# LingoPros

# Software Design Document V.2

February 21, 2018

Josh Shaffer, Luis Montes, Erik Strauss, Matt Quintana

Sponsors: Dr. Okim Kang & Dr. David O. Johnson

Mentor: Ana Paula Chaves Steinmacher

NORTHERN ARIZONA UNIVERSITY

Overview: The software design document will detail the different requirements that we have discussed with our client in order to implement the project to their satisfaction. It will also display the problem statement, our solution vision, and our plan for development.

# Introduction

Non-native English speakers often have a hard time articulating their speech not just in what they say, but how they say it. Proper emphasis on words or just general speech tone is something that native English speakers do not have to think about in their everyday speech as it is something that just comes naturally. For example, in a conversation between a non-native English speaker and a native speaker, the non-native speaker may not be aware of how, by not matching the tone or cadence of the native speaker, the conversation can begin to feel awkward and unnatural. Recognizing these differences in tone, pitch, stress and other prosodic features between native and non-native English speakers, is one of the key focuses of Dr. Okim Kang's research at the Applied Linguistics Speech Lab (ALSL) at NAU.

Currently, students working with Dr. Kang at the ALSL have to do all of the speech analysis by hand, listening to audio samples and recording what features they hear or recognize. This process can take hours on end even for a small 10 second audio clip, so as such, an automated process of speech analysis that is easily available to the students would drastically improve their research capabilities by saving them time and energy. Dr. Kang has already developed her own experimental speech analysis program that can recognize prosodic features, however it is cumbersome to use as it is spread across two different machines and operating systems. Additionally, the model that the current program uses, a framework based on a distinct form of speech analysis posed by David Brazil called *discourse intonation* [1] is not recognized as the standard in speech analysis. A set of conventions called Tones and Break Indices (ToBI) is considered to be the standard for annotating speech features, and so the clients want to show that the Brazil program performs just as well or better than other automatic analyzers such as AuToBI.

For our project, Dr. Kang wants us to develop a program that is able to analyze recordings of non-native English speakers and  grade their English proficiency based on aspects like tone and emphasis, and secondly she want us to implement an easily accessible web portal for users to upload and analyze audio samples in one location. We have been brought on to develop a speech analysis program based on the ToBI model as well as implement the web portal that will house the clients prosodic labeler program based on the David Brazil model. We will develop a machine learning program to score the proficiency of speakers using the output of the AuToBI program as an input. The reason to use machine learning is so it can better calculate a more precise score that is consistent with human scoring. We will also be developing the web application to house Dr. Kang's program (David Brazil Model) so that her program can be accessed from anyone, anywhere, assuming Okim allows them to have full access to the site.

In this document, we will cover our exact plan of implementing the AuToBI prosodic labeler and how we plan on hosting the David Brazil model online. We will talk about specific technologies and software that we will use to accomplish that as well as the architectural designs of our programs. By the end, there should be a clear understanding on how we will accomplish our tasks and how the programs will run.

# Implementation Overview

We have a clear plan in mind for implementing both parts of the project and delivering a finished product. In this section, we will outline the core functionality of the projects, what technologies we are going to use to accomplish this, and some other minor information about the structure of the project

## AuToBI Prosodic Labeler

The first part of our project will be a prosodic labeler that can analyze speech files that are input and output a proficiency score of the speaker. The core of this program is to use a third-party software named AuToBI, which is an automatic speech analyzer that uses the tones and break indies model (ToBI). AuToBI produces raw data about the speech analysis in the form of Attribute-Relation File Format (.arff) files which will contain a series of attributes and a large set of data records, where each entry in the record corresponds with a value for a specific attribute. The produces arff file will be passed into the Java Machine Learning API called Weka so that it can perform an attribute selection process to determine which attributes are the most important for a particular analysis. From the most important attributes the program will find the mean value of those attributes from each file and pass those numeric values into a neural net that we are developing. The neural net will process the inputs, and then make a guess at the proficiency level. If incorrect, the weights on the net will be adjusted and the input will be processed again. We want the net to progressively learn what values constitute “good” English so that it may correctly estimate a speaker’s proficiency in the language. Overall, the AuToBI program will label the speech file with different prosodic features, and the machine learning software will analyze those features and score the speech files accordingly.

### David-Brazil Online Host

The second part of our project is hosting our clients program online to allow them and their students easier access. Our clients have developed their own prosodic labeler, using the David-Brazil model. We are going to take this model and host all the MATLAB applications on a server on the cloud using Digital Ocean. Digital Ocean is a cloud-based computing platform, which allows people to quickly and easily deploy server applications online. Digital Ocean will allow us to easily and reliably host our program without the need of one of our own machines or needing to host through the ITS department at NAU. The site will also be a user-friendly environment where users can login and upload their own audio files into the server to be analyzed. The login database will be built with MongoDB and it will keep track of user-specific information, like past analyses performed. The server will be responsible for running the MATLAB code to analyze the files, then the website will take the results and display it to the user.

