VIDIVOX

Design Decisions

SOFTENG 206

Hannah Sampson

Software Engineering Design

Table of Contents

Table of Figures

Executive Summary

VIDIVOX is a piece of video editing software aimed at a non-technical adult user, with limited experience in video editing. Because of this the layout of the GUI is reasonably basic, with plain colours that ensure that all of the components are very visible and readable. The user can create new commentary files from text with different speeds, add audio files to a selected video, change the

position of these audio files, and edit the volume of each of the audio files added to a video. Finally the finished audio can be saved, as can each of the created audio files.

1 Introduction

There are a number of different video editors and each has its own range of functionality aimed at specific target audiences. Our project was aimed around the idea of being able to create a documentary-type video, where the video plays and keeps its own original soundtrack but a voice is added over the top, describing the video. This meant a lot of emphasis was put on creating appropriate audio commentaries, and being able to dictate where each of these audio files fits into the video.

Our work has involved developing a tool that lets the user take a standard video file and add different audio files to it, whether these audios are music files of commentaries that the user creates. We worked primarily in Java, with a Java specific video manipulation tool: vlcj. By working in Linux, we could also implement Linux system calls to process the media files, as well as working in speech synthesis with festival to create audio files from text.

There are a number of different video editors available, all aimed at a different level of user. VIDIVOX is aimed at a home user, and hence is required to have obvious functionality, and easy use. All of the design decisions throughout this work has taken into account how this kind of user would feel using the product.

All design and implementation decisions had to work towards creating “an easy to use” Video Commentator, which would be easy for a non-technical person to use. The main points are:

- A well designed graphical user interface
- Help on how to use the different features of the tool
- File retrieving abilities
- The ability to create, play back, overlay and save multi-media material

In the discussion section of the report, I have detailed the design decisions made at each stage of development, why these decisions were made, any features of the product and how the GUI interacts with the user, e.g. error handling, system feedback.

The development is broken up into seven sections:

- GUI Design
- Functionality
- Code Design
- Evaluation and Testing
- Future Work
- Conclusions

Each of these subheadings is broken up into sections based on the stages at which feedback from the client and other users was received as this signalled obvious breaks and re-evaluation of design implementation in the project.

This report also includes a user manual, which includes documentation and help with all available functionality.

2 GUI Design

2.1 Colour Consideration

This video player was aimed at a non-technical user, but still an adult. This meant that it was important not to use big bright colours and text as this would make the application look childish and put users off. Instead the buttons and general layout were geared towards being easy to read and view, but not overpowering or taking attention away from the video that was playing. Hence to keep everything easy to read the layout consisted of black text on a light grey frame (not white as if it is too bright this could easily cause eye fatigue), with a small to medium text to make sure users know that it is aimed at adults.

