for different sensor types, adding a new sensor type will require modifying this logic, violating OCP. Instead, use a consistent sensor interface, so adding a new sensor type does not necessitate changes to the Detector class.

2. The Liskov Substitution Principle (LSP)

- **Definition:** "Subclasses should be substitutable for their base classes." [Mar00]
- **Explanation:** A component using a base class should function correctly if a subclass of that base class is passed instead. Subclasses must honor the "contract" of the base class, which includes preconditions (conditions true before use) and postconditions (conditions true after use). Ensure that derived classes conform to these conditions.

3. Dependency Inversion Principle (DIP)

- **Definition:** "Depend on abstractions. Do not depend on concretions." [Mar00]
- **Explanation:** Design components to depend on abstractions (such as interfaces) rather than on specific concrete components. This approach facilitates easier extension and reduces complexity.

4. The Interface Segregation Principle (ISP)

- **Definition:** "Many client-specific interfaces are better than one general-purpose interface." [Mar00]
- **Explanation:** Create specialized interfaces for each major category of clients, specifying only the operations relevant to each category. If multiple clients require the same operations, include them in each specialized interface.
- **Example:** For the FloorPlan class in SafeHome, separate interfaces for security and surveillance functions could be used. The security interface would include operations like placeDevice(), showDevice(), groupDevice(), and removeDevice(), while the surveillance interface would include these plus additional operations like showFOV() and showDeviceID().

Packaging Principles

Components are often organized into subsystems or packages. Martin [Mar00] suggests the following packaging principles:

1. The Release Reuse Equivalency Principle (REP)

- **Definition:** "The granule of reuse is the granule of release." [Mar00]
- **Explanation:** When designing for reuse, establish a release control system to manage older versions of reusable components while users upgrade to the latest version. Group reusable classes into packages to facilitate this management.

2. The Common Closure Principle (CCP)

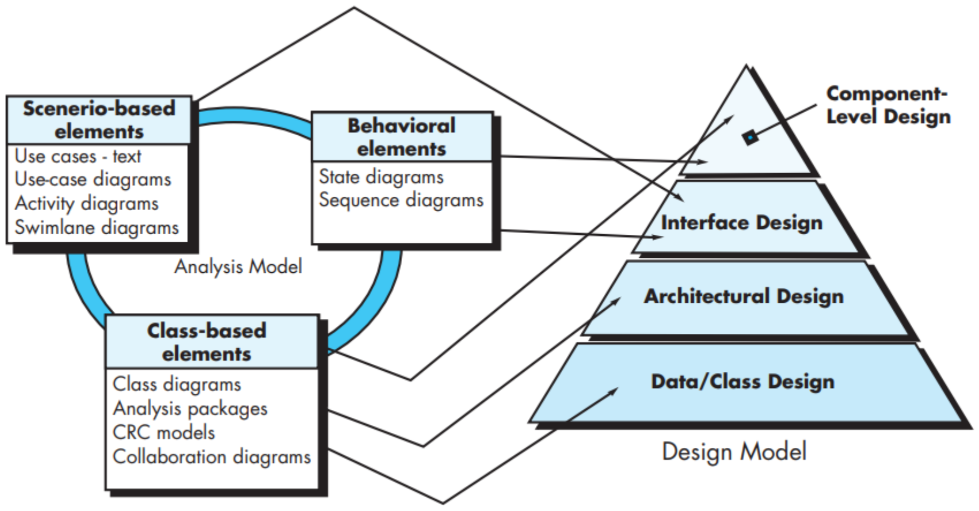
- **Definition:** "Classes that change together belong together." [Mar00]
- **Explanation:** Package classes that are related by function or behavior together. When a characteristic of a package needs to change, only those classes within the package should require modification, leading to more effective change control.

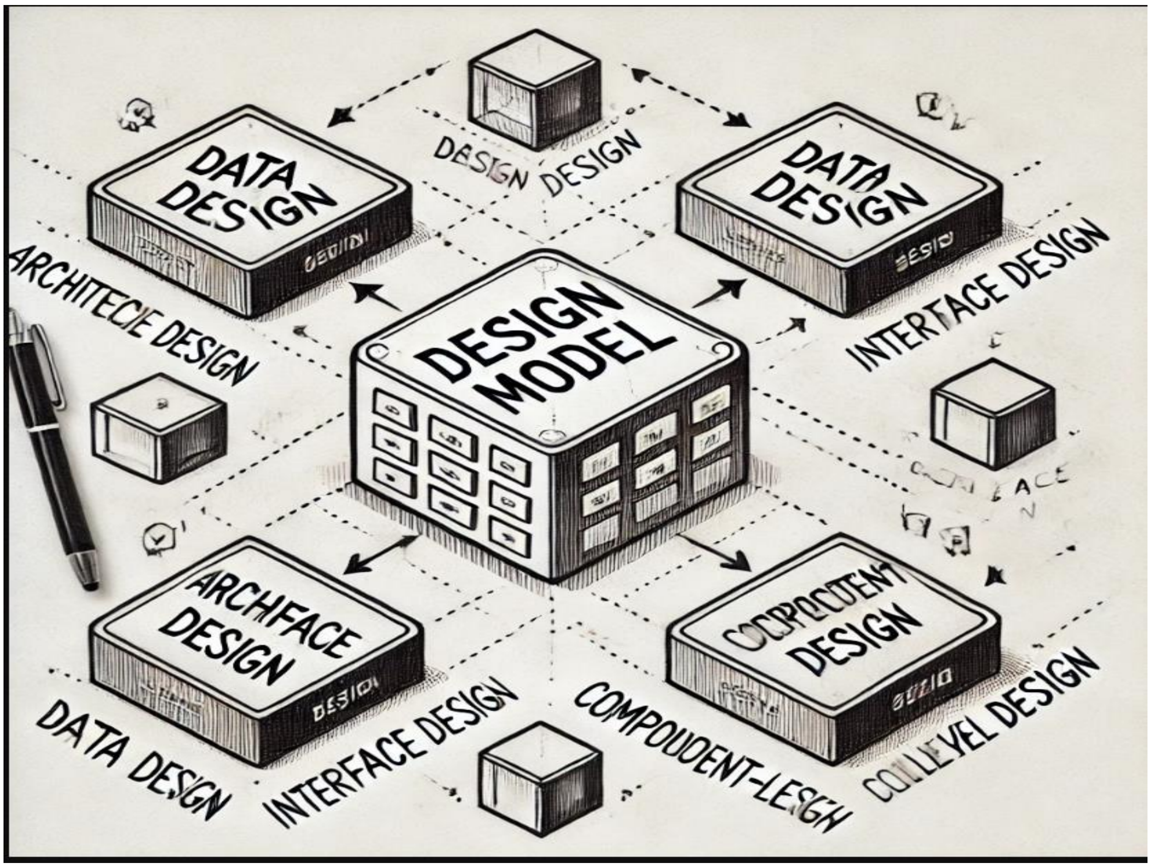
3. The Common Reuse Principle (CRP)

- **Definition:** "Classes that aren't reused together should not be grouped together." [Mar00]
- **Explanation:** If a package changes, its release number changes, and all dependent packages must update and be tested. Group classes that are reused together in the same package to avoid unnecessary integration and testing.

Illustrate dimensions of the design model with a neat sketch:

FIGURE 12.1 Translating the requirements model into the design model





Central Component: Design Model

- **Design Model**: This is the core of the design process, encompassing all the aspects and dimensions of the system's design. It acts as a blueprint for the system, guiding the development and ensuring all parts work cohesively.

