Testing & Integration Summary

Final Testing and Integration Summary

Document Control

Editor	Date	Update
Samuel Raeburn	**03/06/2015**	**Document Created**

Table of Contents

Testing & Integration Summary

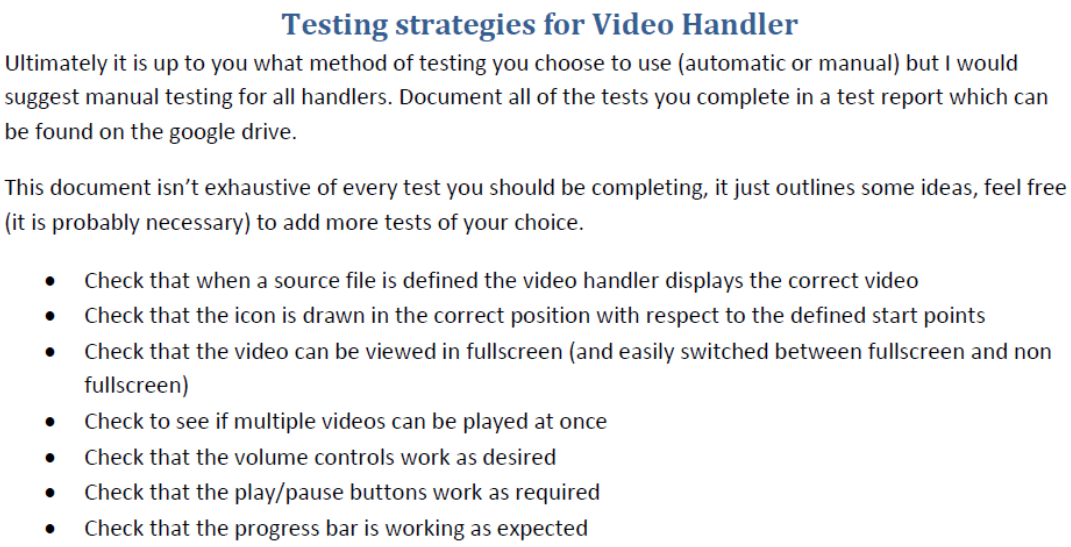
1.0 Introduction

This document summarises the testing, testing procedures and integration of all software produced by sofia thus far.
A brief description of how the testing & integration department functioned is included in order to give the reader of this document an insight into the inner workings of the testing division.
It is stated in the QA manual that the lead software tester will generally oversee all testing and integration; distributing testing strategies and test reports as well as completing tests themselves.
Throughout this document, problems which relate to an issue which has been tracked on GitHub are referenced using ***#x*** where x is the issue number.
This document will summarise the testing & integration of all software produced by sofia. Any problems encountered with testing & integration are outlined here, including, but not limited to, bugs which remain in the software and the justification for them still being present in the software.
This document will be written with reference to the final test & integration plan, which was produced on the 12th of March 2015. Any variances from the plan and how the testing and integration was actually carried out are listed and justified.
An outline of all documentation produced by the department is included along with the proposed purpose of each document.
Finally, future testing is considered and a summary of the testing and integration is reached.

2.0 Coordination of the department

Every member of sofia was involved in this department in some way.
The lead software tester, Sam Raeburn, ensured that the TETO (Test Early Test Often) initiative was used by way of distributing testing strategies to all members on a regular basis. Two methods of distribution were used; the first was used during the initial development of the media handlers. Testing strategies for a particular module were handed out to the pairs who developed that module. These pairs were tasked with swapping strategies with another pair to ensure that nobody tested code which they themselves developed.
The second method of distribution was used towards the end of development when modules of code became increasingly large and people tended to develop features rather than entire modules; it had become convoluted who had developed what module. In order to overcome this a document was compiled containing multiple testing strategies. This document was then given to every member of the group who then were tasked with contacting the lead software tester with which features they wanted to test, the lead software tester then in turn contacted all other members as to ensure that no feature was tested twice.

The test strategies were written in such a way that a method of tested was *recommended* but if the person responsible for testing that module wanted to go about the testing in a different manner that was allowed.
A screen shot taken from a testing strategy is shown in figure 1.



Testing strategies for Video Handler

Ultimately it is up to you what method of testing you choose to use (automatic or manual) but I would suggest manual testing for all handlers. Document all of the tests you complete in a test report which can be found on the google drive.

This document isn't exhaustive of every test you should be completing, it just outlines some ideas, feel free (it is probably necessary) to add more tests of your choice.

