COMP 3004: Deliverable 3

Last Paragraph of Design

Cole Dorma

Overall, our application design has quite low object-oriented coupling. Furthermore, when we were first developing the application, the main goal was to produce a working product with temporal coupling not being a priority but still in the back of our minds. In the beginning, generally, our temporal coupling was at a medium level but once we developed a working product, we quickly worked towards reducing our temporal coupling, resulting in our temporal coupling being at a very low level as of now. The only class that could be considered a small case of temporal coupling would be in the ScheduleGenerator class, in which many operations are happening at once; the adding of the course objects to the schedule objects, the comparison of the schedule objects to the commitments objects, and coming to a final result of the schedules that will be displayed to the user. But the purpose of this ScheduleGenerator class is to basically bring the entire program together and give the final result to the user, in which each operation and action consisted in this class is required. More in depth, our design has low subclass coupling as well. Most of the relationships between the classes are composite and don’t consist of many inheritance relationships. This adds to the fact that most of our classes that are higher on the ‘composition-hierarchy’ do not need to know the state or any information pertaining to the lower level classes. For example, a schedule can be created with just courses and does not need the separate sections and subsections, there just wouldn’t be as many options in the final result. The composition consists between the Schedule class and the Section class, the Section class and the Subsection class, and the Sections/Subsection classes and Timeslot class.

Our system would accommodate and adapt to changes very easily due to the low object-oriented coupling explained in depth above. Generally, if our system needed to be altered, it would evolve easily. More specifically, due to the fact that our system has very low temporal coupling, any changes that would need to be made would be easily adapted by our system because any main-idea actions that are being performed are overall, very separate from each other. Also, if any altering was needed, our system would adapt easily because of our low subclass coupling as well. The majority of our classes are composite relationships, in which they are quite separate from each other and don’t require much communication from the parent to the child. The fact that not much inheritance is used, allows for our program to adapt easily within each class itself. Lastly, we chose the V-Model for our design specifically for the ease of adaption to changes and also for flexibility, maintenance and for agility.

One way that our system would need to evolve in the future would be for adapting our application for other universities. For instance, it would be possible for another university to not have the same course identifying format as Carleton (i.e. COMP 3004, COMP3000), or for another university to not have the same course format with course sub-sections, sections, requirements and tutorials as Carleton. Our applications system would support these changes by simply changing the formats in each class needed along with changing the course structures to the appropriate course structure of the university/college. These could be done without any ‘waterfall’ type effects due to the composition of most of our classes, the little inheritance used throughout the program, and the encapsulation of each class.