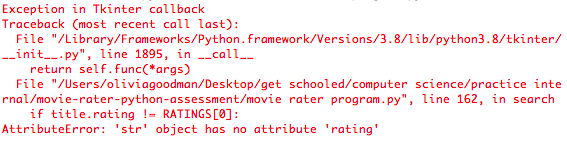
**Implications**

**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. In this case, the program should let the user rate movies and search for them. The specific brief that I had to follow was: *“You will create a program that allows you to scroll through a list of movies, rating each one in turn. At any point in the program the user can search for movies of a particular rating.”* My program fits this requirement, and this is due to me addressing functionality throughout the development of my program.

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 different components of the program that would be a bit more complex to test so that I knew they worked before putting them in the program. All of these tests were screen recorded so that I could look back and see if anything went wrong.

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. One of my testers found one of these bugs while testing, and I was able to work backwards from the error message to find the source of the problem and fix it. If I had not gotten someone else to trial the program I may not have found that bug.



*An error message that I found because of the end user testing.*

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 search frame and I coded it on the same program as the rating 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.

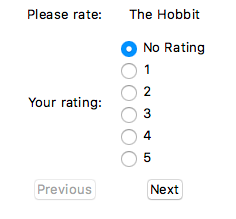


*A screenshot showing three python files. Each are named after a different major component of the program. These files were where each frame was coded and tested before being combined into one file.*

**Usability**

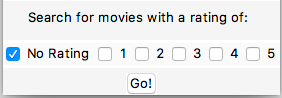
Users should be able to navigate the program and use it without asking for help. The program should be designed in such a way that makes sense to users and prevents them from making errors as much as possible. For my movie rating program, this meant using conventional widgets that users would know how they work, and selecting widgets thoughtfully to prevent input errors.

The majority of people who use computers understand how different widgets work. If they see radio buttons, they know that they can select one 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 happens when that button is pressed, then that should be indicated in some way.

*A screenshot of the rating frame from the movie rater program.*

In the above screenshot, you can see that these usability rules are followed. Each rating option has a radio button next to it which can be pressed, and they are in numerical order so as to not confuse the user. The next button, which would move to the next movie in the list, is in a normal state while the previous button, which would not do anything because the movie is the first in the list, is greyed out.

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 going to be human error.

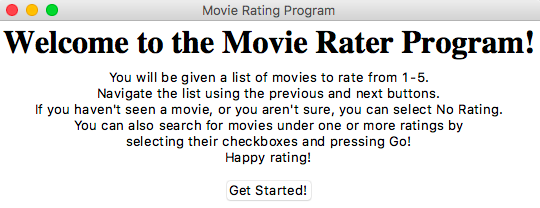


*A screenshot of the rating frame from the movie rater program*

Searching in programs is a very common idea and something that users are familiar with. While an entry box is the most common way of getting search input from the user, I decided to use checkboxes. There are only six ratings to search from, so it would make more sense to just have those options presented to the user so there is no room for error with them typing the wrong thing into the search bar. Although there are numerous combinations of searches possible with those six ratings, using checkboxes allows the user to search for multiple things with ease.

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 as they went through the program, and implemented any changes that they suggested into the end product.

One example of these changes that was very important to usability was the welcome frame with instructions for the program on it.



*A screenshot of the welcome frame from the movie rater program.*

Both end users got a little confused using the program as there were no instructions and they just got dropped straight into an unfamiliar interface. So, I followed the suggestions of one of the testers, and added a welcome screen to the start of the program. This prevents the user from being overwhelmed from starting off directly in the interface, and gives them the information and instructions that they need – because I can’t be there to tell a person what the program does when they start it up.

On the other end of this, too many instructions can also confuse the user and they will not be able to remember it all. Not including the “Happy rating!”, the instruction paragraph is only four sentences long and it uses easy to understand, user-friendly language.