- Check that when a source file is defined the video handler displays the correct video
- Check that the icon is drawn in the correct position with respect to the defined start points
- Check that the video can be viewed in fullscreen (and easily switched between fullscreen and non fullscreen)
- Check to see if multiple videos can be played at once
- Check that the volume controls work as desired
- Check that the play/pause buttons work as required
- Check that the progress bar is working as expected

Figure 1: Testing strategy for the video handler.

The lead software tester ensured that any bugs found during testing were immediately reported to both himself and the lead software developer and that an issue was created on the GitHub.

The GitHub was used extensively for bug tracking as it worked well in conjunction with development; developers could reference bugs as they were fixed and the relevant commits can be seen on the GitHub.

It was paramount that all tests were well documented on test reports in order to effectively track the progression of testing & integration. The test reports offer quick insight into what unit was being tested, when it was tested, what tests were carried out and any fails that occurred. All of the test reports completed are included in the HTML tour.

3.0 Variances from Final Test and Integration Plan

The final test & integration plan can be found in the testing section of the HTML tour; it was completed on the 12th of March 2015 and outlines the plan from that date onwards of the testing & integration of the program. This document not only proved vital in ensuring that the program was developed in time but also that it was developed with minimal bugs and issues.

In reality the testing & integration varied greatly from this document due to unforeseen issues in the development. The key difference between the realised product and the plan is the number of iterations. In the plan it is stated that there will be 2 releases, covering a total of 4 iterations. The developed product did indeed have 2 releases but only 3 iterations. The reason for this reduction in number of iterations is the split in the development team. Once release 1 (the basic functionality of LearnEasy) was completed, the lead software developer began development of TeachEasy and the specialist software developer began to enhance the feature set of LearnEasy to include progress tracking, log in screens, certificates and a home screen. Although these features were included in the testing & integration plan, they made up iteration 4. Since these features were being implemented in parallel with the development of TeachEasy it was decided to merge them into a single iteration, iteration 3. This is explained in further detail in the iterative process review document. This caused all future dates defined in the plan to be incorrect.

Another major difference between reality and the plan is the testing strategy used for TeachEasy. As a company we massively underestimated the task of developing TeachEasy. Instead of testing TeachEasy upon its completion, as defined in the plan, testing was completed in conjunction with the development. The enormous feature set of TeachEasy meant it would be both impractical and irresponsible to wait until

its completion to test it. Every time the development team added a new feature, it was tested immediately, this method helped to avoid the pitfalls associated with waterfall development.

4.0 Bugs still present in software

Although the companies' best efforts went into ensuring a stable and high quality product there are still some bugs within the software at the time of completion of release 2. The solutions to these bugs proved to be either non-existent or very time consuming but research is still ongoing into how they can be resolved.

The video and audio handlers can both stream media from an online source. Implementing this feature introduced numerous bugs, all of which have been resolved (and can be found in test reports) apart from one. The video handler displays the elapsed time and duration of the video to the user, displaying this information requires knowledge of the length of the video, something which does not seem to be possible for an online hosted source. When the video is rendered onto the screen a console error will occur, as shown in figure 2.

```
Exception in runnable
java.lang.NullPointerException
        at javafx.scene.media.MediaPlayer.getCurrentTime(Unknown Source)
        at teacheasy.mediahandler.video.Video.setScan(Video.java:472)
        at teacheasy.mediahandler.video.Video.access$12(Video.java:467)
        at teacheasy.mediahandler.video.Video$PlayerStatusListener.changed(Video.java:821)
        at teacheasy.mediahandler.video.Video$PlayerStatusListener.changed(Video.java:1)
        at com.sun.javafx.binding.ExpressionHelper$SingleChange.fireValueChangedEvent(Unknown Source)
        at com.sun.javafx.binding.ExpressionHelper.fireValueChangedEvent(Unknown Source)
        at javafx.beans.property.ReadOnlyObjectWrapper$ReadOnlyPropertyImpl.fireValueChangedEvent(Unknown Source)
        at javafx.beans.property.ReadOnlyObjectWrapper.fireValueChangedEvent(Unknown Source)
        at javafx.beans.property.ObjectPropertyBase.markInvalid(Unknown Source)
        at javafx.beans.property.ObjectPropertyBase.set(Unknown Source)
        at javafx.scene.media.MediaPlayer.setStatus(Unknown Source)
        at javafx.scene.media.MediaPlayer.access$4100(Unknown Source)
        at javafx.scene.media.MediaPlayer$_PlayerStateListener$4.run(Unknown Source)
        at com.sun.javafx.application.PlatformImpl$4$1.run(Unknown Source)
        at com.sun.javafx.application.PlatformImpl$4$1.run(Unknown Source)
        at java.security.AccessController.doPrivileged(Native Method)
        at com.sun.javafx.application.PlatformImpl$4.run(Unknown Source)
        at com.sun.glass.ui.InvokeLaterDispatcher$Future.run(Unknown Source)
        at com.sun.glass.ui.win.WinApplication._runLoop(Native Method)
        at com.sun.glass.ui.win.WinApplication.access$100(Unknown Source)
        at com.sun.glass.ui.win.WinApplication$3$1.run(Unknown Source)
        at java.lang.Thread.run(Unknown Source)
```

