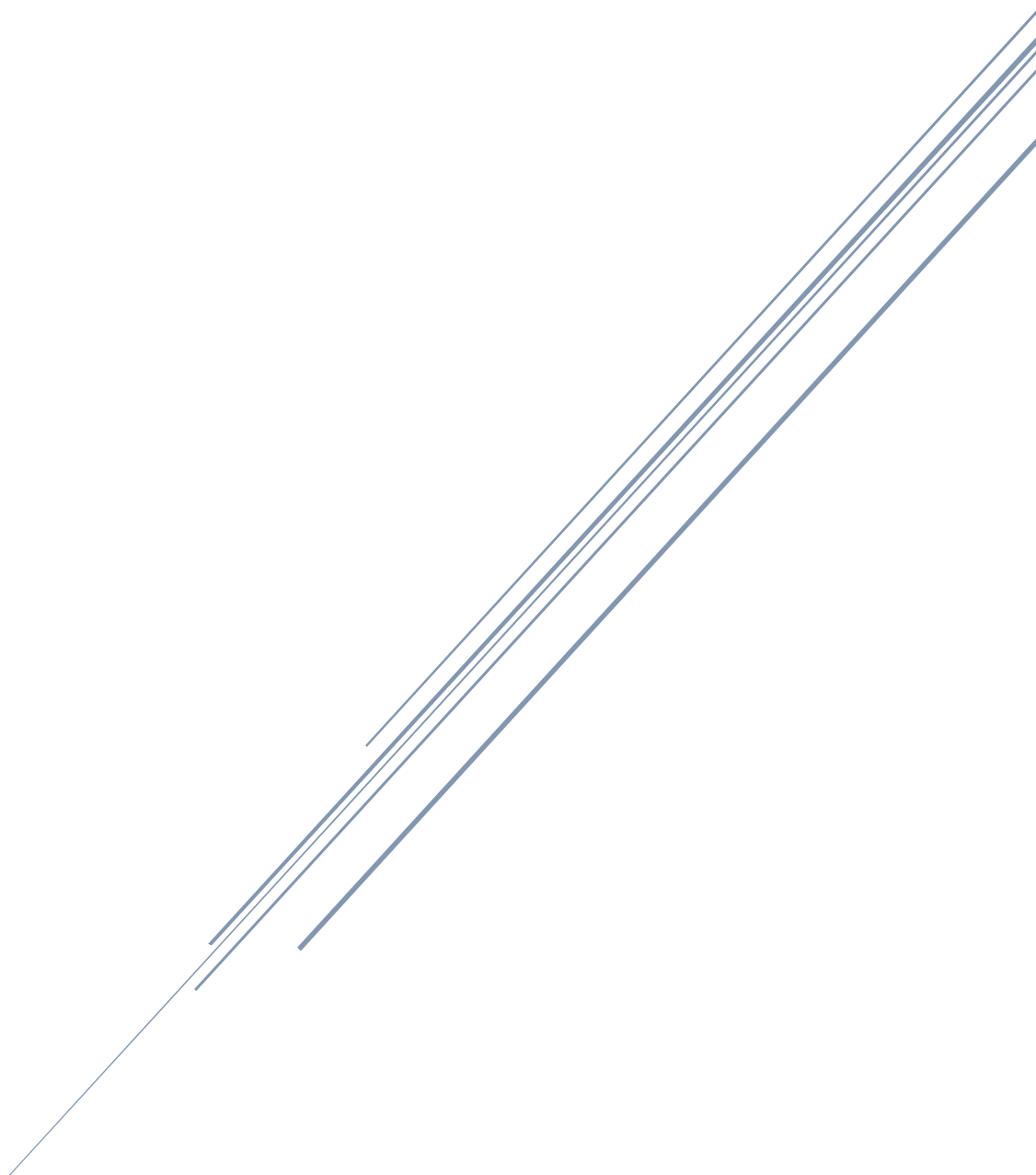


# VIDIVOX REPORT

By Hailun Wang



HWAN561  
6637096

## Executive Summary

A video editing program with the capability of creating synthesized speech and overlaying it onto a video was put forward by a client and needed to be created. The proposed name the program would go by is “Vidivox” and needs to be able to support traditional playback features of video players as well as inserting audio files at specified time points in the video.

Vidivox was made to be a Linux based program with the application being targeted towards lower level users whose exposure to the video editing field is minimal and who would like a program to provide them with a means to develop their skills.