2.2 Discussion of Layout – Main Frames

2.2.1 Video Frame

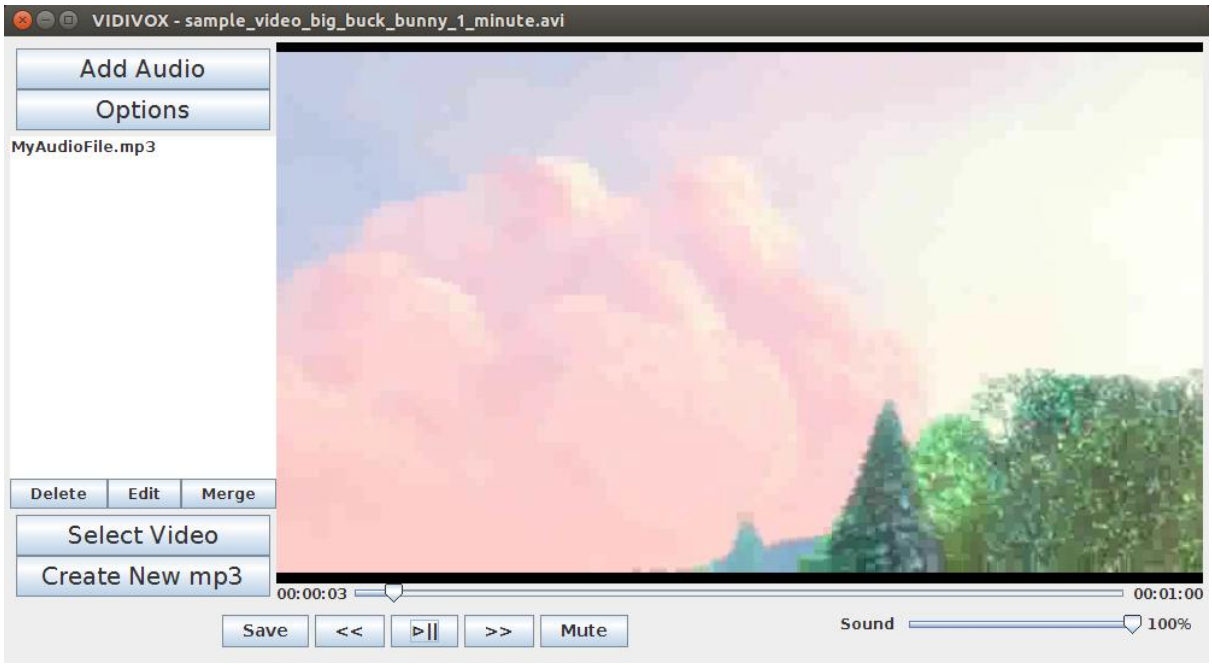

Figure 1: Main video frame, with video selected and one audio file added

The video manipulation buttons are all contained within a panel at the bottom of the screen. This was done as this is fairly standard for the layout of a video player, and the best way to show a user functionality is to make it recognisable, and similar to products that they would have used previously.

The panel to the left is the editing panel. All of the buttons that work with editing a file are stored there.

Buttons are grouped according to functionality, and more specifically what actions need to have occurred for them to be enabled. When VIDIVOX is first started, all of the buttons are greyed out apart from “Select Video” and “Create New mp3”, as all of the other buttons rely on a video to edit being selected before they will do anything. Hence “Select Video” and “Create New mp3” are grouped together, and are also large, obvious buttons so that the functionality of the player is unmistakeable.

Another important feature of editing software like this is that the user can see all the changes that have been made. This manifested in the white panel in the middle of the left hand pane, which contains a list of all the audio files that have been added to the video. This means that the user has a visual representation of the changes made, and can graphically select particular changes and modify,

delete or show (merge) them. The way that this information is available visually is very important as when working with a visual medium (video) having information portrayed in the same way works well.

2.2.2 Adding Audio Files

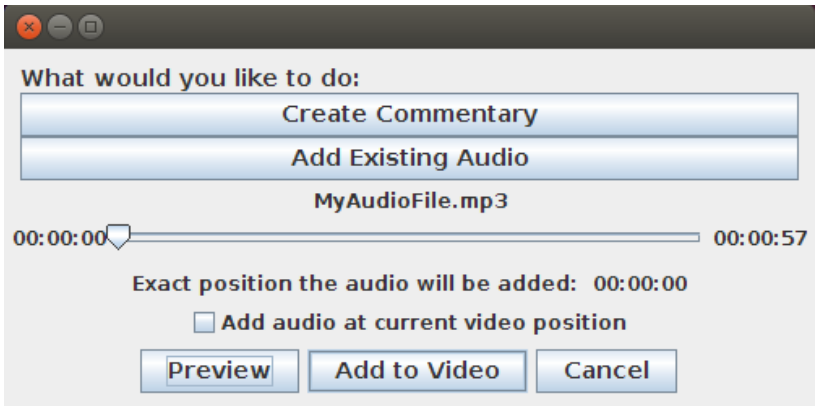

Figure 2: Add audio file to video

When showing the frame that is used to add a new audio or commentary file to the selected video, the most important factor was to make sure that the user knows, and can see everything that is going on.

