**C868 – Software Capstone Project Summary**

**Task 2 – Section A**



|  |  |
| --- | --- |
| **Capstone Proposal Project Name:** | Text to Phoneme Converter |
| **Student Name:** | Edward Sawyer |

**Table of Contents**

Contents

[**Table of Contents** 2](#_Toc43226255)

[**Business Problem** 3](#_Toc43226256)

[**The Customer** 3](#_Toc43226257)

[**Business Case** 3](#_Toc43226258)

[**Fulfillment** 3](#_Toc43226259)

[**SDLC Methodology** 4](#_Toc43226260)

[**Deliverables** 4](#_Toc43226261)

[**Implementation** 6](#_Toc43226262)

[**Validation and Verification** 7](#_Toc43226263)

[**Environments and Costs** 7](#_Toc43226264)

[**Programming Environment** 7](#_Toc43226265)

[**Environment Costs** 7](#_Toc43226266)

[**Project Timeline** 8](#_Toc43226267)

# **Business Problem**

**The Customer**

The customer is potentially anyone developing animation projects. This includes everyone from the beginner YouTube content creator to professional studio animators. The customer is potentially anyone that finds themselves synchronizing audio dialog to mouth animation. The synchronization process is long and tedious when it is done manually.

## **Business Case**

Animation is a tedious process. Unless computers are aiding the process, every aspect of animation takes manual labor and time. Fortunately, computers are wonderful at doing tedious work. Computers are already aiding with most aspects of animation. However, there are still some areas where tools could help reduce production times and make life easier for animators. One such area is the process of synchronizing audio voice tracks to mouth keyframes. The process is still largely done manually.

Some animated scenes are dominated by mouth-audio synchronization. For example, many scenes in the animated television show South Park have very little movement other than the mouths themselves. A tool that could speed up the synchronization process would significantly aid in meeting production deadlines.

As humans are prone to error, having an automated process for synchronization would also foster a greater accuracy in speech reproduction. In a competitive production environment synchronization must be completed as quickly as realistically possible. If a human is doing all the work manually, many small errors are likely to make their way into the final product. Using a conversion tool based on a database provides the ability for consistent and speedy reproduction of synchronized mouth animation data.

## **Fulfillment**

A tool to convert text dialog to a sequence of matching images is possible with the use of an open source pronunciation dictionary and a set of 2D phonemic mouth images. Carnegie Mellon University hosts an open source pronouncing dictionary that provides the data for breaking words down into their phonetic constituents (http://www.speech.cs.cmu.edu/cgi-bin/cmudict). The dictionary lists most words in the English language with the corresponding phonetic symbols that compose the word.

This pronunciation dictionary is then broken down into a database with tables for Words and Word-Parts. Queries are then generated to pull the individual word-part symbols on command. The query results are then mapped to corresponding mouth-phoneme images. The database allows for a structural way to store, add, and edit phonetic word data without redundancy.

The database also requires a table for Mouth Pairs. This table allows for any number of sets of images to be mapped. This table may then be updated for any number of image sets. For example, a set of mouths can be mapped for a lady character, a robot character, or a simple stick figure character. Sets can also be generated for side angles, happy dispositions, or sad dispositions.

The result of this application development project will be a useful tool that greatly reduces the time required to synchronize audio dialog with animated characters. Also, the use of a database will allow for a great deal of expansion. Future versions of this application could be implemented to map not just images, but 3D data sets for more complex animation.

# **SDLC Methodology**

The Software Development Life Cycle methodology chosen for this project is the Iterative Model. This model is well suited for this project because of its inherent flexibility. Also, this methodology allows for revisions and improvements as the development process unfolds.

The Iterative Model is favorable primarily because of its flexibility. In the case of an evolving final product, such as this project, a flexible methodology is great for providing opportunities to change the course of development. For example, two possible paths for developing this application’s database exist. This lack of a clear path forward prevents the use of less flexible SDLC models, such as the waterfall and V-model methods. The Iterative Model is the best choice for this project, but other flexible models could have been used. Models such as the Spiral method, or a variant of Agile.