## Architectural Overview

For the first part of our project, our team has to develop a speech analysis program that processes speech samples using AuToBI and then determines the speakers proficiency in English based on measurements it obtains through the speech analysis. Initially the neural network knows nothing so it must be trained first in order to calculate the proficiency as shown in Figure 3.2. After the network is trained it will run the analysis quickly to guess the proficiency score as shown in Figure 3.1.

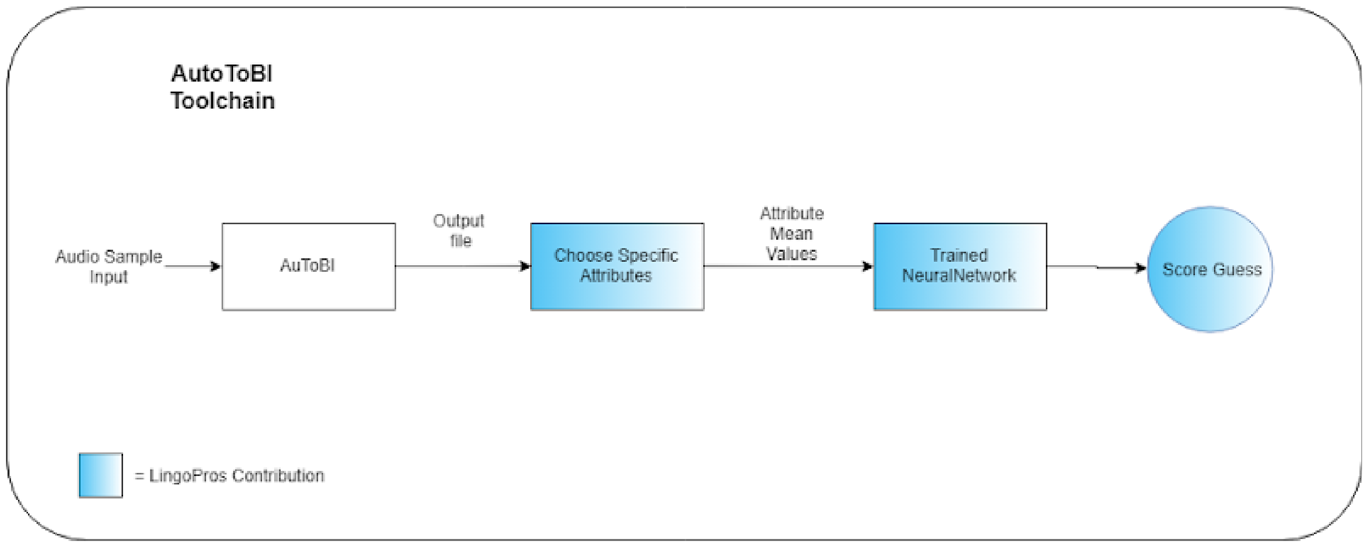

**Figure 3.1-** A diagram of the basic architecture of the AuToBI Toolchain

**Figure 3.2-** A diagram showing the process for training the neural network.

The second major part to our project is the Web Toolchain where we will give our clients a product that allows ease of use for the part of the clients, their students, and other researchers to input phone files. Phone files are output from the Kaldi linguistic program after running an analysis on input sound files, they are used because unlike other audio formats, they are compressed to save space and easier analyze the prosody. Finally the file is analyzed using the clients David Brazil Model (also known as “DBM”) implementation as shown in Figure 3.3.

The process starts with a would-be user creating an account on our website after prior approval from Dr. Kang with a special access code only she can distribute. Next, the would-be user attempts to log in to our system and the server checks for valid input credentials. If successful, the server then awaits input of a phone file for analysis from a now logged in user, else, the server will remind the user to try again or create an account (which requires Dr. Kang’s approval through a valid access code). Provided the user successfully logged in, the user will input a phone file that the server will take in, start an instance of MATLab, and use that file as input for the DBM’s main function to analyze. Upon completion of analysis, the server will now have a results file from DBM that the server then organizes into a results page using the JavaScript PUG view library (previously known as “Jade”) that the user can then look at and draw academic conclusions.

