VIDIVOX

Design Decisions

SOFTENG 206

Hannah Sampson

Software Engineering Design

Table of Contents

[1 Introduction 4](#_Toc433668461)

[2 GUI Design 5](#_Toc433668462)

[2.2 Colour Consideration 5](#_Toc433668463)

[2.3 Discussion of Layout 5](#_Toc433668464)

[2.3.1 Initial Frame 5](#_Toc433668465)

[2.3.2 Adding Audio Files 5](#_Toc433668466)

[2.3.3 5](#_Toc433668467)

[2.4 Presentation of Information 5](#_Toc433668468)

[2.4.1 Pop-up Messages 5](#_Toc433668469)

[2.4.2 Text Messages 5](#_Toc433668470)

[2.5 Main Frame 5](#_Toc433668471)

[2.6 Adding Audio 5](#_Toc433668472)

[2.7 Create a New Audio 5](#_Toc433668473)

[3 Discussion about functionality of product 6](#_Toc433668474)

[3.1 Creating Commentary Audio Files 6](#_Toc433668475)

[3.2 Playing a Video 6](#_Toc433668476)

[3.3 Adding Audio Files to a Selected Video 6](#_Toc433668477)

[3.4 Saving Videos 6](#_Toc433668478)

[3.5 Festival and Sound Functionality 6](#_Toc433668479)

[3.5.1 Speed of Speech 6](#_Toc433668480)

[3.5.2 Volumes of Individual Audio Files 6](#_Toc433668481)

[3.1 What was the motivation of the selected functionality? 6](#_Toc433668482)

[3.2 What were the usability decisions? 6](#_Toc433668483)

[4 Discussion about code design and development of product 7](#_Toc433668484)

[2.1 Choice of programming language and packages used 7](#_Toc433668485)

[2.1.1 Discussion on Use of Java in Project 7](#_Toc433668486)

[2.1.2 BASH processes 7](#_Toc433668487)

[2.1.3 vlcj vs. JavaFX 7](#_Toc433668488)

[4.1 Documentation of software design 7](#_Toc433668489)

[4.2 Development Process 7](#_Toc433668490)

[4.2.1 Development Process Implemented 7](#_Toc433668491)

[4.2.2 How this Affected the Project 7](#_Toc433668492)

[4.2.3 Discussion on Process 8](#_Toc433668493)

[4.3 Innovative Implementation 8](#_Toc433668494)

[4.4 Shortcut keys- motivations, implementation 8](#_Toc433668495)

[4.5 Other developmental issues. 8](#_Toc433668496)

[4.5.1 Version Control 8](#_Toc433668497)

[5 Description of Evaluation and Testing 9](#_Toc433668498)

[5.1 Evaluation and Testing: Developer 9](#_Toc433668499)

[5.2 Evaluation from Clients after Pair Project 9](#_Toc433668500)

[5.2.1 Feedback from Clients 9](#_Toc433668501)

[5.2.1 Changes from Assignment 3 9](#_Toc433668502)

[5.2.2 Justification for decisions 10](#_Toc433668503)

[5.3 Results of Evaluation and Testing of product by allocated Class Peers 10](#_Toc433668504)

[5.3.1 Changes from Beta version 10](#_Toc433668505)

[5.3.2 Justification for decisions 10](#_Toc433668506)

[5.4 Final Design Testing 10](#_Toc433668507)

[5.4.1 Error handling 10](#_Toc433668508)

[5.4.2 Bugs and Fixes 10](#_Toc433668509)

[6 Future Work 11](#_Toc433668510)

[6.1 Festival Functionality 11](#_Toc433668511)

[6.2 Being able to Undo Changes 11](#_Toc433668512)

[7 Conclusions 12](#_Toc433668513)

# 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

## 2.2 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.

## 2.3 Discussion of Layout

### 2.3.1 Initial Frame

### 2.3.2 Adding Audio Files

### 2.3.3

## 2.4 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.

### 2.4.1 Pop-up Messages

#### 2.4.1.1 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.

#### 2.4.1.2 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.

### 2.4.2 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, rather the user could not continue with their use until their input was fixed.