Flexibility is needed for the design of the application's database. There are two possible directions the design of the database may take. The first path is to create a single large table with the word name in the first column followed by columns of the word's phonetic symbols. This approach is simple but does not conform to any of the normal forms. The second path is to create a database from the pronunciation dictionary that fully conforms to at least the 3rd normal form. This path has the potential to run into technical or performance problems. For this reason, a flexible methodology is a great planning tool.

This methodology benefits from an application design focusing on extensibility. The Iterative Model allows for new requirements and functions to be added in each subsequent iteration. The application is first designed around the core functionality of the application. The application must, at a very minimum, convert text to mouth-phonemes at the end of the first development cycle. The subsequent iterations of development implement the non-core functionality of the application. Additional iterations may be pursued if requirements grow or change.

# **Deliverables**

The completed application has a single primary requirement. The application must accurately convert text dialog to a matching sequence of images that can be copied into an animation workflow pipeline. A user must be able to enter in the text, make some style choices, and prompt the application to generate the image sequence. Along the way in this development process, several deliverables are created. They are as follows:

* Project Deliverables
  + Project Schedule
    - As an important part of the planning process, a detailed project schedule should be developed. This aids in keeping production on track. This schedule will be flexible and will likely change as the project develops. However, this schedule will be useful in keeping an eye on deadlines and the next development phase
  + Test Plans
    - Test plans will have to be developed in order to ensure the final product is both fully functional and meets the intended initial requirements. The test plans will focus on the core implementation of the conversion processes. They will need to verify that the input is converted properly, and that the steps from the raw input to the final image sequence are working as intended.
  + Class Diagram
    - A class diagram will be developed. This diagram will link the related methods and classes. Maintenance or future development will find this deliverable to be highly useful in working out the specific functionality of the underlying code.
  + Verification and Validation (Example Animation)
    - A simple demonstration animation will be produced to both show the correct functionality of the application as well as the usefulness of the process in saving considerable time in mouth/audio synchronization.
* Product Deliverables
  + Wireframe
    - A wireframe of the User Interface will also be developed. While this project will be iterative, an interface that makes sense for a user will be very important. Developing this deliverable will help aid in a better user experience.
  + Database Implementation (Convert the Open Source Dictionary to a RDB)
    - A database will be developed for this application. The database will be hosted on a cloud service and will be available as a standalone deliverable.
  + Documentation
    - A user guide and an administration guide will be produced. Future users will find the user guide helpful in the finer points of the operation of the application. The administration guide will be necessary for non-developers to make sense of the database structure and for adding more mouth image sets to the choices available in the application.
  + Final Dialog Converter
    - The final functioning application will be the last deliverable in this project. This deliverable can be distributed and deployed as desired to any interested parties for evaluation or use in animation production.

# **Implementation**

This application will be implemented inside a typical animation workflow. The script and/or the audio dialog recording needs to be available. The image sequences will be generated one sentence or phrase at a time. The final image sequences can be imported into any video editor or compositor.

Most animation workflows will create a script first, then record the dialog from the voice actor. The script, at a minimum, will be needed before any phoneme sequences can be generated. The script will be entered into this application one sentence at a time. The result will be an image sequence that will match the input sentence.

After the image sequences have been exported, they can be imported into the project’s preferred video editor or compositor. Generally, a sequence of images can be imported into an editor as a video clip, or similar object. This object will then be overlaid on top of the relevant section of the dialog audio track. The object can then be stretched to match the audio’s start and end points. The final result is an animated mouth that phonetically matches the actor’s dialog.

# **Validation and Verification**

Validation and Verification of the planned requirements will come in the form of Unit Tests and a demonstration animation. The Unit Tests will test for errors or bugs in the specific implementation of the data model and data flow. The test will look for discrepancies in the produced data against the expected data. This will help ensure the process from raw sentence input to individual words to phoneme symbols to phoneme images is working logically and as expected.

