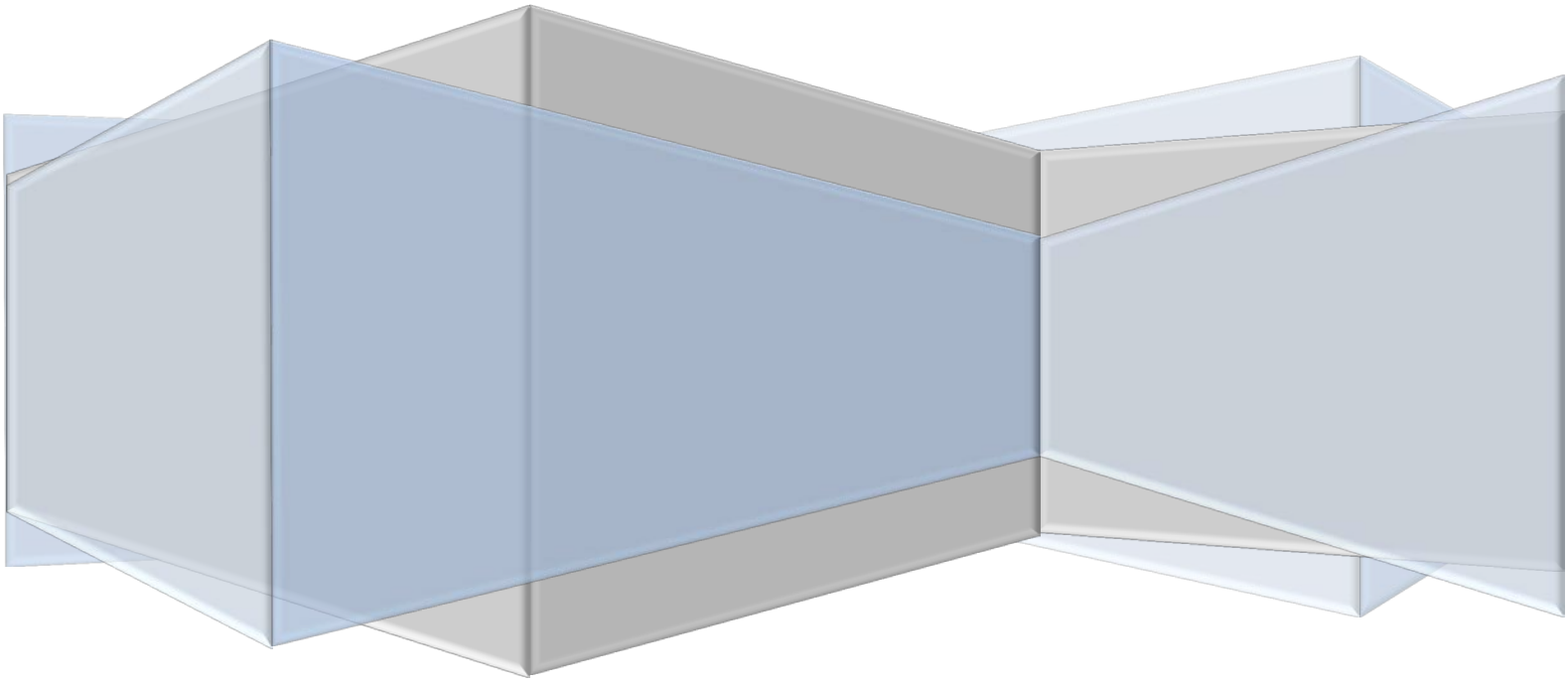
University of Auckland
Softeng 206

VAMIX

Video-Audio Mixer

By Anmol Desai



Executive Summary

Vamix is an open source video and audio mixer. Its target users are young adults (aged 18 to 25). Vamix allows the user to play and manipulate video and audio files. The user interface for Vamix was created using Java, whereas the processing of media files was done with Linux system calls.

