IT5014
Programming Principles
Project - Jeshua Hertzke 20210843

Part 1 Reflective Journal:

Software Development Life Cycle Process:

Planning:

In the planning process we need to decide which development method process we want to follow. I feel that following the Agile Model would be best due to the small timeframe I have to develop a finished product and with me being the only developer working on this project. Agile Model is also more cost effective than the Waterfall Model in small projects as they can be implemented quickly with less planning and documentation required. As this is a small program to develop, it is more cost effective to develop the program ourselves rather than purchase or outsource the program. The client needs a prototype to create a connection between the staff members and the help desk with the issues the staff members have, so I need to design the program that accommodates for both and is user friendly. I need to set a timeframe for steps of the SDLC process to ensure the program is presented on time. I feel it is important to prioritize the testing step as it is very important that the system works. It is a prototype, so construction refinements may need to be made once presented to the client.

Time Plan:

Planning - 2hrs

Requirements Analysis - 2hrs

Solution/Detailed design – 3hrs

Construction – 2 days

Testing – 8hrs

Assessment compilation/submission - 3hrs

Requirements Analysis:

Functional Requirements:

- A staff member needs to submit a ticket to make a request for help and to register their details for a response.
- The help desk needs to be able to view all tickets, to respond to tickets raised by internal staff and reopen tickets if needed.
- The system will need a minimal input constraint, so the help desk has enough information to resolve problems effectively.

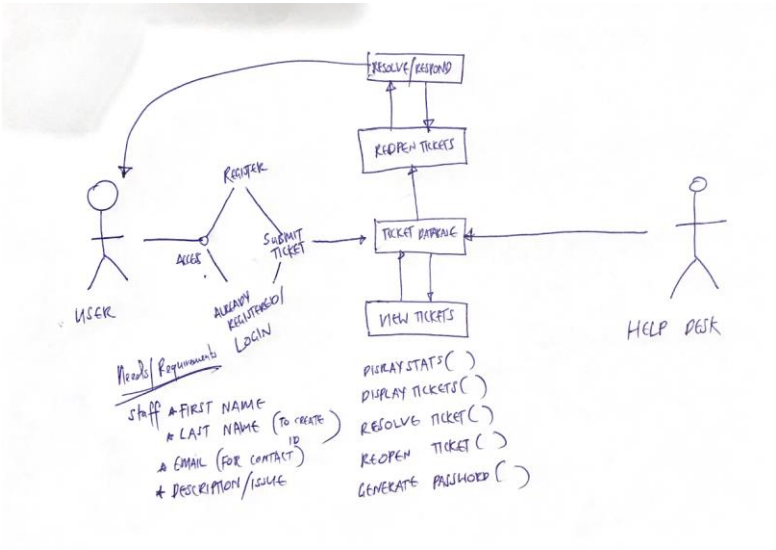
- There needs to be a staff registration process in the ticket system for the help desk to refer to for staff details.
- The program needs to be able to automatically generate a new password for the staff members if they request it in the description of the ticket.
- There needs to be a way for the help desk to keep track of all tickets, whether they are resolved or not and a way for one or all tickets to be displayed.

Nonfunctional Requirements:

- Security/Reliability - The system should be secure. To ensure this, only authenticated members can submit a ticket. If a staff member is not registered, they cannot submit a ticket. Having constraints by setting variables to private with getters and setters will help keep the system secure.
- Performance – Submitting tickets should be a quick process and the system should take less than a couple of minutes to receive the ticket to ensure good customer service. The entire process will need to be simple and reliable.
- Rules and regulations - You cannot store private information about clients permanently, for example credit card details. They should not be stored permanently but temporarily stored through memory, so the system should accommodate this.

Solution Design:

I initially sketched out a map of what would be required in the program.



The system needs to take both staff and helpdesk input. The staff members will need to register their details to create a Staff ID for them to submit their ticket and staff will need to access all tickets to respond and reopen tickets if needed. The program can start at a dashboard which asks what the user wants to do and then starts its path whether it is a staff member or the help desk.

Detailed Design:

OOP:

Creating objects/instances will help with the coding as a single method can be used for all objects/instances. This will help reduce the amount of code required and help keep it clean and understandable. The following objects and their attributes would need to be created:

People: First name, last name, email, staff ID

Tickets: Ticket number, description, response, ticket status

To ensure the system will work effectively and securely the following classes will need to be made:

- **Staff Class:** firstName, lastName, email, staffID
- **Ticket Class** (Ticket: Staff): static counter, ticketNumber, description, response
- **Ticket Stats Class:** ticketsCreated, ticketsOpened, ticketsClosed
- **Password Generator Class:** Static method to generate new password.

As the Ticket Class uses all the attributes from the Staff Class it would be best for the Ticket Class to inherit the Staff Class. I could design the system without the Staff Class but doing it this way makes the system much cleaner and organised and helps build a database of staff. Dividing into classes will also help future developers to easily navigate through the code.