Vidivox’s graphical user interface went through many revisions before a final most suitable one was decided on. The decided user interface was both robust and intuitive with a silver-grey colour scheme chosen due to the colours not being too harsh on the eyes for long time use. Many nested panels were used to achieve the final proposed interface.

Huge emphasis was put on usability during the development process, as the program had to be easy, straightforward to use without being tedious. Processes such as creating of audio files, merging and creation of folders inside the project folder on start-up were all automated for ease of use. The program was also put through extensive error checks whenever a new functionality was implemented to ensure that it works as expected, with error messages popping up to inform the user of anything that went wrong.

GitHub was used to maintain version control during both the pair development prototype process and the solo endeavor of creating the final project. It made merging simple with multiple developers working on the source code at one time, testing out new features convenient due to branching and keeping the project up-to-date and safe in case anything happened to the local repository.

Working on the video editing program ‘Vidivox’ encouraged both good teamwork with the developers as well as the client-developer relationship.

## Table of Contents

Executive Summary .....	1
Introduction .....	3
GUI Design.....	4
Functionality .....	5
File Choosers .....	5
Character Limit.....	6
Displaying Audio Information .....	6
Development and Code Structure .....	7
Use of GitHub.....	7
Evaluation and Testing.....	9
Testing of functionality limits .....	9
Prototype Feedback.....	10
Conclusion.....	11
Future Improvements .....	11
Lessons Learnt.....	11

## Table of Figures

Figure 1 Add Audio Panel-laid out left-to-right, top-to-bottom .....	4
Figure 2 Video Chooser - Selects avi files only .....	5
Figure 3 Audio Chooser - Selects mp3 files only .....	5
Figure 4 Added Audio - Audio to be exported .....	6
Figure 5 Project Audio - Custom files created by user.....	6
Figure 6 GitHub recent commits.....	7
Figure 7 Successful video export message.....	9
Figure 8 File already exists notification .....	9

## Introduction

The task of developing the video editing program “Vidivox” was put forward by a client with the aim of providing a software which is easy to use and intuitive for everyday users. Vidivox must be designed with the intent to be used on Linux based systems and be built dependent on three packages (VLC video player, festival text-to-speech and ffmpeg video editing) which it cannot function correctly without. It is essential for Vidivox to be able to support traditional playback features of video players, create synthesized speech files using the “Festival text-to-speech synthesis package” and merging the speech files with the chosen video at any selected time.

This application is targeted towards low level users whose exposure to the video editing field is minimal and who would like a program to provide them with a means to develop their skills. The usability and intuitiveness of the program was taken into account and the user interface was optimized for use in the Ubuntu operating system. The audience was considered highly during the implementation of the program and interface.

A prototype was first submitted for feedback from peers which was then further developed into the final Vidivox program to present to the clients.

## GUI Design

The development of this program was done in Java due to its object oriented design concepts and its ability to split up groups of similar functionalities into packages and classes. During the planning and development of the program, the creation of the user interface was made in a 'ui' package which contains classes which purely creates the front end of the program which the user sees containing no functionality and a sub package 'utils' which contains the functionality of the user interface.

The colours used were based on the java 'look and feel' Nimbus theme. Nimbus look and feel was used due to its polished cross-platform look as well as being highly customizable. The main colours in this theme is a silver-grey and blue which is very easy on the user's eyes and doesn't strain the user after long periods of use. This is very important as the video-editing is usually a long process and this the reason why the system look and feel was not chosen because a majority of the theme is white, which is very bright and straining after periods of use. Even though the program is aimed at lower level users, the silver-grey also makes the program look very sleek which makes for a better experience as the program should feel like something new and not outdated.

The graphical user interface was designed to be very intuitive to use with out and pre-existing video editing knowledge. All the functionality for traditional video play back, creating/adding audio and exporting the final video is all in one frame. This was done so the user would not need to search for the specific button which opens up a frame for them to create a synthesized mp3 audio as everything is already there. People read from left to right and top to bottom, so therefore it is intuitive to have the components in the panels laid out in this intuitive manner.