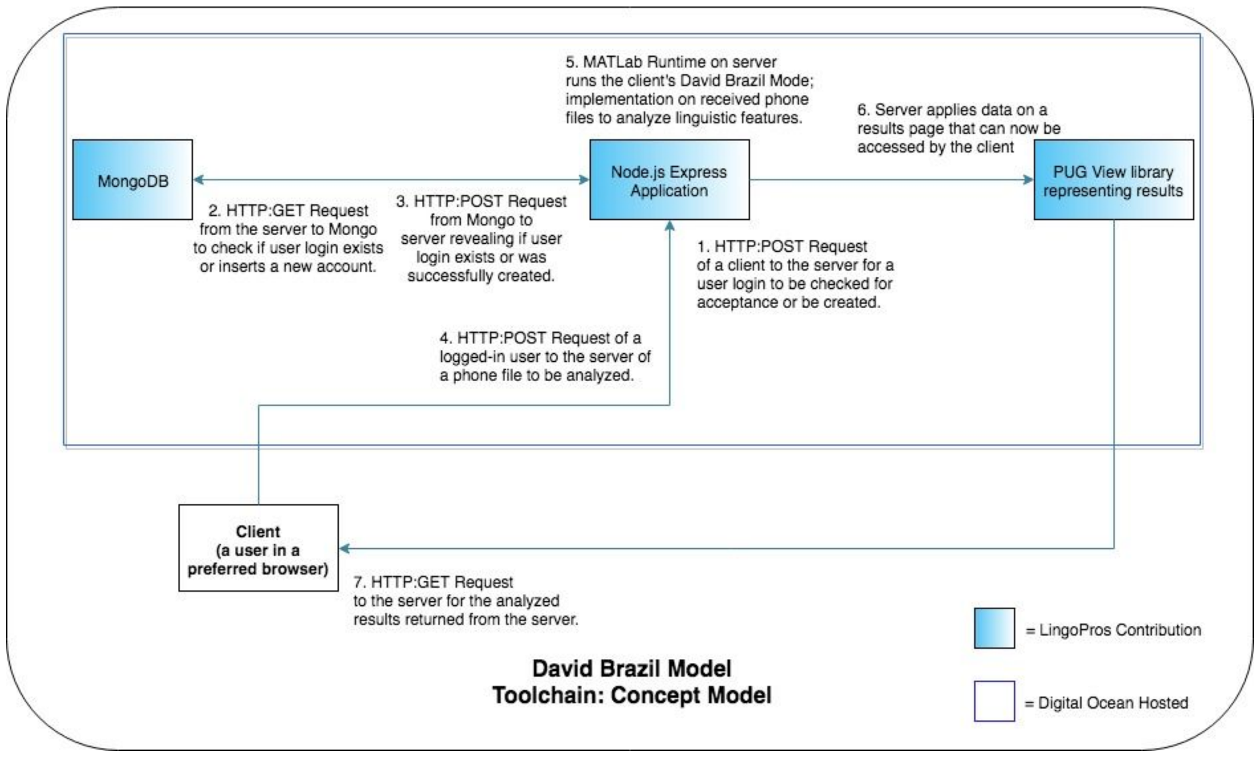
5. MATLab Runtime on server runs the client's David Brazil Mode; implementation on received phone files to analyze linguistic features.
6. Server applies data on a results page that can now be accessed by the client
MongoDB
Node.js Express Application
PUG View library representing results
2. HTTP:GET Request from the server to Mongo to check if user login exists or inserts a new account.
3. HTTP:POST Request from Mongo to server revealing if user login exists or was successfully created.
1. HTTP:POST Request of a client to the server for a user login to be checked for acceptance or be created.
4. HTTP:POST Request of a logged-in user to the server of a phone file to be analyzed.
Client (a user in a preferred browser)
7. HTTP:GET Request to the server for the analyzed results returned from the server.
= LingoPros Contribution
= Digital Ocean Hosted
David Brazil Model Toolchain: Concept Model

**Figure 3.3-** A diagram showing the architecture of the David Brazil Model Web Toolchain.

# Module and Interface Descriptions

In the following section, we will outline all the classes of the project and how they interact with each other. The classes will be accompanied by supporting methods that will be built into the classes. Each section is our project split up based on the major components for our prototype.

## AuToBI Based Analyzer

### Data Analysis

This part of the program will identify which features are chosen the most by Weka as being key features of an analysis. A set of arff (or converted csv) files will be passed through the Weka machine learning suite in which it will perform a feature selection and return the indices of the key features in the arff file. This is challenging because we are not professional linguists so we will need to consult with out client to make sure that the program is selecting the correct features and is working how they intended it to. In Weka there is a method for automatically selecting attributes that seem important, we will be iterating over this process multiple times and analysing the data  and using this to determine which features are deemed most important by Weka. Once we select these features we can choose a certain amount of them to begin training the neural net. After training is completed we can then pass in test arff files that the analyzer hasn't processed before and have it make a guess at th

**Class: autobiRunner**

**Function: RunAuToBIAnalysis(FilePath...)**

Runs AuToBI using the input audio files and configuration and outputs an arff file to be analyzed.

