Megan Mitchell
Professor Eppenger
CS 255 System Analysis and Design
27 September 2025

4-2 Assignment: Evaluate an Object Model

1. What are the different functions of the online storefront? How are they represented in this type of model?

When I look at the object model, the main functions of the online storefront are laid out as methods within each class. For example, the Customer class has functions like register(), login(), and updateProfile(). These represent the ability for new customers to create accounts, existing customers to log in, and for people to update things like their billing, shipping, or contact information. Those are really important functions because without them, customers wouldn't even be able to use the online store. The Shopping Cart class shows functions like addCartItem(), updateQuantity(), and removeCartItem(). This is the heart of the shopping experience, where customers can select items they want to buy, change how many they want, or remove something before checking out. The Order class has the placeOrder() method, which represents the function of actually finalizing a purchase. That ties into Order Details, which stores things like the product name, cost, and quantity. On top of that, the Shipping Info class manages shipping type, cost, and region, and its functions allow the system to calculate costs and tie them to orders. Finally, the Administrator class has functions like updateCatalog(). This means store employees or admins can log in and add new products, remove outdated ones, or update prices. Without this, the store wouldn't stay current. So overall, each class in the diagram holds both variables (data) and functions (actions), which together represent the different things the storefront can do.

2. What are the different classes of “users” represented by this object model? What are the associations between these classes?

From what I can see, there are three main types of users in this object model: Users, Customers, and Administrators. The User class is like the parent class, and both Customer and Administrator inherit from it. This makes sense because both customers and admins need to log in and have credentials, so that shared functionality gets stored in User. The Customer class is more specific. It stores personal information like name, email, password, and credit card info. Customers are the people actually buying products and using the storefront in the normal way. The Administrator class is different because instead of shopping, their role is about keeping the system updated. They can manage the product catalog, which means adding new items, editing descriptions, or removing items that aren’t being sold anymore.

As for associations, the Customer is linked to Orders, meaning a single customer can place many orders over time. Orders are connected to Order Details, which represent each individual item in that order. Orders also connect to Shipping Info, so shipping details are tied directly to each order. The Shopping Cart is also associated with the Customer, since each customer has their own cart. These associations really help show how the system works in real life: one customer logs in, fills their cart with items, checks out, which creates an order, and then that order is shipped out. Meanwhile, admins keep everything running smoothly on the back end.

3. How would the objects “use” their respective variables and functions?

Each object in this model combines two things: variables (data) and functions (actions). For example, the Customer object has variables like customerID, name, address, and creditCardInfo. These variables store all the personal information that’s needed to process an order. Then, its functions like register() or login() allow the customer to create or access an account. The updateProfile() function uses the stored variables so customers can make changes when needed, like adding a new credit card or updating their shipping address. The Shopping Cart object has variables like cartID, productID, and quantity. These are the pieces of data that represent what’s currently in the cart. Then the functions like addCartItem() and updateQuantity() use those variables to either add something new, change the amount, or remove it. Without the functions, the data would just sit there unused, but the functions make it interactive.

The Order object uses variables like orderID, date, status, and customerID. When the placeOrder() function is called, it takes all those variables and ties them together to create a valid order. Then, the Order Details object holds the specifics of that order what product was ordered, how much it cost, and how many were bought. The Administrator object has fewer variables but important functions like updateCatalog(). That means the admin can use this function to interact with the product list, and the variables (like product name, price, description) are updated in the system. So in short, the variables act like the memory of each object, and the functions are how the system makes use of that memory to actually do something.

4. Does this object model capture all of Hamp Crafts' desired functionality? Why or why not?

I think the object model captures most of what Hamp Crafts wanted, but not everything. It definitely covers the basics: customers can create accounts, log in, store billing and shipping info, place orders, and check order history. It also allows admins to manage the product catalog, which is key for keeping the storefront up-to-date. But there are a few things missing. For example, Hamp Crafts said they wanted customers to receive notifications about order confirmations and status updates. That's not clearly represented in the object model. It might be implied through the Customer class and Order class, but it's not obvious.

