Group report – SAW Accommodation office

Ashley Pearson - 20001030

William Robertson - 20005074

Sonia Tadlaoui - 21039395

Strategy planning:

The goal of this project is to create an App for the UWE Accommodation office that would allow the system users to log in using a username and a password. Depending on the clearance level and whether the user is a Manager, Warden or Admin different actions would get enabled or disabled .The manager is only able to add or delete leases. A warden on the other hand can only change the cleaning status of a room to either clean, dirty or off-line. A special user which is referred to as Admin in the SAW system gets access to all the application features, all users are allowed to view room's details from room number, hall number ,occupancy and cleaning status , rental rate, lease duration to the student name .

In order to meet the UWE Accommodation office requirements the "Agile" software development Methodology was applied, this method is known for prioritizing client's satisfaction as well as having frequent deliveries, to not only keep the client in the loop but to also tackle issues as soon as they arise ,benefits of having short but daily meetings. Scrum, a type of agile methodology was used for the development of the SAW accommodation office application .For this specific methodology the group members are called the scrum team, where everyone can work on everything. The scrum master, leader of the team helps the scrum team stay focused on the tasks at hand by removing distractions and organizing the workload.

The whole development process can be summarized by multiple scrum sprints:

First comes the creation of a product backlog by the scrum master , this backlog helps keep track of all the tasks that need to be executed in order to successfully finish the whole project .Second, comes the sprint backlog that would basically get all the tasks that would have to be executed during one sprint cycle only . Not all tasks can be selected as some aspects of the project were judged to be more of a priority compared to others.

Four scrum sprints were needed for the development for the SAW application each sprint took fifteen days.

I. The first scrum sprint :creation of the Graphical user interface

1. Planning and designing:

the planning side of the GUI scrum sprint started face to face with the whole team present .the Graphical user design was first sketched before being created, with a defined idea on what the final system should look like each member of the team was able to start working on their part .

2. Implementation :

Following the designing phase, comes the development part creating a Graphical user interface using python's TKinter module from the sketch designed in the previous phase. The first step was creating frames, buttons and text fields then came the second part of the task where the treeview was populated in the GUI as well as call back function were implemented to make the buttons function

3. Testing and Retrospective:

The next part after the implementation, is to test whether every action button on the GUI would interact with the treeview. When adding a lease to the treeview it managed to create a lease in the treeview table correctly, when it comes to deleting a lease. If a lease is selected the lease is deleted from the system properly, setting most data values back to 'Null'. If update lease is run then the pre-existing lease in the table changed to the values the user inputted, without changing lease num. If Review Lease is clicked, then it will show the correct lease details of the lease selected in the treeview. GUI is fully functional and works as intended, the only thing it could be improved upon is only using one geometry manager like grid, rather than pack so it is more effective.

4.

II. Second scrum sprint : Creating the login page:

1. Planning and design:

For the Login, many ideas came to mind. For the system there needed to be a way to determine what a user could access. We had 3 current users, Hall Manager, Admin and Wardens (one for each hall). For the 3 different user types, there has to be a way to divide them into different system views. The login system was easily created via 5 Frames, 2 text boxes and 1 button. More frames were used than needed for aesthetic purposes. The two text boxes were made to get data from the user as well as manipulating it to allow a specific user to access a different view of the system.

2. Implementation

From the planning, The Login page already had a set out design so the implementation of the Login Page was straight forward. With new found tkinter techniques the pack positioning method was used to create the layout for the GUI. Each Frame was sized according to the screen size for ease of access. The text boxes, labels and other features were implemented just as specified in the planning stage.

3. Testing

When testing the Login function, many validation checks apply. As well as allowing the 3 different user types to be directed to the correct system view. First the username and password for these users were hard-coded, later to be converted database. So an easy if statement was implemented to compare the user's input with the set username and password, if it were not a match error messages would arise. Whereas, if the user was to clear the validation checks, the user's username, password and clearance level would be accessed and depending on that user's clearance level is they would be directed to a different system view. That is how the system would differentiate between Hall Manager, Warden and Admin.