Surrounding Dimensions

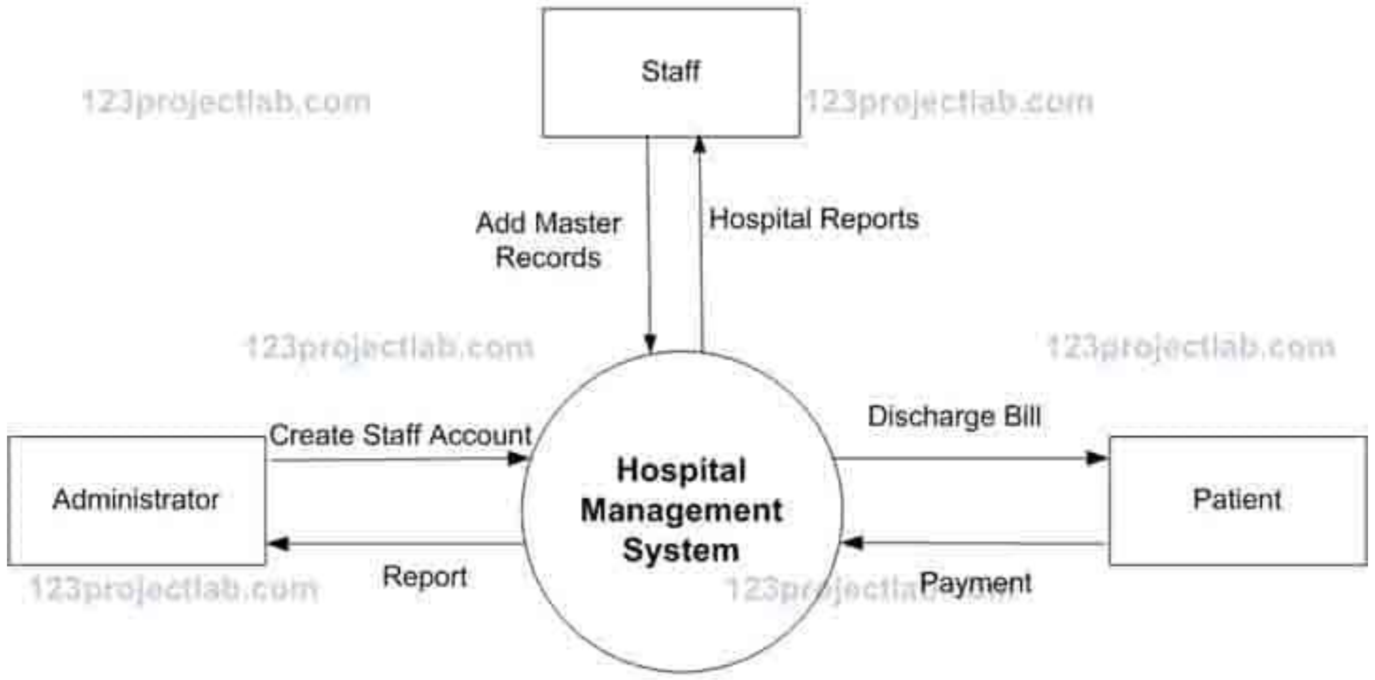
1. **Data Design**
   - **Purpose**: Focuses on the organization, structure, and storage of data within the system. This involves defining data structures, databases, and data flows.
   - **Components**: Includes data models, database schemas, and data storage mechanisms.
   - **Relationship**: Data design provides the foundation for how data is managed and accessed, influencing the architectural and component-level design.
2. **Architectural Design**
   - **Purpose**: Defines the overall structure and high-level framework of the system. This includes the arrangement of components and how they interact.
   - **Components**: Consists of modules, subsystems, and their interactions, as well as design patterns and architectural styles (e.g., MVC, microservices).

  - **Relationship**: Architectural design sets the blueprint for the system's structure, impacting interface and component-level design by dictating how components are organized and interact.

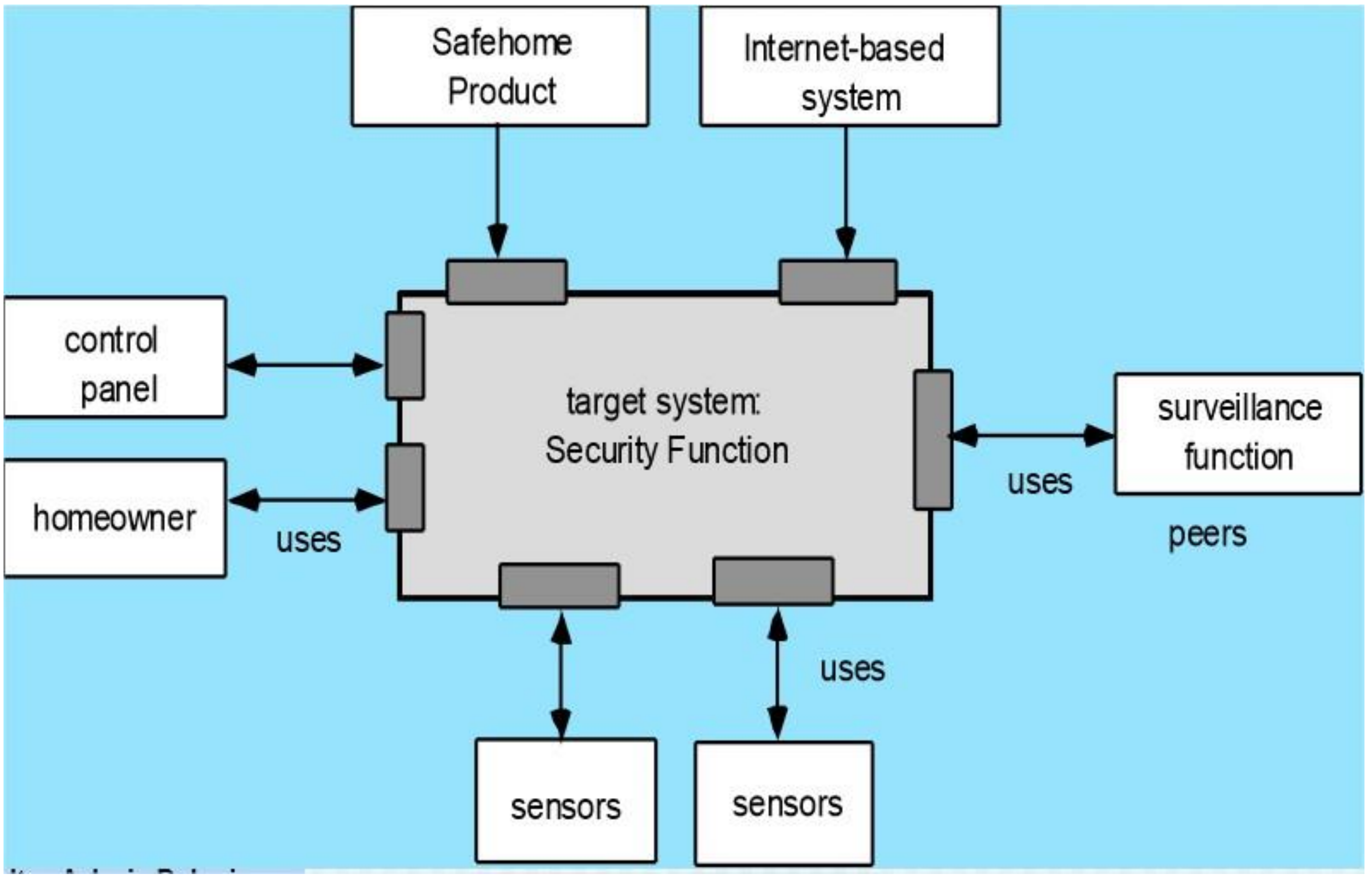
3. **Interface Design**
   - **Purpose**: Focuses on how users and other systems interact with the software. This includes the user interface (UI) and application programming interfaces (APIs).
   - **Components**: Encompasses UI elements, user experience (UX) considerations, and API definitions.
   - **Relationship**: Interface design ensures usability and accessibility, influencing component-level design by defining how components present information and handle interactions.
4. **Component-Level Design**
   - **Purpose**: Involves the detailed design of individual components, specifying their functionality, interfaces, and interactions.
   - **Components**: Includes detailed specifications for classes, methods, data structures, and algorithms.
   - **Relationship**: Component-level design is informed by data design, architectural design, and interface design, ensuring that each component functions correctly within the overall system.
5. **Deployment-Level Design**
   - **Deployment Models:** Shows how software components are allocated across physical computing environments (e.g., servers, PCs, mobile platforms).
   - **Descriptor Form:** Initially, deployment diagrams may be presented in a general form, describing the computing environments and subsystems.
   - **Instance Form:** Later, deployment diagrams are refined to include specific hardware configurations and deployment details.

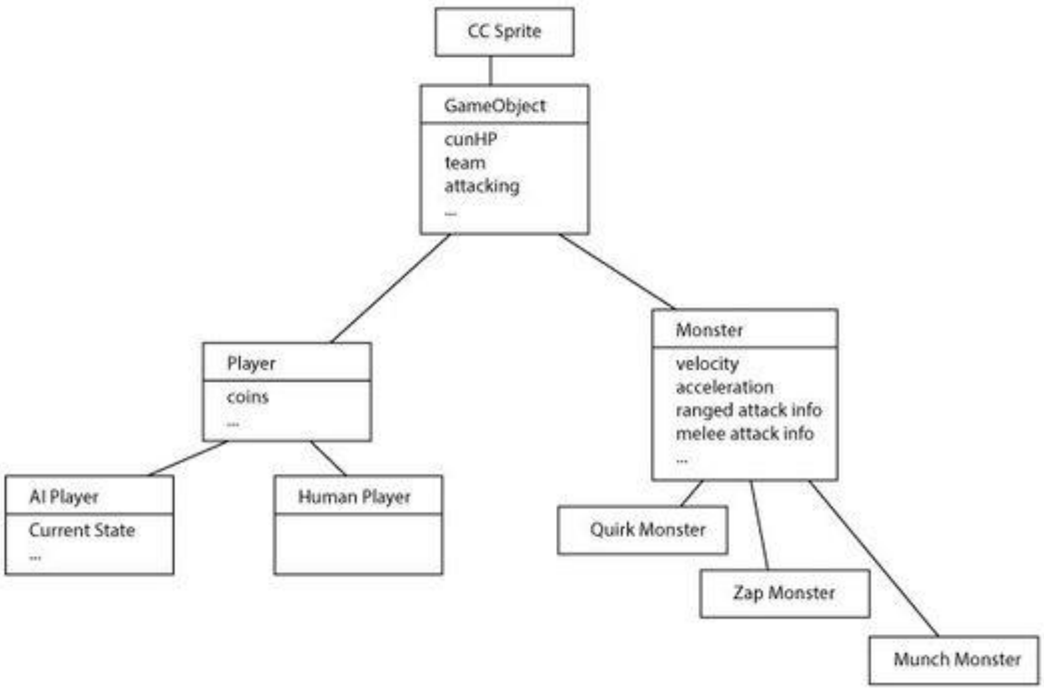
Relationships Between Dimensions

- **Data Design ↔ Architectural Design**: The way data is structured influences the system's architecture, and the architecture dictates how data is managed and accessed.
- **Architectural Design ↔ Interface Design**: The system's architecture affects how interfaces are designed, ensuring that user interactions and API calls are properly handled.
- **Interface Design ↔ Component-Level Design**: Interfaces dictate how individual components interact with users and other systems, guiding the detailed design of components.
- **Component-Level Design ↔ Data Design**: The design of individual components must consider how they handle and process data, aligning with the overall data design.



