COMP47480 Practical 2: UML Exercise

Sukrat Kashyap (14200092)

13 February 2018

1 Work Done

In this practical, we were given to create Use-case model, Domain model and Interaction model of a library system. The library had books and journals which could be borrowed by the members of the library (Students and Staff). There a restriction on the number of items each member could borrow. Journals could only be borrowed by the Staff members. There were two types of loan for the books namely short-term (4 hours) and long-term (4 weeks). We had to represent the system in 3 different model representation. We first created use-case model for the library system which was done by finding different actors (student, staff, and member) and use-cases such as borrow journal, check availability and common use-cases such as borrow book, return, and display. After the use-case model, we picked out nouns suitable for making classes like Student, Member, Staff, Copy, Book, Journal and created links between them. We then created an interaction diagram which showed us scenarios of communication between the classes and had to add a new booking system to our domain model. We then were told to add a new requirement of notifying the members if they had the book or journal for too long. To which we added a booking system which notified the member when the booked in all of our models.

2 Reflections

We started modelling our library system by first created use-case model based on the description. For which we had to find out the actors and use-cases. We thought of actors as the staff and students. Our use cases were borrowed, return, check availability and display. Then we tried to reduce the functionalities by excluding and include generalization. Student and staff were generalized using the actor member. This reduction of both the actors to members was because they shared mostly all the common functionalities except borrowing journals. All the use cases were included in the display because they shared functionality. An actor performs display then borrows/return and then displays new changes in the system. Borrow journal use-case was given separately to staff as only they are allowed to borrow journal. A common return was used instead of return book and return journal as by using simple logic as to when a student cannot borrow journal, it won't be able to return it either. Hence common return for both made sense as they can only return what they borrow. This form of representation

seemed quite useful as it was able to showcase various parts of our system in more readable format. A paragraph of description is hard to understand in one go. This form of modelling enabled to look at the system in much user-friendlier format. Regardless, it failed to showcase much tinier important detail such as students are only allowed to have 3 books and the types of loans of the book available.

The second form of modelling which was done using the description and the use case model was a more of a class diagram that would have been implemented when actually writing the code. It was done by first identifying the nouns. In our case, we thought of Student, Staff, Member, Book, Copy, and journal as our nouns. The second step involved finding the relationship between the same. Student and Staff inherited from the members. This was thought as library system should just care about the member and the limit imposed on them should be handled by their concrete classes student and staff. The reason Copy was created as the book could have multiple copies with a different type of loan but all the other detail would remain the same. Hence N:1 relationship was identified in the Copy vs Book. Since journal didn't have any copies so the copied class was omitted from the journal. The borrow and relationship were straightforward from our use-case model. Comparing the two models above, we see use-case model tends to just identify the actors and what they did irrespective of how they did. Whereas, domain model representation adds a technical aspect of the system. This could be easily seen from the attributes corresponding to each class. Both representations are quite similar to domain model being a bit more technical. Domain model could also show them the fine detail of the limit on each student and staff by assigning a constant value of 3 and 12 respectively to the limit attribute they both shared. We decided to skip such fine details. But it gives us insight into the level of detailing without making the model too complex can be achieved. Although, it seems like a better option than domain model. I feel Domain model has the potential to become too complex real quickly than use-case model.

The third form of modelling was Interaction diagram which tries to picture the exact interaction between the general ideas but more in sequential form. We took member, booking system and item to show the interaction. It shows the sequence of getting items from booking system which in turn called each item and their info. It returned a list of items and their status. Borrow item can be issued later passing the exact item one wants, the result was the confirmation. Notify was included as this was the third requirement which would be done timely until the user returns the exact item. Our sequence diagram was not as perfect as it should be. But I believe it did represent the sequence in the steps for e.g. return can only be done if borrow for that item is issued and notification will be done until the item is returned. Notification step was also added to other diagrams as an addition which didn't seem to be much of a hassle.

After building all the three models, one can see that the use-case model represented the system in a vague manner missing all kinds of detail and giving a very high-level overview. The domain model gave simple and fine details and also somewhat the

implementation of the same whereas sequence diagram gave those implementations and relation in the domain model a sequence in which to be performed. But when it comes to usage of these above three models in real life, I believe it is not as useful. I believe these models were highly required and recommended in the past because of there were none to very few development tools and required a great deal of work to manage and learn about the new system which was already implemented. In today's world, we have so many advanced development tools such as IDE, debugger etc. that it has very much diminished the use of modelling. Also, the changes in the project are very rapid making models before coding redundant. A great example to show this case would be finding the implementations of an abstract class. Before IDE's finding a class in a big system which implements certain class was extraordinarily time-consuming. Hence, the models such as UML helped to figure them out faster. But in today's computing finding the implementation is a task of a mere single click. With that being said, simple to represent the whole system still exists in form of vague pictorial drawing rather than following a standard.