**LINKS**

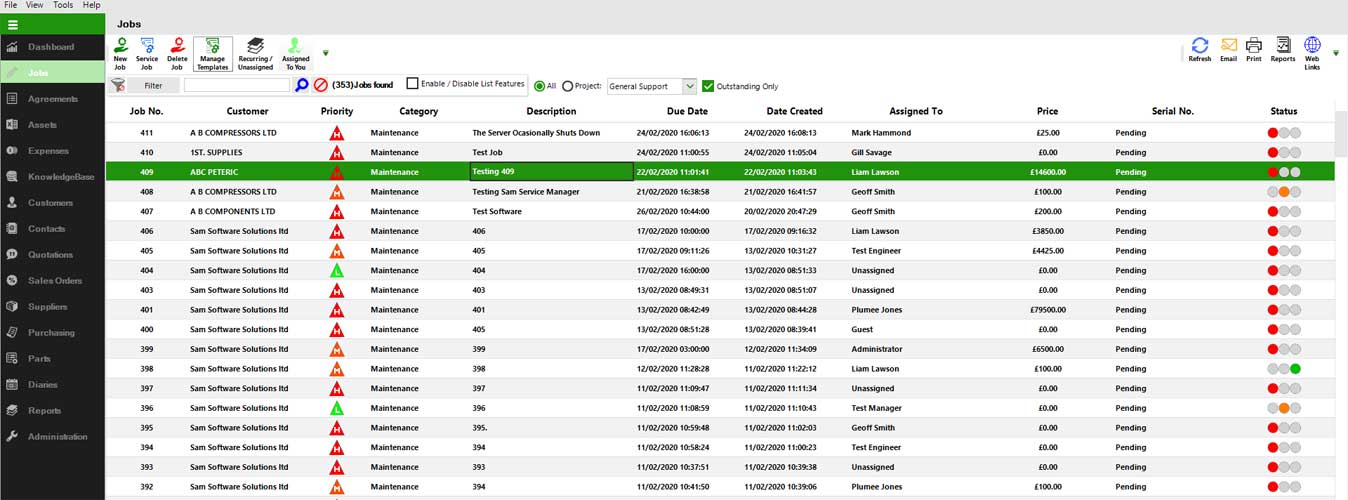
**Trello –** <https://trello.com/invite/b/Vdx2O0Aw/0d90e59ce0dabafbf242b509d4e6ef19/level-3-programming-internal>

**GitHub –** <https://github.com/oliviagoo/level-3-python-internal>

**1 – RESEARCH**

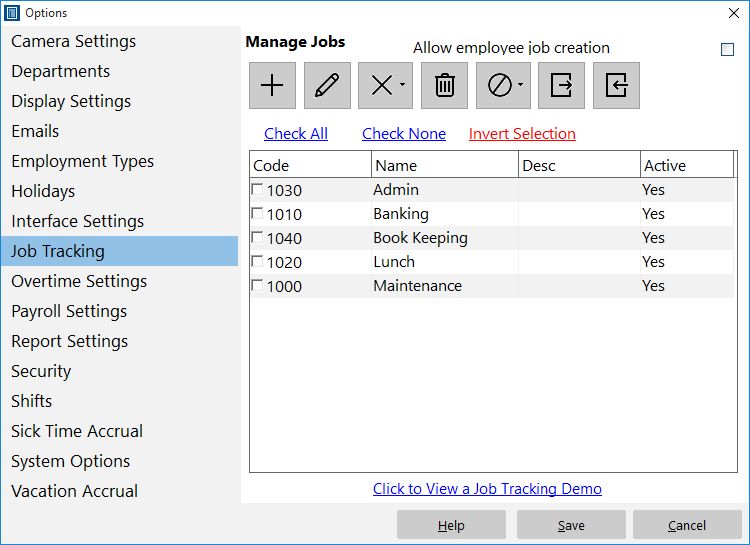
Before even planning my program, I needed to research similar programs to get an idea of how they function / look. The type of program that I will be creating is a job management program for a mobile tech repair/support company. The way the program should function is outlined in the brief, however there is no suggestion as to how it should look. A lot of similar programs require you to register for a trial, involving entering credit card information, so I instead searched for images of interfaces because I already know how I want the program to function, I just need to get an idea of how the interface should look.

**Interface 1:**



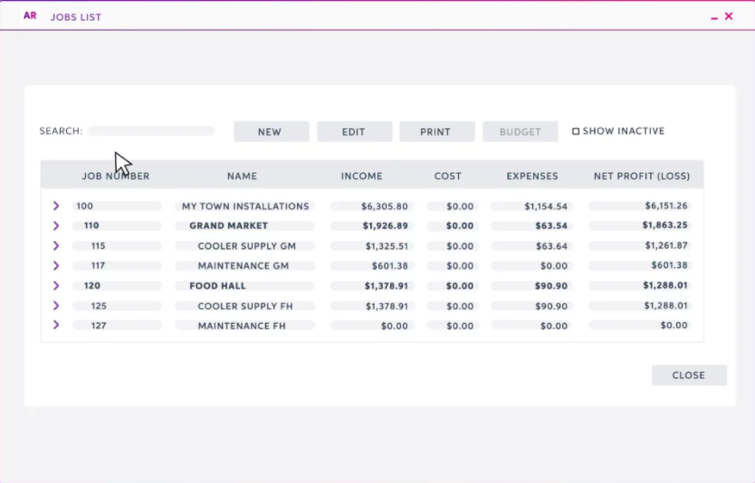
This interface shows a list of jobs that a user has entered. It shows the job number, who the job is for, its priority, description, date it is due, when the job was entered, who in the company will be doing the job, how much it costs, progress, etc. This makes the interface quite cramped and complicated. Because this business is only run by one person and jobs won’t be “in progress” – Suzy goes and completes the job in one visit – there is not a need for all this. I want to keep my interface as simple and easy to use as possible. There are buttons up the top to add/edit/delete jobs, but they are very small and hard for the user to find amongst all the other things.