Many factors had to be considered during the making of Vamix. The design of Vamix was specifically made to appeal to young adults. The layout of the different frames and panels was also modified to ensure that the program was easy to use. There were many design and real time issues encountered during the development of Vamix. For Vamix to become more robust, each of those issues had to be resolved. Due to the large amount of user input, Vamix had to have good error checking. There are many different types of checks done in Vamix to ensure that no invalid file is entered, no field is left empty and no invalid commands are carried out.

The code base for Vamix is very large. This meant that there had to be good documentation of the code so that it can be read by anyone. So, each of the classes and methods have a Javadoc comment to ensure good readability. To ensure that Vamix worked well, it was tested multiple times by different people. Each bug was recorded and fixed. The target users were also asked to use Vamix and give feedback. That feedback was used to make Vamix more suitable for the target users.

Overall, Vamix is a robust and easy to use video and audio mixer. It has achieved all the design requirements and is ready to be used by young adults –its target users.

Contents

Table of Figures:

Introduction

The aim of this project was to design an open source video and audio mixer (Vamix) for young adults (18-25 years old). Vamix was constructed using Java and Linux. The user interface was created using Java while the processes were done by the Linux system (Bash). The final design of Vamix had to be achieving all the functions delegated, be robust, have intelligent file management system and be easy to use.

This report gives information about the different parts of the development and how the project evolved in different areas to give a final and adequate product for people aged 18 to 25.

Features provided:

Download an open source video or audio.
Extract part of a video
Show/hide/delete history
Add text to video
Make a gif image
Extract images from a video
Add Video filters
Make subtitles for a video
Add existing subtitles to a video
Set playback speed
Extract audio from a video
Replace/Overlay audio of a video
Add audio filters
Replay
Playback features (play, pause etc.)
Take screenshot
Add files to library
Make a playlist
Play a file directly from the library
Play the list showing
Load a playlist
Load all media files from a folder
Cancel playing the list.
Display information about each file

Design

Target Users

The target users are young adults (aged 18-25). To make it appealing to that age group, the VAMIX had to be easy to use and appealing to the eye. The design of each feature had to simple, otherwise the users would avoid certain features as they would be intimidated by them. The colour and placement of each feature and frame had to be quite simple but not too generic. If the colours are too generic or bright then the target users would not be inclined to use the program. The placement of components was really important as well. Each of the features had to be distinct from each other in terms of placement. If any features are placed in the same frame as another then the user would get confused and probably make a mistake and quit the program. Young adults do not have much patience, and any mistakes in the design or functionality would prompt them to not use Vamix. I ensured that my Vamix met all the criteria mentioned above and was ideal for use by young adults.

Tools used for the development of VAMIX

The tools used for creating Vamix are:

- Java
- Linux (bash)
- Git
- GitHub
- VLCJ

Java was used as the programming language. It was used to create the user interface, intelligent file management system, creating files/folders and making calls to the Linux system. One reason for choosing Java was its portability. Java runs on any operating system as long as the Java Virtual Machine is present. This means that the project can be expanded in the future on any operating system. Another reason to choose Java was for its large set of APIs and clearly documented JavaDocs. Java has a well-built API, which made the development process very easy. The JavaDocs available are very detailed and that made learning and implementing new things very easy. Finally, Java was chosen as it had great support for GUI development. It has an inbuilt swing package that makes it very easy to make and manipulate the user interface in the present and in the future.

Git and GitHub were used to manage the code in the development process. Git is a distributed version control system and it allowed me to develop Vamix easily. Git allowed me to do many things like: experiment new features safely, keep track of all

source files, merge new features into the existing system and have the updated code when working with a partner.

GitHub was used as it allowed me to store all my source files safely and update them using Git. GitHub was ideal for work as it kept a backup of all the source files and created an easy to use file sharing platform. Me and my partner shared code very easily during the making of the prototype due to GitHub.