Hence the audio file that the user has selected is visible on the frame.

The frame also shows that the user can select to either add the audio at a position in the video that they have already worked out, or they can choose to add the audio at the position the video was up to before the "Add Audio" button was pressed.

The "Exact position the audio will be added:" label is another way of constantly keeping the user informed. This time label shows the exact position that the audio will be added to if they user selects the "Add to Video" button. This will be the same as the slider's position if the checkbox remains unticked, or will be the same as the video's position otherwise.

2.2.3 Creating a New Commentary

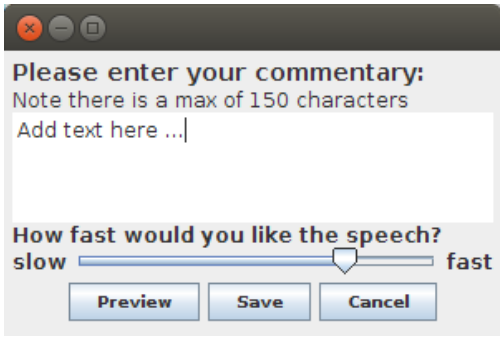

Figure 3: Create new audio

Based on feedback from the Beta submission a textPane (instead of a textField) has been implemented to store the input text so that the user can see all of the text that they type into the frame.

The slider gives an easy visual representation of the changes in speed that the user can apply to the speech.

All the action buttons are at the bottom of the frame. This is because as an English speaker, the user will be most accustomed to working from the top left to the bottom right, so the flow of functionality is laid out the same way. The user needs to input their text before they change the speed or preview the speech, and the preview should occur before the user saves it.

2.3 Presentation of Information

When giving the user information the main requirements were that the information was informative, relevant and non-invasive. An important part of being non-invasive is that when information is presented to the user, the user isn't required to act in order to carry on with their work (minimise pop-up messages), and that the developer doesn't "set the user up for failure" – if particular areas of functionality can only be accessed after something else occurs, make sure that these areas are fully inaccessible as opposed to allowing their use but then presenting invasive error messages.

2.3.1 Pop-up Messages

2.3.1.1 Error Messages

There is only one error message that is displayed to the user when the application is running. This is when the video player tries to play a chosen video and fails. This can occur when an audio file that has been added to the project is deleted (or renamed) and a fix for this has been outlined in the user manual. This is important to have as a pop-up error message, as the user needs to see and acknowledge the error so that they can fix it. If there was merely a small text message somewhere on the screen, this would not draw enough attention to the potential problem and the user could instead end up becoming frustrated with the lack of responsiveness instead of acknowledging that something has gone wrong.

2.3.1.2 Information Messages

Information messages are mainly presented when the user has selected an action that needs to happen in the background to maintain a responsive GUI. They pop up to let the user know that particular tasks have been completed (whether successful or not). The messages appear as dialogue frames as it is important for the user to know (and to acknowledge) that actions have been completed as often this means that they can then move on with the next required piece of functionality.

2.3.2 General Messages

General messages are defined here as text that appears on the frame at a particular stage in the project, and does not require any interaction. It is one of the most non-invasive way to present information to the user, but if not looked after can end up being almost redundant.

The way this product uses text is to show the user what is expected of their interaction, and to (wherever possible) limit their ability to input data that does not adhere to the requirements.

For example, when creating a new commentary file, the user is instructed via a general message that no more than 150 characters can be entered into the text field. When reviewing other video editors it was distressing that such limits were either only presented to the user after an input had already been given, or the limit was not enforced. Hence the decision was made that to make it easier for the user to see when the limit had been reached, they were simply unable to add any more text after they had added 150 characters.

With more time this kind of action would also have been applied to the user's input for filenames, i.e. when adding text they would not have been able to add characters that are not allowed, instead of being informed once they tried to save, that as per the general message, this filename was not permitted.