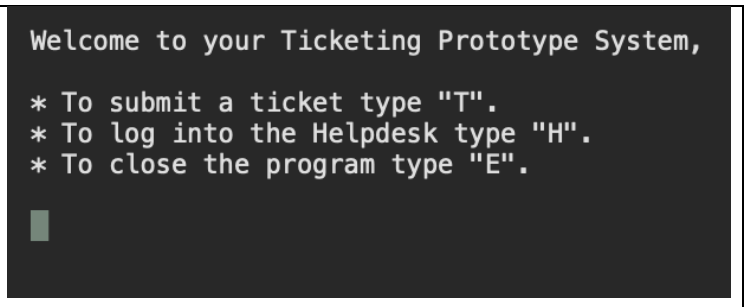
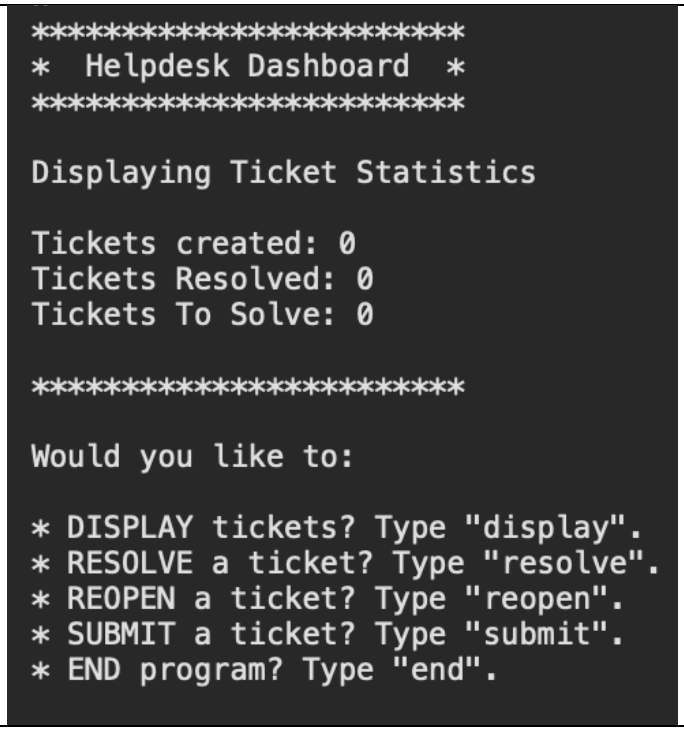
We will also need the following methods/Constructors created:

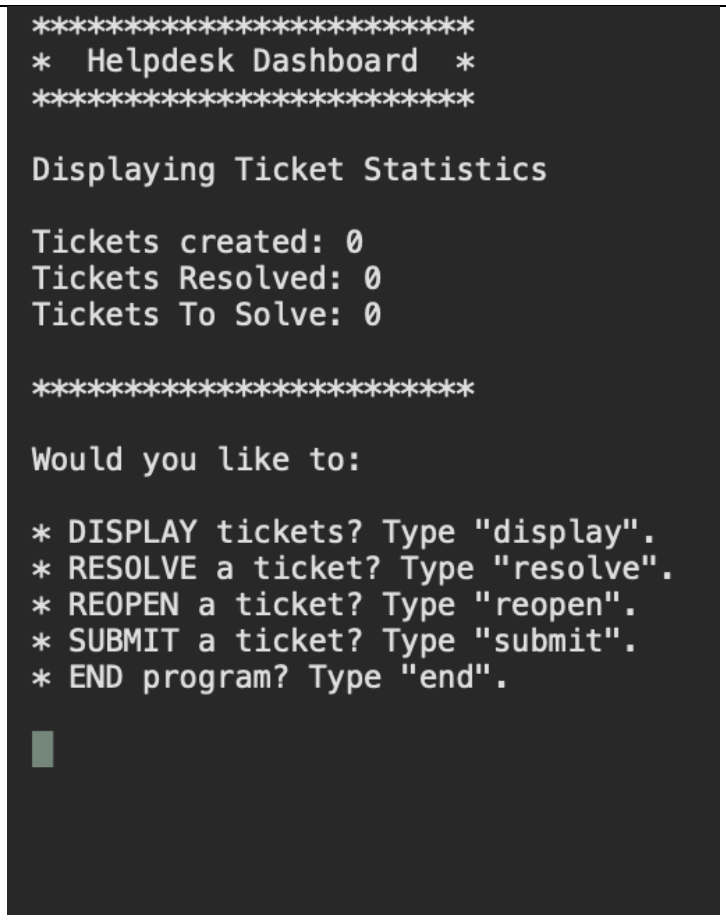
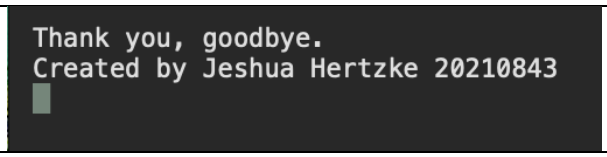
- public Staff(string firstName, string lastName)
- public Staff(string firstName, string lastName, string email)
- public void DisplayTickets()
- public void RespondTicket(string response)
- public void ReopenTicket()
- public int getNum()
- public static string NewPassword(string staff, int ticketNumber)
- public static void DisplayStats()

Using constructors will help initialise objects with attributes faster. Using these methods in the main code will help keep the code to a minimum and avoid repetitive code.