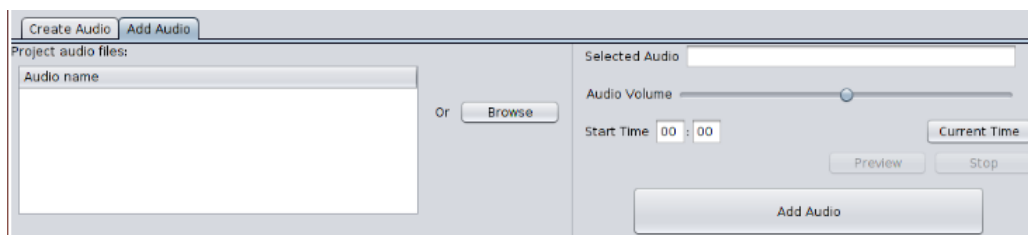


Figure 1 Add Audio Panel-laid out left-to-right, top-to-bottom

Due to the user interface being made from a series of nested panels with different layouts, the main media player frame was made not re-sizeable. The main panel of the media player is a BorderLayout which is resizable, but due to not being able to set the sizes of components such as JButtons, nested FlowLayouts were used which does not automatically resize. Also if the media player was to be resized to a smaller size, certain components will no longer be visible so because of this, the decision was made to make the program not resizable so the distorted interface due to the resizing would not ruin the user's experience of using the program.

## Functionality

The audience that the program is aimed to be used by are entry-level video editing users so it was attempted to automate the more difficult sides of the processes. There was no need for the user to try and adjust voice pitches and speeds as a few select ones were predefined for them to choose from. There was also no need for them to try and create their own voices to use as a selection of different voice diphones were also offered.

The users do not have to select where to save their custom mp3 files each time after their creation as they are automatically saved in an 'Audio' folder inside of their project folder which they chose at the initialisation of the program. This makes sure that these audio files are able to be easily found again when needed to be accessed. The same with exporting videos as the exported videos are saved inside a 'Video' folder inside of their project folder.

Major Vidivox functionalities include:

- Choosing a project work space which files will be saved in
- Selecting videos to edit and play
- Creating synthesized text-to-speech mp3s with options of altering the voice, pitch and speed.
- Previewing and choosing synthesized speech audios and selecting the time which the audio is to start playing in the video when merged.
- The ability to remove audio you have chosen to add to the video.
- Merging and exporting the video with all the added audio into an output avi media file.

## File Choosers

As the video program requires selection of working directories, selecting video files to play and selecting custom audio files which are not part of the project, a JFileChooser was decided on to accomplish all of this. The file choosers also restricted what files may be selected based on what it is being used for.

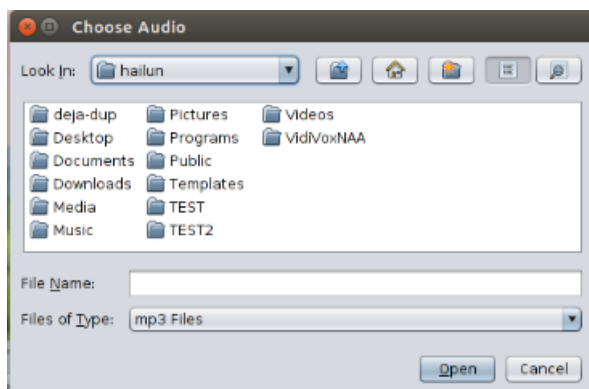


Figure 3 Audio Chooser - Selects mp3 files only

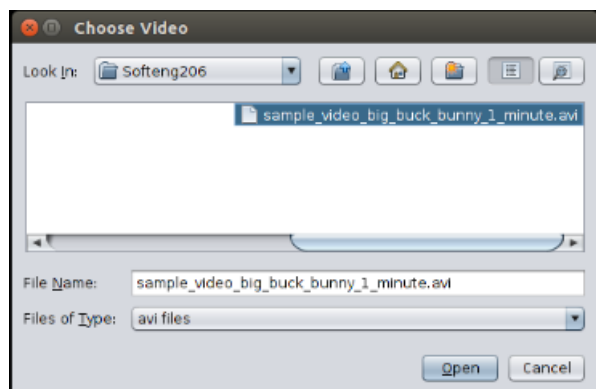


Figure 2 Video Chooser - Selects avi files only

## Character Limit

The create audio panel contains a text area where the user may enter their text to be synthesized. A character limit was set because once the festival voices talk for too long, they eventually 'go out of breath' making the audio no longer audible. This is an example of have ease of use over powerful functionality as there is no need to give the option of creating long mp3 files if they are not outputted to a good quality.

## Displaying Audio Information

Both the audios in the project and audios to be added to the video are displayed in JTables which both have the functionality of selecting a specific audio and previewing them. With the project audio preview the user may alter and select the volume of the audio to be merged with the video.

