# CS 255 Model Application Short Paper

Rachel Iris Aldava

Rachel.Aldava@snhu.edu

Southern New Hampshire University

**Scope**

Before we start on any modeling project, we should determine the scope of our subject. Here, we will be interested in the system our company has been contracted to develop and the data therein. We are not concerned with the type of data associated with pedagogical transfers of knowledge so much as we might be concerned with the data which moves through the system in order to facilitate those activities. Driving lessons occur in the real world and are not processes in our system. We are not designing or modeling the entire business; only a tool that the business uses.

## Process Model Application

If we were to choose to build a data flow diagram, we would need to identify some core element of the system. First, we would need to know where data comes from, what processes are present in the system, where data is stored, and where data arrives at. We would then need to ask ourselves how each element of the system interacts with. Finally, we would need to refine the diagram need by pruning any unnecessary elements or adding elements that we failed to account for, then repeat the above steps. Then, if we wished greater resolution on a particular process, we would need to create sub-diagrams for the relevant processes until we have either reached a suitable level of explanation or even when we have reached the most detailed explanation of the system possible.

On the first pass, I would identify data sources as originating from customers, the DMV, and drivers. The data from the DMV is in the form of updates to the sample test questions or updates to various rules and procedures. The data from drivers would come in the form of available windows for appointments and feedback for the student. The data from the customer would involve account information and specific appointment times. A case could be made that the two administrators also introduce data in the form of adding schedules for employees (and thus, impacts the available appointment times) or has some editorial power over the DMV practice tests, but I would likely choose to overlook these tertiary types of data. While the secretary plays a major role in data collection and organization, he does not seem to be the originator of any data except for temporary passwords for new customers.

The processes in the system could be highly simplified to online tests, in person driving lessons, and customer onboarding. This very simplified number of processes could be broken down into many sub-processes. The next level of in-person driving lessons might feature processes such as appointment scheduling and appointment feedback. The next level of online tests might feature processes such as create new test, pause test, resume test, review test, and refactor test questions. The next level of customer onboarding might feature processes such as create user account, update customer information, and select product plan.

Data storage could be simplified to the database, but destinations for data are a bit more interesting. Data will flow to the customer in the form of available appointments and driver feedback. Additionally, since we must model the flow of DMV sample questions into the system, we may need to ask where that data goes, which would be the customer, however, I personally would omit this for the sake of simplicity. Data will flow to the drivers in the form of scheduled appointment times and updates to DMV rules and policies. Data will flow to the system administrators in the form of DMV alerts and various reports and troubleshooting data.

To fully detail how each element of the system links up to the other components in the system would exceed the page limit guidelines of this paper, however, I have identified the core elements of the system, what data would be collected, and what elements receive what type of data.

## Object Model Application

Let us suppose that the object model we will be creating is a class diagram. One of the first tasks would be to identify the classes by asking what types of objects are in the system. Next, I would think about what types of data about those objects we are interested in storing in variables. I would then look at how the classes would interface with each other in order to determine what methods I would need. Finally, I would spend some time revising by looping back and covering anything I missed this first time around. Another approach would be to ask how the classes relate to each other first, then determine what variables we need once the methods are identified. Such an approach led to an overly complex explanation.

We have several objects in this system. I am not fully confident in my ability to identify objects with a system which has a multi-user sign-on system with various levels of access, however, I will try my best. Objects relating to this role based access functionality could be Session, User, Customer(extends User), and Role. Other objects could be CustomerInfo, Car, Lesson, Actions, SampleTest, SampleTestPool, and LearningPackage.

A User would need to have some basic sign-on information such as a username, a password, pastAppointments, upcomingAppointments, an actionsLog, and maybe an E-mail. All Users would have a Role, so there would be a composition relationship. Roles would dictate user access, so we would like to keep track of what actions each type of Role has access to. A Customer is a special type of user which contains CustomerInfo, a list of SampleTests, a drivingLessonBalance, a classroomLessonBalance, and a true or false value of hasOnlineTestAccess . When someone clicks a button on the website to login, if the user and password is valid, a new Session would be created. All Sessions would have a User, so there would be a composition relationship between the two. We might also be interested in storing the time the Session was created or how long a Session is valid.