VLCJ was used as it provided an excellent framework for playing and manipulating media files. VLCJ gave java bindings to all the native functionality provided by libvlc and made it very easy to manipulate media files.

Colour

The colour implemented was light blue. This was provided by the seaglass look and feel.

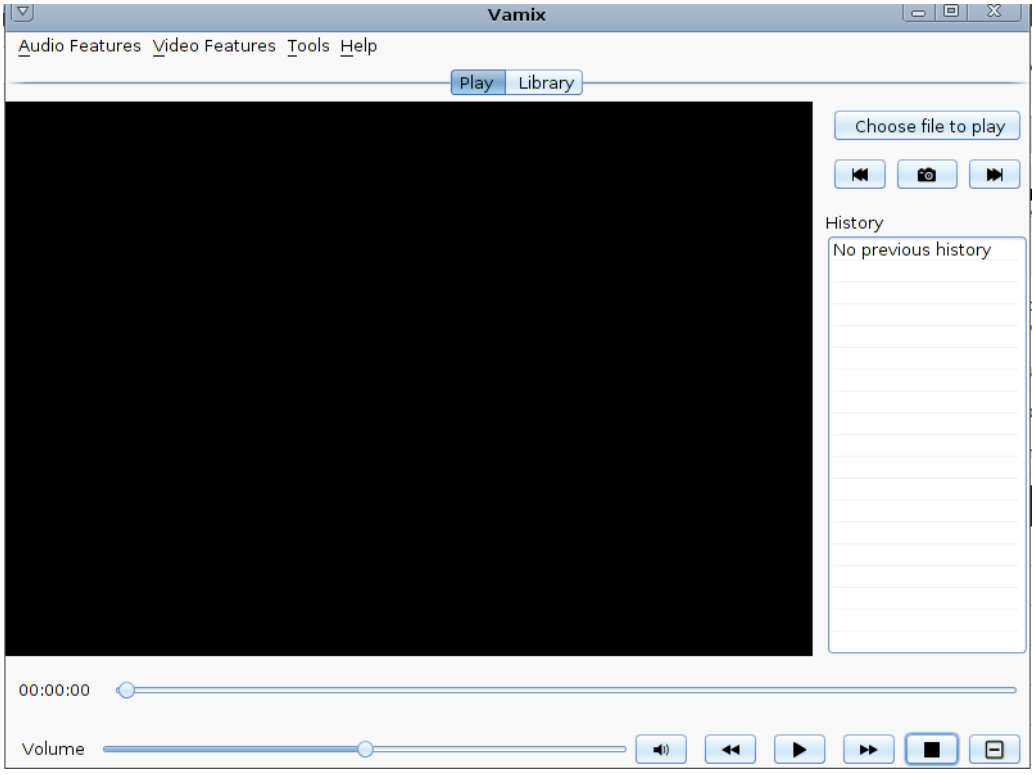


Figure 1 - Main Frame

As seen in Figure 1, the seaglass look and feel makes the Vamix appealing and easy to look at. The seaglass look and feel also made the JTextArea have lines. That made the Vamix look more professional and appealing to young adults. The colour for all the frames is the same as the one showed in Figure 1.

Layout

The layout of Vamix had to be simple but very explicit. Each feature had to be distinct and detailed.

