Video Accessibility for the Virginia Tech Event Capture Group

Prepared By:
Cassidy Heath
Tucker Legard
Edna Morales
Bradley Retterer
Michael Supanich

For:
Keith Gilbertson
Dr. Edward Fox

CS 4624 - Multimedia, Hypertext, Information Access
Virginia Tech
Blacksburg, VA
May 2014

Table of Contents

- Table of Contents
- 1. Executive Summary
- 2. Users' Manual
  - Adobe Premiere
  - Web Application
- 3. Developers' Manual
  - CMAP
  - Requisites Skills and Resources
  - Initial Setup
    - MongoDB Database
      - Downloading
      - Configuration
    - Python Watch Script
      - Installation of Packages
      - Download NTLK Stopwords
      - Configuration
    - Node Front End
      - Install Dependencies
      - Running the Server
  - Project Inventory
    - Root Files
    - App Folder
    - Index File
    - CSS
    - Scripts
  - Overview of Code Structure
    - End Points
    - Views and Layouts
    - Event Driven Programming
- 4. Manual Transcription Findings
- 5. Lessons Learned
  - Timeline Schedule
  - Problems and Solutions
- 6. Acknowledgements
- 7. References

1. Executive Summary

We worked with Keith Gilbertson to come up with a way to make the videos on VTechWorks more accessible. Not accessible in the sense of where the videos are stored but accessible for searching for the videos and assistance while watching them. Initially, Keith wanted us to focus on both the ability to make the video full-text searchable and to add captions to assist with viewing. They wanted to make the videos easy to find and easy to watch if the video had tough accents or the viewer had a disability. He also suggested used CMU's Sphinx software; software that was developed to automatically transcribe videos to a text document. We also worked with Therese Walters of the VT Event Capture Group and Paul Mathers, a UNIX System Administrator with the library. Both of whom provided us valuable information and suggestions to move forward and complete this project.

After doing research on the currently available software for manual and automatic transcription and for captioning, we determined that captioning was far beyond the scope of this class project. Many of the automatic transcription tools we found were extremely poor at correctly identifying the words in a video and anything used for captioning was even worse. We even tested YouTube's captioning and the accuracy was quite low. In the end we decided that working on a process for manual transcription and using Adobe Premiere for automatic transcription; captioning would have to wait for the technology to catch up.

For manual transcription we used a standard word document and wrote each word as we heard it. This process proved to take far longer than we anticipated for each video that was transcribed. The automatic transcription was run through Adobe Premiere and proved to be much more time consuming that we thought it would be as well. Typically, it seems to take just as much time to manually transcribe a video as it does to automatically transcribe a video. There appears to be a large separation between what a computer can process in the english language and what the human brain is able to interpret. There are also many other factors that go into transcribing a video, manually or automatically, which are discussed in further detail below.

In the end, there is no easy way to transcribe video, at least that is readily available. However, we believe the processes we researched can still be of great benefit to the library and the VTechWorks groups. We can further shift the focus of the project to focusing more on just the keywords than on transcribing the text word for word. This allows videos to be fully searchable and easily located on the VTechWorks site and can prevent matching the filler words; such as, the, and, it, and so on.

We want to thank Dr. Fox, Therese, Paul, and especially Keith for the guidance, support, and cookies given during the project. Below we explain in detail how to use and setup Adobe Premiere, the results of manual transcription research, and what we learned from the project.

2. Users' Manual

To ease the process of transcribing a video, we created a web application that works with the Adobe Premiere speech analysis. Essentially, the user runs the video through Adobe Premiere, a transcript is created, and the transcript is then imported into the web application.

Adobe Premiere

Before editing the transcription of a video, the video must first be processed through Adobe Premiere's speech analysis to create a rough version. The first step is to open Adobe Premiere and create a new project. Once a new project is created, the video must be imported. Simply go to File -> Import, as shown in Figure 2.1 below.