A simple animation will be produced to validate the overall process from the perspective of an end user. If the application can pass a complete set of Unit Tests, but does not produce a visually coherent image sequence, then the project has not met its intended goal. A successful demonstration video will also visually demonstrate the usefulness of the tool developed in this project.

# **Environments and Costs**

## **Programming Environment**

The primary programming language for the development of this standalone application is Java 8. The development IDE is IntelliJ IDEA Community version 11. This IDE is free and open source and implements the modern JUNIT 5 testing framework. NetBeans and Eclipse were two other valid choices for the IDE, but the friendly and intuitive user interface in IDEA creates a more comfortable platform for development.

A significant project deliverable is a Java Executable file. This executable is eligible to run an any system capable of running the Java Virtual Machine. However, the initial development will be on a Microsoft Windows 10 machine.

MySQL is open source and comes with a powerful workbench application that is vital to the development of this application. This workbench is essential for developing the exact queries required for data retrieval. Database queries are crafted and tested inside this workbench before being migrated to the application source code.

The database structure is also developed inside MySQL Workbench. Tables and their relations are created and linked inside the workbench before being applied to a MySQL instance. Also, the database tables are populated in the workbench using the Table Data Import Wizard. The operational database will be deployed on the Amazon Web Services RDS service for easy delivery and access.

## **Environment Costs**

The expenses for this project are minimal, but not entirely free. The use of open source platforms in the programming language, IDE, and Database avoids most expenses. The only expenditure in development, other than time, is the hosting of the database on Amazon Web Services. As it turns out, the Free-Tier services on AWS are not entirely free. Thankfully, they are rather cheap. Expenses are expected to run close to $15.00 per month for hosting the cloud-based database.

# **Project Timeline**

Once the feasibility of both converting words to phonemes and converting the pronunciation dictionary to a relational database are established, the project can begin. Initially, two iterations will be expected. The first cycle will develop the core functionality of the application. Then once the input-text to image-sequence process has been verified, the second iteration will focus on fulfilling the rest of the requirements. It is possible more iterations will be needed to complete the project if a course correction is needed. A functioning core application is eligible as a deliverable after the first complete iteration. The complete application and user guides should be available as deliverables after the second iteration.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Cycle | Event | Details | Dependencies | Hours | Calendar Day |
| 0 | Pre-Project | Verify the feasibility of converting words to phonemes, and the pronunciation dictionary to a database. | None | Many | Pre-Project |
| 1 | Planning | Core Needs | Write project proposal and obtain approval. | 2 | June 1 |
| 1 | Requirements | Core Needs | Review project requirements. | 3 | June 1 |
| 1 | Design | Core Needs | Decide on modular design approach. | 3 | June 2 |
| 1 | Implementation | Core Needs | Install IntelliJ IDEA IDE. Host database on a cloud service. | 14 | June 2-5 |
| 1 | Verification | Core Needs | The word to phoneme process must be in place for testing. | 4 | June 6 |
| 1 | Evaluation | Core Needs |  | 4 | June 6 |
| 1 | **Milestone** | Core Application Functional |  |  |  |
| 2 | Planning | Extra Requirements | Only begin the second iteration after the core functionality is in place. | 3 | June 7 |
| 2 | Requirements | Extra Requirements | A clear familiarity of the requirements will be helpful. | 2 | June 7 |
| 2 | Design | Extra Requirements | A modular approach should have been established. | 3 | June 7 |
| 2 | Implementation | Extra Requirements |  | 18 | June 8-9 |
| 2 | Verification | Extra Requirements | Testing can take place as the modules are completed. | 6 | June 10 |
| 2 | Evaluation | Extra Requirements |  | 8 | June 11 |
| 2 | Documentation | Delivery Prep | All major changes should have been implemented. | 10 | June 12-15 |
| 2 | **Milestone** | Full Application Functional |  |  |  |
|  |  |  |  |  |  |
|  | Total |  |  | 80 | 15 |
|  |  |  |  |  |  |
|  | Total Planning |  |  | 16 | 3 |
|  | Total Development |  |  | 54 | 8 |
|  | Total Documentation |  |  | 10 | 4 |