CustomerInfo would store basic information such as name, address, and phone number and an actionsLog. Car would store some form of identifier, upcomingAppointments, and potentially in the future service records. A Lesson would have a lessonType, startTime, endTime, participants, notes, an assignedCar (if it’s a driving lesson rather than a classroom lesson), and actionsLog. There could also be a Lesson class with child classes of DrivingLesson or ClassroomLesson. Actions would be generated whenever a user successfully changes data; it would keep track of the time, the type of commandUsed, previousValues, and newValues. SampleTest would be comprised of random elements from the SampleTestPool and might store a map or dictionary of question:(answerList, correctAnswer, studentAnswer) as well as a time the test was taken, a status, and a grade. SampleTestPool might store a map or dictionary of all questions:(answerList, correctAnswer). LearningPackage would store values for drivingLessonCredits, classroomLessonCredits, and hasOnlineTestAccess.

Fully listing out the methods of the model may cause me to grossly exceed the page limit. Let us assume there are appropriate getters and setters for the class variables as well as methods to store and load objects from storage. Throughout the website, there might be buttons which pass through the session object, have their user access permissions checked, then call various actions methods. There are a few important methods which we have time to look at in more detail. An important method would be createAppointment() which would check the participant’s current upcomingAppointments, appointments for the Car if applicable, and if there are no conflicts it would create a new appointment object and add it to the Car’s and participants’ upcomingAppointments lists. Similarly, a function like getAppointments() would return information about the appointment and finalizeAppointment() would allow for notes to be added and transfer the appointment from the participants’ upcomingAppointments into pastAppointments. When a createSampleTest() function is called it would select a set number of questions from the pool of sample questions. answerQuestion() might capture input and update the data in real time whereas finishTest() might change the status of the SampleTest to “Completed” and update the grade variable. createUser() would, again, check the Session for permissions and receive data from the web form which populates the variables for the user, then clone that information to a new Customer object if applicable.

**Process and Object Model Comparison**

Both models I outlined above are useful in describing different snapshots of the system, but there are certain strengths and drawbacks of each type of model. A drawback of both types of models are that they don’t fully capture the user experience when interacting with the system; They might tell us what data is collected or what happens to the data, but they don’t tell us how that data is captured. They don’t inform us about how the interface is designed. They don’t inform us about whether the customer is expected to provide input via tick buttons, slider buttons, or text boxes. They don’t provide us with knowledge about how a user would navigate through the system or even what device a user might be using. The models don’t tell us what operating platform the system uses and only glancingly hint at the specifications of hardware which would be required.

The process model I outlined was a Data Flow Diagram, which was useful in tracking data, identifying different functions that the system would have, and providing information about how each element in the system relates to other elements. Different levels of data flow diagrams can yield different levels of specificity which is useful in either quickly identifying the basic functionality of the system or diving deep into the functionality and seeing what is happening to data on a granular level. Another strength of this type of model is that it can represent clear boundaries of where our system begins and where it ends. A system can be circumscribed within a DFD and can frame what a project needs to accomplish. Finally, a strength of this type of model is that it can be more accessible to a non-technical audience than other types of models. A drawback of this type of model is that it doesn’t explicitly inform us of what that data actually is; it can give us an explanation for what happens to the data, but we don’t know the actual contents of the data nor can we be fully clear what happens whenever that data is divided at any particular node. This type of model can be ambiguous and present interpretation challenges to a developer. Another major issue is that the only clear stopping point of designing the model is reducing a system to its most low-level components, which could be a very large task for more complex systems; prior to developing such a model an organization may need to communicate to an analyst what an acceptable level of resolution would look like.

The object model I used was a class diagram model, which also has some strengths and weaknesses. One of the strengths is the fact that it can clearly list the different object is the system as well as the attributes and actions which relate to those tasks. It can provide a very clear view of the types of relationships between different classes and even the numerical range of objects which exist in such a relationship. The actual model contains a dense amount of information which a software developer could easily interpret and use that as the basis for much of the software framework. This model does have several disadvantages, however. Because this type of model contains a dense amount of information, it could be slightly difficult to interpret by a non-technical audience. Developing such a model also necessitates a working plan for how the system will operate, and as such may be subject to numerous revisions throughout the planning process and the life of the system. Another potential issue with software development under an agile framework when using a class diagram is that there may be pressure to spend time modeling aspects of the system which may not be the primary focus of the current sprint, and as such could introduce distractions. While this model may be able to list what types of data are stored in the system and what classes accept which types of data, it does not explicitly outline where that data comes from, how a method uses that data, or where that data arrives at. Tracking data flow using a class diagram would involve disjointedly jumping to various places on the class diagram, a bit of guesswork, and a fair amount of mental bandwidth keeping track of each node; it is not as suited to modeling data flow as a data flow diagram.

The two models have tasks which they are well suited to, they both have some level of overlap in what elements of a system they describe, and they can both have weaknesses in areas which neither one of them were designed to excel at. They are both valuable tools in capturing aspects of the system from certain viewpoints, but may not be fully sufficient in describing a system to all parties involved.