The program will need to greet the user and ask them what they want to do, either submit a ticket or view tickets to resolve. We can do this through IF statements. The user may want to submit more than one ticket or resolve multiple tickets so FOR and WHILE loops and will need to be used respectively. I may need to also include TRY/CATCH for exceptions to prevent the program from crashing as there will be a lot of user input. Because the system will be storing personal information it needs to be secure, setting appropriate variables to private with properties for access will help provide constraints and restrictions. There also needs to be a static int counter that increases every time a new ticket is made, this will be done in the Ticket Class.

Below is a design of the program:

This is the design of the Home section, this prompts the user to input what they would like to do. Here they can submit a ticket as a staff member, log into the Helpdesk or end the program. Inputting **"T"** will prompt the user to submit a ticket. Inputting **"H"** will take the user to the Helpdesk Dashboard. Inputting **"E"** will display a thank you message and terminate the program.	
When the user selects "T" the program will ask if they are a registered staff member. If they are not they will be asked to register which will automatically generate their Staff ID after which they can finish submitting their ticket. If they are a member, the program will ask for the description of their issue and if they would like to include their first name and email in the ticket. Once the ticket is submitted, they are asked if they want to submit another ticket. If not, then the program returns to the Home section.	

Here is the design of the Helpdesk Dashboard when “H” is selected. The program automatically displays the ticket stats and the options for the user. Once the user has completed the option the program will return to the Dashboard. **Display** will show all tickets that have been submitted in order of the ticket number. **Resolve** will ask the user which ticket number to resolve and what their response is. It will then print TICKET NOW CLOSED and go back to the dashboard. **Reopen** will ask the user which ticket number to reopen, make the change, print TICKET NOW REOPENED and go back to the dashboard. **Submit** is where the user can submit a ticket from the Helpdesk end. **End** will close the Helpdesk and return to the Home section.	
If the user selects to close the program, the program will display a thank you message and then ends.	

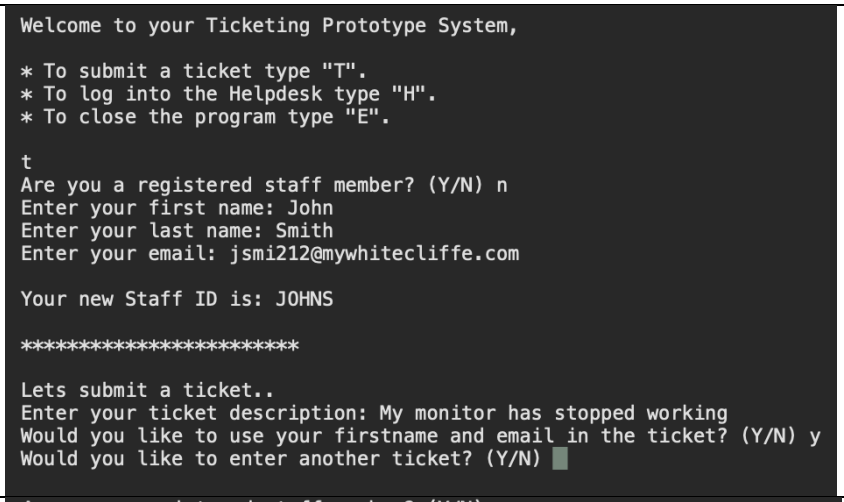
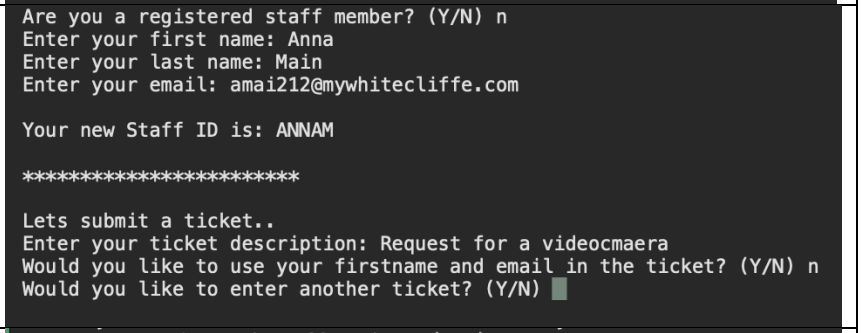
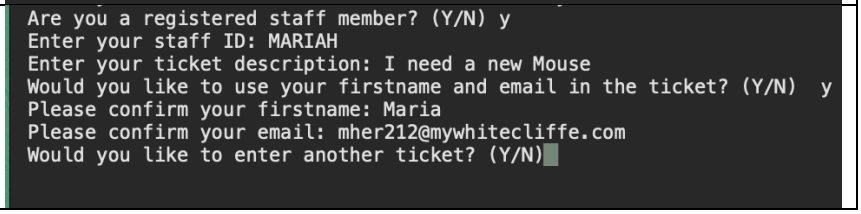
Construction:

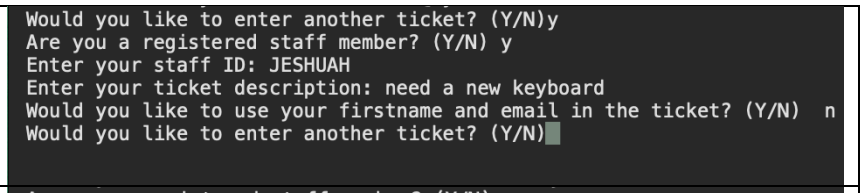
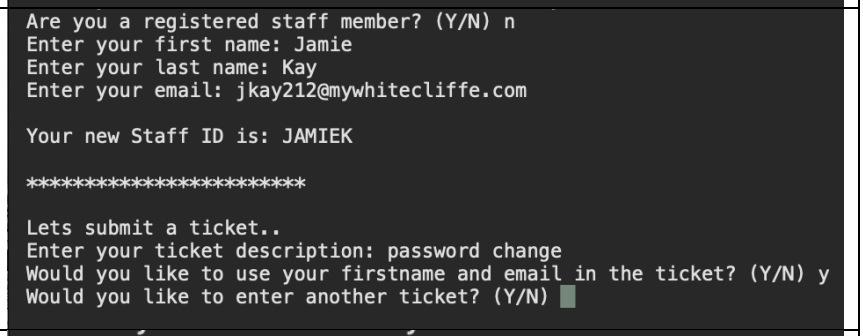
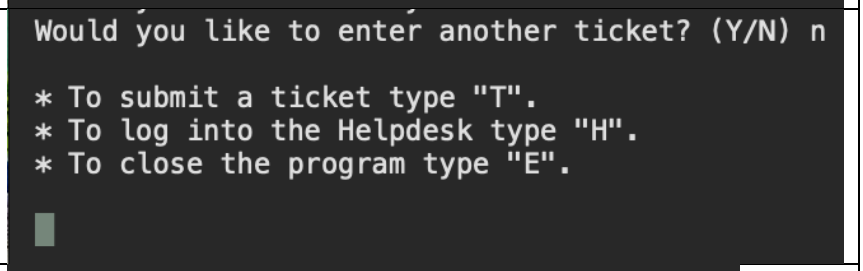
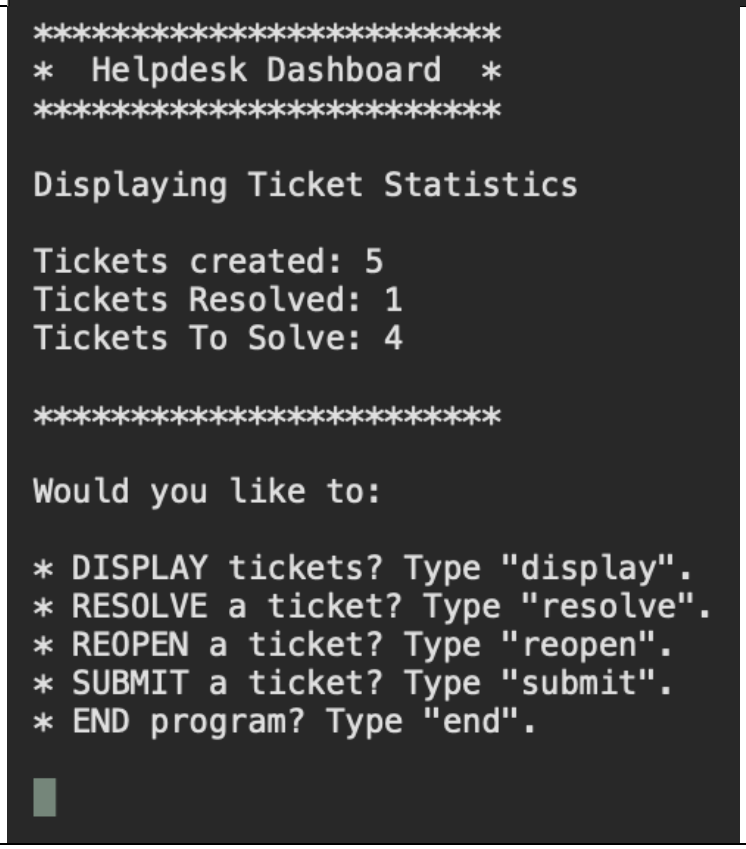
In starting the construction of the project, I created two copies, one being a master and another for testing purposes. This was done so the master copy was always clean with no bugs and working effectively, I would write the code initially in the test copy, check it works, try to simplify it and once it is good, I would transfer it into the master copy. If I were working on a portion of code and moved to another portion, I would leave comments about what needs to be done. All work was done through Visual Studio to help with error detecting and debugging.

Testing:

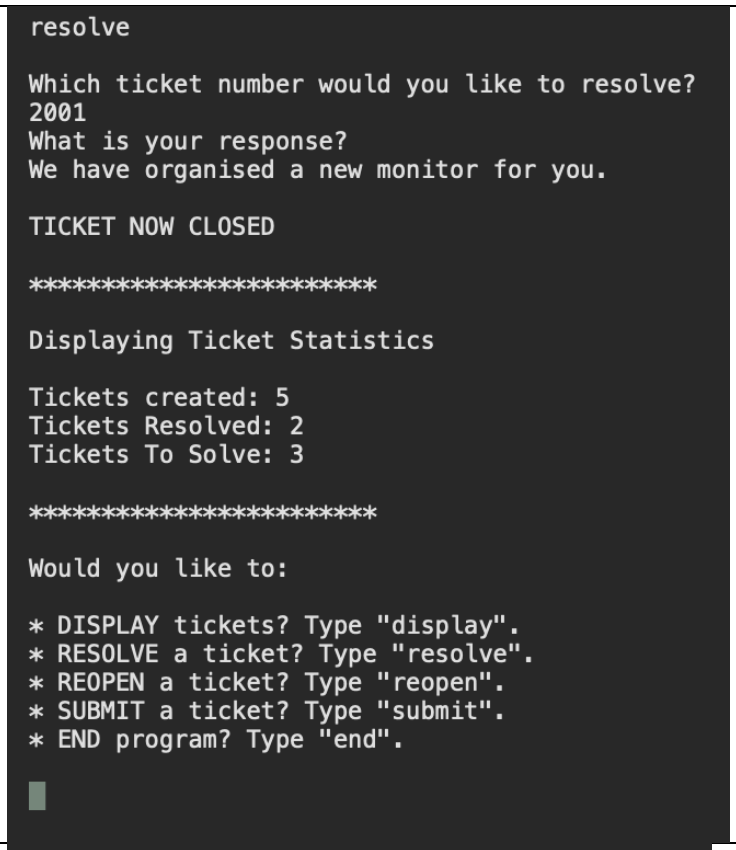
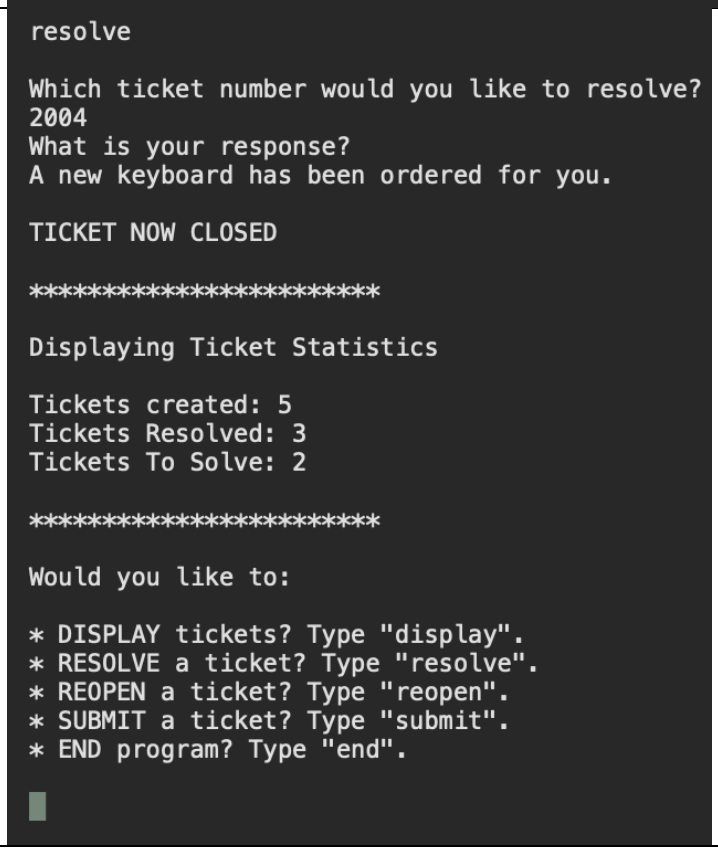
In the duration of construction, I would always routinely test the code as I found not doing so would build up mistakes and bugs. Once the code was finished, I spent many hours rigorously testing the program through functional testing by creating test cases and adding, resolving, and reopening many tickets with some tickets having all information and others with the minimum. I tested the password generator and realised It would only accept the phrase “password change” if it was lower case. I used .ToLower() and .Contains for the method to work more effectively. I also data type tested the program, for example I would input an int when a string is requested to see what happens at each stage of the program, if an error occurred, I would change the code to handle the exception. I also ran the program for other users to see how easily they navigated through the system any errors they expected. Through testing I removed any unnecessary code to reduce the amount of code and added notes to the code for reference. I spent a lot of time ensuring the system worked correctly.

Below are screenshots of testing the program to the assessment criteria:

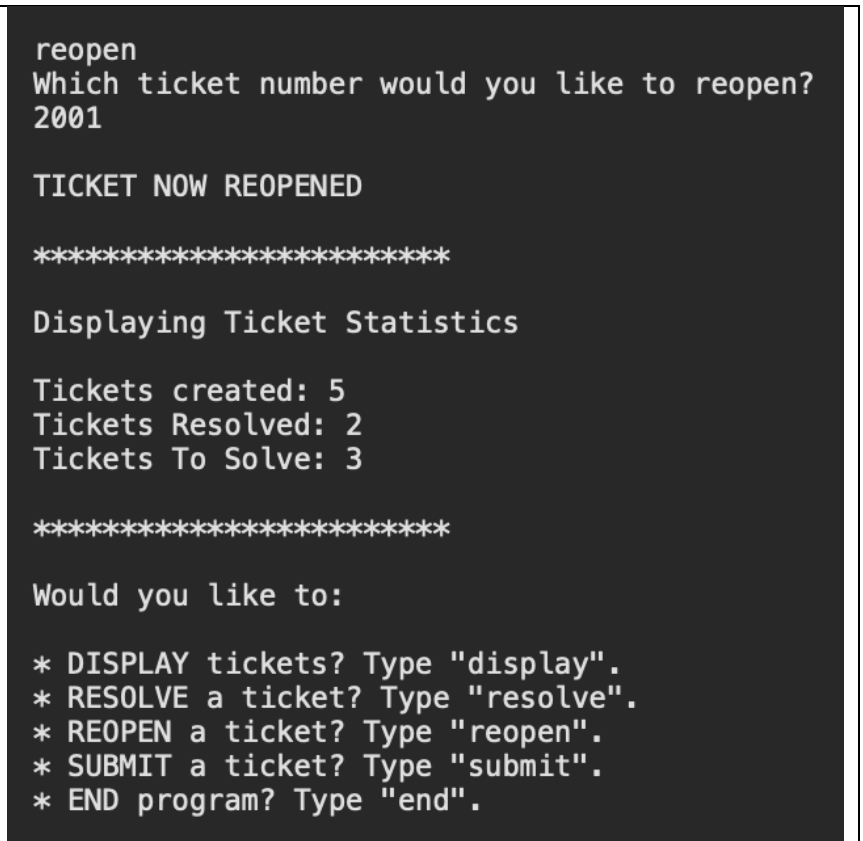
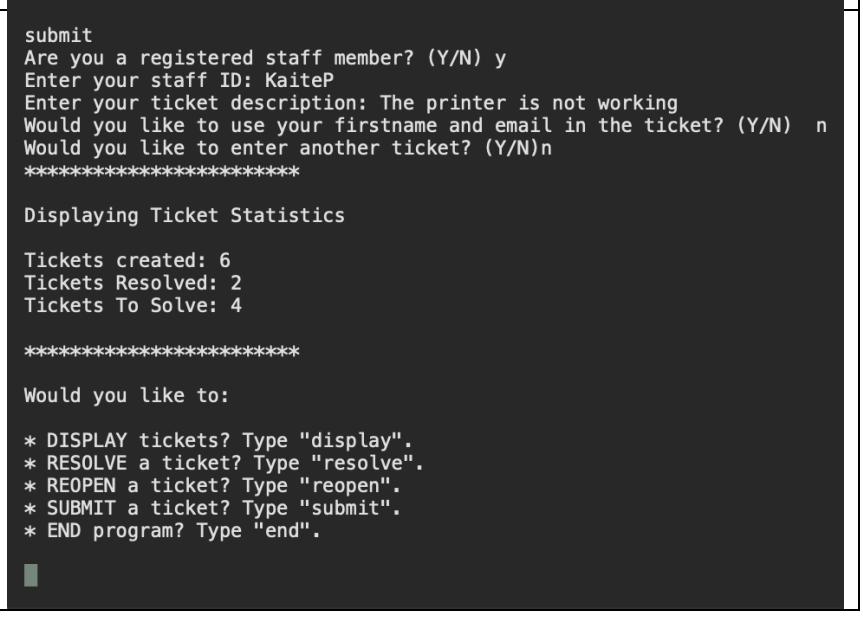
Submitting Tickets	
Submitting a ticket as a new member and including first name and email in ticket.	
Submitting a ticket as a new member and not including first name and email in ticket.	
Submitting a ticket as an already registered member and including first name and email in ticket.	

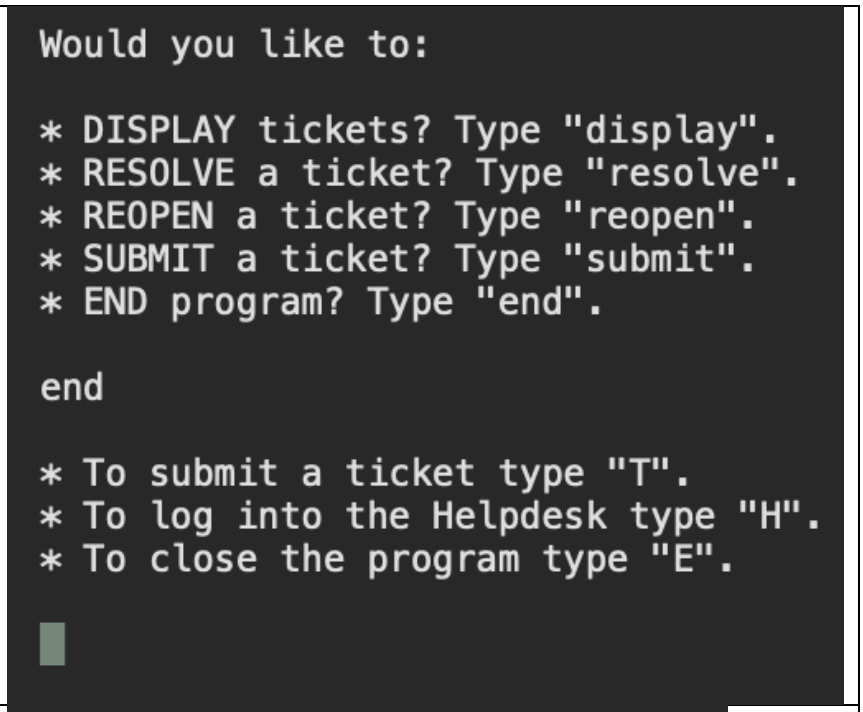
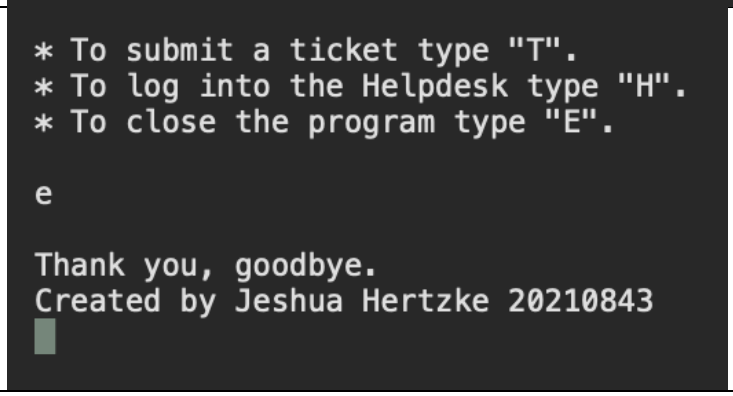
Submitting a ticket as an already registered member and not including first name and email in ticket.	
Submitting a ticket as a new member and including first name and email in ticket where “password change” is in the description.	
User has decided not to add any more tickets so the program returns to the Home area. Input **“H”** for the Helpdesk.	
Helpdesk displaying all ticket stats and options for the user to choose from when first entering.	