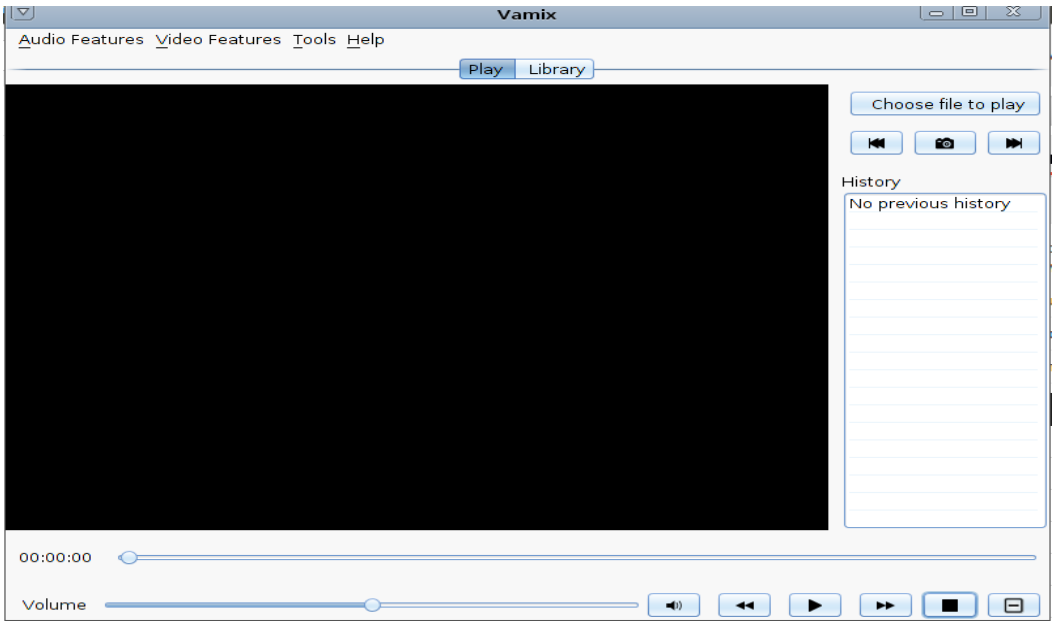


Figure 2 - Main Frame (play tab)

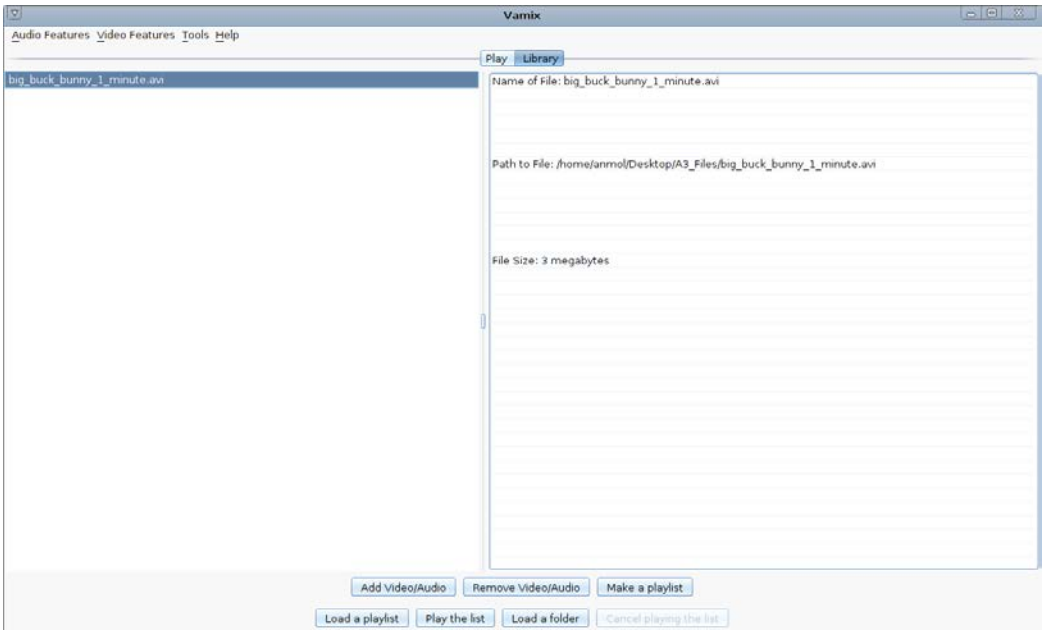


Figure 3 - Library tab

The features in Vamix were separated by the menu bar. There are 2 main tabs that are present. First tab is the “Play” tab. It is visible when the Vamix opens. The second tab is the “Library” tab and that is not visible when Vamix first opens. Both are shows in figure 1 and 2

respectively. The “Play” tab was made to open at the start as the user mainly wants to use the program to play media files.