4. Retrospective:

Once showing the Login function to the group, a database was immediately implemented so that the user's details were no-longer hard-coded. Since working on the main page GUI, in retrospect The

Login page could be a lot more efficiently coded and less frames were to be needed and the whole code would look a lot nicer.

III. Third scrum sprint: Creating the python classes:

1.Planning and design:

With the existing UML diagrams from the initial stages of the project the team was happy to continue largely as planned with only minor grammatical errors and type errors, such as getHalls returns a type of Halls, as opposed to the correct, hall. Initially barebone classes were created with only structural components and the bare minimum methods defined. Following a planned scrum meeting it was agreed that the classes were for the most part ok to be created and built upon, with any small errors/inconsistencies to be ironed out within the implementation stage. Classes were initially designed to utilize both composition and inheritance based on scenario, with setters and getters being shared amongst classes to reduce redundancy.

2. Implementation

Building upon the aforementioned barebone class structures the team began to flesh out the class attributes and build upon the basic getter and setter functions outlined during the planning stage. Throughout the implementation process it became clear that certain attributes were redundant, repeated code and would be better implemented in the parent 'User' class, doing so enabled the child classes to drop rarely used functions such as 'getWardenID' and user super to inherit the more comprehensive 'getID' function. Likewise, the 'getHalls' function was ultimately assigned to 'User' given that all user levels can view all halls, returning 3 separate lists as initially planned only served to allow for errors to arise, instead the level of editing allowed for each user was assigned as a class attribute with 'TheWarden' and 'HallManager' themselves, with functions within the GUI determining a user's privileges, but ultimately on only one list as opposed to 3.

3. Testing

Testing was an ongoing task as the classes continued to grow and adapt to the GUI's demands, especially the Treeview and lease creation. The Primary interest throughout was to ensure the OOP classes had retained integrity and were consistent, on multiple occasions changing the Lease class would break the room objects associated with it for example. Toward the end of the implementation the classes were solid and faced little issue, toward the end of development an error arose in which the lease number of a room was no longer incrementing correctly, the team reviewed the code and found the error with the rooms leaseID attribute, and all errors were resolved. Due to the level of error checking within the program itself it is highly unlikely bad data reaches the classes themselves and as such the team stuck to using loosely typed data to allow for potential edge cases in future, geographical area codes as part of a phone number for example were of note and the reason we chose to not type Telephone number to int.

4. Retrospective

On reflection, the classes created served their function from inception through to deployment, with only structural changes required within inherited classes. In retrospect it is possible that composition could have been used more effectively to streamline code further and it is possible that the student record class could have been 'absorbed' as attributes into the lease object, thought this arguably offers very little benefit.

IV. Fourth scrum sprint: Creating the database :

1. Planning and designing:

The database was planned and designed before the python classes were implemented in the program, the initial diagram contained three tables (hall, room and lease) tables, those t were deleted from the program when the objects of classes were added and proved to be a more effective way of storing the data, the user table on the other hand was designed to hold all the system's users data to make the log in happen.

2. Implementation:

The user table contains three attributes username, password and clearance_level, this table was populated with rows in the form of list of lists where each object's getters represent the list elements, once the table populated a select statement was executed to get the username and password from one of the user table rows depending on the user input in the data field of the logging page

5. Testing and retro perspective:

In order to know if the database is storing the data it needs to store a printed select statement would be extremely helpful for testing purposes as well as printing out in terminal whatever the database is collecting from the table

Team communication and simplicity of the program:

During the winter break; when all of the team had to split up, working online was a real challenge for the team to overcome and work efficiently. However, the group managed to persevere with daily calls and long discussions over various social media sites, as well as making use of programming-oriented websites such as GitHub to commit any changes made to the code. Since we all have re-converged the final coded product, is very simplistic and easily. The bulk of the code uses classes and get methods to transfer data throughout the program, these classes are what makes the code flexible.