Display selected. Helpdesk displays all tickets created and their details. Notice Ticket Number 2005, because the description contained “password change” the program automatically responded with a new password generated to the assessment criteria and then the ticket status is set to closed. The Tickets stats are updated because of this. Options are automatically displayed again asking the user what they would like to do.		

Resolve is selected, Ticket Number 2001 is selected and a response is made. TICKET NOW CLOSED is printed Ticket Stats are displayed again. Tickets resolved are increased. Tickets to resolve is decreased.	
Resolve is selected, Ticket Number 2004 is selected and a response is made. TICKET NOW CLOSED is printed. Ticket Stats are displayed again. Tickets resolved is increased. Tickets to resolve is decreased.	

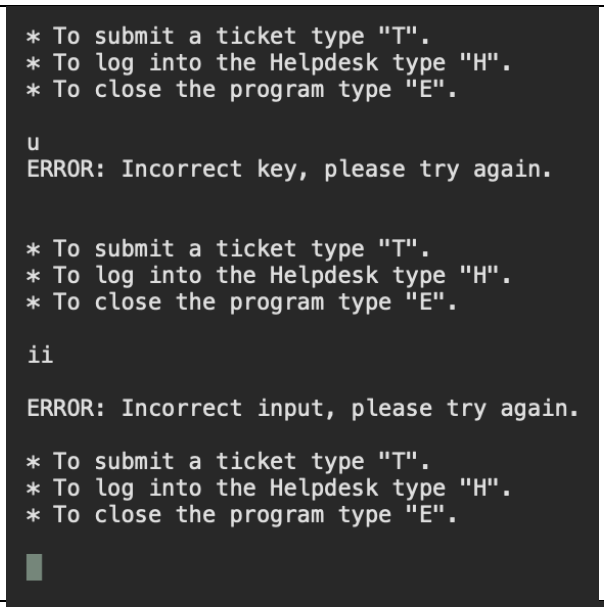
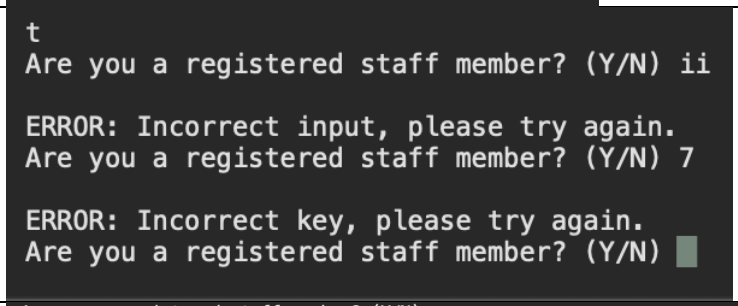
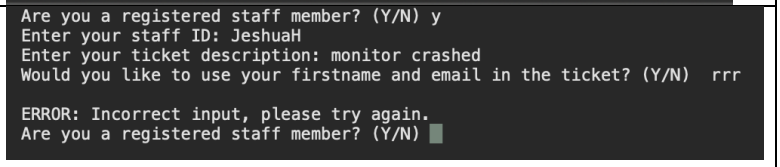
Display is selected. Note this now shows the tickets that have been resolved earlier.	display ********PRINTING TICKETS******** Ticket Number: 2001 Ticket Creator: John Staff ID: JOHNS Email: jsmi212@mywhitecliffe.com Description: My monitor has stopped working Response: We have organised a new monitor for you. Ticket Status: Closed Ticket Number: 2002 Ticket Creator: Not Specified Staff ID: ANNAM Email: Not Specified Description: Request for a videocmaera Response: Not Yet Provided Ticket Status: Open Ticket Number: 2003 Ticket Creator: Maria Staff ID: MARIAH Email: mher212@mywhitecliffe.com Description: I need a new Mouse Response: Not Yet Provided Ticket Status: Open Ticket Number: 2004 Ticket Creator: Not Specified Staff ID: JESHUAH Email: Not Specified Description: need a new keyboard Response: A new keyboard has been ordered for you. Ticket Status: Closed Ticket Number: 2005 Ticket Creator: Jamie Staff ID: JAMIEK Email: jkay212@mywhitecliffe.com Description: password change Response: New password generated: JA7D579 Ticket Status: Closed ********** Displaying Ticket Statistics Tickets created: 5 Tickets Resolved: 3 Tickets To Solve: 2 ********** Would you like to: * DISPLAY tickets? Type "display". * RESOLVE a ticket? Type "resolve". * REOPEN a ticket? Type "reopen". * SUBMIT a ticket? Type "submit". * END program? Type "end".	