**Function: sortAttrArray(int[]; int[])**

Sorts the array so that the most frequent attributes are first and the less frequent are last.

**Class: wekaRunner**

**Function: useLowLevel(Instances)**

Uses Weka library to perform an attribute selection on a specific data source, aka on a single arff or csv file. It returns the indices of the attributes selected as a list of integers. This will be called from the countFrequency() function in order to as it loops through all of its input files. This function returns a list of indices that correspond to specific attributes in the arff file.

## Function: countFrequency(File[])

Takes in a list of arff files and passes each one to the useLowLevel function that selects important attributes and returns a list of the attributes selected. Once each file is passed through the useLowLevel function, countFrequency counts how many times each attribute was chosen by useLowLevel and returns a sorted, descending order, list of the attributes with the highest frequency. This function will be called during the training session

## Function: buildMeanArray(File, int[])

This function will take in an arff or csv file and an array of integers which represent the a set of attributes at certain indexes. From these specified attributes, the mean value each attribute from the data generated will be extracted and placed into an array of double values to be returned. The array of mean values will be passed into the neural network to determine the proficiency.

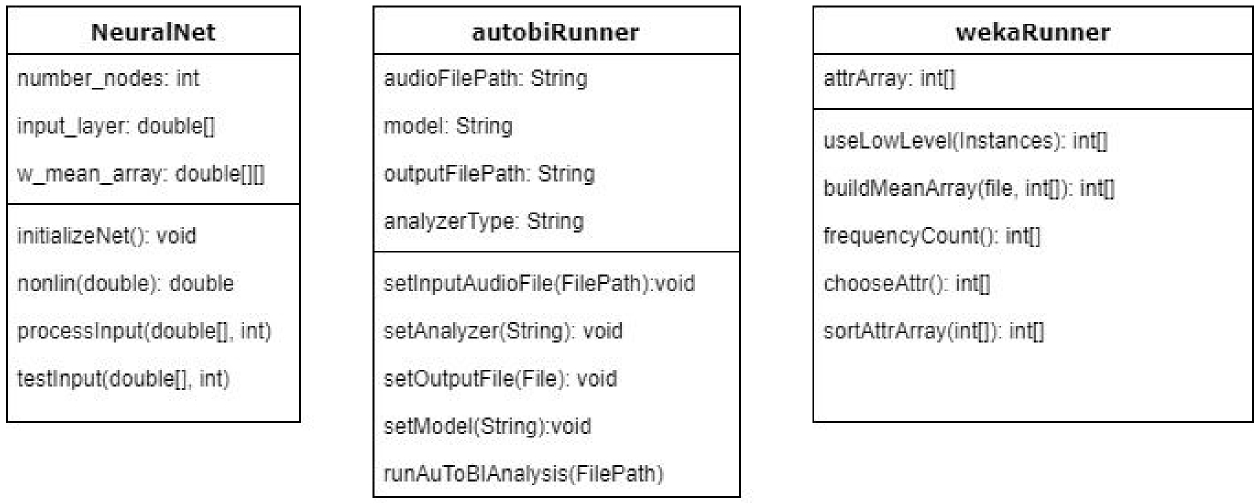

**Figure 4.1**- UML diagram of different classes used in the AuToBI analyzer. AuToBIRunner is ran first and that output is passed into the wekaRunner which selects the attributes from the arff file which then passes the respective attributes and their data into the NeuralNet.

## Training the Neural Net

This part of the program will serve to train the neural network to recognize what measurements of features determine the English proficiency of a speaker. The mean values of several different attributes are passed into the neural network and based on the weights of the graph, it will make a guess at the English proficiency of the speaker. If the guess is wrong, then the weights of the network will be adjusted and a new guess will be made until the network arrives at a correct guess. After several training files with different proficiency levels are passed in, the network should have adjusted itself to correctly determine what values for each attribute accurately determine a speaker's proficiency.

### Class: NeuralNet

### Function: initialize_net()

Creates a new neural network and randomizes the values of the weights of each node. This will allow us to start with an unbiased neural network for training. This function will be called when a neural network object is instantiated.

### Function: nonlin(double)

Normalizes a value between 0 and 1. Will be used to normalize the weights on the neural network after passing values through the net. This function will be called after each pass of data through the network.

### Function: processInput(double[])

Passes the input data through the neural net, and makes a guess at the English proficiency. If the guess is wrong, the function checks the difference in error and adjusts the weights of the net so that on the next guess, it can have a better chance at making the correct guess.