**Interface 2:**



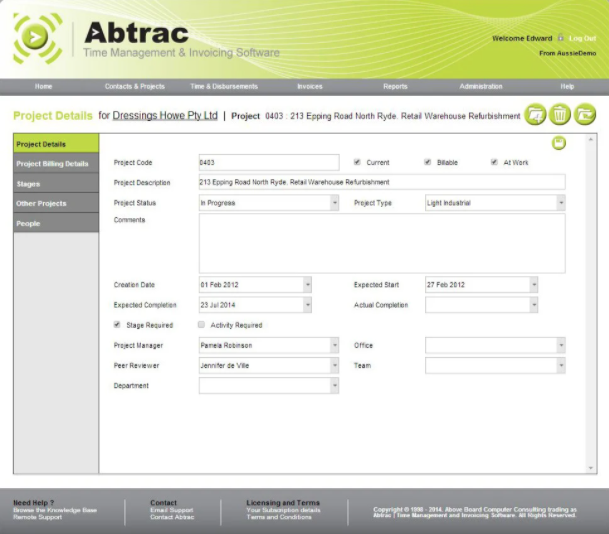
This interface is a lot simpler, and likely a bit closer to what mine will look like. It is not too crowded and the icons/buttons etc. fit in with the computer’s interface. The sidebar allows the user to switch to different screens, which is something I could possibly implement in my program. In this interface the user can edit jobs while on the job list screen, which I’m not sure if I want in my program.

**Interface 3:**



This interface is really good because it is simple and clear to the user what they can do. There is a list of jobs, they can search, create a new one, edit the jobs, and print. Not all of the functionality will be relevant to me – you cannot print from python tkinter – but this is a good interface to reference.

**Interface 4:**



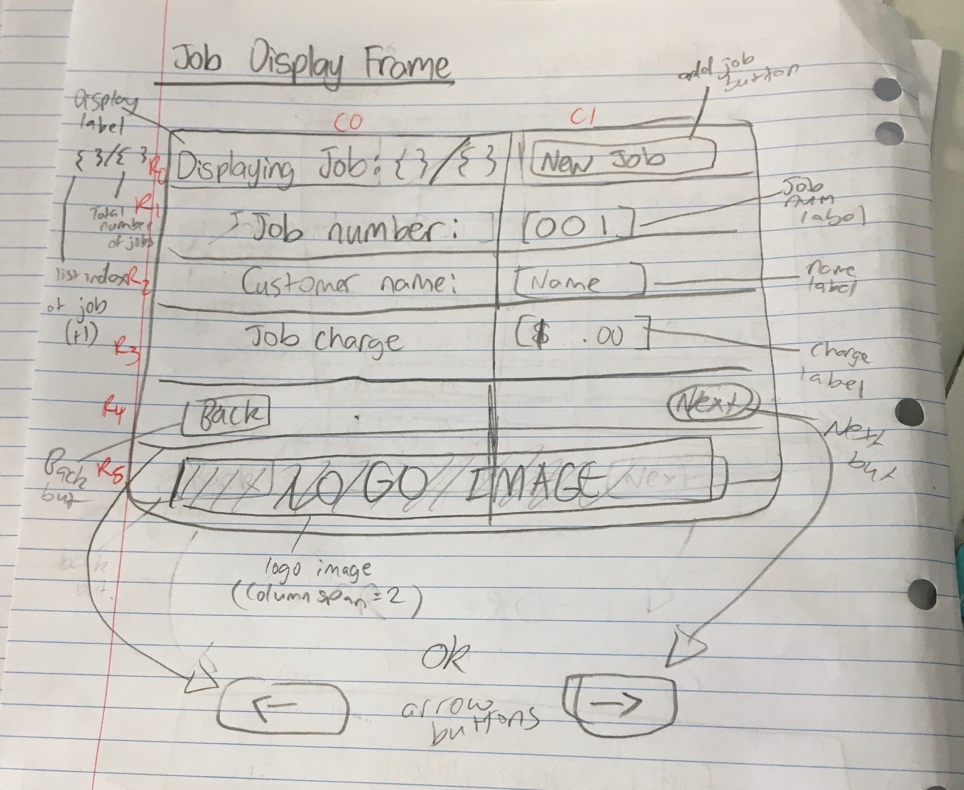
This interface shows the job entry part rather than the displaying jobs part of the program. There are lots of entry boxes, but also some checkboxes and drop-downs to limit user entry error. In my program I should try and use things like this as much as possible (without making the interface too clogged and confusing for the user) to prevent errors. ­­ However not all of these entries will be relevant for my program. I will only include the things outlined in the brief. So, for example, there can be a checkbox for “WOF and tune” rather than the user having to put that in an entry box.