The library was made a tab as to allow the user to take media files and manipulate them easily. The user can select a media file, add it to the library and then just double click on the name of the file to start playing it.

The library was split into 2 panes to allow the user to see information about each file easily while doing other functions. In the “Library” tab, the functions are clearly labelled on the buttons. These functions allow manipulation of media files existing in the library.

In the “Play” tab the media player was given most of the space. The reason for that was to allow the user to see as much of the playing media file while still allowing them to look at the features provided. A toggle panel was also implemented. The toggle panel allowed features in it and made the main frame less cluttered. The toggle panel can also be minimised. That allows the “Play” tab to become more spacious and allow the user to just focus on basic features like play, pause etc.

As seen in figure 1 and 2, the menu bar has all the features in it. The features are separated according to their traits and then put in their respective menu. There are 4 menus: Audio Features, Video Features, Tools and Help. Each menu item opens a different frame that allows the user to use that specific feature. The idea of a new frame for each feature makes it really easy for the user to use. The user will not get confused about what feature they are working on as each frame will have a name of the feature at the top.

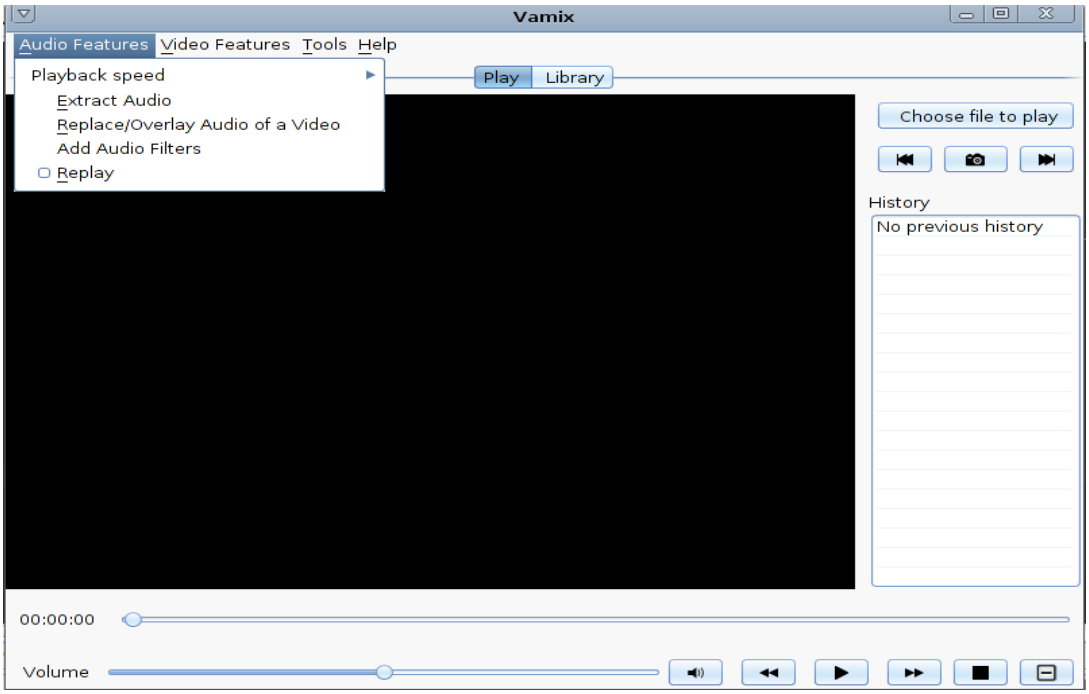


Figure 4 - Menu Bar

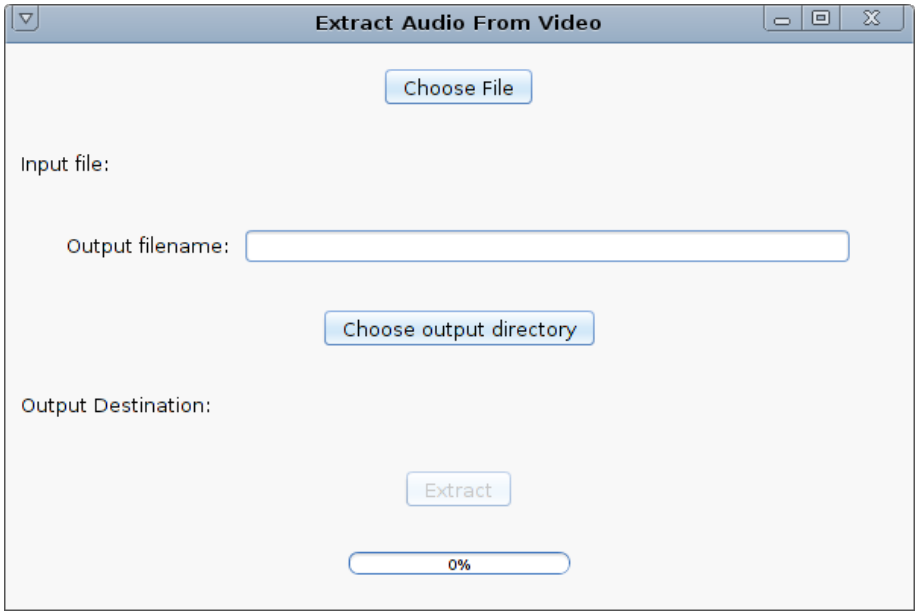


Figure 5 - Example of a feature frame

Figure 4 shows how the menu bar opens. Figure 5 shows the frame that shows up when the menu item “Extract Audio from Video” is clicked in the menu bar from figure 4. All other features are opened in a new frame (similar to the “Extract Audio from Video” feature shown).

Menus were chosen because they allow the Vamix program to be developed further without changing the existing layouts or features. More features can be added to an existing/new menu without changing any of the current menu items. This layout makes the Vamix very robust and easy to use. It is also easy to test the features easily in this layout. In case new features are added, only that new menu item has to be tested because it is separate from the other features/menu items.