Context Level DFD for Hospital Management System


Safehome Product
Internet-based system
control panel
homeowner
uses
target system: Security Function
uses
surveillance function
peers
uses
sensors
sensors


CC Sprite
GameObject
cunHP
team
attacking
...
Player
coins
...
Monster
velocity
acceleration
ranged attack info
melee attack info
...
AI Player
Current State
...
Human Player
Quirk Monster
Zap Monster
Munch Monster

Design architectural context diagram for the following systems:

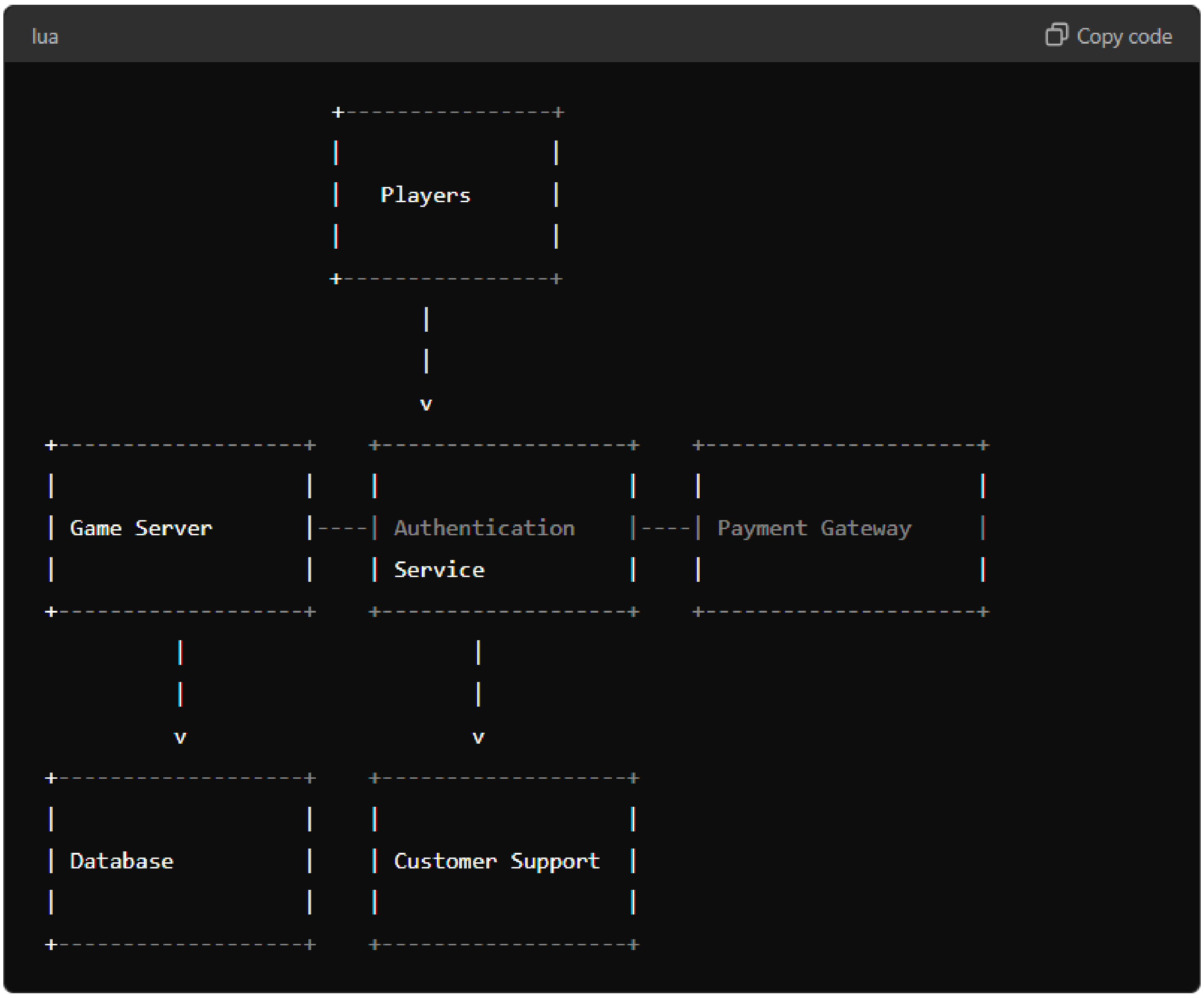
(i) Online Gaming System

(ii) Agricultural Products Management System.

(i) Online Gaming System

- **Players:** Users who participate in online games.
- **Game Server:** Central server managing game logic, player interactions, and real-time gameplay.
- **Database:** Stores player profiles, game data, scores, and other relevant information.
- **Payment Gateway:** Manages transactions for in-game purchases.
- **Authentication Service:** Handles user login, registration, and security.
- **Customer Support:** Provides assistance to players for issues or inquiries.

Architectural Context Diagram:

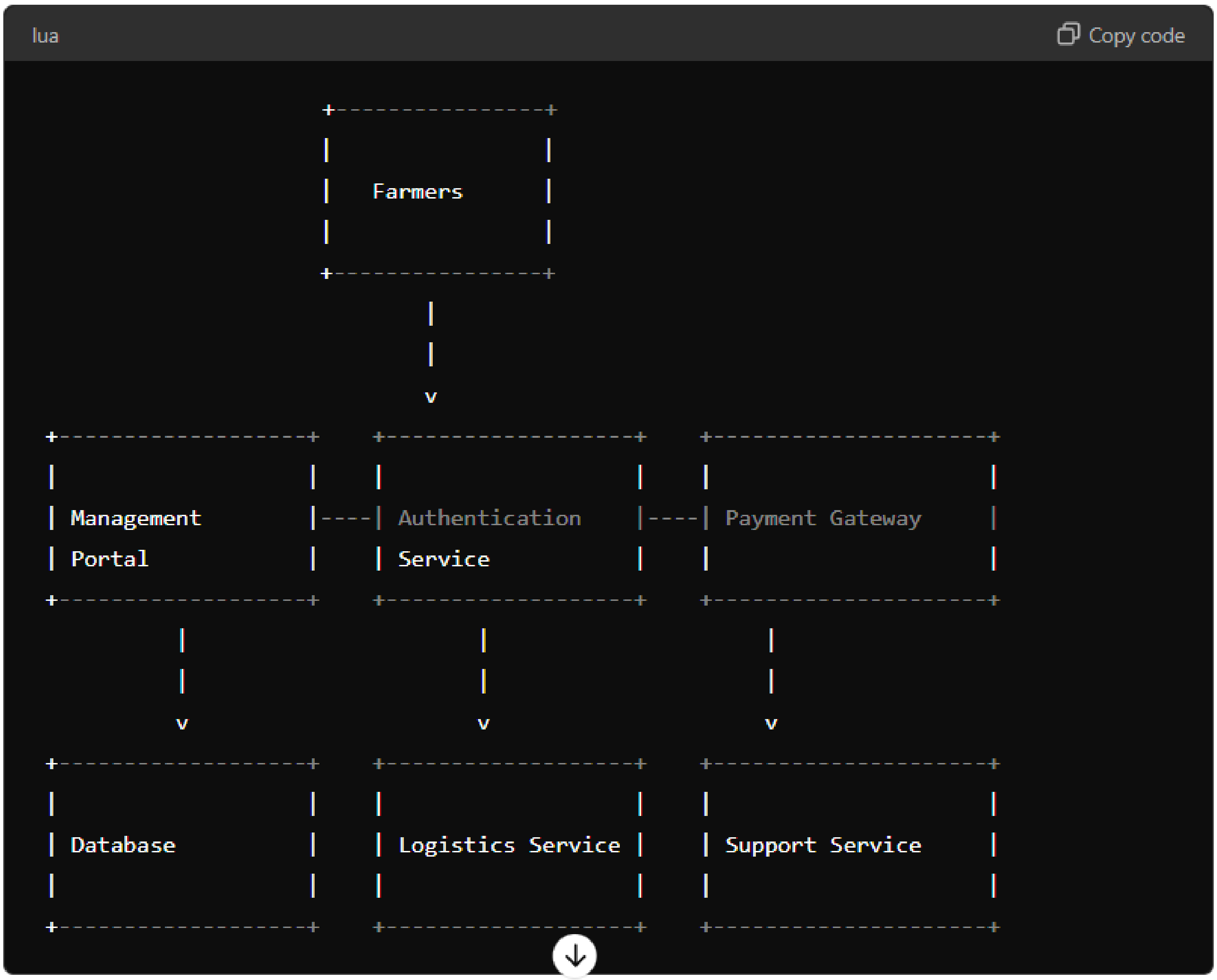


```
               +---------------+
               |               |
               |   Players     |
               |               |
               +---------------+
                     |
                     |
                     v
+------------------+    +------------------+    +---------------------+
|                  |    |                  |    |                     |
| Game Server      |----| Authentication   |----| Payment Gateway     |
|                  |    | Service          |    |                     |
+------------------+    +------------------+    +---------------------+
         |                       |
         |                       |
         v                       v
+------------------+    +------------------+
|                  |    |                  |
| Database         |    | Customer Support |
|                  |    |                  |
+------------------+    +------------------+
```