## Testing the Neural Network

This part of the program will be to test our trained net with analysis results that it hasn't seen before. If the net guesses correctly on a high enough percentage of the test data, then we can confidently say that it is prepared for further samples.

### Function: testInput(double[])

Passes the input data through the neural net, and makes a guess at the English proficiency. This function will not make any adjustments to the network.

## Website

The website, or more thoroughly, the web application, is the second half of the Capstone. Here we should make a site where logged in users can upload phone files to our service and have them analyzed in the David Brazil Model way. The users can then see their results, and due to the logged in requirement: Dr. Okim Kang can also control who and how many can use her and Dr. Johnson's copyrighted David Brazil Model implementation. The site will be hosted through DigitalOcean's service because it can run the MATLab interpreter on board and DigitalOcean as a server is very easy to manipulate files using Node.js. Our database will be Mongo.db and we will store it on the mLab.com site, which is a common place to store Mongo databases.

## Database

This part of the web application will store users that Dr. Kang has approved to be able to use the DBM analysis on our system.The plan is that first Dr. Kang will insert access codes that user accounts can be created with. She will have access to a special admin page (that only she has access to) where she can create and manage access codes for the users who want to use the site. Next, a person seeking to make an account will input simple profile information such as a username, password, but unique to our system: an access code. If such an access code exists on our database and a unique username was chosen: an account is created. Upon account creation, the access code is dropped from our table of acceptable access codes to avoid one access code allowing multiple accounts. This database gets checked from the server anytime the server takes in a user login attempt to see if a user is allowed to analyze their phone file with our web application by having an account in this database. Now a user can log in when desired and use our application to input phone files for DBM analysis.

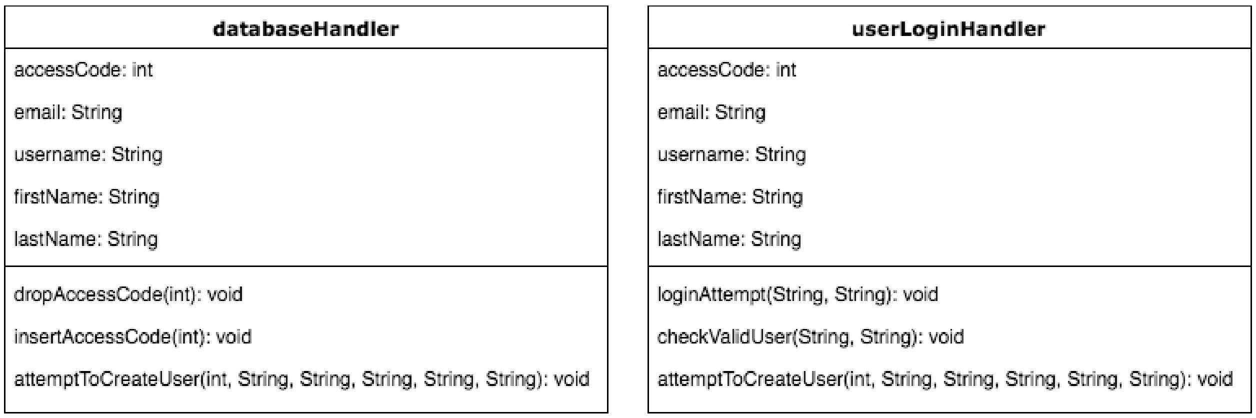
databaseHandler
accessCode: int
email: String
username: String
firstName: String
lastName: String
dropAccessCode(int): void
insertAccessCode(int): void
attemptToCreateUser(int, String, String, String, String, String): void
userLoginHandler
accessCode: int
email: String
username: String
firstName: String
lastName: String
loginAttempt(String, String): void
checkValidUser(String, String): void
attemptToCreateUser(int, String, String, String, String, String): void

**Figure 4.2**- UML Diagrams for the web toolchain using DBM. Database handler functions are ran to allow new users to be potentially made access codes that Dr. Okim chooses. User Login handler functions are then used for user to try to make a new account and try to login in with that account.

The following functions are necessary for our database system. These functions are listed on the main javascript file of our application, namely “app.js”, which has the the input handlers and most of the computation functions (save for the MATLab code) of the entire web tool chain:

**insertAccessCode(int)**

Passes an input number from a super user, Dr. Kang, from the website to our database that will then allow for creation of a new account should someone know the number (which will be at Dr. Kang’s discretion).

**dropAccessCode(int)**

Drops an input integer access code if it exists in our database. This happens automatically on the server immediately after a new user is created. This is stop one user from sharing his or her access code to make multiple accounts when Dr. Kang only allowed one more to be created.