2.4 Interface Issues

2.4.1 Trouble Determining State of Playback

During testing it became obvious that checks intended to work out whether the video was paused, muted, fast-forwarding or rewinding were not working as intended. To start with the check were focussing on the media player component itself, and what state it believed itself to be in, by accessing methods associated with it. This was important mainly when implementing the functionality of different playback buttons.

In order to make the state of the video more obvious to the user, the plan was to change the text on particular buttons depending on what was happening. For example, the "mute" button would change to "unmute" when the selected to mute the audio, and the "play" (⊳) button would change to "pause" (||), when it was playing, and vice versa when paused again (or fast-forwarding/rewinding).

Because the state of the media player was not consistent enough (it would sometimes state is was paused while still playing, or that it was muted while sound was playing), it was decided that it would be easier if the buttons' text did not change, and this would not affect functionality but could save confusion if anything went awry.

However there were other issues surrounding this that did require a fix as the consistency could affect functionality. Here Boolean values were implemented that were manually set to hold the state of the player, set when VIDIVOX was first started up and then updated in the code every time the user changed something. As long as this was tested thoroughly, this was a much safer way of implementing particular functions. For example when the video is paused, fast-forwarding and rewinding cannot occur, or when the video is fast-forwarding and the user wants to rewind it, the video has to be played normally and the rewound, instead of just implementing rewinding.

3 Discussion about functionality of product

3.1 Creating Commentary Audio Files

When creating the new commentary files the user needed to be able to input text, hear it played back to them and then be able to add this to a video.

One of the considerations that needed to be taken into account was making sure that the speech created by festival was clear and audible throughout the recording. To do this the amount of text had to be limited as with too much speech, the festival voice begins to die, much as a human's voice falters out when they don't take a breath when speaking long sentences. A limit of 150 characters was selected as this is above the average number of words in a compound sentence, but not so far above that problems occur. This means that most sentences can be completed in this length, and if they go over the limit it is very likely that the sentence could be broken up into more than one sentence anyway.

A preview of the speech was necessary as when working with a computerised voice such as festival the user needs to be able to play around with extending vowel sounds and adding punctuation, in order to get the feel that they really want.

Because the nature of the project is to be able to work on multiple videos and add large volumes of audio, the product also supplies functionality to save the created audio file. This makes it easier for users as to reuse something the created audio file can simply be called to hand rather than each addition requiring all text to be written out again.

One of the other functionalities that was lacking in the first part of development was the ability to not only preview the input text, but also, once the audio has been created, preview this audio file with the intended video (at the appropriate position in the video).

3.2 Playing a Video

Playing a video is one of the most important features of the product, as for the user to be able to decide where they want to add their audio, and the features of this audio, they have to be able to view the video they want to edit. They also have to be able to view their completed design. Hence the functionality had to be simple, quick and reliable, as well as ensuring that the GUI remained responsive.

3.3 Adding Audio Files to a Selected Video

The user is able to select a position in the video file to add an audio to. The code automatically handles any errors that could occur from the user attempting to position the audio over the end of the film by

only allowing the slider (on the Add Audio Frame) to go up to the total length of the audio minus the total length of the audio file the user wants to add.

This is an example of limiting the functionality without being invasive (so that errors and error messages don't occur).

3.4 Saving Video Files

The way this product works is that any edited video that is played is actually a hidden temporary file. This means that the real work behind being able to save a video file is done when changes are made to the video and then played, rather than when the save button is pressed. This worked well as it meant that the video the user wanted to save would always be exactly the same as the one they were currently playing, as it is just a renamed copy of this video, and any changes that the user makes are guaranteed to work in the saved copy if the user can see that they worked in the preview.

3.5 Making Changes to Audio Files that have Already Been Added

Just as the user can select a specific position in the video for the audio to be added, once it has been added to the video this can still be edited.

3.5 Losing Changes

In this project, if the user is editing a video and doesn't save any of their changes, if they exit the application these changes will be lost (there is no system of storing information for the next start-up) the application makes sure to check that the user knows that exiting the program will discard all their changes, and gives them the chance to save it. If the user has not made any changes to the file since their last save, then this message does not appear, as there are no unsaved changes.

