Sam Maclean

CS-255

SNHU

10/3/21

I would apply the Process Model approach to the DriverPass project by beginning with the customer who can schedule drive times or take online tests and practice tests. Since taking tests is strictly online and scheduling driving practice takes place elsewhere, the scheduling a drive part of the process model will flow to choosing a package amongst the available three packages. The first package will be six hours with the trainer, the second package will contain 8 hours with the trainer and a lecture on the DMV rules and policies, while the last package will contain twelve hours driving with a trainer along with the lecture on the DMV rules and policies. Package three will also include access to practice tests and the online class with all content and material.

Just before the selection of the packages, the chart flow to the checkout/payment plan which will flow to the payment provider. This is necessary because there needs to be a payment credit/debit processing system such as Square to make these transactions happen. Following the payment, the chart will flow to the availability of the driving appointments, tests/practice tests, and/or online class with all materials depending on the package that the customer selects. From there, the customer will simply access this information depending on their needs and can interact with the user interface that supplies them with all of the resources required to learn. This part also includes the learning itself which will be done through on-site practice driving with an instructor or practice tests that will help the customers succeed when they attempt their DMV tests.

The object model approach lets you see the detailed view of the overall system including the variable and function names within each object. For the DriverPass object model, I would model some of the objects along with their dedicated functions and variables such as the Customer which would include variables like -customerName, -customerID, -address, -email, -phoneNumber, -paymentInfo, and -packageInfo which would all be strings. These variables would store the basic information about the customer to both keep the information on file for data storage and purposes of saving customer information so they do not need to keep re-entering their data whenever they checkout. Additionally, customers will be able to optionally opt-out of saving their private information on their own devices for next time. Some functions that would be included in the Customer class would include +addCustomer() which would add the customer to the database and register them in the system, +viewAccount() which would allow the customer to view their account information on file, and +login()/+logout() which would either log the customer in or out of the website.

For the next class, User, there would be variables such as -userID, -password, and -adminPermissions which would determine whether or not the user has administrative privileges. These would all be strings and this list of variables would also include -registrationDate, a date type variable, which will keep the date the user signed up for the website on file. Functions for the User class include +permissions() which will grant respected permissions to users and +verifyLogin which will determine true or false to whether the user successfully logged in or not. The Customer class inherits from the User class which means they share the same features because the Customer class is a child of the User class. Next, The Administrator class will determine whether or not the user is an administrator with the function +verifyAdmin() and will use the function +updateReservation() to update users appointments by modifying the appointments as needed. Administrator uses variables -adminName, -email, and -onlineStatus to store personal information and verify whether or not the administrator is currently online. The Administrator class also inherits from the User class which is represented by an arrow that is not shaded in.

Stemming from the Customer class, there will be Tests and Drives classes which will both have a composition relationship to Customer which indicates the classes are dependent on each other in terms of functionality. The customer is able to either take a practice test or schedule a drive using the dedicated tools within the website. The Tests class will use variables like -testName, -testDate, -numAttempts, -grade, and -timeRemaining. Functions like +checkScore(), +viewPastTests(), +checkCumulativeGrade(), +startTest(), and +endTest() will be included to navigate the online test results. The Drives class will contain variables like -instructorName, -numDrives, -carName, -driveDate, and -driveRoute. Functions included in this class will be +checkDrivingScore(), +carDetails, and +driveDetails. Both Tests and Drives will have a 0 to many relationship with Customer because one customer can make 0 or more appointments and take 0 or more tests.

A great advantage to the process-modeling approach to addressing this scenario is how the steps in the chart can be easily followed like a set of instructions for how to accomplish the task. The objects flow right into each other and it is a visually straight forward chart to follow when using a process model to accurately approach the development of this interface. On the other hand, a disadvantage to this strategy is a lack of detail for each part of the process model which leads to vague ideas potentially being analyzed in the wrong way which could lead to problems during the development process. The process model is a great guide to follow for a general idea of how to go about developing a system but should not be used on its own due to lack of important technical details and simplicity.

The object-modeling approach provides the technical details of the system that the process-modeling approach lacks. Object-modeling offers a guideline for how to go about the development of a system by introducing direct variable names and types along with functions names. These objects are represented as having relationships to one another which are represented through arrow notation. These relationships can also be described as 0 to many, 1 to 1, 1 to many, etc. to maximize the impact of the visuals. The downsides to this approach are lack of flexibility since oftentimes, there will be great detail in each object and having no order in how classes/objects flow into each other can also be problematic. This is where you could introduce the process-model to use in combination with the object-model for maximum impact.