Figure 2: The error which occurs as a result of streaming videos from online sources.

The bug was also once present in the audio handler but a fix was found and the bug no longer occurs, for this reason the team is hopeful that a similar resolution will be found shortly.

Another issue associated with the audio and video handlers concerns CSS style-sheets. When the controls are rendered for an audio or video object an error is thrown, this bug can be tracked on the GitHub repository using #134. The error is shown in figure 3.

```
java.net.ConnectException: Connection refused: connect
java.net.ConnectException: Connection refused: connect
        at java.net.DualStackPlainSocketImpl.connect0(Native Method)
        at java.net.DualStackPlainSocketImpl.socketConnect(Unknown Source)
        at java.net.AbstractPlainSocketImpl.doConnect(Unknown Source)
        at java.net.AbstractPlainSocketImpl.connectToAddress(Unknown Source)
        at java.net.AbstractPlainSocketImpl.connect(Unknown Source)
        at java.net.PlainSocketImpl.connect(Unknown Source)
        at java.net.Socket.connect(Unknown Source)
        at java.net.Socket.connect(Unknown Source)
        at sun.net.ftp.impl.FtpClient.doConnect(Unknown Source)
        at sun.net.ftp.impl.FtpClient.tryConnect(Unknown Source)
        at sun.net.ftp.impl.FtpClient.connect(Unknown Source)
        at sun.net.ftp.impl.FtpClient.connect(Unknown Source)
        at sun.net.www.protocol.ftp.FtpURLConnection.connect(Unknown Source)
        at sun.net.www.protocol.ftp.FtpURLConnection.getInputStream(Unknown Source)
        at java.net.URL.openStream(Unknown Source)
        at com.sun.javafx.css.Stylesheet.loadBinary(Unknown Source)
        at com.sun.javafx.css.StyleManager.loadStylesheetUnPrivileged(Unknown Source)
        at com.sun.javafx.css.StyleManager.loadStylesheet(Unknown Source)
        at com.sun.javafx.css.StyleManager.access$1900(Unknown Source)
        at com.sun.javafx.css.StyleManager$StylesheetContainer.gatherParentStylesheets(Unknown Source)
        at com.sun.javafx.css.StyleManager$StylesheetContainer.getStyleHelper(Unknown Source)
        at com.sun.javafx.css.StyleManager$StylesheetContainer.access$1300(Unknown Source)
        at com.sun.javafx.css.StyleManager.getStyleHelper(Unknown Source)
        at javafx.scene.Node.impl_createStyleHelper(Unknown Source)
        at javafx.scene.Node.impl_processCSS(Unknown Source)
        at javafx.scene.Parent.impl_processCSS(Unknown Source)
        at javafx.scene.control.Control.impl_processCSS(Unknown Source)
        at javafx.scene.Parent.impl_processCSS(Unknown Source)
        at javafx.scene.Node.processCSS(Unknown Source)
        at javafx.scene.Node.processCSS(Unknown Source)
        at javafx.scene.Node.processCSS(Unknown Source)
        at javafx.scene.Node.processCSS(Unknown Source)
        at javafx.scene.Node.processCSS(Unknown Source)
        at javafx.scene.Scene.doCSSPass(Unknown Source)
        at javafx.scene.Scene.access$3800(Unknown Source)
        at javafx.scene.Scene$ScenePulseListener.pulse(Unknown Source)
        at com.sun.javafx.tk.Toolkit$5.run(Unknown Source)
        at com.sun.javafx.tk.Toolkit$5.run(Unknown Source)
        at java.security.AccessController.doPrivileged(Native Method)
        at com.sun.javafx.tk.Toolkit.runPulse(Unknown Source)
        at com.sun.javafx.tk.Toolkit.firePulse(Unknown Source)
        at com.sun.javafx.tk.quantum.QuantumToolkit.pulse(Unknown Source)
        at com.sun.javafx.tk.quantum.QuantumToolkit$9.run(Unknown Source)
        at com.sun.glass.ui.win.WinApplication._runLoop(Native Method)
        at com.sun.glass.ui.win.WinApplication.access$100(Unknown Source)
        at com.sun.glass.ui.win.WinApplication$3$1.run(Unknown Source)
        at java.lang.Thread.run(Unknown Source)
Exception in thread "JavaFX Application Thread"
```

Figure 3: Error thrown by audio and video handlers.