(ii) Agricultural Products Management System

- **Farmers:** Producers who list agricultural products.
- **Buyers:** Consumers or businesses purchasing agricultural products.
- **Management Portal:** Interface for managing product listings, orders, and inventory.
- **Database:** Stores product details, farmer profiles, buyer profiles, order history, and inventory.
- **Payment Gateway:** Manages transactions between buyers and farmers.
- **Logistics Service:** Handles the delivery and tracking of agricultural products.
- **Support Service:** Provides assistance for both farmers and buyers.

Architectural Context Diagram:



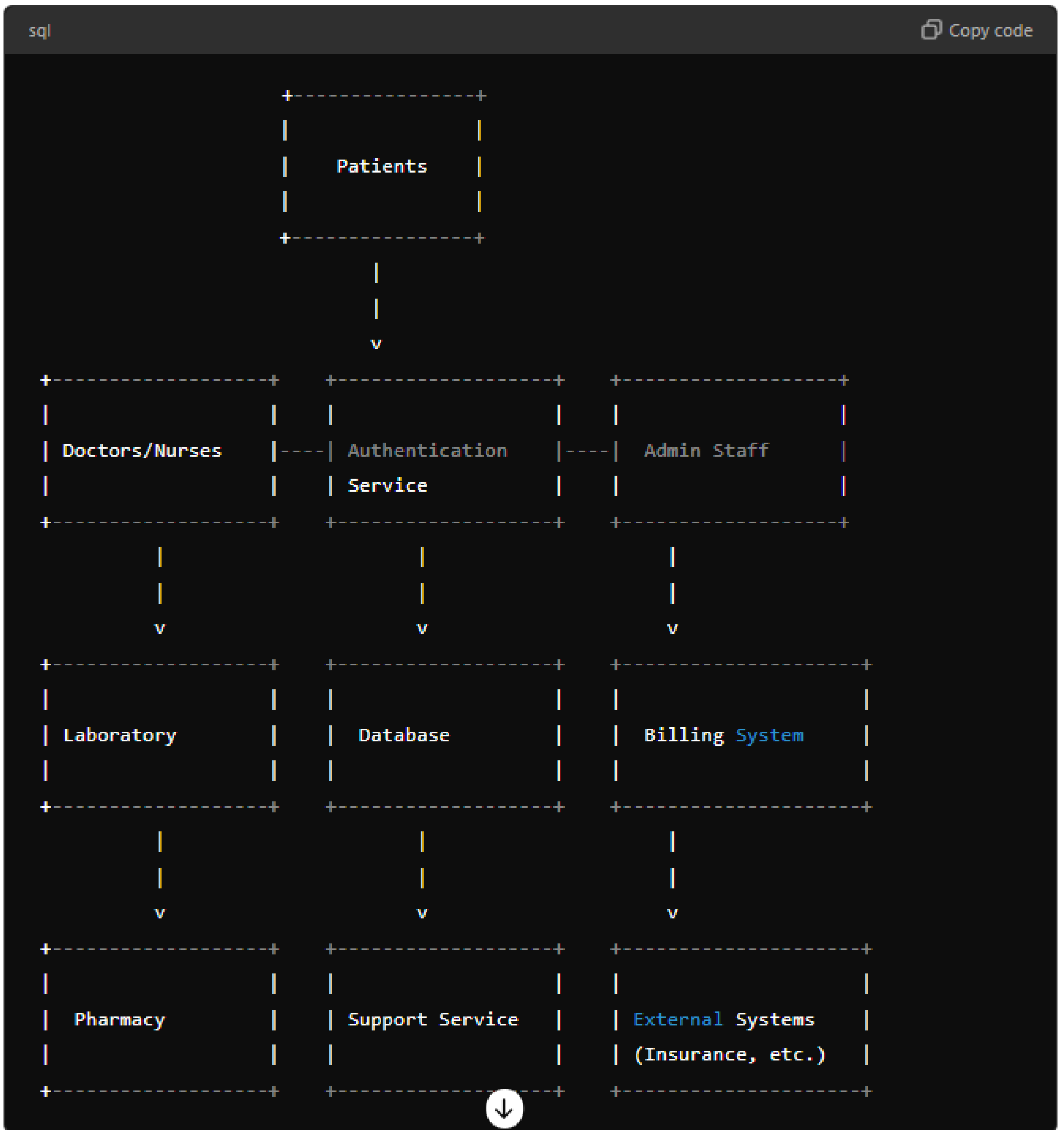
```
                    +---------------+
                    |               |
                    |    Farmers    |
                    |               |
                    +---------------+
                           |
                           |
                           v
+------------------+    +------------------+    +---------------------+
|                  |    |                  |    |                     |
| Management       |----| Authentication   |----| Payment Gateway     |
| Portal           |    | Service          |    |                     |
+------------------+    +------------------+    +---------------------+
          |                      |                        |
          |                      |                        |
          v                      v                        v
+------------------+    +------------------+    +---------------------+
|                  |    |                  |    |                     |
| Database         |    | Logistics Service|    | Support Service     |
|                  |    |                  |    |                     |
+------------------+    +------------------+    +---------------------+
```

Draw the architectural context diagram for Hospital Management System

- **Patients:** Individuals seeking medical care.
- **Doctors/Nurses:** Medical professionals providing care to patients.
- **Admin Staff:** Hospital administration handling patient records, appointments, and billing.
- **Pharmacy:** Manages medication inventory and dispensation.
- **Laboratory:** Handles diagnostic tests and results.

- **Database:** Stores patient records, medical history, billing information, and other relevant data.
- **Authentication Service:** Manages user login, registration, and security.
- **Billing System:** Handles patient billing and payment processing.
- **Support Service:** Provides assistance for patients and hospital staff.

Architectural Context Diagram:



```
                        +---------------+
                        |               |
                        |    Patients   |
                        |               |
                        +---------------+
                                |
                                |
                                v
+------------------+    +------------------+    +------------------+
|                  |    |                  |    |                  |
| Doctors/Nurses   |----| Authentication   |----|  Admin Staff     |
|                  |    | Service          |    |                  |
+------------------+    +------------------+    +------------------+
          |                       |                       |
          |                       |                       |
          v                       v                       v
+------------------+    +------------------+    +--------------------+
|                  |    |                  |    |                    |
| Laboratory       |    |  Database        |    |  Billing System    |
|                  |    |                  |    |                    |
+------------------+    +------------------+    +--------------------+
          |                       |                       |
          |                       |                       |
          v                       v                       v
+------------------+    +------------------+    +--------------------+
|                  |    |                  |    |                    |
|  Pharmacy        |    | Support Service  |    | External Systems   |
|                  |    |                  |    | (Insurance, etc.)  |
+------------------+    +------------------+    +--------------------+
```

Suggest and justify the architectural style for the online Jewellery Mart system. Also identify major components used in designing online Jewellery Mart system

Suggested Architectural Style: Microservices Architecture

Justification:

1. **Scalability:** Microservices architecture allows each service to be scaled independently. This is crucial for an online Jewellery Mart system where different services (like product catalog, user management, payment processing) may experience varying loads.
2. **Flexibility:** Each service can be developed, deployed, and maintained independently. This provides flexibility to use different technologies for different services based on their requirements.
3. **Fault Isolation:** With microservices, a failure in one service does not necessarily impact other services. This increases the overall reliability and availability of the system.
4. **Continuous Deployment:** Independent services facilitate continuous deployment, allowing for frequent updates and quicker delivery of new features.
5. **Ease of Maintenance:** Smaller, focused services are easier to understand, maintain, and test. This reduces the complexity compared to a monolithic system.
6. **Reusability:** Services can be reused across different projects or even different parts of the same system.