3.6 Festival and Sound Functionality

3.6.1 Speed of Speech

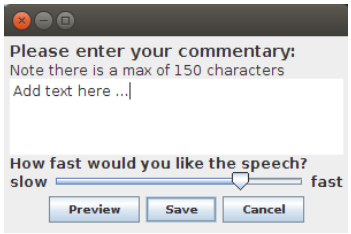

Figure 4: Use of slider to control speech speed

The festival functionality that was implemented was to do with how fast festival speaks the text it is given. This functionality was implemented as giving the user the ability to control the speed of their speech is important as this can give each audio quite a different feel, and make audio files suit different purposes with greater ease. When the user is creating a new commentary, they enter their text and then can use a slider to change and then preview what effect the different speeds have on the text.

In festival to make the speech slower you set Duration_Stretch to a higher value, but when a user looks at a slider like the one above, it is more instinctual for faster speech to have a higher value (to be to the right). Hence the code takes (4 – the slider value) to actually set the stretch, so that the view on the GUI was easier for users to understand and use.

3.6.2 Volumes of Individual Audio Files

VIDIVOX also allows the user to set the volumes of each individual audio file that they add to the video. This was useful as it meant that if there were louder sections of the original video that the user still wanted to hear speech over they could simply change the volume for the commentary that they placed over these sections, and vice versa.

4 Discussion about code design and development of product

4.1 Choice of programming language and packages used

4.1.1 Discussion on Use of Java in Project

Java was an appropriate language to use, especially with its support for GUI design. As it is an object oriented language, components with particular functionality were easy to create, use and most especially reuse. GUI specific features like buttons and messages are already created, so putting together the layout and flow was very easy.

An important feature of a good GUI is that it remains responsive at all times during its use. The main way this project implemented background functionality was through the use of SwingWorker, i.e. tasks could be given to a different thread to complete so that the main thread would always remain responsive. SwingWorker worked well when creating files using ffmpeg, as BASH commands were easy to run and execute in the background.

4.1.2 BASH processes

By implementing BASH processes, the product was able to make good use of the linux environments that it was built and designed for. While one of the main objective of code design is that it is transferable (doesn't rely on being run in a particular environment), by designing for linux the product could work with

4.1.3 vlcj vs. JavaFX

This project was centred on using vlcj code to play the video. It made for a good basic video player, but a lot of the video effects and playback functionality were time-consuming to code, and the overall layout of the application ended up looking very simplistic when compared with other video editing software. A simplistic layout does still work for an inexperienced user, as it doesn't feel overly complicated, but a more sophisticated layout would have given a more professional edge to the project.

With more time, using JavaFX as the basis for the video player would have been an asset to the application, as it makes implementing video playback easier, and improves the look of the GUI in the long run. However with the time limit, learning about a new bit of software that there had been no previous exposure to was too difficult.

4.2 Documentation of software design

Each class has a JavaDoc comment above it that states the main purpose of the class and how it fits into the overall scheme of the project. This helps to identify where failure occur or at least where they are made visible as this documentation clearly outline the purpose of the class and the tools it works with.

4.3 Development Process

4.3.1 Development Process Implemented

The development process that was implemented in the project was that of feedback driven design. While a standard brief was supplied and adhered to, this gave only the minimum functionality requirements, and after the time working on this product, did not necessarily give a fully formed view of the client's requirements.

By constantly receiving feedback on the work being done, through client meetings, individual presentations and peer reviews, at each stage of the project there were clear definitions of how to progress with the development, and what improvements could be made.

This is a very important way of being able to develop code, especially when the product is primarily concerned with providing a good user experience and positive interaction with the GUI design. GUI design is hard to test from a developer's point of view, so constant access to client feedback was very important to keep the design moving in the right direction. Design requirements can state many things about layout and design, but until the client is given a visual idea of what they could have, it is hard to determine what is necessary and what is not.