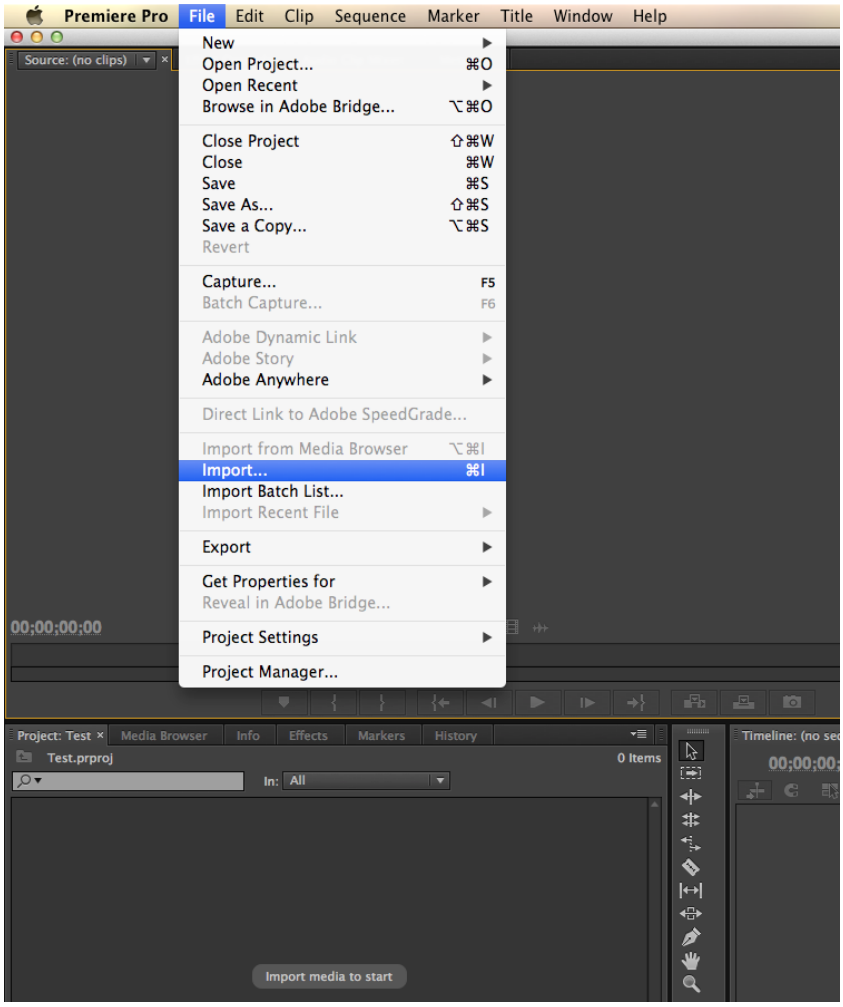


Figure 2.1: Importing a video in Adobe Premiere

Next, right click on the video and select Analyze Content, as shown below in **Figure 2.2**.

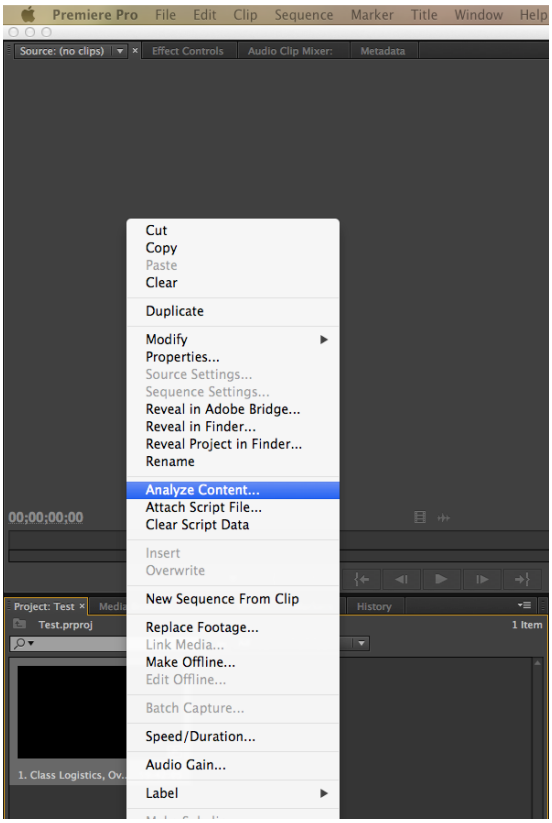


Figure 2.2: Analyzing the content of a video in Adobe Premiere

Ensure that Identify Speakers is selected in the Analyze Content window, as shown in Figure 2.3 below.

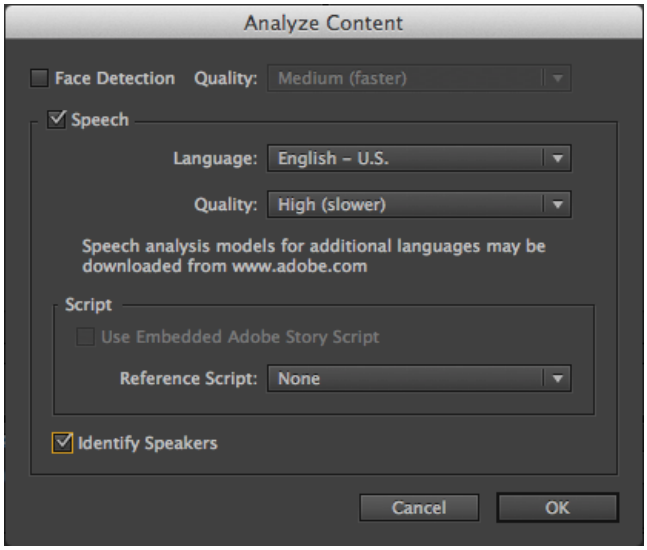


Figure 2.3: Settings for Analyze Content in Premiere Pro)

Afterwards, Adobe Media Encoder will open and begin to analyze the speech. Figure 2.4 below shows Adobe Media Encoder processing the speech and showing a status.