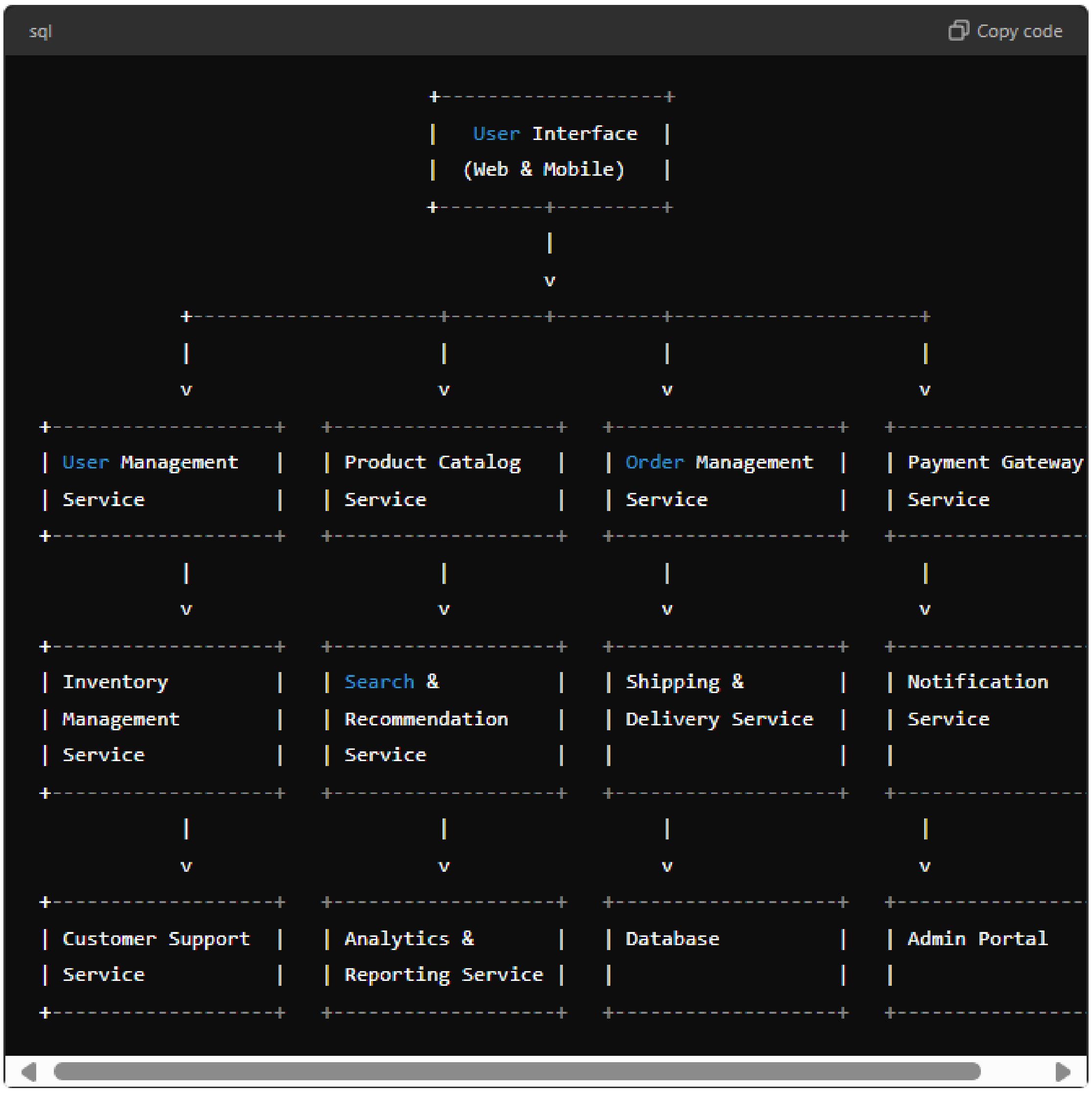
Major Components in Online Jewellery Mart System:

1. **User Management Service:**
   - **Authentication & Authorization:** Handles user registration, login, password management, and roles/permissions.
   - **User Profile:** Manages user details like personal information, order history, and preferences.
2. **Product Catalog Service:**
   - **Product Listings:** Manages product information including descriptions, prices, images, and availability.
   - **Categories & Filters:** Supports categorization of products and filtering options for users.
3. **Inventory Management Service:**
   - **Stock Management:** Keeps track of inventory levels, stock replenishment, and notifications for low stock.
   - **Warehouse Integration:** Integrates with warehouse systems to ensure real-time inventory updates.
4. **Order Management Service:**
   - **Order Processing:** Handles the creation, updating, and tracking of orders.
   - **Order History:** Maintains a record of all past orders for users.
5. **Payment Gateway Service:**
   - **Payment Processing:** Integrates with third-party payment gateways to handle transactions.

    - **Security & Compliance:** Ensures secure payment processing and compliance with relevant standards (e.g., PCI DSS).

6. **Shipping and Delivery Service:**
    - **Logistics Management:** Manages shipping options, tracking, and delivery schedules.
    - **Integration with Delivery Partners:** Connects with various delivery partners to streamline the shipping process.
7. **Customer Support Service:**
    - **Helpdesk:** Provides support for customer inquiries, complaints, and returns.
    - **Live Chat & Communication:** Facilitates real-time communication with support agents.
8. **Notification Service:**
    - **Email & SMS Notifications:** Sends notifications for order confirmations, shipping updates, promotions, etc.
    - **Push Notifications:** Sends updates through mobile apps or web push notifications.
9. **Search and Recommendation Service:**
    - **Search Functionality:** Provides advanced search capabilities for products.
    - **Recommendations Engine:** Suggests products based on user behavior and preferences.
10. **Analytics and Reporting Service:**
    - **Business Analytics:** Tracks sales, user behavior, and other key metrics.
    - **Reports Generation:** Generates reports for business insights and decision-making.
11. **Admin Portal:**
    - **Dashboard:** Provides a centralized interface for managing the entire system.
    - **Management Tools:** Includes tools for managing products, orders, users, and settings.

Architectural Context Diagram:



```
sql                                                        Copy code

                    +------------------+
                    |   User Interface  |
                    |  (Web & Mobile)   |
                    +---------+---------+
                              |
                              v
       +--------------------+---------+---------+--------------------+
       |                    |                   |                    |
       v                    v                   v                    v
+------------------+  +------------------+  +------------------+  +----------------
| User Management  |  | Product Catalog  |  | Order Management |  | Payment Gateway
| Service          |  | Service          |  | Service          |  | Service
+------------------+  +------------------+  +------------------+  +----------------
       |                    |                   |                    |
       v                    v                   v                    v
+------------------+  +------------------+  +------------------+  +----------------
| Inventory        |  | Search &         |  | Shipping &       |  | Notification
| Management       |  | Recommendation   |  | Delivery Service |  | Service
| Service          |  | Service          |  |                  |  |
+------------------+  +------------------+  +------------------+  +----------------
       |                    |                   |                    |
       v                    v                   v                    v
+------------------+  +------------------+  +------------------+  +----------------
| Customer Support |  | Analytics &      |  | Database         |  | Admin Portal
| Service          |  | Reporting Service|  |                  |  |
+------------------+  +------------------+  +------------------+  +----------------
```

Develop a complete architectural design for Safe home product. Also identify various components used in Safe home product.

Safe Home Product: Complete Architectural Design

The Safe Home product is a comprehensive smart home security system designed to ensure the safety and security of a household. It integrates various sensors, cameras, alarms, and automation systems, controlled via a centralized interface accessible through mobile and web applications.

Architectural Style: Microservices Architecture

Justification:

1. **Scalability:** Each service can be scaled independently based on demand.
2. **Flexibility:** Different components (like monitoring, alerts, user management) can be developed using the most suitable technologies.
3. **Fault Isolation:** Failures in one service do not affect the entire system.
4. **Continuous Deployment:** Independent services allow for frequent updates.
5. **Ease of Maintenance:** Smaller, focused services are easier to maintain and test.

