Report on VOXSPELL for SOFTENG206

Luke Tudor
Department of Software Engineering
The University of Auckland
Auckland, New Zealand

***Abstract*—This report provides an overview into the processes and motivation used in the development of the VOXSPELL application for the teaching of spelling. This report covers the basics of the development of this application including the GUI design, functionality, development and evaluation.**

I. Introduction

For this project I produced a spelling aid using Java and Festival, to provide a way for people to practice spelling and develop confidence in the learning of new words. The intended audience for my spelling aid was the elderly user. As a result, my goal was to focus on an easy and enjoyable to use system which focused less on the learning of words and more on the competitive side of spelling. For this reason, the design was also intended to have descriptive labels for options and overall ease of use for any user, be they technically able or not. In addition, a focus was placed on a simplistic and clean look and feel in which all functions were easily identifiable and accessible. One of the special features of this program is the video manipulation through FFMPEG. This application allowed for this FFMPEG use by the user through an easy to use GUI. This GUI for video editing was designed to simplify the normally daunting process of using FFMPEG to modify a video in some way. Part of the reason FFMPEG is daunting is because of the large number of options available for video and audio editing. As a result, the video editing functionality is more a proof of concept, but it does allow for some powerful effects to be added to a video despite the limited control range that a GUI allows for, compared with the command line.

II. Discussion about the GUI design of the Product

A. Choice of Programming Language and Packages Used

For this project, I decided to use JavaFX, an extension of the Java library that is standard on recent Java Software Development Kits (JDK). The reason for this decision is that JavaFX not only allows for a greater range of functionality, JavaFX also allows for the streamlining of GUI development through such paradigms like Cascading Style Sheets (CSS) for styling. The developed project did not contain too much CSS owing to the fact that I preferred the specific control and type checking that the Java compiler allows for. However, if the same style is to be used more for more than one particular class, CSS is certainly a good option. JavaFX also allows for programming using FXML, a variation of XML (Extensible Markup Language). FXML was not used in this project since it seems like a minor optimization when it comes to the fulfilling of certain tasks. However, in the hands of an expert, FXML seems like a powerful tool for GUI creation in Java. As far as the external libraries are concerned, the only dependencies are the JDK, which is necessary for JavaFX, and festival. The reason for this is that JavaFX comes with an MP4 media player built into the standard library. This is great as it allows the removal of external libraries and external applications such as VLC, instead relying on the much used, and thus more likely higher quality, JavaFX standard library.

B. Colour Consideration and Display Layout

As far as color is concerned, the application does not rely on color much. Indeed, if the application were entirely grayscale, it would still be usable by almost everyone. This lack of reliance on color was designed to enhance the experience of elderly users who may have color recognition issues, such as colorblindness. Indeed, it is untested, but this application should even be suitable for those affected by any sort of color blindness, such is the lack of reliance on color. Despite the lack of reliance on color, the application is not completely devoid of color. The default coloring that JavaFX uses contains a number of colors. Most prominently, light gray, blue and black. These colors are used together to create a display that stands out, but is easy on the eyes. In contrast, a display entirely made from green and red widgets would be very difficult to use because of the clashing colors. The colors added to the application were some darker grey colors and small areas with red and green. These red and green shapes were not necessary for the components that they represented (Ticks and Crosses), however these colors were bright enough to stand out and draw the user's attention to them. This was done to highlight the importance of the shapes they represented. The darker grey colors were also used to separate different parts of the window to make it more obvious what the user should be looking at, at a particular point in time.

C. Presentation of Information

In terms of display layout, the application is designed in such a way as to promote simplicity and cleanness. Fortunately, this is the same design philosophy behind the default look-and-feel used in JavaFX. Because the intended style of the application was so similar to this default look and feel, there were not many changes that had to be made. This simplistic and clean style was used to allow the user to take in all possible options quickly and easily such that it is immediately obvious how to use the application. Most adults do not particularly care about a colorful and detailed layout, instead preferring a simple design which is easy to use and does what you expect. Since the target audience was the elderly community, a larger than normal sans serif font is used to enhance readability for those with diminished eyesight. In order to make functionality as obvious as possible, information