4.3.2 How this Affected the Project

This meant that the project, instead of having concrete requirements had rolling requirements, a list of features and needs that were constantly evolving.

One of the main things to come out of such a process was the importance of only implementing as much functionality as was needed, rather than steaming ahead and

4.3.3 Discussion on Process

This was the most appropriate process for the project, however even with the changing requirements, more structure and planning could have been applied to the development process to ensure that work on the project was carried out evenly with a set of goals firmly in mind. Because this planning process, while present, was not as thorough as was required for this scale of project, it meant that a disproportionate amount of time was spent on the functionality at the end of the time frame, when it would have been more appropriate early on to give more time for bug checking.

4.4 Innovative Implementation

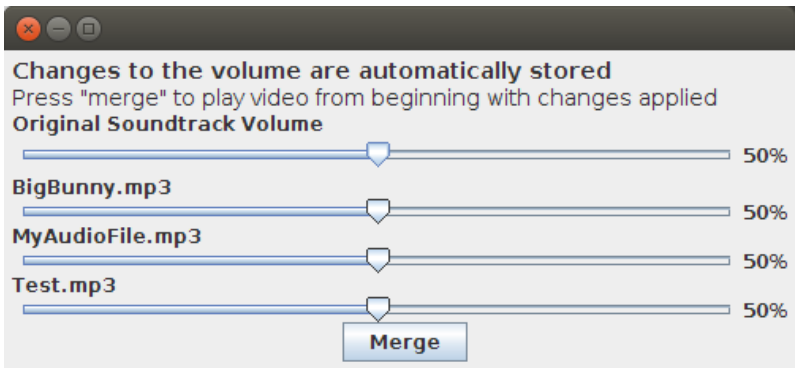

Figure 5: Editing audio volumes in project

The most innovative part of this product is the ability to edit the volumes associated with each audio file that has been added to the video. This also (in effect) gives the user the option to keep the sound track or strip it from the video.

The reason that this is so innovative is the way that the user does not have to edit the volume of each audio file in isolation, every audio associated with the project is present on the one screen to so that it is easy for the user to see the relative volumes of each of their files.

This required the use of a for loop based on the list of audio files stored in the main frame, which translated to a for loop, that for every audio file, added a new special class of panel (which contained the slider and labels) to a list, which were then accessed and added to the frame using the same index number from the for loop.

4.5 Other developmental issues.

4.5.1 Version Control

As in previous projects, the use of Github as a way of managing version control was highly recommended. This meant that any changes made along the way could easily be reversed, especially if these changes caused faults in other areas of the code. It also meant that the project could be accessed on any machine, and while majority of the work was done in the UG4 labs, it meant any work done elsewhere could be quickly checked in UG4 to make sure everything ran smoothly.

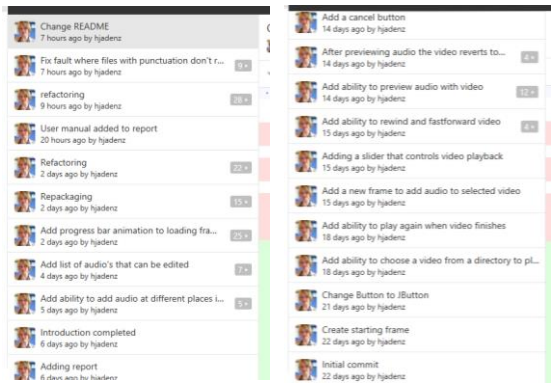

Figure 6: Evidence of Github use

Because Github had already been used for assignment 3, it meant it was easy to work with the code already implemented, but still rework it so that changes in flow and layout could be implemented, as well as refactoring and additional functionality.

Having the ability to test new features out in branches was also an asset, as it meant that testing could occur without worrying about breaking a team mate's work, or upsetting any progress being made.

