Elliott Rose

Prof. Huang

COSC 625

05/04/20

Socrates Final Report

**Introduction**

This project was a semester long software engineering project meant to put together the knowledge gained from our studies in an applied project. My choice for the project was a voice-controlled research assistant that would be capable of taking and executing commands. My initial requirements were modest, specifying that the program should be able to accept voice commands to search and read Wikipedia, answer general questions (math problems, etc.) and read back my personal notes. Further, the program should be able to search within a returned document or query once it began reading back to the user. These answers should then be reported back to the user with a voice engine that is interruptible. This paper will outline the development of this project over the semester, its challenges, and the details of the final product.

**Overview**

To manage the overall project, I would need to depend on open-source software to take care of the basics of each of the required pieces of the project. In researching the project, I found that some of the best libraries for speech recognition and speech-to-text were available in Python. Python was a great choice for this project as it has reliable support, an easy and concise syntax, and support for concurrency.

Next, I needed to find the components of the project. First, I started with the text-to-speech engine (TTS). I needed a TTS engine that was reliable, quick, and ideally local. After some searching, I found the Pyttsx3 TTS engine was well suited as it allows for text to be fed into the engine with almost no lag on turn around, it requires no internet connection, and it offers reliable and instant stoppage of the speaker. These were some of the more important aspects of the project as many of the TTS engines available do not instantly speak when commanded, as they have to first convert the specified text into an MP3 file before they can begin playing. This is also problematic because the files need to be stored either temporarily or permanently. Further, many other speech engines did not offer support for stopping the speaker. All this being said, I did end up partially using Google’s GTTS for common short commands, that are used repeatedly. I will explain this in more detail in the following section.

The next piece of the project was the speech-to-text converter. This ended up being one of the bigger challenges in the project as the technologies were much more complicated to implement. I had to make some compromises here to ensure that the project would be completed during the specified timeline. The first software that I attempted to use was the a project out of Carnegie Mellon University called Sphinx (in this case PocketSphinx). PocketSphinx is an open source, lightweight speech-to-text engine that is optimized for mobile phones. The problem I ran into with this project is that the documentation was rather deep and confusing. I did get the software up and running and integrated within Socrates, but the accuracy was much to poor to be effective. I do believe that with more time I will be able to further train the model and implement it successfully. For the future, at a bare minimum, I will use PocketSphinx as a hotword detector for touchless startup of Socrates.

But because of these issues, I had to find another software for speech-to-text conversion. I tried two web-based options: IBM’s Watson, and Google’s speech recognition. Both of these were easily implemented through the Speech Recognition library available for Python. They are both free to a certain specified number of queries or words detected. In trials with both, I never ran into a limit. In the end, I found Google’s software to be more reliable, although sacrificing some speed.

Next, I had to enable Socrates with some skills. I made this happen with some basic web scraping with BeautifulSoup. I created functions to scrape Wikipedia and MIT’s START system which is provided for free through the web. The START system is capable of answering natural language questions in diverse fields. Similarly, I used BeautifulSoup to parse and scrape the answers after querying the system. Lastly, to keep things from getting out of hand and a bit more secure, I only made files from within the program folder searchable and readable for reading personal notes or files. This means that files need to be directly moved into the Socrates folder and then put into a notes subdirectory to be found by Socrates.

All of these skills are setup so that they return text. Once the text is returned, I then used regular expressions to make the text searchable. So, the text is then sent to the speaker and a text navigator. While the text is being read by the speaker, the user can query Socrates to find a keyword. When this keyword is located, the regular expression traces back to the beginning of the sentence and returns the entire sentence and the rest of the text. This way the speaker reads the keyword in its context and then can continue with the rest of the information in the text, being able to stop whenever necessary.

Finally, all of this had to be put together into a User Interface (UI). For the UI, I wanted something that I can deploy across multiple platforms, including mobile platforms. In my search, the most applicable UI software was Kivy, which is built to work mainly with python. This allowed me to keep from having to integrate too many technologies when all I needed form an interface was a simple on/off button and emergency stop. And with all of these technologies, Socrates was rounded out and ready for development.

**Development**

Working through the project, I tried to work iteratively, with AGILE concepts in mind. I would complete modules so that basic functionality was created in growing complexity. Important to me in this entire development was modularity. I wanted to ensure that decoupling pieces would not be too complicated as the technologies in this space are rapidly developing, with new technologies becoming available constantly. This modularity consisted of keeping the listener, the speaker, and the skills of Socrates as separate pieces, that return simplified outputs from their requests. Functionally, this meant that the listener just returns what it hears, the speaker only speaks and stops when requested, and the skills just return text to be sent to the speaker. Coupling these together after was one of the bigger challenges, as they needed to be strung together in a way that would accomplish the task, which when looked at individually, seems easy. But, in reality, all of these pieces need to be more or less continuously available during operation of the program. Because of this, the

**Challenges**

**Final Product**

**Conclusion**