**2 – PLANNING**

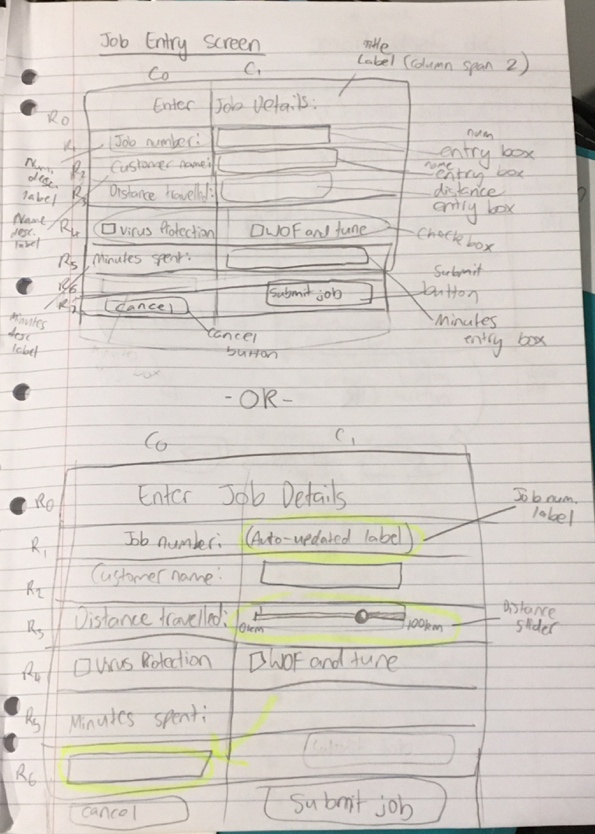
Now that I had conducted some research, it was time to start planning my program.

**Layout Diagrams**

First, I planned how I wanted my program to be laid out based on the specifications from the brief using a layout diagram.



This rough diagram shows what I want the job display frame to look like, and where I want different labels and buttons to be positioned on the screen.

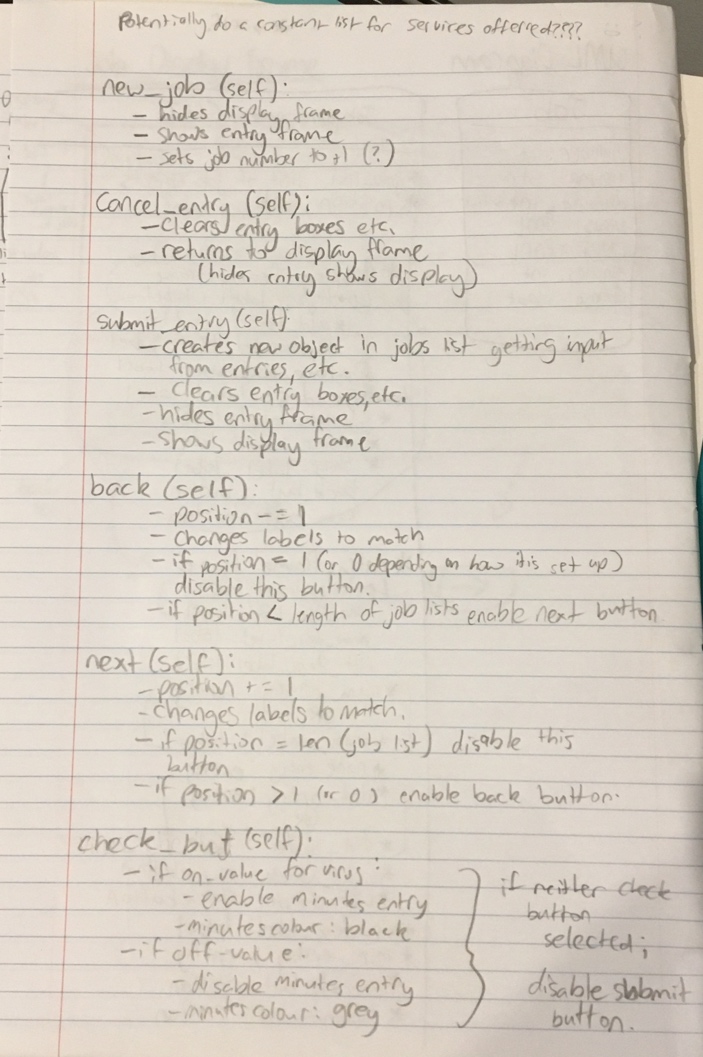
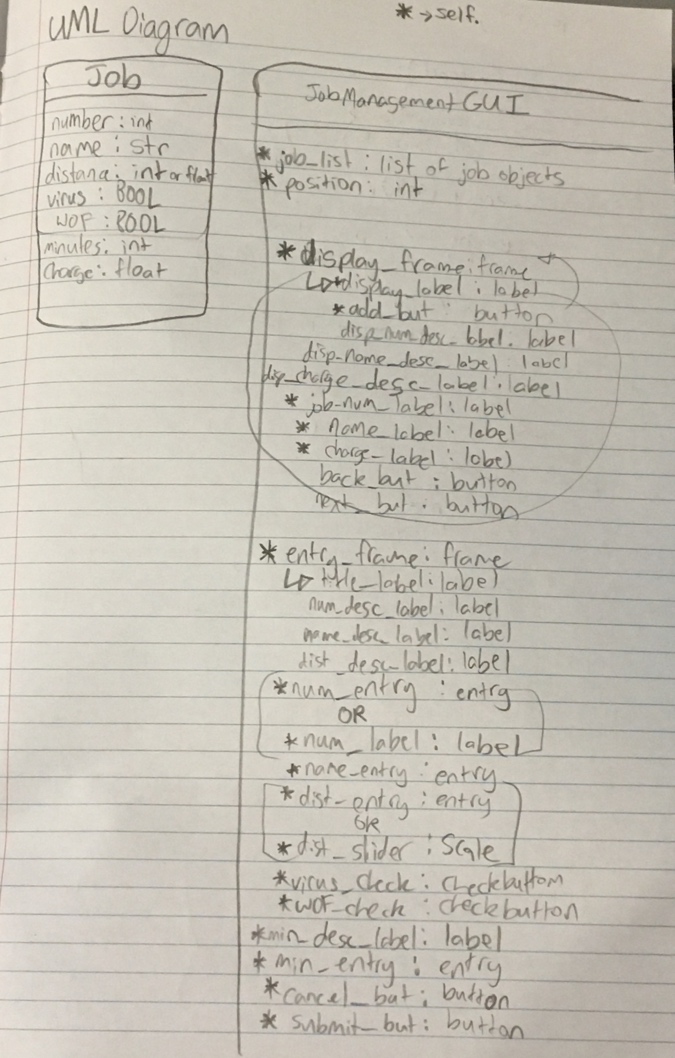
For the entry frame, I made an alternate version of the diagram with a few changes – an auto updating label instead of an entry box for the job number, a slider scale instead of an entry box for the distance travelled, and the positioning of the minutes spent entry box.