is displayed as simply as possible, providing the maximum level of abstraction whilst still providing the user with enough information to easily determine what is going on at any stage. Many examples of this philosophy are found throughout the application. The main menu itself is a simple vertical array of buttons. This makes the application more accessible to a first time user as it is immediately obvious what each of the buttons do from their names. Another example is the lack of tick labels for the video editor sliders. It is not helpful to know that contrast can be between -2.0 and 2.0 for FFMPEG. This is information that is not at all useful to the user. There are many applications that overwhelm the user with more information than they will ever need. But for such a simple application, like this one, it is important to provide only enough information that the user will need at any given time. This allows the user to explore the functionality on their own, making interaction with the application a far more natural experience.

D. Other Interface Issues

A situation that I experienced during development of the application was that it seemed that the application was somewhat limited by the screen size of my laptop. This was caused by the spacing out of components on the GUI combined with the larger than normal font size that the application was using. On a larger screen such as that of a PC monitor or terminal, these concerns would have been trivial. It is interesting to note though, the possible and unexpected limitations of developing GUI applications from a laptop alone. Another interesting interface issue was the interaction with Festival. Festival is interesting in that it seems that the way that festival speaks a word is dependent on what voice is used to speak that word. This may seem obvious, but it means that often a word can be said with one accent or dialect, but sound very different to the word spoken with a different accent. This means that increasing the number of accents that the user can choose, will also allow the user to more consistently determine exactly what Festival is saying. Another limitation of Festival is that it is not possible (as far as I know), to specify phonetically how a word is to be pronounced. This can lead to ambiguities, such as the lack of difference in pronunciation of words such as ‘and’ and ‘an’.

III. Discussion about functionality of product

A. What was the motivation of the selected functionality?

The motivation for the selected functionality was to make the application as easy to use as possible for as many users as possible. Some of the motivation and inspiration for this were the many GUIs that I had used previously, which required too much technical knowledge or information to use properly. The goal of the application was also to cater more toward the elderly population. This decision dictated the use of precise labelling and instructions for those who may not be as technically able. As a consequence of the decision to design the product towards elderly people, a lower focus was placed on the learning of words, which elderly people probably don’t care about. Instead, the application leaned more towards making the application an enjoyable and non-technical experience of usually very technical tasks, such as voice synthesis and video filtering. Aside from that, the core functionality was brought about as a product of discussion with clients as to what functionality they expected from such an application.

B. What were the userability decisions?

For the application functionality, I focused on designing the application such that the user would require very little knowledge of how the application worked. Part of this meant creating little to no need to go to the install location of the app to change or configure files. This is certainly fine for a developer to do, but users shouldn’t have to worry about the architecture of a particular application. The specific things that I tried to streamline, were the use of symbolic links to lists that the user added. This allowed the user to make small modifications to a list, if they spelt the word wrong when making the list. But it also allowed for easy removal of a list by simply deleting the file that the symbolic link pointed to. Another aspect of usability was from the overall design and way that the user interacted with the application. In a previous iteration of the design, in assignment three, The GUI was quite limited in functionality such that a new window would pop up for most of the new screens in the application. This was bad since the user was forced to think about many screens at the same time. The change I made was to use a single window to display all information, and to provide ‘Back’ buttons to go back to the parent scene. This decision greatly increased usability as a result. In the video editor part of the application, usability was also very important. FFMPEG can be very complex because of the sheer number of operations that it can perform. Therefore, the interaction with FFMPEG had to be simplified substantially in order to make it usable for someone inexperienced with FFMPEG. This was done by using components like radio buttons and sliders to limit the inputs to FFMPEG to within acceptable bounds. For example, the ‘saturation’ value can be between 0.0 and 3.0, this is information that was hidden from the user by means of a slider without the tick labels. If the user had to enter that value using a text field, then the user would have to be aware of the valid range for saturation values and the user would have to be trusted not to enter an invalid number like 4.0.