Design and Real time Issues

Design issue:

The resizing of the frame was a big design issue. The expansion of the main frame meant that the media player and other tabs and their components would have to be resized. The resizing was causing the components to break the layout and made the program hard to use. I solved this issue by using GridBagLayout for the any panels or frames that could be resized. That ensured that all the components ended up staying in the layout they were originally arranged in.

Real time Issues:

Some of the features were not giving an output in real time. This was causing the program to lag or often freeze completely. One way I got around that was by using a SwingWorker. I shifted all the time consuming tasks to the background thread by using a SwingWorker. This allowed the GUI to be fully functional at all times while multiple processes were being run.

Error Checking

Input File

Vamix takes in a lot of files from the user. This meant that there was a chance that some file was actually not a media file. To assure that no invalid file was worked on by the program I put in a file checker class. This class had a method that took in the selected file path and then did the “file” function from the Linux system. The “file” command was then checked for the words “audio” or “video”. If the file was valid then it would have either one or both of the words mentioned above. If the file was a valid media file then the method returned true and the process is continued. If the file was not a valid media then the user was warned and the process is discontinued. The buttons that start the process also become disabled until a valid media file is entered,

Input Fields

The user is prompted to enter input into different fields in each feature. This means that the user could leave some input blank and try to carry on with the process. This was stopped by having a check before the process starts. If any of the fields were empty then the process does not start, it just warns the user with a message in a dialog box. The process is only started when all the fields are filled with the appropriate values.