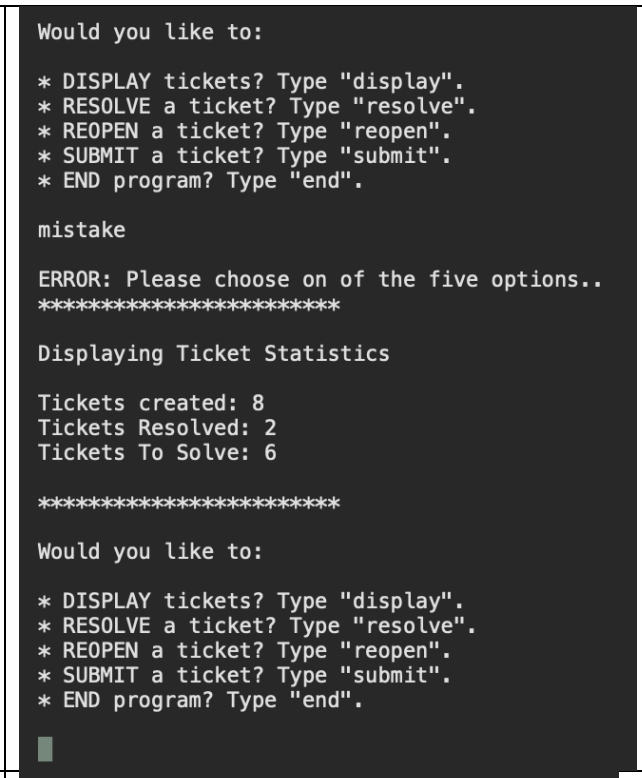
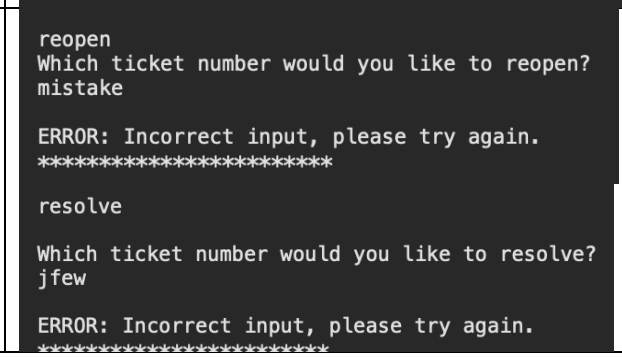
Reopen is selected. Ticket Number 2001 that is currently closed is selected. The program displays TICKET NOW REOPENED. Ticket Stats are updated and displayed along with options.	
Submit is selected. The ticket submission method is called, if the user selects not to submit another ticket the program returns to the dashboard. Ticket Stats changed accordingly.	

End is selected. Program returns to the Home area from the start.	
"E" is selected. Message is displayed. Program is terminated.	

Exception Handling:

Below are some test cases where input mistakes are made to test the program. I have tested every user input instance and cannot crash the system which is good.

User input at **Home** area. Wrong char and string entered. Exceptions handled correctly by Try and Catch. Program prompts user to try again and does not crash.	
User input at Staff **registration** area. A string and an int is entered which is incorrect. Exceptions handled correctly by Try and Catch. Program prompts user to try again and does not crash.	
User input at **submit** (constructor**)** area. A string is entered which is incorrect. Exceptions handled correctly by Try and Catch. Program prompts user to try again and does not crash.	

User input at **Helpdesk** area. Wrong input is made by the user (“mistake”) Exceptions handled correctly by Try and Catch. Program prompts user to choose one of the 5 options.		
User input in the **resolve** and **reopen** area. Images show wrong input made by the user Exceptions handled correctly by Try and Catch. Program prompts user to try again.		

Deployment:

Once I was satisfied with the system, I referred to the assessment requirements for submission through iQualify and the recommendations of the lecturers.

Operation/Maintenance:

As the system is a prototype, I would imagine refinements may be required to fully satisfy the clients' needs. I would usually discuss this with the client once presented. Routine patching and maintenance would be required in the future for operating system updates and routine software updates. I would also need to accommodate feedback from the help desk and staff users after a period of use to improve the system and changes in the clients' needs in the future. This may require repeating the SDLC process, hence why it is called a cycle.

Personal Reflection:

I found that writing down what classes, objects and attributes were required first before coding helped me build the C# program faster. Using the Agile Model to develop the program helped me to move between tasks when I was struggling with something, I would leave a note and come back to it later. Also, through rigorous testing I was able to find ways to make the code shorter and using the commenting feature to temporarily hide code and to refresh my mind on its purpose helped with organising the structure of the code.