I then asked which version testers preferred, so that I could get an idea of what people wanted from my program. The second version of the diagram with the auto updating label, slider, and moved entry box was the preferred option – it was said to be more user friendly and clear.

Now I know, when I code my program, to go off of the second layout diagram.

**UML Diagram**

This diagram will help me create the classes that I need for my program. In this diagram I outlined the two classes that I plan to have – the GUI class and a jobs support class – and their instance variables and methods.

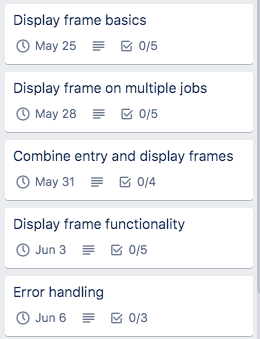


**Decomposing the Program**

Because this is quite a complex program, I need to break it down into smaller components so I can focus on programming one thing at a time and getting each part to work before combining everything into one final working program.

1. Entry frame basics  
   - Get the entry frame working on expected input by printing jobs into the shell.
   1. Set up layout
   2. Get customer name and distance travelled working
   3. Get checkboxes working
   4. Get minutes entry working
   5. Get auto updating job number working
   6. Get charge calculation working
2. Display frame basics  
   - Get the display frame working on ONE hard-coded example (buttons should not be working at this point)
   1. Set up layout
   2. Add in logo image
   3. Get display labels to work
3. Display frame on multiple jobs  
   - Get the display frame to cycle through MULTIPLE hard-coded example jobs
   1. Add next button functionality
   2. Add back button functionality
   3. Ensure display label (e.g. job ¾) is updating correctly
4. Combine entry and display frames  
   - Lets the user switch between the two frames by pressing a button
   1. Add functionality to the cancel and submit buttons – hides entry frame and shows display frame
   2. Add functionality to the new job button – hides display frame and shows entry frame
5. Display frame functionality  
   - No more hard-coded examples and printing to the shell. Jobs entered in the entry frame should appear in the display frame.
   1. Remove hard coded jobs and printing code
   2. Save entered objects to a list when submit is pressed
   3. Get display labels to update based on what is in the list
6. Error handling  
   - Making sure that the user cannot break the program
   1. Boundary error handling
   2. Exceptional error handling

After planning the decomposition, I put these into cards on trello.



I put due dates on each of these to help with my time management. Here is a screenshot of part of the calendar showing when these different components are due:

**Considering Implications**

There are some implications that I need to address throughout the creation of my program:

* Functionality
  + It is important that when you have a program it does what it is meant to do. No matter how good something looks, it is useless if it does not function as intended. I need to make sure that my program functions as outlined by the brief I was given so Suzy can use it for her business.
* Usability
  + Users should be able to navigate the program and use it without asking for help. The program should be designed in such a way that makes sense to users and prevents them from making errors as much as possible.
* Accessibility
  + I need to ensure that Suzy is able to access the program. I will use language that is user friendly and relevant to her business and find a way to allow her to use the program without having to download python.