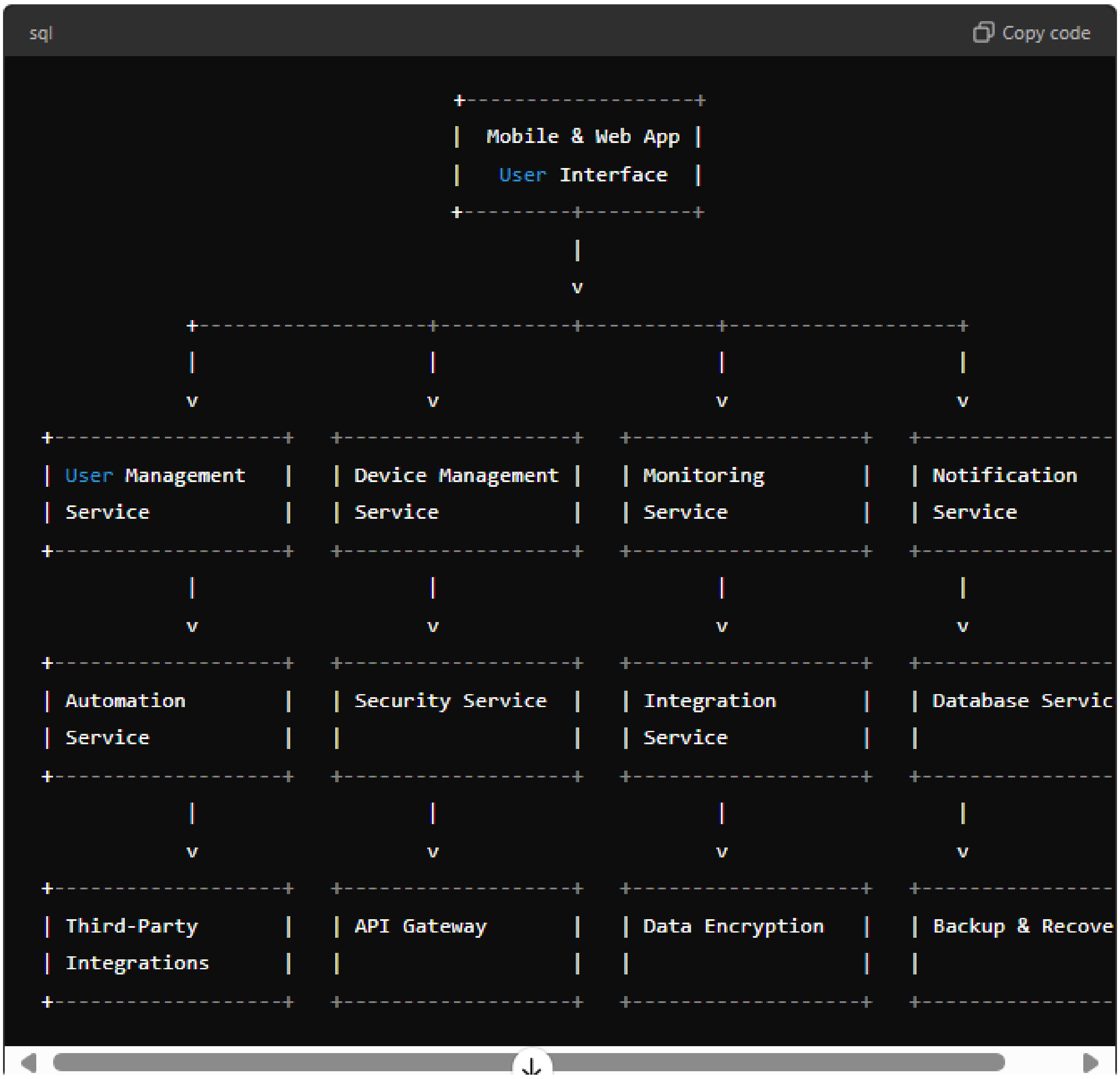
Major Components:

1. **User Management Service:**
    - **Authentication & Authorization:** Handles user registration, login, and role-based access control.
    - **User Profile:** Manages user details, preferences, and settings.
2. **Device Management Service:**
    - **Device Registration:** Manages the registration and setup of smart devices.
    - **Device Control:** Interfaces for controlling devices like cameras, sensors, and alarms.
3. **Monitoring Service:**
    - **Real-time Monitoring:** Streams data from cameras and sensors to the central system.
    - **Event Detection:** Analyzes data for events like motion detection, door/window opening, etc.
4. **Notification Service:**
    - **Alerts:** Sends notifications via email, SMS, or push notifications for events.
    - **Logs:** Maintains a log of all events and alerts for audit purposes.
5. **Automation Service:**
    - **Rules Engine:** Allows users to define automation rules (e.g., turn on lights when motion is detected).
    - **Scheduled Tasks:** Manages scheduled tasks like turning on/off devices at specific times.
6. **Security Service:**
    - **Data Encryption:** Ensures secure communication between devices and the central system.
    - **Access Control:** Manages secure access to devices and data.
7. **Integration Service:**
    - **Third-Party Integration:** Integrates with third-party services like voice assistants (e.g., Alexa, Google Assistant) and other smart home ecosystems.
    - **API Gateway:** Provides APIs for integrating with external systems.
8. **Mobile and Web Application:**

- **User Interface:** Provides an interface for users to interact with the system, view live feeds, control devices, and configure settings.
- **Notifications:** Receives and displays alerts and notifications.

9. **Database Service:**
   - **Data Storage:** Stores user data, device data, event logs, and configuration settings.
   - **Backup and Recovery:** Ensures data backup and recovery mechanisms

Architectural Context Diagram:


sql

Copy code

```
                              +------------------+
                              |  Mobile & Web App |
                              |   User Interface  |
                              +---------+--------+
                                        |
                                        v
                  +-------------------+-----------+-----------+-------------------+
                  |                   |                       |                   |
                  v                   v                       v                   v
    +------------------+   +------------------+   +------------------+   +---------------
    | User Management  |   | Device Management |   | Monitoring       |   | Notification
    | Service          |   | Service          |   | Service          |   | Service
    +------------------+   +------------------+   +------------------+   +---------------
                  |                   |                       |                   |
                  v                   v                       v                   v
    +------------------+   +------------------+   +------------------+   +---------------
    | Automation       |   | Security Service |   | Integration      |   | Database Servic
    | Service          |   |                  |   | Service          |   |
    +------------------+   +------------------+   +------------------+   +---------------
                  |                   |                       |                   |
                  v                   v                       v                   v
    +------------------+   +------------------+   +------------------+   +---------------
    | Third-Party      |   | API Gateway      |   | Data Encryption  |   | Backup & Recove
    | Integrations     |   |                  |   |                  |   |
    +------------------+   +------------------+   +------------------+   +---------------
```

Detailed Component Descriptions:

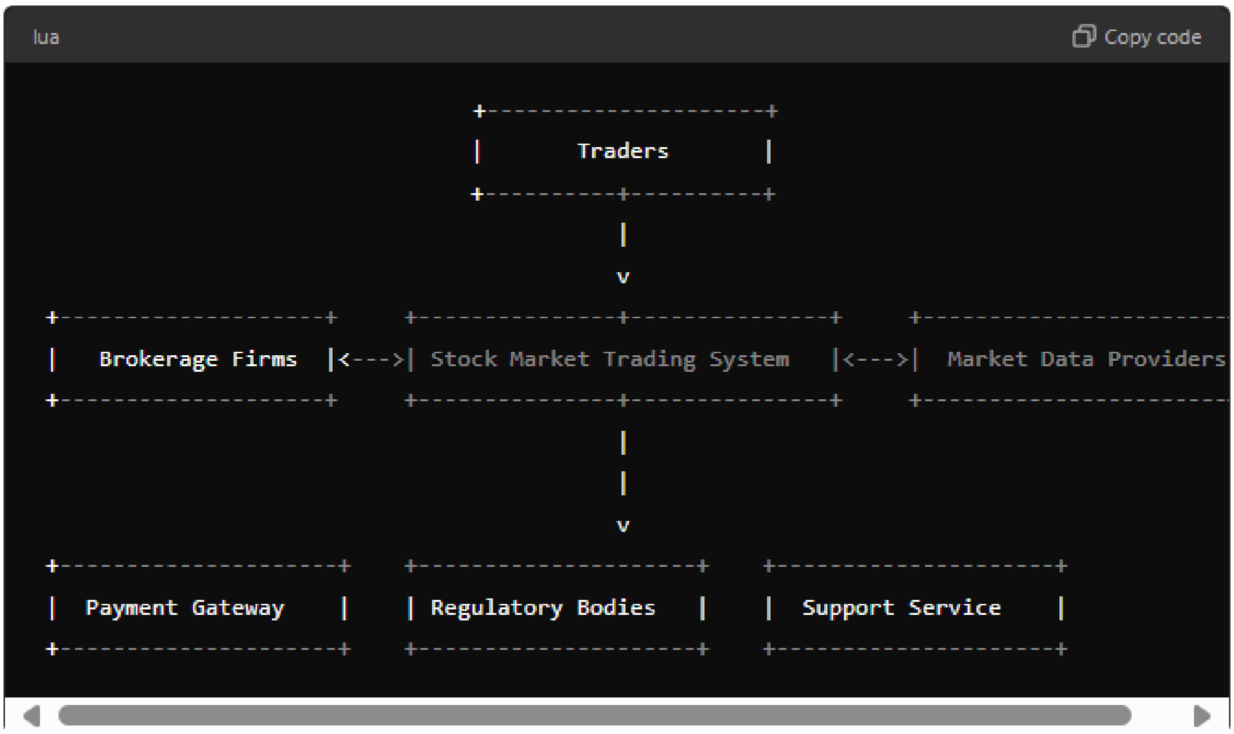
1. **User Management Service:**
    - **Authentication:** Verifies user identity using methods like passwords, biometrics, or OTP.
    - **Authorization:** Ensures users have appropriate permissions to access and control devices.
    - **Profile Management:** Allows users to update their personal information and preferences.
2. **Device Management Service:**
    - **Registration:** Adds new devices to the system, ensuring they are recognized and configured.
    - **Control:** Provides interfaces for users to control smart devices (e.g., turning lights on/off, locking/unlocking doors).
3. **Monitoring Service:**
    - **Live Feed:** Streams live video and sensor data to the central system for real-time monitoring.
    - **Event Detection:** Uses algorithms to detect significant events (e.g., unusual motion, unauthorized access).
4. **Notification Service:**
    - **Alerts:** Configures and sends alerts for predefined events (e.g., security breaches, system malfunctions).
    - **Logs:** Maintains a detailed log of all events and alerts for tracking and audit purposes.
5. **Automation Service:**
    - **Rules Engine:** Allows users to create rules for automating device actions based on triggers (e.g., time of day, sensor activity).
    - **Scheduled Tasks:** Manages tasks that need to occur at specific times (e.g., daily routines).
6. **Security Service:**
    - **Encryption:** Secures data in transit and at rest to protect against unauthorized access.
    - **Access Control:** Manages who can access which devices and data, ensuring robust security policies.
7. **Integration Service:**
    - **Voice Assistants:** Integrates with platforms like Alexa or Google Assistant for voice control.
    - **Smart Home Ecosystems:** Works with other smart home systems to provide a seamless user experience.
8. **Mobile and Web Application:**
    - **Dashboard:** Provides a central interface for users to view system status and control devices.
    - **Live View:** Displays live feeds from cameras and sensor data.
    - **Settings:** Allows users to configure system settings, automation rules, and preferences.
9. **Database Service:**
    - **Storage:** Manages the storage of user data, device data, and event logs.
    - **Backup:** Ensures data is regularly backed up and can be restored in case of data loss.