**attemptToCreateUser(int, String, String, String, String, String)**

Attempts to create a user for our application. When the server finds that a valid access code has listed on our database has been input, first and last names are not empty, and that a username is not already in use, the server inserts the new account into our list of created accounts in our database and tells the user that an account has been created for them. The dropAccessCode is called here if the function had inputs that lead to successful account creation. If any of the parameters could not be used to create an account (e.g. Duplicated username, empty first/last name fields, invalid access code), the user is then prompted to try again from this function.

## User Login

The following functions are necessary in the user login module of the web application:

**attemptToCreateUser(int, String, String, String, String, String)**

This function, though previously mentioned, is also associated with this module thus needs to be included in this section as well. This is because when the user login input correct arguments to create a new account, this will be called if the server found no problems with his or her attempt and thus was successful.

**loginAttempt(String, String)**

This function gets called when a user tries to log in with the fields in our login page. It contains proper sanitization for the fields so as to avoid SQL injections or other potentially malicious attacks on our system. We will use the JavaScript library “body-parser” to accomplish this as it is well tested and certainly protects against more kinds of attacks than we can think of just us four alone. After sanitization, checkValidUser is called with the same input, but it is now sanitized.

**checkValidUser(String, String)**

The server here checks to see if user logged in with valid input. Valid input would mean that the input email and password were findable in our database and associated with one profile. If found to be valid input, the user is alerted that his or her login was successful. If found to be invalid input, the user is prompted to try again or ask Dr. Kang for an access code to create a new account. Upon successful login, the user may now input phone files to be analyzed by the hosted DBM in our application.

## Server

This part of the program is truly the workhorse of our application. Here the server has the job of interacting between the user and database, taking in phone files from logged in users, running the DBM analysis, and formatting results in a readable way to the user.

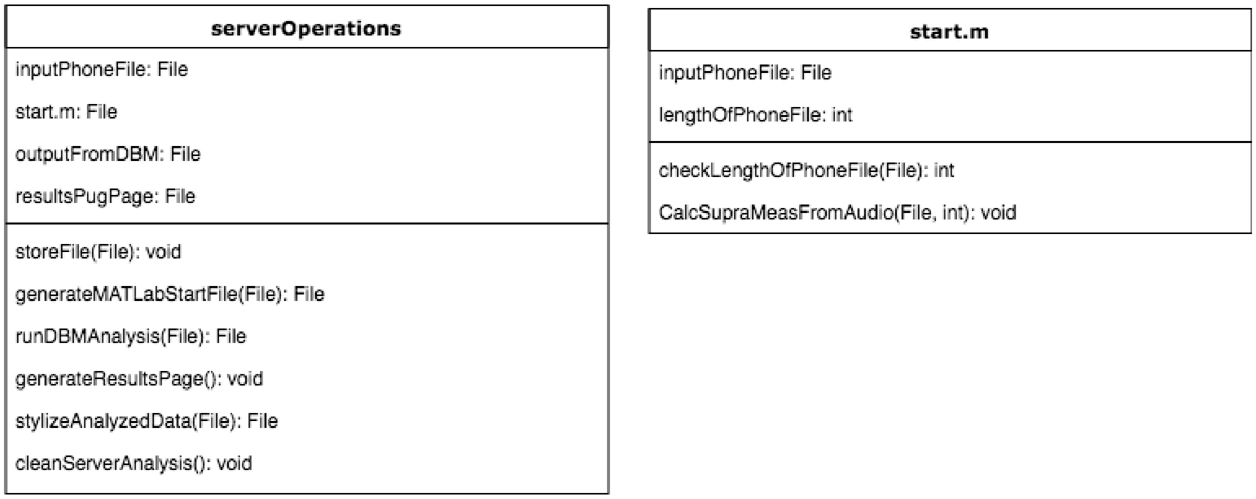
serverOperations
inputPhoneFile: File
start.m: File
outputFromDBM: File
resultsPugPage: File
storeFile(File): void
generateMATLabStartFile(File): File
runDBMAnalysis(File): File
generateResultsPage(): void
stylizeAnalyzedData(File): File
cleanServerAnalysis(): void
start.m
inputPhoneFile: File
lengthOfPhoneFile: int
checkLengthOfPhoneFile(File): int
CalcSupraMeasFromAudio(File, int): void

**Figure 4.3**- UML diagrams of the server and model of our web toolchain for the David Brazil Model. Server functions store a user's input phone file on Digital Ocean's provided local storage. Upon successful file upload of a true phone file, the server then calls the file in the generateMATLabStartFile function to make a second file named "start.m" containing proper MATLab run commands specifically for the user's individual analysis, also overriding any previously made "start.m" file. This is what the MATLab interpreter will use to run and make a server side analysis from based on the David Brazil Model MATLab code Dr. Johnson gave the team. That new start.m file is then passed to runDBMAnalysis which simply calls the MATLab interpreter on that file and produces a third file of analyzed results in the .csv format. Next, a call to generateResultsPage will trigger stylizeAnalyzedData to create a resultsPugPage.pug file that is the user's results page.