**Test Plan**

|  |  |
| --- | --- |
| **Test** | **Expected Results** |
| Name Entry | |
| [Any name] | It should go through no problems – it is a string input so anything you enter aside from leaving it blank should be accepted |
| Leave it blank | It should not allow you to submit the job |
| Distance Entry | |
| 1km | All of these are valid entries, the slider should not allow any values that are not valid. |
| 56km (or any random middle amount) |
| 99km |
| 100km |
| Checkboxes | |
| Ticking only virus | Should enable the minutes spent entry box. |
| Ticking only WOF | No visible change, but saves that it is a WOF job |
| Ticking both | Should enable the minutes spent entry box. |
| Ticking neither | Should not allow the job to be submitted. |
| Minutes Spent | |
| 0 | Should not allow the job to be submitted. |
| 1 | Will be a valid option and will be saved to the job |
| 37 | Will be a valid option and will be saved to the job |
| Any text e.g. “one” | Should not allow the job to be submitted. |
| Leaving it blank | Should not allow the job to be submitted. |
| 3.7 (or any float value) | Should not allow the job to be submitted. |
| -56 | Should not allow the job to be submitted. |
| Cancel and Submit | |
| Press the cancel button | Will clear all of the fields and take the user back to the display frame |
| Press the submit button | Will do the same, but the job entered should now show up on the display frame |
| Charge amounts | |
| Just WOF and tune  Distances: |  |
| 1km  3km (or any in between value)  5km | 110.00 |
| 6km | 110.50 |
| 46km | 130.50 |
| 99km | 157.00 |
| 100km | 157.50 |
| Virus only (Distance set at 1km)  Minutes: |  |
| 1 | 10.80 |
| 7 | 15.60 |
| 50 | 50.00 |
| 129 | 113.20 |
| (For both WOF and virus, just add 100 to these charges) | |
| Distance – 20km  Virus – 38 minutes  WOF – yes | 147.90 |
| Next and Back buttons | |
| Press the next button until you reach the end of the list | Next button should disable until you press the back button |
| Press the back button until you reach the start of the list | Back button should disable until you press the next button |

**3 – Developing the Components**

With the planning done it was now time to start developing the program. The development process will be documented in the Testing and Evidence Log document. Each component has its own test plan and subsequent testing via screenshots with explanations and/or screen recording videos. This table is broken down into sections of the different components, and details my testing/development process throughout, as well as frequent reflections on what I had achieved so far and how I was doing timewise.

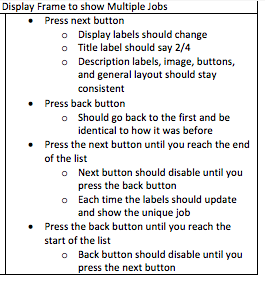
**4 – Implications**

This is an explanation of how I addressed the relevant implications throughout the development of my program.

**Functionality**

I made sure when developing my program that it functioned as expected, as outlined in the brief so that Suzy had a program that fitted her need.

I tested extensively to ensure that everything I put in the program worked. Before I started coding I wrote up a basic test plan that covered how I wanted the program to work, so when I had finished I could go back and test that the program worked as outlined in the plan. I also wrote a small test plan for each component of the program for any component that wasn’t at a point where it could work with the main test plan so that I knew what worked and what needed fixing before combining the components into the program. All of these tests were screen recorded so that I could look back and see if anything went wrong.

A test plan for one of the components

For example, when I was testing component 4, I noticed an error with the current job label in the display frame not updating upon returning to the display frame after adding a new job. This was a really subtle error that I might not have noticed if I hadn’t set time aside to test carefully. I was able to identify the problem and fix it (and then test again to make sure my fixing hadn’t caused any more problems!)

Another thing I did to address functionality was get end users to trial my program. Because I know how the program is meant to work, it’s easy for me to miss bugs that may be in the code. It’s also always useful to get another person’s opinion on how something like a computer program should function, as a fresh perspective can introduce some ideas that I would not have thought of before.

Python files containing the different components

Splitting the program into different components and programming each major part on different files before integrating them into one also helped with functionality. I could develop and break each one as needed without breaking the whole program, meaning it would be easier to figure out where the problems were stemming from. For example, if there was an error with the entry frame and I coded it on the same file as the display frame, it would be harder to tell which part of the program was not working. I ensured that each part of the program was working as I wanted before combining them, and then tested again once they were combined to make sure nothing had broken.

**Usability**

I wanted my program to be used easily by Suzy, and also for anyone else that this program could be repurposed for. This meant using conventional widgets that users would know how to work, and selecting widgets thoughtfully to prevent input errors.

The majority of people who use computers understand how different widgets work. If they see check buttons, they know that they can select one or more of the options in the list by clicking on it with the mouse. They know that if they see a button they can press it and something should change on the screen based on that input. If nothing obvious on the user end happens when that button is pressed, then an acknowledgement/explanation of what happened when the button was pressed should be indicated in some way.