Another missing piece is the alert system for Hamp Crafts' owners. The model doesn't show how the business itself would be notified of transactions in real-time. Also, there's nothing that represents integration with a third-party payment vendor like Square or Shopify, which the store said they wanted to use. So while the object model is solid and captures the main flow of how customers and admins will use the site, it doesn't fully represent every piece of functionality Hamp Crafts asked for. Some of those pieces might need to be added in later or shown in more detail.

5. The above diagram uses a solid diamond shape to represent a form of aggregation. What type of aggregation does this represent? What does it imply about the relationship between the classes? Why is a solid diamond the appropriate choice here?

The solid diamond in the UML diagram represents composition, which is a strong type of aggregation. Composition means that one class is made up of other classes, and the parts can't exist without the whole. In this case, a good example is the relationship between the Order class and the Order Details class. An Order is composed of multiple Order Details, but those Order Details can't exist on their own. They only make sense as part of an order. If you delete the Order, the Order Details should also disappear.

The solid diamond is the right choice because it makes that relationship clear. It shows that the smaller pieces are completely dependent on the bigger one, unlike normal aggregation (which is shown with a hollow diamond) where the parts can still exist independently. So composition here is important to accurately show how orders and order details work together.

6. Compare the two different models (process and object)

When I think about the process model (the DFD from Module Three), I see it as really helpful for understanding the flow of activities. It shows how data moves between processes, like when a customer places an order and how that information flows to inventory, payment processing, and shipping. It's great

for seeing the sequence of steps and who does what. But what the process model doesn't do well is show the structure of the system like what kinds of data are stored in which objects, or how different pieces of information are grouped together.

The object model, on the other hand, does a great job of showing the structure. It breaks the system into classes with variables and methods, so you can see exactly what data is being stored and what actions can be performed. It also clearly shows relationships between classes, like how an order is linked to a customer and shipping info. What's harder to see in the object model is the actual sequence of events. For example, you can see that there's a placeOrder() function, but you don't see the step-by-step process of how it flows from cart to payment to shipping. So I think both models complement each other. The process model helps explain the "how," while the object model explains the "what." Together, they give a full picture of Hamp Crafts' system.

1. What are the different functions of the online storefront? How are they represented in this type of model?

As I look at the object model I am able to see that the main functions of the online storefront are arranged as methods within each class that describe the functions. In the Customer class, there are functions such as register(), login(), and updateProfile(), which are used to register customers. These represent the ability for new customers to create accounts, existing customers to log in, and for people to update things like their billing, shipping, or contact information. Those are really important functions because without them, customers wouldn't even be able to use the online store.

The Shopping Cart class shows functions like addCartItem(), updateQuantity(), and removeCartItem(). This is one of the most important parts of the shopping experience, where customers can choose items they would like to purchase, change the number of items they would like to purchase, or remove items before checking out.

The Order class has the placeOrder() method, which represents the function of actually finalizing a purchase. That ties into Order Details, which stores things like the product name, cost, and quantity. On top of that, the Shipping Info class manages shipping type, cost, and region, and its functions allow the system to calculate costs and tie them to orders.

Finally, the Administrator class has functions like updateCatalog(). The key benefit of this system is that the store employees or administrators can log in and update prices, add new products, or remove the old ones. Without this, the store wouldn't stay current. So overall, each class in the diagram holds both variables (data) and functions (actions), which together represent the different things the storefront can do.

2. What are the different classes of "users" represented by this object model? What are the associations between these classes?

From what I can see, there are three main types of users in this object model: Users, Customers, and Administrators.

The User class is like the parent class, and both Customer and Administrator inherit from it. This makes sense because both customers and admins need to log in and have credentials, so that shared functionality gets stored in User.

The Customer class is more specific. It stores personal information like name, email, password, and credit card info. Customers are the people actually buying products and using the storefront in the normal way.

The Administrator class is different because instead of shopping, their role is about keeping the system updated. They can manage the product catalog, which means adding new items, editing descriptions, or removing items that aren't being sold anymore.

As for associations, the Customer is linked to Orders, meaning a single customer can place many orders over time. Orders are connected to Order Details, which represent each individual item in that order. Orders also connect to Shipping Info, so shipping details are tied directly to each order. The Shopping Cart is also associated with the Customer, since each customer has their own cart.