The reason that this issue has not been resolved is that the conditions required for this to occur have not yet been found. This issue is currently only thrown on 40% of the companies' members' machines. The same code can be compiled on another members machine and the bug not be found, for this reason we believe this bug to involve the preferences and settings of eclipse more than the code which has been developed.

Another issue that is yet to be resolved concerns the colour picker in the editor. The colour picker, as supplied by Java FX, was used because of its intuitive interface and because its style fits in nicely with the rest of our product. Although it works as desired, the custom colour section doesn't. If the custom colour button is selected, a dialog is brought up; this dialog box has a modality such that the application cannot be used until a colour has been selected. It was found that under some circumstances the custom colour dialog would load off of the screen, causing the user to be unable to do anything with the program. The

issue can be tracked as #170 on GitHub. This issue is not to do with our code but to do with Java FX, a solution was created and formed part of the release of Java FX 8, which requires the use of Java 8; something which cannot be used for this project. This is the reason that this issue remains unsolved. As a team we feel that this bug is of high priority because it is the only known bug which can actually cause the program to crash, for this reason research is still ongoing.

5.0 Major bugs discovered through testing

Due to our rigorous testing procedures and the use of the Test Early Test Often initiative many major bugs have been found in the software and successfully removed. Some examples are given here. All issues found through testing can be seen in the comprehensive “testing failures” document.

Arguably the most serious issue found related to memory. If our program was used for an extended period of time an error would be thrown stating that the Java Heap Space had run out; leading to numerous null pointer errors which in turn would cause the program to crash. This is referenced on the GitHub as #215. Research was done into this issue and it was realised that there must be a memory leak somewhere in the program; something notoriously difficult to solve.The development team theorised that the memory leak could be coming from array lists which are never cleared, meaning every time the screen is re-rendered all visible objects are added to the array list, regardless of whether they are already present in the list or not. This had an exponential effect on the memory usage of the program. All of the array lists were correctly cleared and this issue ceased to occur. It is however entirely plausible that there are other memory leaks at different points in the program and that this issue may re-surface. In an attempt to check for this stress testing was utilized. A lesson comprising of over 65000 pages was created and opened by the program. The lesson behaved normally with no memory issues.
Another stress test was completed, this time a lesson comprising of a single page with multiple videos on it was opened. Although no actual Java Heap Space error occurred, the program did hang and other errors were thrown when more than around 25 videos were included on a single page. It is therefore not recommended that an excessively large amount of content is included on a single page. If excessively large amounts of content are required for a lesson it is advised to spread it over numerous pages since the first stress test proved that the software can cope with this.

Another serious issue encountered concerned the XML parser. If an XML file had warnings or errors associated with it, these warnings and errors would be stored in an array list produced by the parser upon parsing. If another, error and warning free, XML file was then parsed it would appear to have the same errors as the previously parsed file. This issue was tracked on GitHub using #145. This was due to the array list not being cleared between parses. Although the solution to this problem was a single line of code, if this bug had not been found a huge portion of our development work would be rendered useless by this error. It serves as a reminder of how important proper testing is in the development of a large scale project.

6.0 Testing procedures

As previously stated there were three distinct testing techniques used in the testing & integration of software, these were module testing, integration testing and stress testing.
The media handlers created by us were tested extensively, with extra emphasis placed on any handlers which were being sold to WaveMedia. This was done to ensure that they were of a high standard when they were sold.
The media handlers purchased from WaveMedia were also tested exhaustively as any bugs had to be reported within the warranty period.

Once all of the handlers had been tested they were integrated with the rest of the program, from this point onwards they were only tested using XML files; this was a form of integration testing. This method of testing proved invaluable in discovering bugs.
As aforementioned the testing of TeachEasy was done in parallel with development, this, again, proved invaluable. If this had not been done it is doubtful that the product would have reached the standard it is at today.

Once all integration testing was completed stress testing was used in order to see how the program would cope with large amounts of data – the program generally performed well, more detail in test reports.