4.5.2 Short-cut Keys

This product does not make use of any short-keys. This is because so much of this projector text based (creating commentaries, saving files) that it would be very easy for a user to accidentally cause a lot of damage if they end up closing a text screen that they were typing into.

As all of the functionality of the project is shown through buttons, all the functionality is obvious from the beginning, and inexperienced users would find navigating functionality visually a lot more approachable than hidden functionality accessed through the keyboard.

5 Description of Evaluation and Testing

5.1 Evaluation and Testing: Developer

As a developer most of the evaluation and testing was to do with the user experience and how the GUI responded to any actions. This meant that testing involved looking at what possible flows through the functionality could be implemented, and making sure each of the steps in each of these paths responded as was expected.

Some of the more valuable feedback came from clients and experienced peers reviewing different stages of the project. This gave valuable insight into possible user experiences surrounding the project, and gave views that weren't from someone that was immersed in development already.

5.2 Evaluation from Clients after Pair Project

5.2.1 Feedback from Clients

5.2.1.1 Comments on Code

"Think of better names for packages. Good to see lots of classes, but consider what sub-packages will make sense. Comment at class level, use correct indentation and get rid of unused imports. You do comment at class level, but be consistent in the format used. "MainPage" isn't the best, as this is not a website. Even better would be to have the main method in its own file at high level easy to find. Some constructors were getting too long, break them into smaller methods."

These comments were all very appropriate for this stage of the project, and changes were made to incorporate all feedback. By breaking the code up into more descriptive packages and sub-packages, it made it easier to add functionality, as now it was easy to see where the required functionality would fit overall. Good commenting made everything easier to understand, and hence changes were easier to make and faults easier to find.

By acknowledging that some of the class names ("MainPage") did not correctly show the actual functionality based on what the application was, this meant that more descriptive names were implemented which again helped with making the code more understandable.

5.2.1.2 Comments on Functionality

"Very straight forward to use. Liked how it created custom directory off home, and also suggested file names. I also liked how it indicated what files were selected, and overlay option straight forward. You might want to offer users chance to keep background sound – that seems to be stripped. Also would want to play video, without necessarily overlaying, so would need that option."

Again all of this feedback was taken on board, especially as for the next stage of the project it was necessary to implement this functionality. The feedback on being straightforward to use was definitely appreciated as it meant the layout and GUI designs, as well as the flow of the application were heading in the right direction and could just be taken further instead of needing reworking.

5.2.1 Changes from Assignment 3

Assignment three was focused on the base requirements of the project: being able to create audio files from text, and being able to add an audio file to a given video file. Between Assignment 3 and the Beta submission, the audio functionality remained the same, but work was put into being able to change what time the audio was added to the video, as well as being able to add multiple audio files, and keep the original soundtrack (points made in client feedback).

The Beta version of the project also had a different starting screen. Where previously the start frame had been a frame with just two buttons ("Create Commentary" and "Select Video"), this did not show user's the full functionality until they had selected a course of action. The two side of the project were too separated, and starting everything on the one screen (building to Figure 1) was more unified.

However as it was the Beta version, the attempted functionality was not actually in place, and there were a number of known issues (all documented in the README)

5.3 Results of Evaluation and Testing of product by allocated Class Peers

5.3.1 Feedback from Peers

5.3.1.1 Comments on Code

"The SaveVideo class could be moved into the backgroundtasks package, as it is a background task. More commenting is required to make things more understandable."

In terms of general code structure the feedback from peers was generally positive. As there was still functionality to be added, the information gathered from this feedback was continue to put more thought into packages and make sure everything contains helpful comments.

5.3.1.2 Comments on GUI

"When typing in commentary to add, it is hard to see what is typed as it all appears on a single line. Using Icons on buttons. Using a slider when selecting the position in the video for the audio file instead of needing to type things in."

Most of the issues with the GUI were to do with making things easier for the user, which is a very important part of any user/client based project. All comments were taken into account for the final design. Using icons instead of words makes the player more accessible and more intuitive.