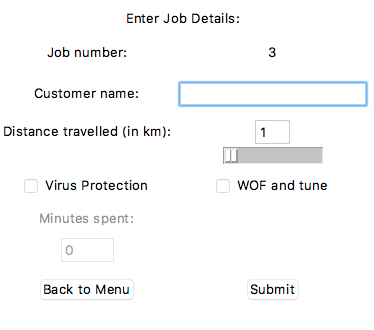
The confirmation message that appears after submitting a job

In my program, there is no obvious change when the user presses the submit button to enter a job. Unlike the other buttons, it doesn’t change the screen. The submit button’s work is very much behind the scenes – saving the details that have been entered so they can be displayed on the display frame. So that the user knows when their changes have been saved instead of pressing the button repeatedly, I made the button trigger a confirmation message in green at the bottom of the page (and, if the user is entering new jobs, it clears the entry labels so they don’t have to backspace or untick boxes themselves).

 A disabled button compared to an enabled button

One other thing I did to improve usability related to buttons was disable buttons that cannot be used. For example, when there are no jobs entered the view all and edit job buttons are disabled because there is no job to edit and there are no jobs to view. And, when there are no jobs or only one job entered the back and next buttons are disabled because there is nothing to scroll through (these buttons also disable relative to the user’s position in the list, e.g. if they are at the end of the list they cannot press the next button anymore). This means that, instead of the user repeatedly pressing the button and being confused as to why nothing is happening, they can see the greyed out button and immediately know that it is something that they cannot do yet.

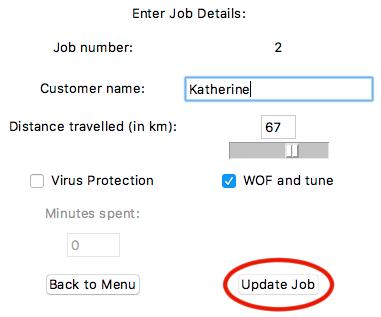
Another important part of usability is error prevention. To make the user’s experience as smooth as possible, it’s better to prevent the user from making errors in the first place rather than recovering from those errors later on. With any sort of user input there is likely going to be human error – typos, mis-clicks, etc.



The distance slider and the check buttons

To help prevent errors as much as possible in my program, I chose the widgets that I thought would make inputting data the easiest. I used check buttons where there were only two options for WOF and/or virus instead of something else like typing in yes or no. Not only does this stop errors with typing in values and having to cover every variant of Y or N, but it’s easier for the user and check buttons are very recognisable methods of input. For the distance input I used a scale (and a linked entry) that made it impossible to enter an invalid input. Choices like this help ensure that the program runs smoothly, and that it is usable.

End user testing was an important way I addressed usability. The best way to tell if users will be able to use your program easily is to give it to them and get their feedback. I gave my program to multiple end users and took notes of any changes that they suggested be made to the program.



Entry frame in edit “mode” so the user can make changes to the job

One example of these changes that was very important to usability was the edit button. The user can choose a specific job on the display frame and press the edit button to go and make changes to that job. A major part of usability is being able to recover from errors. People make mistakes entering data all the time, and if I had not added the edit button then there would be no way to go back and fix any mistakes. This is why it was a good idea to take user suggestions, as they helped improve the usability of my program.

**Accessibility**

It’s important that my program is accessible not only to Suzy, but to anyone she hires or anyone that uses a repurposed version of the program for their business.

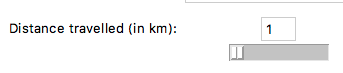
Firstly, even if I sent Suzy my python program she would have to download software to actually run it – the program is not an executable file, it is run through the python launcher. To get around this, I uploaded my code to a site called repl.it that can run my python program – tkinter GUI and all – on a web browser. This means that she (or anyone who I give the link to) can access the program without any problems. You do not need an account to access the program.



A picture of the Job Management Program accessed in a web browser through repl.it

<https://replit.com/@oliviagoo/Suzys-Job-Management-Program>

One other small thing I did to help with accessibility was change my scale on the entry frame to have an entry box linked to it. One of my user testers pointed out that the scale was quite hard to achieve specific values with. This was something that I had not previously thought of, as some people find it easier to do precise movements than others. Thanks to the tester pointing this out, I was able to link an entry box that would be easier for some people to use to get an exact value. Having an accessible program means that it is able to be used by all sorts of people with different motor skills.

