# CS 255 Model Application Short Paper

Nate Dukes

nate.dukes@snhu.edu

Southern New Hampshire Universit

## Process Model Application

Process models are one of many different ways to visually interpret and represent software systems. They utilize various different visual elements common in business to make the understanding of the system easier. These elements consist of the usual UML diagram elements, arrows, circles, all the other basic shapes you know and love. Process models are most often used when there are not so many branching pathways in the software. They are most effective when the logical chain of events for the aforementioned software is clearly connected, with little deviation. These models usually have multiple levels of abstraction, so as to give a better idea to the viewers of what the final product might look like at the end of a design phase. They are also equally useful for when the system has already been developed.

In order to implement a process model for DriverPass, we first need to understand what the software is going to be doing each step of the way. We can check in with the client to see if we are on the right track along the way. DriverPass is an online website that has a basic function- provide tests and scores to students. On top of this layer we have instructors, potentially communication between the two, and a final administrative layer that oversees the ongoing and effective use of the software. Encapsulating it all we have some added levels of security, such as login management and encryption. These features are all integral parts of our software’s process, so we will add them to the model.

The model might begin with the customer, followed by them logging in and security checks. Afterwards they might be presented with the various site features, such as studying, taking tests, communicating with their instructor, all of which would have their own branching pathways across the site and the model alike. The instructors might have a more complex branching path to implement into the model, where in they have a few more options, such as leaving feedback, grading tests, or otherwise do their duty to the student; in this instance we would need to model another branch on the model. And similarly, the admin will have other options and necessary branches within the model. There are a lot of different paths to take.

## Object Model Application

An object model, in the context of computer science, is used to model a program’s functionality visually. This usually manifests in a big image that describes each of the classes and their methods, all their components, and the requirements for each function of the software. They are in depth, and describe each aspect of each object type, and demonstrate the connections between the objects and how they are to interact. This is distinctly different from process models, which are on a more “real-world” layer of abstraction, where as object models more directly describe the functionality.

DrivePass would need to implement quite a few different classes to the model; among them, the most important would be a basic user class, an administrator extension, an instructor extension. These classes would have all sorts of information tied to them, such as login information, name, date of birth, history, scores, and all other manner of relevant information. The admin and instructor extension objects might have less or more available methods, impossible to know without actually designing an object model. These methods and classes would be linked intrinsically with the functions of the site, such as test taking, a test object, or study object- all tied together. How these might look is again, an unknown without developing one, but it is clear that there would be a lot of branches from the user types and the tests.

## Process and Object Model Comparison

I believe that an object model much better represents the software for DrivePass. DrivePass seems to know exactly what they want, exactly what classes and objects would be needed, and while there are many, it is mostly clear what methods and data each object would need. A process model, in my opinion, is more useful for the design phase of the software development lifecycle, where as the actual development of the software is more fitting with an object model. Object models clearly indicate the functionality and necessary connections between each class on a programmatic level incredibly clearly, where as process models might only offer an elementary understanding of these connections in comparison. An object model is also much more in depth; while this can be a bad thing, I think that DrivePass is at the perfect size for it to be more easy to understand to an outsider- exactly enough details are necessary and sufficient such that an object model is not overburdened, but also not unhelpful in developing the system.