5.3.1.3 Comments on Functionality

At this stage in the project the functionality was quite limited and while creating the commentary files still worked, actually adding audio to a chosen video had not yet been implemented. As all of the feedback was mainly around this issue, the feedback was definitely taken into account, as all of his was required functionality.

"Would be good if the user could create a new commentary file without having to select a video file."

This was one piece of functionality that was lost from Assignment 3, and adding it back in is not only easy to achieve but also increases usability, so these feature was added back in for the final product.

5.3.2 Changes from Beta version

Since the Beta version VIDIVOX has progressed quite rapidly. It has kept the main design layout and flow as was originally intended, and required functionality like adding audio files has been implemented.

The other functionality that has been added is the ability to manipulate the speed of speech in festival. This makes it more evident to the user that this software, though good for editing videos, is equally concerned with being able to make speech files, and having as much control over the computerised voice as is viable.

5.4 Final Design Testing

The final design testing was mainly down to looking at all the possible paths that a user could take through the project, and working through all of these trying to cause failures.

It also involved taking a step back from the developer aspect of the project and looking at how a user would see the project, and how it would meet their needs.

5.4.1 Error handling

Error handling is mainly performed through checking for existing files, and taking outputs from either the media player or any processes that are executed. While error handling could have been implemented further (i.e. looking at likely spots of failure and estimating which failures could possibly occur at these points), the most likely errors have been accounted for, and these let the user know of problems through pop-up messages.

5.4.2 Bugs and Fixes

The biggest issue with bugs were the ones based around timing issues. Because of the way some components interact with others, certain actions need to take place before others, and if this fails fatal errors can occur.

The biggest issue experienced with this was what happened when the video reached completion.

6 Future Work

6.1 Festival Functionality

Currently the only festival functionality that has been implemented is the ability to change the speed of the audio. In order to make the computer voice more usable, being able to change the pitch of the voice too, or making it so that the user can choose from a number of pre-set voices would mean that

if the user wanted to add character voices to the video, it would be possible to distinguish between them, instead of just listening to the same monotone voice.

6.2 Being able to Undo Changes

Because this is aimed at a normal user, the ability to immediately undo changes is important, especially when most software implements usability features that let ctrl+Z undo the most recent change. While the user may not notice the existence of such a feature, they are very likely to notice its absence, and as such if this product was to be developed further this would be one of the features to be added.

6.3 Saving workspace to come back to

When working with a video editor such as VIDIVOX, it is likely that the videos being edited would

6.4 Adding more error checking – e.g. files being deleting while working

One of the known errors is that if an audio file is removed or renamed outside of the application, this will cause the media player to be unable to play if any changes are made inside the application. With more time the product could be developed to include file checking while building each file, so that if a file no longer exists it skips it and deletes the audio from its list, instead of simply showing an error message. Changes like this make the application more robust.

6.5 Being able to trim video files

Because VIDIVOX is advertised as a video editor, and easy way to develop this tool further would be to let the user edit the beginning and end of the video before they start adding audio files to it.

7 Conclusions

7.1 Final VIDIVOX Product

The final product contains all the required functionality, as well as all the added functionality that was hinted at in feedback and client meetings. It is a robust piece of software that is fit for purpose and visually easy to navigate and use.

7.2 Learning Opportunities

This project was a good opportunity to have control over the cycle and processes involved in the development of a real software tool from beginning to completion. It required understanding how to work to deadlines and be able to interact with clients, which as software engineers is a valuable life skill.

Being in charge of the design process, and figuring out the pros and cons of feedback driven design is very important, especially in a field that is driven by client demand. It stressed the importance of a well-structured plan before getting lost in implementation details.

7.3 Teamwork and Communication

This project was split into multiple stages, and having the first stage as a chance to learn to work in a small team was a very different experience from previously in this degree. Having to learn to share workloads evenly, and distribute tasks, as well as trust another peer with working on different section of the overall product was a very different environment from individual projects, but one that important for later on in life.