## 2.5 Main Frame

## 2.6 Adding Audio

## 2.7 Create a New Audio

# 3 Discussion about functionality of product

## 3.1 Creating Commentary Audio Files

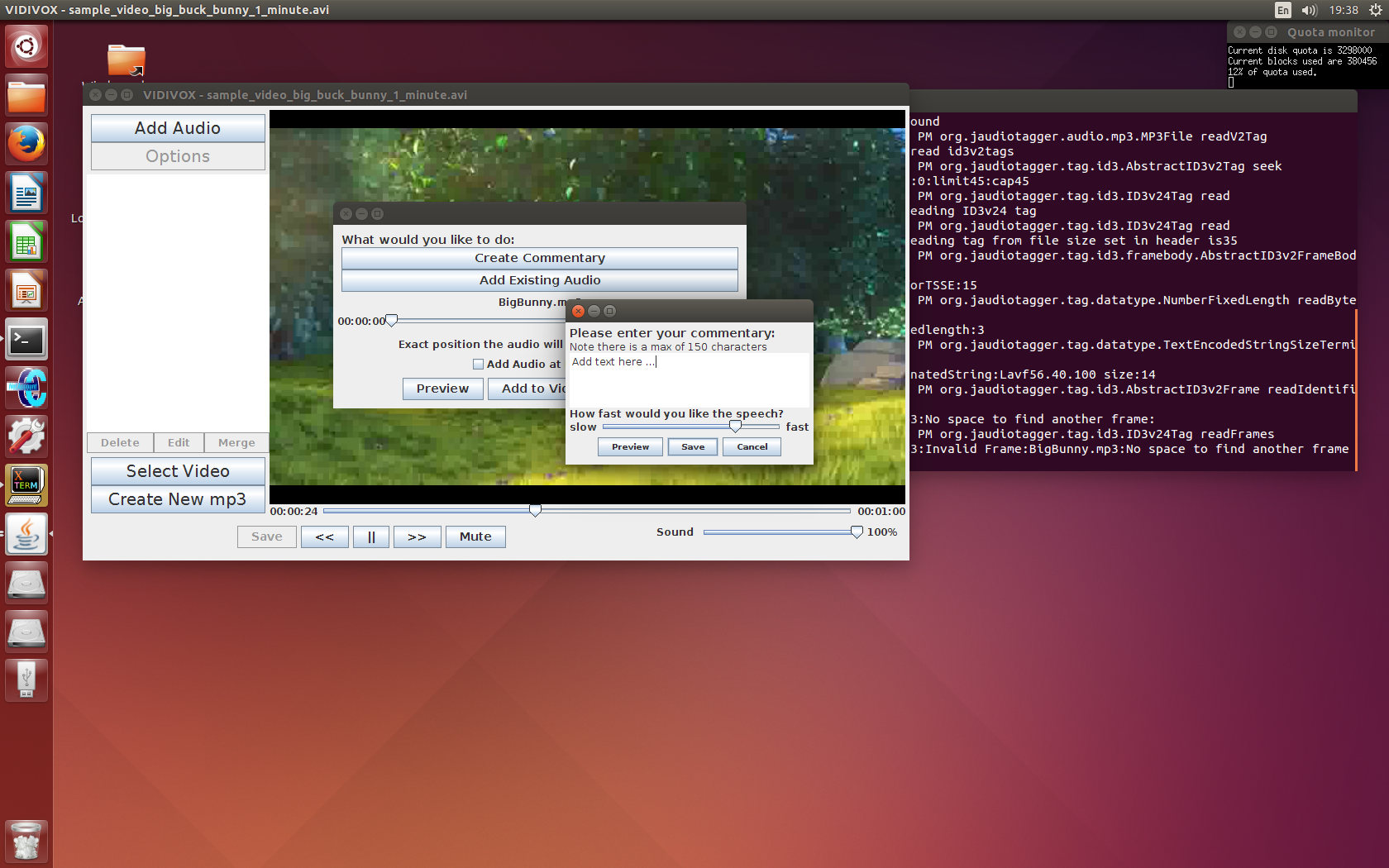
## 3.2 Playing a Video

## 3.3 Adding Audio Files to a Selected Video

## 3.4 Saving Videos

## 3.5 Festival and Sound Functionality

### 3.5.1 Speed of Speech

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.