IV. Discussion about code design and development of product

A. Documentation of software design

When it comes to programming languages, I find that Java can be somewhat verbose. This is caused by Java’s roots as a C inspired imperative programming language. What ‘verbose’ means in this circumstance is that there is seemingly a large amount of unavoidable repeated code in Java, especially when it comes to configuring a GUI. An example of this is the need to explicitly set the ‘padding’ for each container widget individually. This is somewhat undesirable as it makes for code that is, in general, less readable and more difficult to maintain. JavaFX seeks to remedy this with the integration of CSS and FXML. Of the two, CSS seems far more useful in reducing code duplication than FXML for a GUI. Unfortunately, the application was finished with the configuring stages before I realized the power of CSS in simplifying the styling of a GUI.

In light of this, it cannot be said that Java was the best choice of language. However, I feel that CSS can fix many of the problems caused by imperative programming whilst still providing the flexibility of the more imperative Java style. The only other libraries used in the application were the JavaFX standard libraries and of course, Festival. The development process was a form of test driven development in that small changes were made, tested and then committed to Git. This style created an evolution over time of the project and allowed for simple 'plug in' style changes for the GUI. For instance, if a new button was needed to be added to the main screen, it was trivial to add that button to the button panel, create an event handler, and implement the functionality. This design philosophy also encouraged the use of many small classes to allow for a large degree of extensibility and flexibility in implementing the design. One thing that could have made the development process easier would have been the use of CSS from the start. This would have allowed for a hierarchy of CSS files that could layer styling on top of each other, in much the same way that inheritance works in Java. It is interesting how in the development of the application, that the changing requirements provided for a greater impetus for the maintenance of previous software but also made some aspects of maintenance in the previous iterations of software seemingly irrelevant. An example of this is in the commenting of specific parts of the code. Commenting on specific parts was sometimes rendered meaningless in the face of constantly changing requirements, making the code itself, not the comments, the main focus of maintenance. This project is therefore a good example of the negative impact that changing requirements can have on software and the need for as much generalization as possible in the beginning stages of an application. In addition, the changing requirements also prompt a serious look into the perils of object oriented programming. It is telling that despite efforts to minimize the size of classes and to maximize the number of classes and packages, that some classes remained stubbornly immutable. In order to shrink such large classes, anonymous inner classes and overriding would need to be employed to such a large extent that the two classes might as well have been one. A possible solution to this problem would be the use of temporary files to store data to be transferred between class. However, this would be undesirable as this would require the entire application to be built around this principle from the start, in order for the class hierarchy to make sense. Another possible solution would be to use inheritance to share data between classes. However, this is also undesirable because at that point, inheritance would be so far from its intended use that it would be made meaningless.

B. What was the development process?

The development process used was a very iterative one. The way it would work was that I would think of, or recognize a requirement, then think about and study the best way to implement that particular requirement. Once the best way of implementing a requirement was determined, I did so and performed rigorous tests to make sure that everything worked as expected. The tests were similar to those used in equivalence partitioning since that is still the optimal method for testing most GUI applications. Once the tests were performed and the changes validated, I used a Git repository to store each iteration of the program. Git was a great help because a number of times mistakes were made in the particular iteration. Most often, these mistakes were caused by a lack of understanding of the difficulty of implementing such a requirement. This simple test driven development process allowed me to be sure at every stage, that the product did not cause any error states to occur. This iterative approach also had the advantage of providing me a usable application at every stage of its lifecycle. This was good as it allowed me to more easily visualize what further steps needed to be taken to improve the project. One disadvantage of such an iterative approach, is that a constant change of requirements often meant constant refactoring so that the individual pieces of the product fitted together in a more sensible way. This is fine in a small project in which one person is aware of the functionality of all individual components at once. However, in a larger project, much longer iterations are required to prevent the constant refactoring. Interestingly, this principle is taught in the SOFTENG 254 course, in which it is seen that a development process such as RUP (Rational Unified Process) that is intended for large development teams, will have a longer iteration time than XP (Extreme Programming), a process used by much smaller teams. From my experiences with this project, it seems that in future, adopting a process such as XP would be advantageous. Even if there is only one person involved and especially if the requirements are somewhat unclear.