(i) Stock Market Trading System

Actors:

- **Traders:** Individuals or entities participating in buying and selling stocks.
- **Brokerage Firms:** Intermediaries facilitating trades between traders and the stock market.
- **Stock Exchange:** Platform where stocks are listed and traded.
- **Market Data Providers:** Provide real-time market data (stock prices, indices).
- **Payment Gateway:** Handles transactions for buying and selling stocks.
- **Regulatory Bodies:** Monitor and regulate trading activities.
- **Support Service:** Provides assistance to traders for issues or inquiries.

Architectural Context Diagram:



```lua
                        +--------------------+
                        |       Traders      |
                        +---------+----------+
                                  |
                                  v
+-------------------+     +-------------+--------------+     +----------------------
|   Brokerage Firms |<--->| Stock Market Trading System |<--->|  Market Data Providers
+-------------------+     +-------------+--------------+     +----------------------
                                  |
                                  |
                                  v
+--------------------+    +--------------------+    +--------------------+
|  Payment Gateway   |    | Regulatory Bodies  |    |  Support Service   |
+--------------------+    +--------------------+    +--------------------+
```

Major Components of Stock Market Trading System:

1. **User Management Service:**
   - **Authentication & Authorization:** Manages user registration, login, and access control.
   - **Profile Management:** Handles user profiles and preferences.
2. **Trading Engine:**
   - **Order Processing:** Manages the placement, matching, and execution of buy/sell orders.

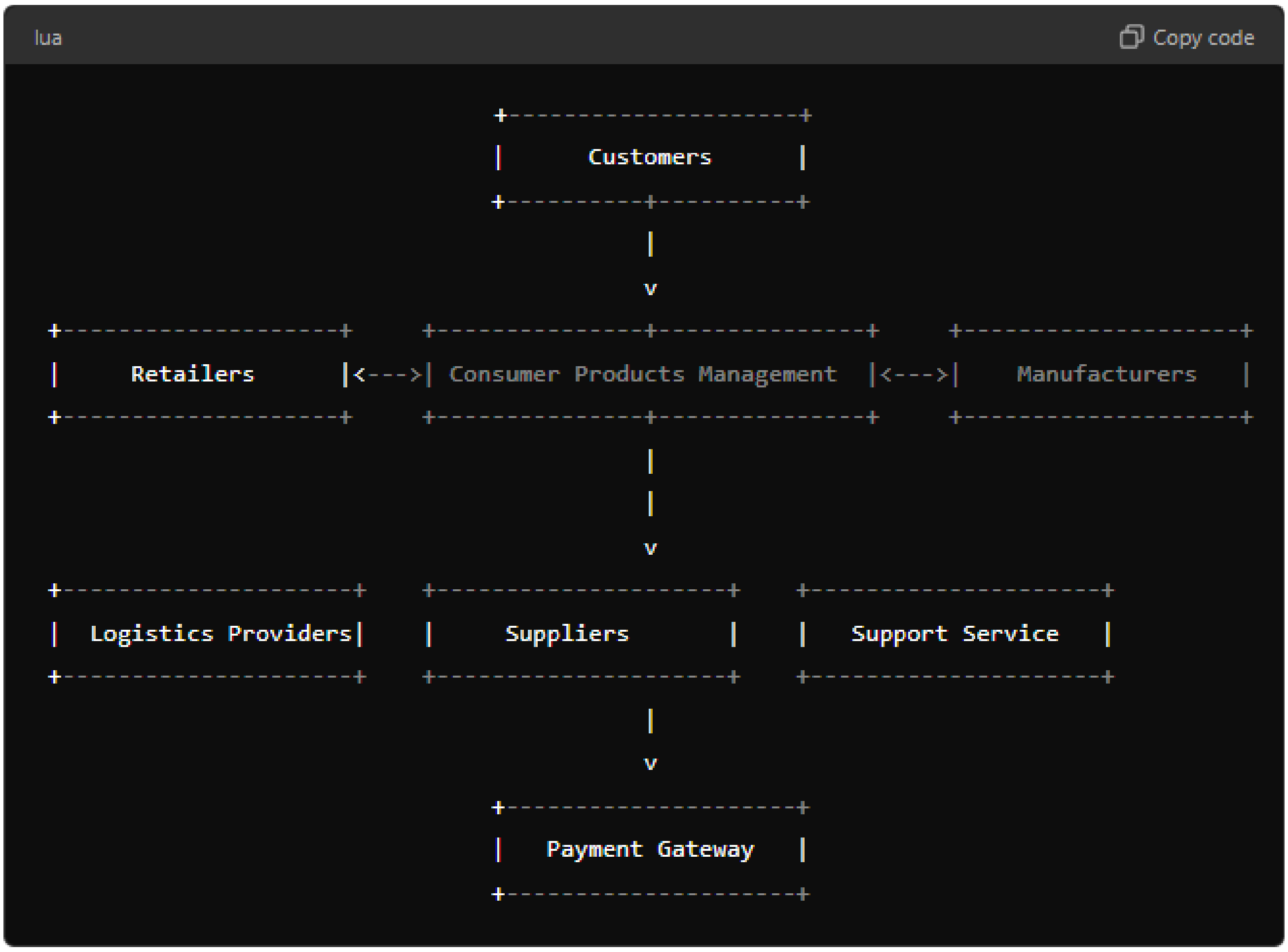
    - **Trade Matching:** Matches buy and sell orders based on price and availability.
3. **Market Data Service:**
    - **Real-time Data Feed:** Provides real-time market data to traders.
    - **Historical Data:** Stores and provides access to historical market data.
4. **Risk Management Service:**
    - **Monitoring:** Monitors trades for compliance with regulations and risk limits.
    - **Alerts:** Generates alerts for suspicious or non-compliant activities.
5. **Payment Processing Service:**
    - **Transaction Handling:** Manages financial transactions for buying and selling stocks.
    - **Settlement:** Ensures timely settlement of trades.
6. **Reporting and Analytics Service:**
    - **Trade Reports:** Generates reports on trading activities.
    - **Market Analytics:** Provides insights and analytics based on market data.
7. **Support Service:**
    - **Helpdesk:** Assists traders with technical and account-related issues.
    - **Live Chat:** Provides real-time support to traders.

(ii) Consumer Products Management System

Actors:

- **Manufacturers:** Entities producing consumer products.
- **Retailers:** Businesses selling consumer products to end customers.
- **Customers:** Individuals purchasing consumer products.
- **Suppliers:** Provide raw materials and components to manufacturers.
- **Logistics Providers:** Handle the transportation and delivery of products.
- **Support Service:** Provides assistance to customers and retailers.
- **Payment Gateway:** Handles transactions for product purchases

Architectural Context Diagram:


lua

Copy code

```
                    +--------------------+
                    |     Customers      |
                    +---------+----------+
                              |
                              v
+-------------------+    +-------------+-------------+    +-------------------+
|     Retailers     |<--->| Consumer Products Management |<--->|    Manufacturers    |
+-------------------+    +-------------+-------------+    +-------------------+
                              |
                              |
                              v
+--------------------+    +--------------------+    +--------------------+
| Logistics Providers|    |     Suppliers      |    |  Support Service   |
+--------------------+    +--------------------+    +--------------------+
                              |
                              v
                    +--------------------+
                    |  Payment Gateway   |
                    +--------------------+
```

Major Components of Consumer Products Management System:

1. **Product Management Service:**
   - **Catalog Management:** Manages product listings, descriptions, prices, and availability.
   - **Inventory Management:** Tracks inventory levels and manages stock replenishment.
2. **Order Management Service:**
   - **Order Processing:** Handles order placement, updates, and tracking.
   - **Order Fulfillment:** Manages the fulfillment process from order receipt to delivery.
3. **User Management Service:**
   - **Authentication & Authorization:** Manages user registration, login, and access control.
   - **Profile Management:** Handles customer and retailer profiles.
4. **Supplier Management Service:**
   - **Supplier Integration:** Manages relationships and transactions with suppliers.
   - **Supply Chain Management:** Coordinates the supply of raw materials and components.
5. **Logistics Management Service:**
   - **Shipment Tracking:** Tracks the shipment and delivery of products.
   - **Logistics Coordination:** Manages relationships with logistics providers.