The following functions are unique to the server module:

### storeFile(file inputPhoneFile)

Here the server takes in an alleged phone file from a logged in user, checks if the file has a valid phone file extension, and if so stores this in the DigitalOcean's local storage. This storage is fine as the phone files do not have to be hidden to outsiders should DigitalOcean ever get an attack that leaks server documents.

### generateMATLabStartFile(file inputPhoneFile)

The server looks up the input file in the local storage to make sure it was added, and based on that file name the server makes a "start.m" file in the server's local storage that is a list of commands MATLab understands and those commands conform to the function call of our given DBM implementation.

Looking inside "start.m":

1. **cd ../../../PhoneFiles ;**

   Here we have a terminal like command to the effect of setting the MATLab environment to point to where we put the user input phone file.
2. **lengthOfPhoneFile = checkLengthOfPhoneFile(file inputPhoneFile);**

   This function in the David Brazil Model MATLab implementation will see a file in the server's local storage and determine its length or size in this case. We need this number of to start the entirety of the analysis in the successive function, and thus after getLengthOfPhoneFile is called we want to store the number it returns in lengthOfPhoneFile.
3. **CalcSupraMeasFromAudio(file inputPhoneFile, integer lengthOfPhoneFile);**

   This is a MATLab function that we will call using our start.m file that actually kicks off the DBM analysis in MATLab. The input fields here have already been generated in the start.m file command list that came from generateMATLabStartFile(file inputPhoneFile). Upon the function's completion, the MATLab instance terminates. After this termination, our server's local storage will have a new output file from the MATLab run. Now the server has an output stream that will be stored in a unique file on our server.

Back outside of "start.m":

### runDBMAnalysis(file start.m)

This is where the server starts a MATLab instance and uses the newly written start.m file to run the DBM with. We needed to make a start.m file first because headless

MATLab in the server can only be ran one argument at a time, but MATLab commands can be stacked in a start.m file that the MATLab runtime environment can run as a single argument.  This function will output a stream we will save as a file called outputFromDBM to be used in stylizeAnalyzedData.

### generateResultsPage()

Here is just a simple call to stylizeAnalyzedData(file outputFromDBM), but for purposes of good code modularization, we nest it in here to keep to convention of "generate" + some page (or file) consistent.

### stylizeAnalyzedData(file outputFromDBM)

We input outputFromDBM to this function so that we can organize and present the analyzed results in a clear way with JavaScript because of the view library PUG. After following PUG syntax, we'll have a webpage to show our user that displays the analyzed data in a concise and organized way.  This function outputs the "resultsPugPage.pug" file that the user will be presented with client side and can see the results of his or her analysis.

### cleanServerAnalysis()

This function call happens after a user logs in successfully.  The server deletes from its local storage any previously made "start.m" file, input phone file, and output analyzed file.  This makes room in local storage for our server for any more input, and allows for a new user user to analyze the same file again, or analyze a file that happens to be named the same as a previous entry.

## Implementation Plan

In this section we will describe our course of action over the next four months in order to implement our project and release a final product. Considering Figure 5.1, the development of this project is split up into three main components that will be necessary to have completed in order for us to deliver the prototype. Each of the team members will be working on a portion of these components as Figure 5.2 depicts. The plan is to work on these components that also consist of smaller subsections as described in the module and interface descriptions above, to create this functional prototype. Once this is complete we will then extensively test and upgrade the prototype until our clients and we are satisfied with the final product.