C. Any innovation in implementation?

In the implementation of my design, I added a 'stopwatch' to monitor how long it took the person to complete the quiz. This is not a feature that I noticed many people do and was accomplished quite simply, using the JavaFX animation framework. Another innovation in my design was the ability to let the user manipulate videos as a reward, using a GUI which adapted the user inputs to FFMPEG commands. The use of FFMPEG in this way was interesting as it highlights just how powerful the creation of new processes from inside a Java program can be. One interesting thing about the video editor, was that the application could track the progress of the video rendering. This is obviously trivial when using FFMPEG from the command line, since detailed messages with timestamps are constantly displayed. This functionality, combined with the other information FFMPEG supplies, provides an easy way to track the progress of a video rendering. However, I found that this method of tracking the FFMPEG progress would be very difficult to implement directly in a Java program. The solution would be to use some sort of heuristic to determine the progress of the rendering. The heuristic used was a calculation of the ratio of the size of the generated file with the size of the original file at certain intervals. This monitoring was done in a similar way to the updating of the stopwatch. Namely, by using the JavaFX animation framework to create a timer. Obviously, this method would not work very well if the generated file was significantly different in size to the original file. I found that this was only the case when the framerate was decreased significantly. In this case, the significant decrease in file size is to be expected. In all other cases, this heuristic provided a reliable way to track rendering progress without having to perform complex input stream reading functions.

D. Shortcut keys- motivations, implementation

In this project, the application only contained one shortcut key that was not in the standard JavaFX library. This shortcut was for entering the word in the quiz. If the user typed in a word into the text field and they pressed 'ENTER', the word would be entered into the system as if they clicked the 'Enter' button on the GUI. This allowed for very rapid entering of words and much more natural usage. More importantly, this shortcut allowed the user to constantly keep their hands on the keyboard whilst typing. When testing the product, I found it very annoying to constantly switch to the mouse just to enter a word. This shortcut allowed users to concentrate on the quiz by not forcing them to worry about clicking a button and distracting them. What is interesting about JavaFX, is that many of the other shortcuts that most users would be familiar with, such as 'CTRL+A' or 'TAB', which are very prevalent in most text editors, are also available by default. I acknowledge that these other shortcuts also make using this program easier. However, the motivations for including them will not be discussed, since they were not implemented in the project. The way the project implemented the 'ENTER' shortcut, which is in my opinion, the most useful one, was not entirely obvious. This was accomplished in the project by setting a lambda as the key listener of the text field, then running the method that the 'Enter' button lambda called when the 'Enter' button received an action event. This method was only called by the text field lambda, if the 'key code' of the key pressed was equal to the 'ENTER' key code, and the 'Enter' button was not disabled.

E. Other developmental issues

Some aspects of JavaFX are not particularly helpful for development. One example of this is that, some standard library classes should not be extended, but there is no way to tell what these classes are. For instance, the shape class should not be extended, instead custom shapes should be an aggregation of other shapes. The reason for this is that in order to subclass the shape class, a protected method of unknown function has to be overridden. This problem is not just found in the shape class. There are many other classes which should not be extended. Not because the classes are final, but because in order to extend these classes, unusual abstract methods have to be overridden. This problem of overriding classes is also an issue in the Swing framework. In the case of Swing, the problem is that serialization warnings need to be suppressed whenever a widget is overridden, an understandably unsatisfying solution. Another development issue I had was with the way that JavaFX works with action event handlers. In my application, it seemed that the action events would be fired by sources that were not the user's mouse click. Instead, it seemed that JavaFX fired action events whenever a property changed as well. One example of this was when the program set a combo box, an action event was fired. This event was difficult to handle as it was undesired and indistinguishable from a combo box selection event. This difficulty made working with combo boxes much more difficult as a consequence.

V. Description of Evaluation and Testing

A. Evaluation and Testing by ones self

When it came to testing the application, I was very diligent in testing and created a test class for every major GUI component developed. This was quite easy to do, since the goal of each test class was to create an instance each GUI component in isolation. This was done so that I didn't have to navigate through the menus and pass all the tests every time tests on a particular component needed to be done. This testing methodology was good since it cut out the dependencies on all the other components, meaning that any bug found was with the current class for the current iteration, and not on some other class. The style that used was similar to a test driven development style. The main difference was that because of the few requirements for all of the components, features of the components had to be built up piece-by-piece to achieve the desired functionality. The isolated test classes would then be used to see if the result was as expected. Because the development of all the components was incremental, the source code had to be laid out in such a way as to promote the easy addition of new requirements. This meant the usage of large numbers of symbolic constants to easily provide the necessary information to the new code in the classes.

