**CS-255 System Analysis And Design**

**Module 4 Assignment**

**Evaluate An Object Model**

**Submitted By:**

HAMNA KHALID

ID: 2902671

[hamna.khalid@snhu.edu](mailto:hamna.khalid@snhu.edu)

**Submitted To:**

PROFESSOR PARUL HIRPARA

p.hirpara@snhu.edu

Southern New Hampshire University

Sunday, September 28, 2025

**Hamp Crafts Object Model**

Object modeling is a crucial part of systems analysis and design because it allows analysts to describe a system in terms of objects, classes, attributes, and behaviors. While process models focus on how information flows through the system, object models concentrate on the structure of the system and the relationships between its components. This object-oriented approach is particularly beneficial now that programming languages such as Java and C++ rely heavily on encapsulation and polymorphism.

For Hamp Crafts, a family-owned craft store that currently operates through a brick-and-mortar shop, object modeling provides a way to translate business requirements into a system design that supports online sales. By defining customers, administrators, orders, shopping carts, and shipping information as objects, the model creates a clear blueprint for building the new storefront. Using the Unified Modeling Language (UML) also ensures standardization, making the design clear not only to developers but also to non-technical stakeholders. In this paper, I will interpret the UML class diagram provided for Hamp Crafts’ online storefront, evaluate whether it fully supports the owners’ needs, explain the meaning of composition relationships, and compare the strengths of object and process models for understanding this system.

# **Functions Of The Online Storefront**

The UML class diagram for Hamp Crafts’ online storefront represents a variety of critical functions through its classes, attributes, and methods. These functions ensure that both customers and administrators can interact effectively with the system.

## **Customer Functions**

Customers can create accounts, log in, and update their profiles (*register(), login(), updateProfile()*). They also manage their orders, save payment details, and check out through the shopping cart.

## **Shopping Cart Functions**

Customers can add items, update quantities, calculate totals, and finalize purchases using *addCartItem(), updateQuantity(),* and *checkOut()*.

## **Order Management**

Orders are tracked with attributes such as *orderId, dateCreated,* and *status*. The *placeOrder()* method finalizes transactions, while order details ensure products and quantities are properly stored.

## **Payment Functions**

Payment information, including credit card details, is stored in the Customer class and linked to orders for secure processing through a third-party vendor. Without explicit modeling of payment vendor integration, financial records could be delayed or mismatched, leading to customer dissatisfaction or business risk.

## **Shipping Functions**

The *Shipping Info* class maintains shipping type, cost, and delivery regions, and can be updated via *updateShippingInfo()*. These details are linked directly to orders to ensure accurate fulfillment.

## **Administrative Functions**

Administrators can manage the storefront by updating the product catalog (*updateCatalog()*) and receiving alerts for completed transactions, ensuring smooth operations and customer support.

Together, these functions form a structured blueprint for building a functional and efficient online storefront.

# **User Classes And Their Associations**

The UML class diagram for Hamp Crafts’ online storefront includes three major user classes, each with distinct responsibilities and relationships:

### **User**

A parent class responsible for core authentication functions such as *verifyLogin()*. It provides the foundation for all user interactions in the system.

### **Customer**

Inherits from the *User* class and is the primary participant in the storefront. Customers create accounts, manage shopping carts, and place orders. This class maintains associations with *Order* and *Shopping Cart*.

### **Administrator**

Another user type focuses on backend tasks, such as updating the product catalog and managing store operations. Administrators also monitor system activity and ensure order processes run smoothly.

## **Associations Between Classes**

The relationships in the model show how users interact with other components of the system:

### **Customer → Order**

A single customer can place multiple orders, each containing detailed information about products and quantities.

### **Order → Order Details**

Through composition, each order is broken down into order details. These details cannot exist independently of their parent order.

### **Customer → Shopping Cart**

Customers can maintain one or more shopping carts while browsing and shopping.

### **Administrator → Product/Order Management**

Administrators are responsible for product catalog updates and order oversight, as well as supporting system maintenance and customer service.

This structure ensures that both front-end interactions (customers) and back-end management (administrators) are represented and integrated, while composition enforces the dependencies between orders and their details.

# **Variables And Functions In The Object Model**

Each class in the UML diagram encapsulates its own variables (attributes) and functions (methods), demonstrating the principle of **encapsulation**, where data is protected and accessed only through defined operations.

## **Customer**

Contains variables such as *customerName, creditCardInfo,* and *accountBalance*. Methods like *updateProfile()* allow the customer to modify stored information securely. By limiting access through functions, sensitive data such as payment details remains protected.

## **Shopping Cart**

Identified through variables like *cartId* and *productId*. Its functions, including *addCartItem()* and *updateQuantity()*, enable customers to build and adjust their cart before checkout.

## **Order**

Uses variables such as *orderId, dateCreated,* and *status* to track progress. The *placeOrder()* method finalizes the purchase, linking order details and payment confirmation.