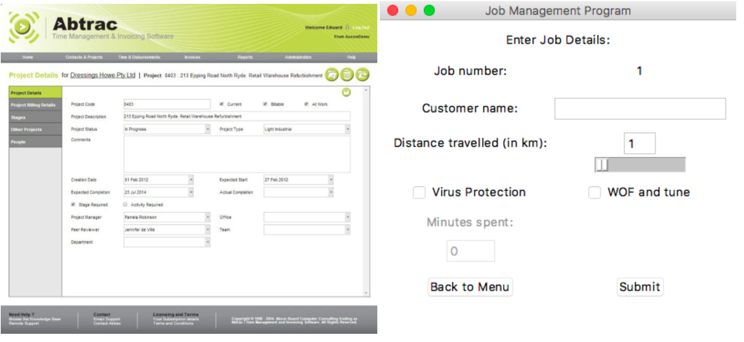


Distance slider with the entry box

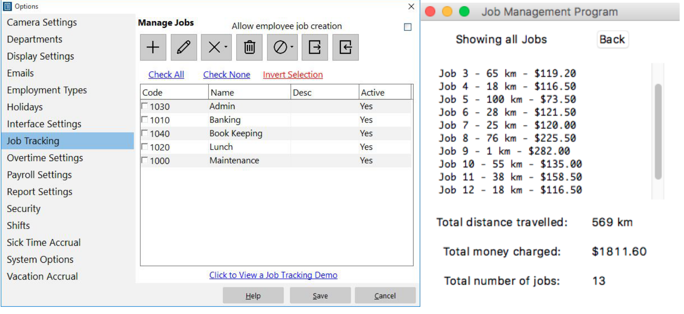
**5 – Discussion**

The information I gathered throughout the entire process of developing this program – from the initial research, to the planning stage, testing throughout, user feedback, and final in-depth testing/trialling all helped me to create a program that is high-quality to the best of my ability.

The very first thing I did, before getting into even the planning, was research the interfaces of similar types of programs and think about how I could implement the relevant and successful parts of these interfaces into my own program.



Here is a comparison of one of my researched interfaces vs. what my entry interface ended up looking like. You can see the similarities, but it is not identical because I did not have that many entries, and I tried to use buttons/sliders where I could. I took influence from the simple and structured nature of this interface to make my entry frame as orderly as I could.

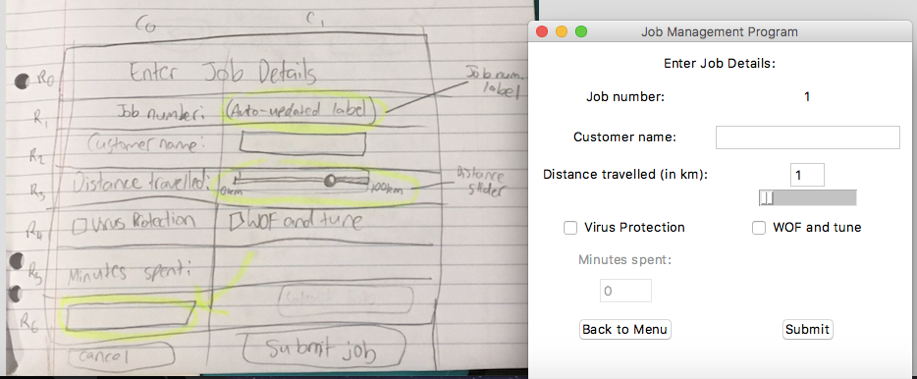


This is another comparison of my interface with the research. I listed all the jobs going down in columns like in the interface and had a button to go back to the main view. However, I kept this view separate from the editing and other functions, because this research interface is a little bit cramped and that is not what I wanted for my program.

The next thing that helped with creating a high-quality outcome was the planning stage. My diagrams, decomposition of the program, and test plan were especially useful in preparing for this project and staying on track throughout.

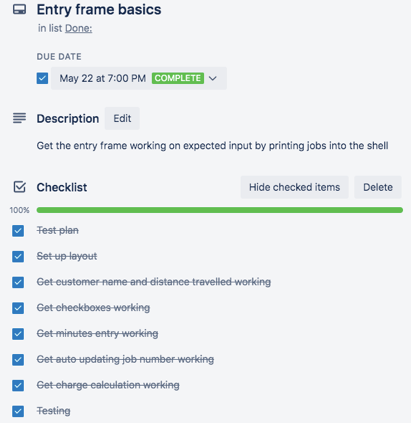
The diagrams that I made while planning were extremely helpful for creating my program. The UML diagrams showing the classes and methods I wanted helped me structure the essential basis for my program in the early stages of coding. While I ended up adding to this a lot (details of all the changes I made are in the testing and evidence log), outlining this before I started coding helped me get the earlier and more simple programming done faster as I didn’t have to stop and think about what variables I needed and what I wanted each method to do, leaving time to work on increasing the quality of my outcome. The layout diagrams helped similarly, aiding me in figuring out where I needed to grid each component to achieve the layout that I wanted.

When I created the layout diagram for the entry frame, I could think of multiple ways to do several parts of it. I wasn’t sure whether users would prefer an entry text box or slider for the distance, if they would rather input the job ID manually or have it update automatically, or if the minutes label and entry box should be next to each other vertically or horizontally. Instead of making these decisions all by myself, I decided that it would be best to ask potential end users, as they know what they want from the program, and their input as an outside view is valuable. I took their opinions as to which layout was preferred, and went with that one through the developing process.



My final program’s entry frame compared to the diagram that the users liked more.

During my planning, I created a Trello board and decomposed the different aspects of the program into “cards”, with separate descriptions and/or checklists. For each card, I set a deadline to help me keep track of my time management for this large project. I could keep track of when I expected each component to be done by on a calendar, and plan my work around this to make sure that I was not in a rush, and that I had time to improve the quality of the program.