6. **Payment Processing Service:**
    - **Transaction Handling:** Manages financial transactions for product purchases.
    - **Payment Reconciliation:** Ensures accurate payment records and reconciliation.
7. **Support Service:**
    - **Helpdesk:** Assists customers and retailers with inquiries and issues.
    - **Live Chat:** Provides real-time support to users.
8. **Reporting and Analytics Service:**
    - **Sales Reports:** Generates reports on sales performance.
    - **Inventory Analytics:** Provides insights into inventory levels and turnover.
9. **Marketing and Promotions Service:**
    - **Campaign Management:** Manages marketing campaigns and promotions.
    - **Discounts and Offers:** Handles discounts and special offers for products.

Briefly describe each of the four elements of the design model.

12.4 The Design Model

The design model can be viewed through two dimensions, as illustrated in Figure 12.4. These dimensions are the process dimension, which indicates the evolution of the design model as design tasks are executed as part of the software process, and the abstraction dimension, representing the level of detail as each element of the analysis model is transformed into a design equivalent and then refined iteratively. The dashed line in the figure indicates the boundary between the analysis and design models, though sometimes this distinction is not clear-cut as the analysis model can gradually blend into the design model.

The design model elements use many of the same UML diagrams as the analysis model. However, these diagrams are refined and elaborated during design to include more implementation-specific details. Architectural structure and style, components within the architecture, and interfaces between components and with the outside world are emphasized.

It's important to note that the development of model elements along the horizontal axis does not always follow a sequential order. Typically, preliminary architectural design is followed by interface design and component-level design, often occurring in parallel. The deployment model is usually developed once the design has been fully fleshed out.

Design patterns can be applied at any stage during design to solve domain-specific problems using established solutions.

12.4.1 Data Design Elements

Data design, sometimes referred to as data architecting, creates a high-level model of data that is progressively refined into more implementation-specific representations. The data architecture significantly influences the software architecture. Data design spans multiple levels:

- **Program-component level**: Designing data structures and algorithms.
- **Application level**: Translating a data model into a database.
- **Business level**: Reorganizing data from disparate databases into a data warehouse for data mining.

12.4.2 Architectural Design Elements

Architectural design for software is akin to the floor plan of a house, depicting the overall layout and relationships of the components. The architectural model is derived from:

1. Information about the application domain.
2. Requirements model elements such as use cases or analysis classes.
3. Available architectural styles and patterns.

The architectural design element is depicted as a set of interconnected subsystems, each potentially having its own architecture. Techniques for deriving specific elements of the architectural model are covered in Chapter 13.

12.4.3 Interface Design Elements

Interface design is analogous to detailed drawings for a house's doors, windows, and utilities. It depicts information flows into and out of the system and communication among components. There are three key elements of interface design:

1. **User Interface (UI)**: Incorporates aesthetic, ergonomic, and technical elements, and is considered in detail in Chapter 15.
2. **External Interfaces**: Interfaces to other systems, devices, or networks.
3. **Internal Interfaces**: Interfaces between various design components.

An interface is often modeled like a class in UML, defined as a specifier for externally visible operations without specifying internal structure. An example is the ControlPanel class in the SafeHome system, which realizes the KeyPad interface to provide keypad operations to other classes.

12.4.4 Component-Level Design Elements

Component-level design provides detailed drawings for each room in a house, depicting all internal details. It fully describes the internal detail of each software component, defining data structures and algorithmic detail for processing within a component, and an interface for accessing component operations.

In UML, a component is represented diagrammatically, such as the SensorManagement component in the SafeHome security function. Component diagrams and further details on component design are discussed in Chapter 14.

12.4.5 Deployment-Level Design Elements

Deployment-level design elements indicate how software functionality and subsystems are allocated within the physical computing environment. For example, SafeHome operates within three primary computing environments—a home-based PC, a control panel, and a CPI Corp. server.

A UML deployment diagram is developed and refined to show the computing environment and the subsystems housed within each element. Initially in descriptor form, the diagram describes the environment in general terms. Later, in instance form, it details specific hardware configurations.

With suitable examples, describe Architectural styles and Architectural Genres.

13.2 Architectural Genres

Although the underlying principles of architectural design apply to all types of architecture, the architectural genre often dictates the specific architectural approach to the structure that must be built. In the context of architectural design, genre implies a specific category within the overall software domain. Within each category, you encounter a number of subcategories. For example, within the genre of buildings, you would encounter the following general styles:

- Houses
- Condos
- Apartment buildings
- Office buildings
- Industrial buildings
- Warehouses

Within each general style, more specific styles might apply (see Section 13.3). Each style would have a structure that can be described using a set of predictable patterns.

In his evolving *Handbook of Software Architecture* [Boo08], Grady Booch suggests the following architectural genres for software-based systems:

- Artificial intelligence
- Communications
- Devices
- Financial
- Games
- Industrial
- Legal
- Medical
- Military
- Operating systems
- Transportation
- Utilities

13.3 Architectural Styles

When a builder uses the phrase “centre hall colonial” to describe a house, most people familiar with houses in the United States will be able to conjure a general image of what the house will look like and what the floor plan is likely to be. The builder has used an architectural style as a descriptive mechanism to differentiate the house from other styles (e.g., A-frame, raised ranch, Cape Cod). But more important, the architectural style is also a template for construction. Further details of the house must be defined, its final dimensions must be specified, customized features may be added, and building materials are to be determined, but the style—a “ centre hall colonial”—guides the builder in his work.

The software that is built for computer-based systems also exhibits one of many architectural styles. Each style describes a system category that encompasses:

1. **A set of components** (e.g., a database, computational modules) that perform a function required by a system,
2. **A set of connectors** that enable “communication, coordination, and cooperation” among components,
3. **Constraints** that define how components can be integrated to form the system, and

4. **Semantic models** that enable a designer to understand the overall properties of a system by analysing the known properties of its constituent parts [Bas03].

An architectural style is a transformation that is imposed on the design of an entire system. The intent is to establish a structure for all components of the system. In the case where an existing architecture is to be reengineered (see Chapter 36), the imposition of an architectural style will result in fundamental changes to the structure of the software, including a reassignment of the functionality of components [Bos00].

An architectural pattern, like an architectural style, imposes a transformation on the design of an architecture. However, a pattern differs from a style in a number of fundamental ways:

1. The scope of a pattern is less broad, focusing on one aspect of the architecture rather than the architecture in its entirety.
2. A pattern imposes a rule on the architecture, describing how the software will handle some aspect of its functionality at the infrastructure level (e.g., concurrency) [Bos00].
3. Architectural patterns (see Section 13.3.2) tend to address specific behavioural issues within the context of the architecture (e.g., how real-time applications handle synchronization or interrupts).

Fundamental Software Design Concepts

1. **Abstraction**
   - **Procedural Abstraction:** Focuses on sequences of operations (e.g., `open` for a door).
   - **Data Abstraction:** Focuses on data objects and their attributes (e.g., `door` with attributes like type and dimensions).
2. **Architecture**
   - Defines the overall structure of software, including components and their interactions.
   - Includes structural, framework, dynamic, process, and functional models.
   - Uses Architectural Description Languages (ADLs) to represent the models.
3. **Patterns**
   - Named solutions to recurring problems within a context.
   - Helps in reusing design solutions and guiding new designs.
4. **Separation of Concerns**
   - Divides complex problems into manageable pieces, allowing for easier handling and optimization.
5. **Modularity**
   - Divides software into distinct modules to make it more manageable and understandable.
   - Balances between too few and too many modules to optimize cost and effort.
6. **Information Hiding**
   - Ensures that modules hide their internal details and only expose necessary information.
7. **Functional Independence**
   - Designs modules with a single-minded function and minimal interaction with other modules.
   - Evaluated through cohesion (internal strength) and coupling (interdependence).
8. **Refinement**
   - A top-down design strategy that elaborates details progressively from high-level abstractions.

9. **Aspects**
    - Represents crosscutting concerns that affect multiple modules or requirements.
    - Ideal for separate modules that address concerns spanning the entire system.
10. **Refactoring**
    - Reorganizes code to improve its internal structure without changing its external behavior.
    - Aims to simplify and enhance maintainability.
11. **Object-Oriented Design Concepts**
    - Includes principles such as classes, objects, inheritance, messages, and polymorphism.