### 3.5.2 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.

## 3.1 What was the motivation of the selected functionality?

## 3.2 What were the usability decisions?

# 4 Discussion about code design and development of product

## 2.1 Choice of programming language and packages used

### 2.1.1 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.

### 2.1.2 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

### 2.1.3 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.

## 4.1 Documentation of software design

## 4.2 Development Process

### 4.2.1 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.

### 4.2.2 How this Affected the Project

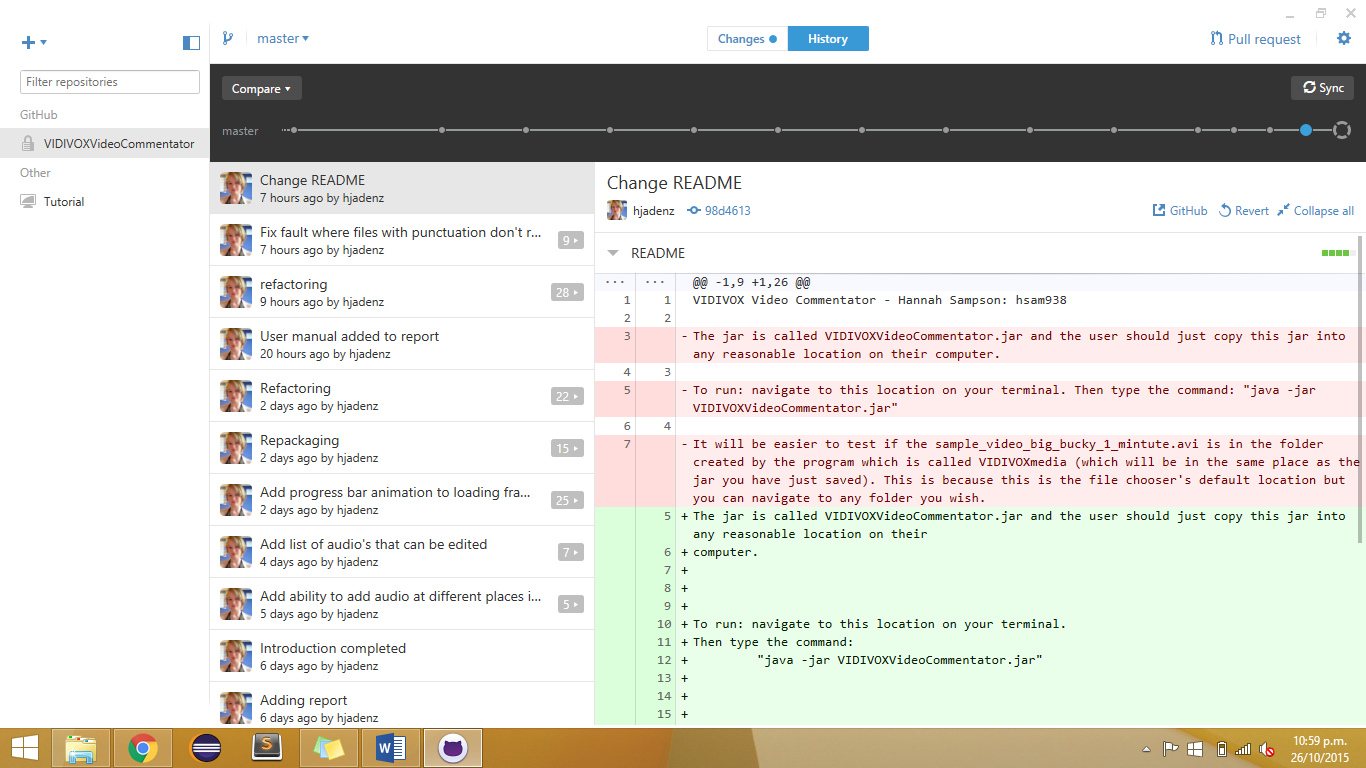
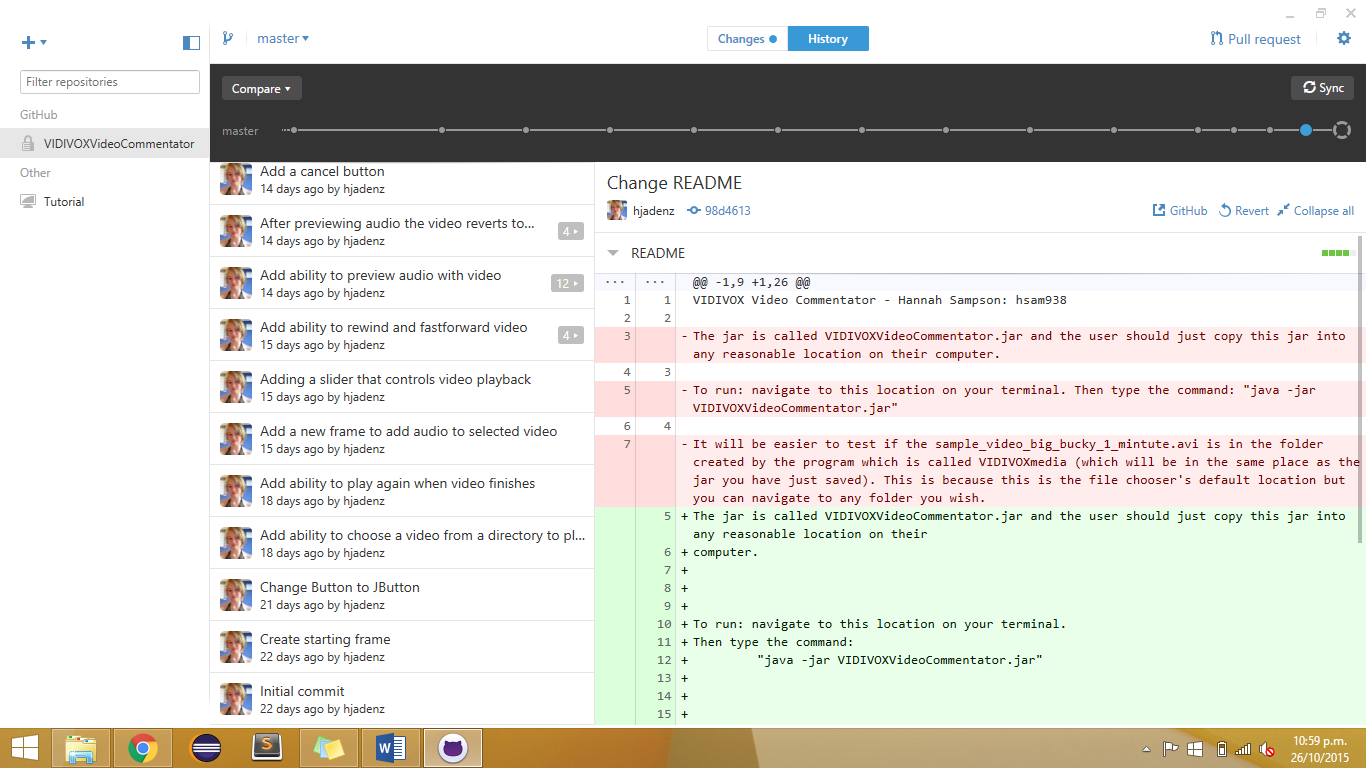
### 4.2.3 Discussion on Process

## 4.3 Innovative Implementation

## 4.4 Shortcut keys- motivations, implementation

## 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.

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

# 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.

### 5.2.2 Justification for decisions

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

### 5.3.1 Changes from Beta version

### 5.3.2 Justification for decisions

## 5.4 Final Design Testing

### 5.4.1 Error handling

### 5.4.2 Bugs and Fixes

# 6 Future Work

## 6.1 Festival Functionality

## 6.2 Being able to Undo Changes

## 6.3 Saving workspace to come back to

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

# 7 Conclusions