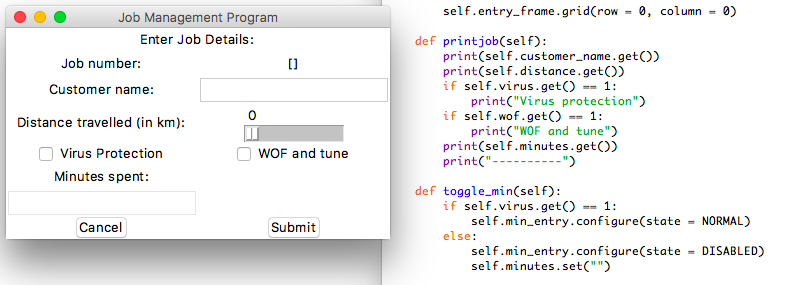
 A close-up view into one of the component cards. I created a checklist to keep track of each little task that made up the component, then once I had tested it and was sure that the component was working as I wanted to I would tick off the due date and move it to the done column.

For example, when I was running many days ahead of schedule I made the error handling due a little bit earlier. Error handling is important, but it doesn’t take as long as the final testing and acting on user changes process takes (plus that final testing process quite often picks up bugs that error handling doesn’t). Making this change allowed me to make more ambitious changes to my program, like adding an entirely new frame to the program and adding an edit mode. Having this timetable that was strict enough to keep me on track but also flexible enough to allow for change for the better helped me use my time wisely to complete this program to the best of my ability.

Another way the trello board was useful was for decomposing the final outcome into manageable chunks. This was a very large project that seemed a bit overwhelming when I first read through it. So in my planning, I broke the whole program down into six main components, then created a checklist for each of the little tasks that I needed to complete to get that component working. I was able to focus on getting the program to work, one small aspect at a time, instead of wasting time worrying about how exactly I would code the finished product from the very beginning.

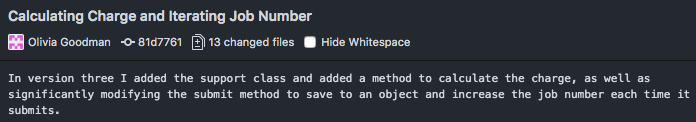
The test plan was another part of planning that proved useful, but I will touch on that more when I discuss the final testing process.

When I got into the actual development of my program, I took things step by step. I coded the program one step at a time, until the steps turned into bigger components, and until that turned into a working program. For example, the very first component I coded was the entry frame printing input (that was expected, no boundary or exceptional input yet) into the shell so I knew it could take the user input and work with that. This seems really simple, but the user input is the real backbone of the program. If the user can’t enter jobs and have that working properly then there is no point to the program at all. Working out the essential basics and putting my effort into coding them carefully early on made sure my program could go on strongly from there and build up, resulting in a high-quality outcome.



This is version 2 of my program next to the only two methods that I had. The layout was messy, the input just printed to the shell and didn’t save, and the minutes label only disabled once the box had been ticked and unticked, but what was important was that I knew I could get input and I had the widgets there. This provided a basis for the changes I made later.

Throughout the development of my program, I used GitHub to keep track of changes that I was making and versions I was creating. After I finished coding something – this could be an entire component or a few tasks on the checklist of a component depending on their size – I would commit the change to GitHub with a concise but explanatory name, and then write the details in the description box. This means that when I look back on the different versions and the changes I made within them, I know exactly what I did in each commit, and if I am looking for a specific change/version I can find it easily. Saving the program in versions is important, because if I made a change that ended up breaking my program to a point where I did not know how to fix it, I could retrieve a previous version of the program and start anew. Without GitHub, I would have to clog up my project folder with separate files for each version. This is messy and takes up extra storage space. Making my project folder a GitHub repository works around this, and also is one (of multiple) ways to back up my project.



One of the GitHub commits – the name is descriptive, but the description itself goes into more detail.

As well as creating a full test plan for the program, I wrote test plans for each component of the program. As mentioned in my implications writeup, this helped me identify bugs and improve each component before integrating them into the program to reduce the number of errors in my code – the more there are of them, the harder it is to find all of them. It would not have been good for my outcome if there were still errors that I had not picked up on after I finished.

Once I had finished all the components that I had planned for my program, I gave it to some users of a variety of ages and technical experience for them to use it and give me feedback. I took note of the changes that were suggested – some were as small as changing the wording on a label to make it less confusing, and some were as large as adding an entirely new screen to the program to help see all the jobs and keep track of things in a more general sense. I go into this in a bit more detail in my implications discussion, but this was one of the very useful aspects of the development process for ensuring the quality of my outcome. At the end of the day, this program is being made for the users. The best outcomes cater to the user’s needs above all else. Getting feedback from the users in both the planning and program development stages helped me to craft a program that was what the users wanted. If the users are happy with my program, then that is an indication that it is of a high quality.

Finally, an in-depth final testing process also helped with my outcome. I mentioned this as well in implications, but I crafted my test plan before even starting on the coding and I used it as a way to keep myself on track and know that I hadn’t strayed too far from the original idea for the program. I had written it to cover every possible scenario to make sure that no bugs were going to get through to the final outcome. When my program had advanced to a point where there were some components that the original test plan did not *quite* plan for, like adding an edit function, I simply improvised and tested it as systematically as possible. Because of this intensive testing process, I am now confident that my program is smooth-running, and of a bug-free quality.