These associations really help show how the system works in real life: one customer logs in, fills their cart with items, checks out, which creates an order, and then that order is shipped out. Meanwhile, admins keep everything running smoothly on the back end.

3. How would the objects "use" their respective variables and functions?

Each object in this model combines two things: variables (data) and functions (actions).

For example, the Customer object has variables like customerID, name, address, and creditCardInfo. These variables store all the personal information that's needed to process an order. Then, its functions like register() or login() allow the customer to create or access an account. The updateProfile() function uses the stored variables so customers can make changes when needed, like adding a new credit card or updating their shipping address.

The Shopping Cart object has variables like cartID, productID, and quantity. These are the pieces of data that represent what's currently in the cart. Then the functions like addCartItem() and updateQuantity() use

those variables to either add something new, change the amount, or remove it. Without the functions, the data would just sit there unused, but the functions make it interactive.

The Order object uses variables like orderID, date, status, and customerID. When the placeOrder() function is called, it takes all those variables and ties them together to create a valid order. Then, the Order Details object holds the specifics of that order what product was ordered, how much it cost, and how many were bought.

The Administrator object has fewer variables but important functions like updateCatalog(). That means the admin can use this function to interact with the product list, and the variables (like product name, price, description) are updated in the system.

So in short, the variables act like the memory of each object, and the functions are how the system makes use of that memory to actually do something.

4. Does this object model capture all of Hamp Crafts' desired functionality? Why or why not?

I think the object model captures most of what Hamp Crafts wanted, but not everything. It definitely covers the basics: customers can create accounts, log in, store billing and shipping info, place orders, and check order history. It also allows admins to manage the product catalog, which is key for keeping the storefront up-to-date.

But there are a few things missing. For example, Hamp Crafts said they wanted customers to receive notifications about order confirmations and status updates. That's not clearly represented in the object model. It might be implied through the Customer class and Order class, but it's not obvious.

Another missing piece is the alert system for Hamp Crafts' owners. The model doesn't show how the business itself would be notified of transactions in real-time. Also, there's nothing that represents

integration with a third-party payment vendor like Square or Shopify, which the store said they wanted to use.

So while the object model is solid and captures the main flow of how customers and admins will use the site, it doesn't fully represent every piece of functionality Hamp Crafts asked for. Some of those pieces might need to be added in later or shown in more detail.

5. The above diagram uses a solid diamond shape to represent a form of aggregation. What type of aggregation does this represent? What does it imply about the relationship between the classes? Why is a solid diamond the appropriate choice here?

The solid diamond in the UML diagram represents composition, which is a strong type of aggregation. Composition means that one class is made up of other classes, and the parts can't exist without the whole.

In this case, a good example is the relationship between the Order class and the Order Details class. An Order is composed of multiple Order Details, but those Order Details can't exist on their own. They only make sense as part of an order. If you delete the Order, the Order Details should also disappear.

The solid diamond is the right choice because it makes that relationship clear. It shows that the smaller pieces are completely dependent on the bigger one, unlike normal aggregation (which is shown with a hollow diamond) where the parts can still exist independently. So composition here is important to accurately show how orders and order details work together.

6. Compare the two different models (process and object)

In my view, the process model (the DFD from Module Three) is one of the most helpful tools for understanding how activities flow through the organization. This chart shows how data moves from one process to another, such as when a customer makes an order and how that information flows through the various processes of inventory management, payment processing, and shipping. It's great for seeing the sequence of steps and who does what. But what the process model doesn't do well is show the structure of the system like what kinds of data are stored in which objects, or how different pieces of information are grouped together.

The object model, on the other hand, does a great job of showing the structure. As a result, the system is broken down into classes that have variables and methods, and as a result, it is possible to visualize exactly what data is being stored and what actions can be performed with it. In addition, it shows how orders are related to customers and shipping information, so that it is very easy to follow relationships between classes. What's harder to see in the object model is the actual sequence of events. For example, you can see that there's a placeOrder() function, but you don't see the step-by-step process of how it flows from cart to payment to shipping.

So I think both models complement each other. The process model helps explain the "how," while the object model explains the "what." Together, they give a full picture of Hamp Crafts' system.