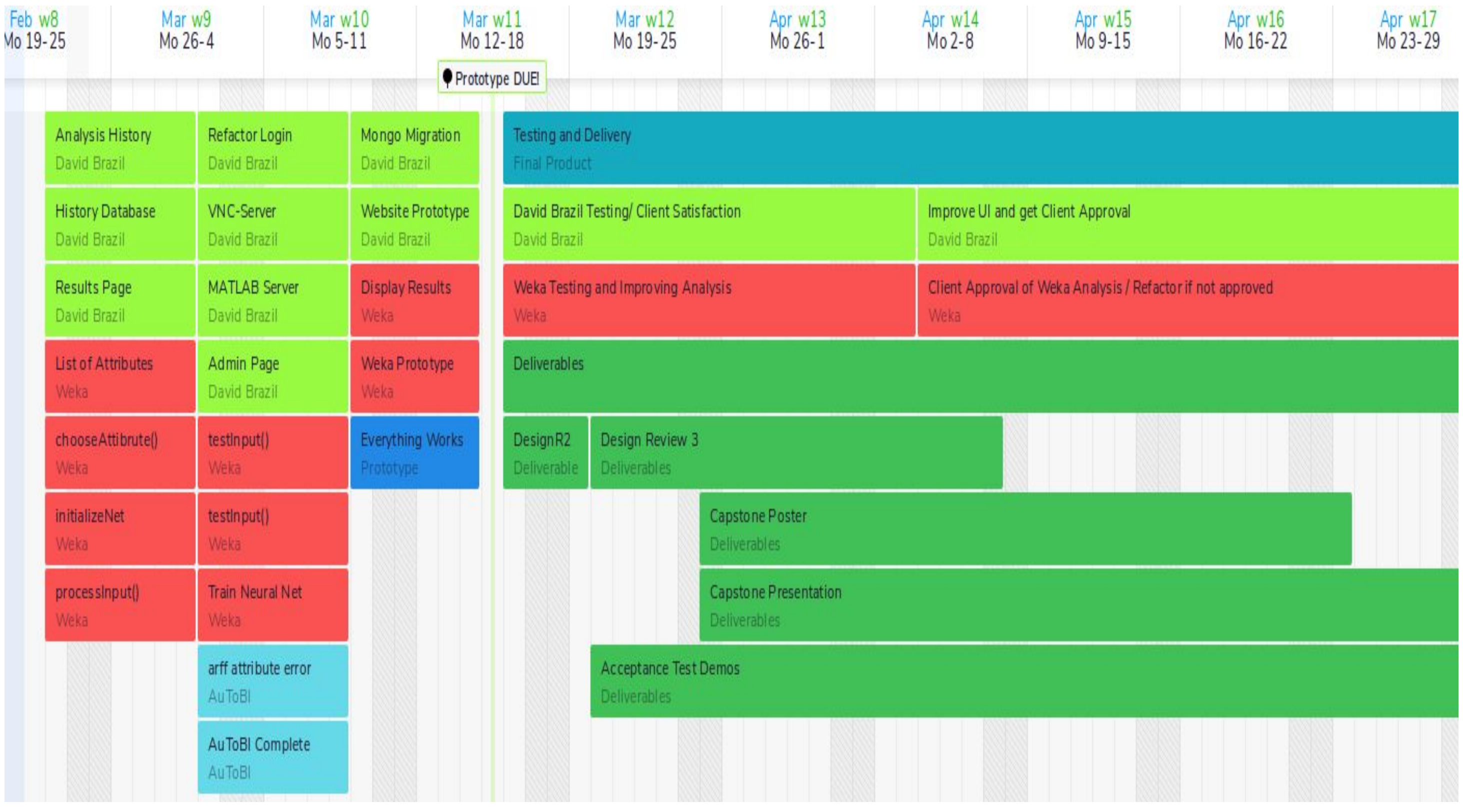

**Figure 5.1-** A Gantt chart showing when we will be completing work on the implementation of each module and their respective methods. Figure 5.2 displays a more general overview of the sections and the work delegated to each member.

| Work Assignments | | | | |
|---|---|---|---|---|
| **Project Details** | **Matt Quintana** | **Josh Shaffer** | **Luis Montes** | **Erik Strauss** |
| **AuToBI Weka Analysis** | ✔ | ✔ | | |
| **David Brazil Server** | | | ✔ | |
| **Web Portal** | | | ✔ | ✔ |
| **Team Website** | ✔ | ✔ | ✔ | ✔ |
| **Testing** | ✔ | ✔ | ✔ | ✔ |
| **Quality Assurance** | ✔ | ✔ | ✔ | ✔ |

**Figure 5.2-** A table describing how work will be delegated among team members.

# Conclusion

Automatic speech recognition is a topic that has been around for several decades. However, only recently the field has advanced to the point where researchers are able to study the deeper subtleties of speech due to advancements in computational capability and machine learning. Linguistics researchers are now taking advantage of these new, better performing tools to carry out their analysis so that they can gain a better understanding of how humans communicate. Dr. Okim Kang and Dr. David O. Johnson are such researchers who are taking advantage of these new technologies and using them to develop a new framework for automatic speech analysis.

Dr. Kang has asked our team to help develop a speech analyzer based on the standard speech analysis framework ToBI so that she may compare the results against her Brazil framework. In addition we have been asked to build a web application that can provide our client and other users a website to analyze speech sample using this Brazil framework that our clients created. The current system they use is cumbersome and time wasting because they have the framework spread across different computers with specific operating system installations. Our solution will help Dr. Kang and her students at the ALSL conduct better research so that the field of speech analysis as a whole can gain a better understanding of how humans communicate. Although we might not be LingoPros we believe our product will be.