# Process Description

*orator.io* by Team Gucci


Sung Kim (sunghyun) Yina Zhu (yinazhu)
William Bhot (willbhot) Hao Hu (hhu94)
Logan Girvin (lsgirvin) Joey Li (joeyli96)


## Software Toolset

Our project is a web-based service that provides feedback and training for public-speaking. The front-end will be a combination of HTML, CSS, and Javascript. The Javascript can be built on frameworks such as React or Angular. The back end will be based on Python 2.7, version-controlled using Github. Our team chose Python as the main programming language as everyone was familiar with it and since most of our features are based off of machine learning or analysing data, functional programming is currently more useful to us than object-oriented programming.

There are many APIs that we are planning on using "off the shelf" as well as many features that we plan on developing from scratch.

**Existing Services**

**Google**

-Cloud Speech API: This API can transcribe word audio to text. We originally planned on using this API to create a transcript of the user's speeches to analyze and give feedback; however, we felt that IBM Watson was better suited to our needs--explained below. Google also provides speech to text for other languages if we want to expand our service in the future.
-Google Account: Allows user to link their Google account with service. We have currently implemented Google login system into our site. We choose Google account over other web-based login services as most people own a Google account.

**Microsoft**

-Emotion API: This API can analyze faces to detect a range of feelings. We plan on using this API as a possible stretch goal for users to receive feedback on their speech in combination with their actions.
-Bing Speech API: This API can transcribe word audio to text as well as understand intent for commands. We may use this if the Cloud Speech API does not prove to be consistent or to do a side-by-side comparison of the transcripts to get a more accurate transcription from a speech.

**MySQL**

-MySQL Database: An SQL-based database to store information. We can use this to store past speeches and feedback for a user.

**Amazon**

**-**DynamoDB Database: A NoSQL cloud-based database to store information. We can possibly use this as an alternative to SQL as the lookups are faster and no need to pay for long-term storage.

**Development**

We are planning on developing several key features in Python for other analysis purposes. All of the APIs that are listed above are compatible with Python's libraries. We are looking into using Django, a framework for Python, to utilize in our service.

# Group Dynamics

Joey Li is our project manager. Everyone is developing and writing tests for their own software. Once the work is finished, we submit our code for review and other members of the team check to make sure everything seems correct. We have people writing tests for their own software since the person who best understands how their software works and the possible cases for it should be the person who wrote it. The review is there a safety net and not as a dependency for catching errors. If disagreements arise, we are escalating to the project manager who will make the decision on what they think will be ultimately better for the team. We chose this structure as looking for a majority vote on decision will slow development process as well as create friction between groups who voted differently.

Currently, we are designating people to work on the front-end and getting the speech audio from the user to the database and other people to work on the back-end analysis. The back-end is also split into two groups, machine learning and data analysis. The data analysis team will be working on features such as pace control, synonym recognition, and hesitation reduction. The machine learning team will be working on the features such as recognizing speech patterns, frequency, and volume to evaluate speech quality.

# Schedule / Timeline

**Web Development Front-End (3-4 weeks):**
-Web base structure (3-5 days)
-Audio recording / upload (2-3 days)
-User login service (2-3 days)
-User recording retrieval (1 week)
-Webcam mirror (2-3 days)
We predict that the hardest part of this section will be integrating the service with the database since it will require different API calls to upload and retrieve the audio.

**Python Back-end (Analysis) (3-4 weeks):**
-Pace control (1 week)
-Synonym recognition (1 week)
-Word Count (3-5 days)
-Hesitation Reduction (1 week)
We predict that each analysis method will take around a week to implement along with the unit tests and implementation tests.

**Python Back-end (Machine Learning) (2-3 weeks):**

-Find databases of past speeches (2-3 days)
-Machine learning algorithm (1-2 weeks)
We plan to get most of the basic features done by the beta release and then work on optimizing and giving more analysis of the speeches for the final release.

# Risk Summary

**Speech to Text:**
Currently, many of our features depend on a reliable method for transcribing audio to speech. Some services like IBM Watson that we have tested with are not as accurate and less reliable. We are mitigating this risk a few different ways. For one, we are using the Cloud Speech API by Google which has a well-known reputation for being robust. We are also looking into comparing transcripts produced by the Bing Speech API from Microsoft to find differences in transcripts produced and find the more likely one. We are also looking the possibility for people to upload their transcripts to get an exact representation of what was said vs intended.

We are also looking into machine learning which would not require a transcript at all but only focuses on the dictation, frequency, and pace of the speech and see if it has the properties of a well-made speech.

**Unhelpful feedback:**
Many qualities of public speaking are subjective. We may not necessarily give the correct feedback. We look to mitigate this risk by testing the service ourselves with real speeches. We are also going to be looser on the specifications so that decent speeches do not get unnecessary criticisms and possibly discourage the user.

**Machine Learning**
Machine learning is often a hit-or-miss. We are checking on the feasibility of the project by looking into audio to feature vector APIs as well as databases of speeches produced by famous public speakers (e.g. presidents). If the project is falling behind, this is one of the first features we will cut as even if it does end up implemented correctly, we do not know how well it will work until we test on real users.

---

## Design changes and rationale

The team decided to move over to Python 2.7 from 3.5. We have done this for a few reasons. One, it is easier to install and compatible with more machines. Two, it is the default python version that comes with the UW CSE lab computers and virtual machines. We also found it was more compatible with the libraries and web servers we were using. By switching to 2.7, we are giving up the “latest” version of Python which means we might miss future implementations as a tradeoff.

The process description has also been updated to include the services our team ended up using. We chose IBM Watson to convert speech-to-text as it had some of the features we wanted such

as word beginning and end times and tone analysis. For databases, we are using SQL as it is supported directly through the Django framework.