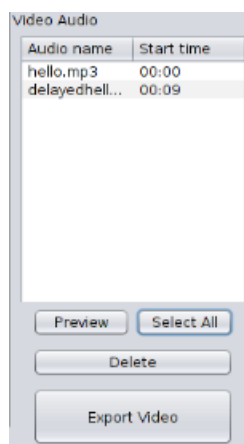


Figure 4 Added Audio - Audio to be exported

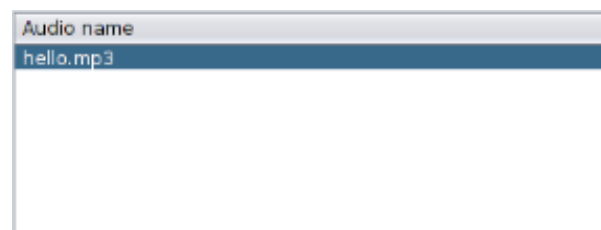


Figure 5 Project Audio - Custom files created by user

## Listeners

Many action listeners were used to call methods that implemented functionality of certain buttons when clicked and sliders when slid. More custom listeners were created to listen for changes regarding creation of new files and the addition and removal of keys in data structures such as hash maps which would then notify the respective tables displaying the information to refresh and update. Listeners were also attached to media player components to notify other components of the code when the media, either video or audio has finished playing, and what to do then.

## Development and Code Structure

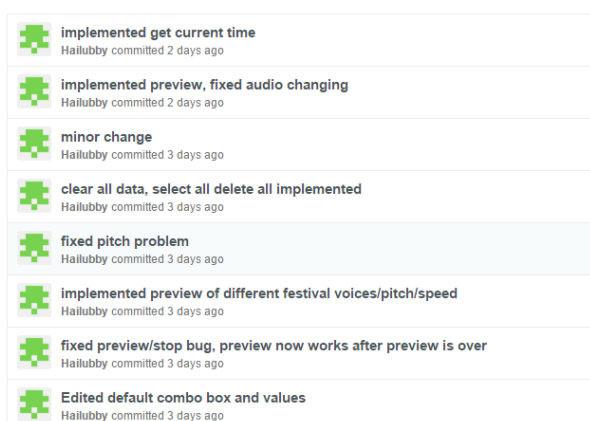
Due to java being of object oriented nature and its grouping of classes through packages to keep code with different behaviour separate, it was excellent for the development of the Vidivox program. The Java Swing classes also provided more than enough components to be used to implement functionality for the user interface and provided a good design layout to create the user interface from.

VLCJ was also a very useful framework used as it provided robust media and audio player components to integrate into the Vidivox application. The VLCJ components also came with their own video manipulating functions which provided all of the traditional video player playback options.

During the development process, the code was split up into three packages, the ui package, ui.utils(nested package) and audio package. These three packages took care of specific classes which were all related to each other. The audio package contained an AddedAudio class which defines the path and start time of the specific audio files, AudioConverter class which takes care of all the processes that deal with the festival and ffmpeg packages to preview, create and merge audios, an AudioPlayer class which creates an audio media player and an AudioConverterListener which listens for audio changes. As seen here, everything in this package has something to do with dealing with audios and audio objects, while in the ui.utils package it deals with functionality of the user interface components such as progress bars and buttons.

Issues in the development process includes goals of implementing certain design and functionality decisions which could not be accomplished easily such as previewing multiple audio files at ones along with the video and synchronising them together to respond to a single click of the playback options. It was not efficient or practical to include this function, an alternative option of previewing only a single audio with the video was implemented.

## Use of GitHub











	implemented get current time Hailubby committed 2 days ago
	implemented preview, fixed audio changing Hailubby committed 2 days ago
	minor change Hailubby committed 3 days ago
	clear all data, select all delete all implemented Hailubby committed 3 days ago
	fixed pitch problem Hailubby committed 3 days ago
	implemented preview of different festival voices/pitch/speed Hailubby committed 3 days ago
	fixed preview/stop bug, preview now works after preview is over Hailubby committed 3 days ago
	Edited default combo box and values Hailubby committed 3 days ago

Figure 6 GitHub recent commits

Initially the creation of the Vidivox program was a pair project to design the prototype. Because of this, GitHub was used to keep track of changes made to the code by both developers to handle version control. It allowed for a time efficient partnership as multiple developers may work on the same piece of code and then merge their changes together before pushing to the repository. The ability to make use of branched on Git was very useful too as different branches were made off from the main branch to test out different features and deciding on the best one. After the completion of the prototype, the rest

of the development of the Vidivox program was done as a solo endeavour.