B. Results of Evaluation and Testing of product by allocated Class Peers

The feedback received in the peer evaluation was very useful. In general, most of the feedback was very helpful in determining areas where my application was lacking. In particular, one of the peer reviews detected a bug in my application that I hadn't considered. One peer review made me think more carefully about the possible sources of error from the lists that the user was to provide. Another comment from that peer review was that the statistics could be displayed in a table. This was an idea that I hadn't considered so the feedback was much appreciated. A peer review also gave me more tips as to how the layout should be done. The other peer review suggested improvements to the way that voices are selected in the quiz. Due in part to how busy I was at the time, the beta version submitted is significantly different to the final version. A number of changes were made. Notably the many bugs in the beta version were removed. These bugs ranged from errors caused by invalid user input to errors in the application logic which prevented certain aspects of the quiz from working. Therefore, the non-bug changes will be documented in this report instead. The more notable changes include the switch from opening new windows for each of the different tasks the application performed, to the use of the same window and the addition of a back button to go to the parent screen. This was an obvious change, but at the time the best way to implement this functionality was not known. The advantage of using the same window, is that it is much more difficult for users to get the application into some sort of error state. This technique also makes window managing easier for the user who don't have to keep track of many windows at a time. A change that was prompted by the peer review process was the use of a table to display statistics. It was difficult at first to figure out how tables work in JavaFX. It seems like the implementation of tables is a bit more complex than is necessary. Despite this, I

was pleased with the decision since tables provide a much needed way to easily sort and manage data. Something that would have been difficult and tedious to implement. A significant part of the final submission that was not in the beta version was the ability to manipulate and view videos from inside the application. I had previous implemented both these functionalities separately but had not found a satisfying way of combining the two parts. As such, the video manipulation was not in the beta. This was unfortunate as feedback on my somewhat novel way of integrating FFMPEG into Java would have been much appreciated. Despite the lack of feedback, I was pleased with the result of the video editor reward as it shows the extent of what is possible in the combining of GUIs with shell commands.

C. Other Evaluation ?

A notable source of other evaluation was from assignment three. In particular, my partner Aprajit Gandhi, and Nasser Giacaman. Although these sources of feedback were not directly for the project, it was useful to receive feedback on a related product. The feedback from Aprajit was primarily directed at the shortcomings of Swing. This encouraged me to try and create the project using the more recent JavaFX framework. At the time I was already interested in JavaFX and was researching some of the ways to use JavaFX. In the hands of an experienced designer, JavaFX would be more powerful, and easier to use than Swing. In addition, JavaFX seems to allow for more complex graphical design, which benefitted my application as a result. Nasser also provided evaluation for assignment three, which I considered and took forward in the project. Some of the helpful feedback was the use of a singular size for all windows or in my case, all states of the window. This feedback was important in making me think about what space there was to work with, and how the space could be used best. Amongst his other feedback were discussions on the logic of the quiz itself and how the quiz flowed. Ideas which made me think more critically about what a user would most want out of a quiz. These other evaluations were certainly helpful for me and allowed me to think more about ways to improve my program.

VI. Future Work

Some of the work that could be done in future might be to provide definitions for the words when prompted. This could eliminate the confusion that would arise from homophones like ‘effect’ and ‘affect’. The reason why this functionality was not implemented in my application was because I did not want to introduce the unreliability of internet access to an otherwise robust program. Another task for the future could be to somehow improve the speaking ability of the speech synthesizer. This could be achieved by simply using another type of speaker instead of festival. This could also be accomplished by modifying the way that festival speaks. However, this would be a very complex way of accomplishing this goal.

Conclusions

The time spent developing this application has made me consider more deeply, the difficulties that can arise from the development of an application. This project has taught me the importance of very relevant aspects of software development such as version control, software architecture, development processes, and even the general approach to take when tacking a non-trivial GUI design project such as this one. Overall I am pleased with the resulting application, and glad that I had the opportunity to design such an application and improve my development skills as a result.