## **Shipping Info**

Maintains variables like *shippingType* and *shippingCost*. The *updateShippingInfo()* method allows updates to shipping preferences, ensuring accurate delivery information.

## **Administrator**

Provides backend control through the *updateCatalog()* method, which manages product listings and ensures inventory is kept current for customers.

This structure shows how each class is self-contained but interacts with others through defined methods, ensuring both modularity and system security.

# **Evaluation Of The Object Model’s Completeness**

The object model for Hamp Crafts’ online storefront is strong, but it does not fully capture all required functionality. The diagram successfully represents customer account creation, order management, checkout processes, catalog updates, and shipping integration. These elements form the foundation of the system and demonstrate how customers and administrators interact with the storefront.

However, several important features are not explicitly represented in the model. For example, integration with third-party payment vendors such as Square or Shopify is missing, which creates uncertainty around how transactions would be processed securely. Additionally, customer notifications, such as confirmations or order status updates, are absent, leaving out a critical aspect of the customer experience. Similarly, while administrator involvement is implied, the model does not clearly show alerts for completed transactions, which are essential for timely order management. Without these features, Hamp Crafts could face gaps in both customer trust and operational efficiency.

Overall, the object model provides a strong structural base, but it requires refinement to ensure that all business requirements, particularly payment processing, notifications, and administrative oversight, are fully met.

# **Aggregation In The Object Model**

The solid diamond in the UML diagram represents **composition**, the strongest form of aggregation. This implies that:

* Child classes cannot exist without their parent.
* For example, *Order Details* cannot exist independently without an *Order*. Deleting the order removes all its details.
* Similarly, *Shopping Cart* items only exist within the context of a shopping cart and its customer.

This use of composition is appropriate for Hamp Crafts, as it reflects real-world dependencies between orders, carts, and their components.

# **Evaluation Of The Process Model (From Module 3)**

As I discussed in my Module 3 assignment, the process model highlighted valuable insights into Hamp Crafts’ current purchase and supply workflow. One of its greatest strengths was its ability to clearly illustrate the sequence of activities, from customer purchase to supplier interaction. This visualization made it easy to identify gaps, such as the reliance on manual inventory management, which could result in delays or errors if not addressed.

At the same time, the process model was limited in scope. While it excelled in showing workflows and data movement, it did not provide details about system logic, object attributes, or class relationships. These omissions mean that while the process model is excellent for understanding how work flows through the system, it is less effective at representing the structure of the system or its long-term maintenance requirements. If Hamp Crafts continues to rely on manual inventory processes, these issues could carry over into the object model unless addressed through automation.

# **Evaluation Of The Object Model**

The object model addresses many of the weaknesses of the process model by shifting focus from workflows to system structure. Its main strengths include the clear definition of classes, attributes, and methods that are essential for coding and system maintenance. It also accurately captures critical relationships, such as the composition between orders and order details, which reflects real-world dependencies.

Despite these advantages, the object model has its own limitations. It does not emphasize the sequence of processes or the timing of activities, both of which are important for workflow clarity. Furthermore, the model does not explicitly include external vendor integration or automated customer notifications, leaving out key elements of the business requirements. Without these additions, the system risks inefficiencies in financial processing and customer communication.

When compared to the process model from Module 3, the object model is stronger for long-term development and system design, while the process model is more effective for showing workflow clarity. Together, the two provide a complementary understanding of Hamp Crafts’ systems: the process model explains how the system works, while the object model explains what the system is.

# **Conclusion**

The UML object model provides a solid foundation for Hamp Crafts’ online storefront but needs refinements to fully capture payment vendor integration, customer notifications, and administrative alerts. Its use of composition accurately depicts dependencies between orders and order details. Compared to the process model from Module 3, the object model better supports coding and long-term system design, while the process model is more effective for visualizing workflows.

By leveraging the strengths of both models, Hamp Crafts can confidently modernize its business, ensuring efficient operations today while also laying the foundation to scale into future channels such as mobile apps or third-party marketplaces.

By using both models together, Hamp Crafts not only modernizes its business today but also creates a flexible foundation for future innovations, such as loyalty programs or omnichannel retail.

# **Acronyms And Full Form**

* **UML –** Unified Modeling Language
* **OOD –** Object-Oriented Design
* **DFD –** Data Flow Diagram
* **DB –** Database

# **References**

* Dennis, A., Wixom, B. H., & Tegarden, D. (2015). *Systems analysis and design with UML* (4th ed.). Wiley. [PDF]
* Fowler, M. (2003). *UML Distilled: A Brief Guide to the Standard Object Modeling Language* (3rd ed.). Addison-Wesley. [PDF]
* Object Management Group (OMG). (2017). *Unified Modeling Language (UML) specification*. Object Management Group. [PDF]
* Valacich, J. S., & George, J. F. (2020). *Modern systems analysis and design* (9th ed.). Pearson. [PDF]
* Southern New Hampshire University (2025). *CS-255 Systems Analysis and Design course materials*. [PDF]