GitHub was still used to make sure a copy of the work was safe in case anything should happen. But a few problems were encountered when trying to continue using the repository of the prototype for the rest of the development. New repositories were made due to the projects starting anew.

## Evaluation and Testing

During the development of the program, new functionalities went through numerous tests to ensure that it was performing as expected. In particular, when nearly all of the functionality was implemented, extensive testing was undergone to check the error handling of the program making sure that no exceptions were thrown and not caught, and that the program reacted appropriately to user input.

Error handling was much needed to check for invalid inputs to file names of video and audio files as the user is not permitted to name a file with whitespace, null pointers for when the user decided to cancel out of the JFileChooser as the path would equal null, invalid number format for selecting the delay/start time of audios and numerous other exceptions. A big problem was encountered in when the user chose to cancel the naming of an output video file as the processes would still try to run and merge mp3 files together with the video when there was no output for the video to be outputted to. This was handled by ensuring that the merge audio and video methods were only called when the user entered a valid output video name.

Also to prevent the user accidentally causing errors, many JComponents were disabled when they shouldn't be able to be used such as the video play back options (play, stop, rewind, fast forward and volume control) were all disabled when there is no video selected. This is because if there is no video to play, those buttons should not work, due to these buttons being disabled, it informs the user that they must select a video to use those options. When a valid video was chosen, the playback options were then enabled. The same thing was done with the preview of the festival text in the create audio pane. This is because the festival preview function gets the text from the text area to synthesize, but if there is no text to be synthesized, there will be no audio, so there is no need to give the user false security in waiting to hear a preview when nothing is written. What was done was if the text area was empty, the preview, stop and clear buttons were all disabled until something was typed in.

### Testing of functionality limits

The limits of functionality was tested, and the user was informed by JMessageDialogs if there was an error in their inputs, if a file already existed with the same name or if processes completes successfully.

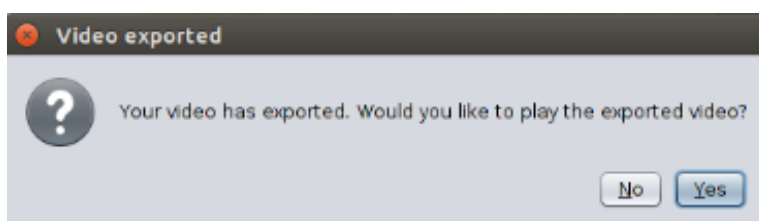


Figure 7 Successful video export message

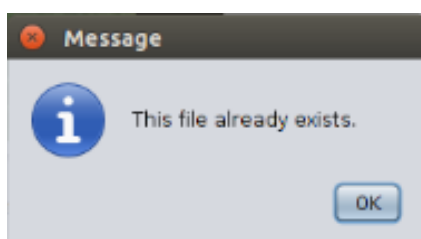


Figure 8 File already exists notification

## Prototype Feedback

The Vidivox prototype that was developed was put up for peer reviews from other developers of the same program to get feedback and constructive criticism of their opinion of the program at that point in time. It was both very helpful and encouraging as flaws of the program and what needed to be fixed was pointed out, also improvements that could have been made. It was made apparent that the graphical user interface of the media player was not very 'visually appealing' so taking this into consideration, the graphical user interface was remodelled so it was more intuitive and appealing. Also putting a limit on what types of files can be chosen to be played as the prototype had a problem with playing mp4 files from the user, so a change was made so that only avi and mp3 files were able to be selected.

## Conclusion

- This Vidivox program is compact, visually engaging and appeals to the intended audience as it provides easy to use functionalities to edit videos with.
- Vidivox is very easy on the eyes as the components are not all packed and cluttered into a small space, it is intuitively laid out with a soothing colour scheme which is good on the eyes for long periods of use.
- Few problems arisen throughout the development was taken care of efficiently and critique from prototype reviews was taken into account when developing the final project.

## Future Improvements

Further improvements would include:

- Additional refactoring of the source code to shorten classes and reduce coupling.
- Finding and implementing an efficient way of previewing all of the audio to be added alongside the video at one time.
- Synching the fast forward and rewind functionalities with the preview of added audio with the video.
- Next upgrade can include option of adding subtitles at any chosen point into the video

## Lessons Learnt

- Effective time management
- Client's requirements are constantly changing, so the programs developed must be flexible to changes and additional features are easily added
- Version control of code when working alone and alongside other developers is very important due to loss of code and losing work other people have worked on due to manual merging. Use of GitHub was very important.