Existing File

Most of the processes required a user to give a name for the output file and then choose a directory to store that file when the process was completed. However, if that directory already has a file with the same name then the user is warned and asked to either override the existing file or change the name of the output file.

File Chooser

The file chooser that was implemented also had error checking. Previously, the user could choose any file and just pass it in as input. I resolved this issue by making sure that there is a file filter for each file chooser. For example, an audio input file chooser will only show all the audio files and a destination chooser will only allow the user to choose folders. This created an added layer of security so that any invalid files had no chance of being chosen. If there is an invalid file that does get chosen then the invalid file check will stop and warn the user.

Documentation

The code was continuously documented throughout the development process. This allows the reader to understand every part of the code and how it is related to the rest of the code. All classes and methods have a Javadoc comment. Parts of the code that are hard to understand are also well commented.

Special Features

Extract part of a video

One of the methods to extract part of a video is to double click the media player screen. To start recording the user had to double click the media player screen. Then the user can double click the media player screen again to stop the recording. The video between the 2 double clicks will be extracted and saved to the user's home folder. There were lots of problems with this command. When part of the video was being extracted, it used up the time between the start and end of recording. This meant that if the time between the start and end of the recording was too small then the output file would not play properly. One thing I did to avoid that problem is put a message dialog that warned the user if the video being extracted was smaller than 8 seconds.

Subtitles

The subtitles feature allows the user to make their own subtitles and add them to a video. If a subtitle file already exists for that video then the new subtitles are just appended to the original subtitles. The user can also add an existing subtitle file to a video.

Library

The library allows the user to add arbitrary number of media files. Those media files go in the list on the left and can be manipulated in many ways. The user can make and load a playlist as well. Currently the playlist feature takes some time to be created. The playlist is just a text file with the paths of the chosen media files for that playlist. In the future, that function will make the playlisy into an international playlist format.
The user can also play the list that is currently present in the frame by pressing the “Play the List” button. A folder can be loaded into the library as well. Loading a folder is just loading all the media files in that folder. That increases the range of functions for the user and allows them to manipulate those media files further.

Evaluation and Testing

Peer Review

Recommended changes by peers:

- Increase the number of comments made.
- Allow the help button to open a readme file.
- Store the files currently in the library and show them when VAMIX is reopened.
- Fix the layout of the add text frame so that it is more visible.
- Combine the history and library function.
- Add subtitle functionality

Changes made from BETA version:

I commented all the classes and their methods. I also commented certain statements of the methods that were not explicitly stating what they were doing. I fixed the help button so that it shows a readme file with instructions on each feature. I added the subtitle functionality so that the user can write their subtitles or add an existing subtitles file to a video. The layout of the add text frame was also changed so that it was not hard to read.

Changes I did not make:

I decided not to store the files from the library for future use. I want the library feature to be one that the user can use when the VAMIX is open. I have already provided a “Make a playlist” option if they want to store the files from the library. Also, I did not combine the history and library function. I want the library function to be a separate entity that does not depend on the history. The history is just for the user to see what files have been played, whereas the library function is for the user to manipulate media files. I did not want those two features to cross over as that would make the program overly complicated and not fun to use.

Testing

I did thorough testing of each function after it was completed. I went through each possible path that the user could take for that function. This allowed me to find minor bugs and check unique test cases. Also, if I added a feature that shared code with any other features, I tested all those features again because the new feature might have accidentally affected the existing features. Finally, I asked my friends and family to test each feature and write down all the bugs that occur and what improvements they would like to make to Vamix. Their feedback allowed me to improve my Vamix further. The feedback my friends gave was very useful as they are the target users. So, all their comments allowed me to evolve the Vamix to become more suitable for my target users –young adults.

Conclusion

- The final VAMIX product has all the required functionality.
- The final VAMIX product meets all the design requirements outlined for the target user (young adults).
- The final VAMIX product is easy to use and did not fail at any time.