Arguably the most challenging units to test were the XML Parser and the XML Writer. These modules change as the program grows, introducing more bugs. Several protocols were in place to try and overcome this. If a module was being retested it would be done so by creating XML files and opening them. This was done to test all of the handlers. This meant that the XML Parser/Writer was tested simultaneously with the handlers. A full description of the 70 XML files written for the purpose of testing is included in the description of XML files document. A spreadsheet was produced to aid with the testing of the XML parser and writer, the parser requirements spreadsheet outlines what should be classed an error and what should be classed a warning in the parsing of a lesson. A lesson with warnings can still be used whilst any errors mean that it is not possible to use the XML file in the program.

A standard test report template was used for all testing. This document not only provides space to write about tests being completed and the outcome of tests but also areas to write about prerequisites to the tests and the priority of the tests. This information proved very valuable when there were multiple tests to be completed at the same time as they could easily be prioritised.

7.0 Testing contributions

The following table outlines the contributions of the team to the testing of the software. Dates of testing are also included.

Test Report	Completed by	Date of completion
XML Test	Sam Raeburn	26/02/2015
GUI 1.0	Lewis Thresh	10/03/2015
Answer Box Handler	Sam Raeburn	07/03/2015
Answer Box Handler Retest	Sam Raeburn	15/05/2015
Audio Handler	Dan Berhe	10/02/2015
Audio Handler Retest	Sam Raeburn	21/05/2015
Graphics Handler	Sam Raeburn	20/04/2015
Image Handler	Sam Raeburn	18/03/2015
Multiple Choice Box Handler	Alistair Jewers	11/03/2015
Multiple Choice Box Handler Retest	Sam Raeburn	22/05/2015
Text Handler	Sam Raeburn	20/04/2015
Video Handler	Sam Raeburn	05/03/2015
Video Handler Retest	Sam Raeburn	21/05/2015

LearnEasy home screen	**Alex Cash**	**31/05/2015**
LearnEasy Log In	**Daniel Berhe**	**31/05/2015**
LearnEasy Progress Tracking	**Jake Ransom**	**25/05/2015**
TeachEasy Templates	**Emmanuel Olutayo**	**02/06/2015**
TeachEasy Log In	**Daniel Berhe**	**31/05/2015**
TeachEasy Copy & Paste	**Jake Ransom**	**02/06/2015**
XML Warnings	**Daniel Berhe**	**02/06/2015**
XML Retest	**Sam Raeburn**	**24/05/2015**
Integration Testing	**Sam Raeburn**	**10/05/2015**
Stress Test	**Sam Raeburn**	**25/05/2015**

It may appear that not all of the media handlers have been retested; some of them were retested in the creation of TeachEasy. If you consult the testing failures document you will see this.

8.0 Summary of submitted documentation

A lot of document has inherently been produced throughout the project, all of which will be supplied in the HTML tour. In order to aid with the analysis of this documentation this section will outline the purpose of each document.

- Description of XML files
  - This document describes the intended use for each of the XML files generated for use in testing.
- Testing XML files
  - These are XML files that were created to aid with integration testing. They were used so that the XML Parser can be tested in conjunction with the renderer, media handlers etc.
- Parser Requirements
  - This spreadsheet outlines the requirements which the Parser has been tested to. It defines whether a missing piece of data should result in an error or a warning.
- Final Test & Integration Plan
  - This document outlines the plan for both testing & integration, it was created on 12th March 2015 and was not kept to for reasons outlined in this document. The testing & integration GANTT chart is also included in this document.
- Test Strategies
  - These documents were created by the lead software tester to assist with the testing of modules. They were distributed to the company outlining what tests to perform on specific units.
- Testing Failures
  - This document outlines every test failure that occurred and if the issue has been resolved/retested or not.

9.0 Future Testing

As has been outlined above several bugs remain in the software. There is also potentital scope for development to continue and a cloud service to be implemented. If development does continue there will inherently be more bugs as a result. In order to ensure that the products being produced by sofia continue

to be of the same high standard as the software produced for this release the testing will not stop. The Test Early Test Often initiative has proved a viable option. As well as this a beta release is planned. The users of the beta test will have the opportunity to report any bugs they encounter to the company. These issues will be prioritised and resolved as soon as possible.

10.0 Summary

In summary the testing & integration of the product has been a success. Although the deadlines set in the final test & integration plan document were not met, there are good reasons for this. Not all desired functionality was implemented; the ability to play and pause lessons could not be included. The reason for this is that the implemented runtime data structure did not allow for local file saves; a requirement if this was to be implemented. Although it would have been possible to adapt the runtime class time restraints meant this was not feasible.

The testing department worked closely with the development department throughout the project which is reflected in the quality and feature set of the finished product.