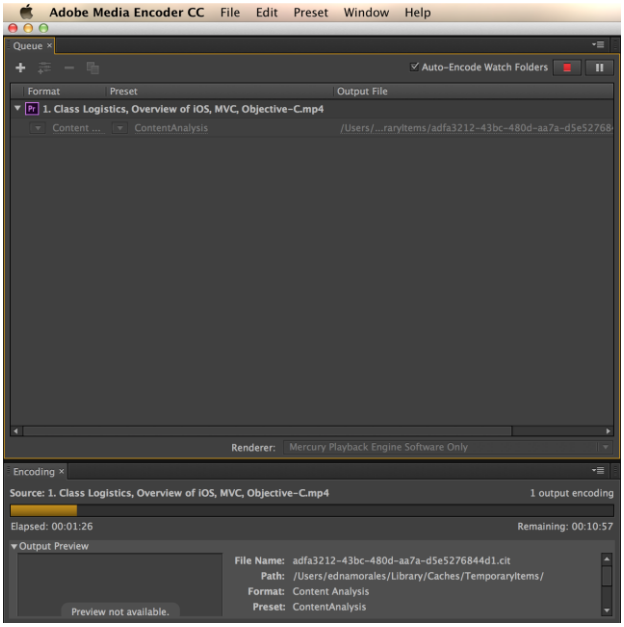


Figure 2.4: Adobe Media Encoder processing speech of video

The transcription file will get created after the speech analysis is completed. This file will automatically get picked up by a script and be imported into the the web app library. If you want to look at the transcription in Adobe Premiere, simply go to Window and select Metadata, as shown in the Figure 2.5 below.

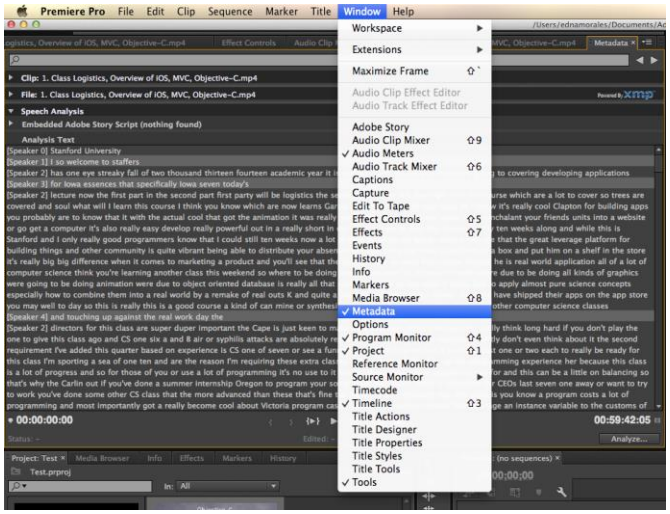


Figure 2.5: Displaying transcription in Adobe Premiere

Web Application

Once the video is processed through Adobe Premiere, a script will import the transcription of that video into the web application. Figure 2.6 shows what the user sees when first opening the web app.

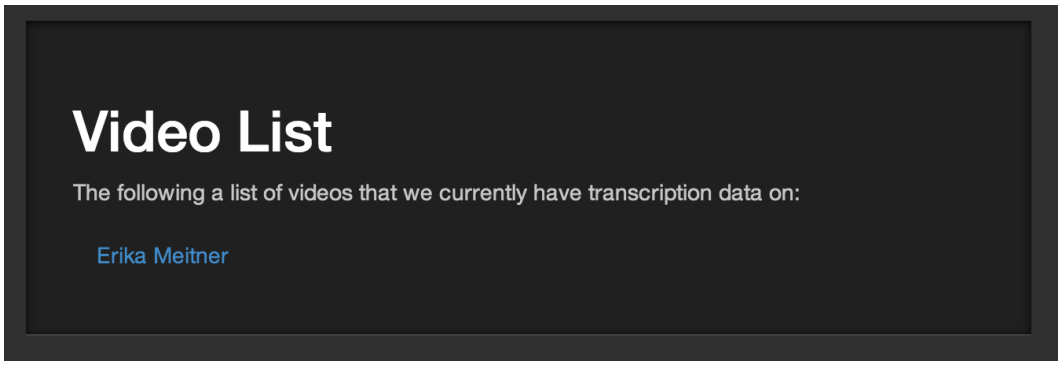


Figure 2.6: List of transcribed videos displayed in the web app

A list of the transcribed videos is displayed. From there, the user can select a video to work on. Figure 2.7 shows the layout of the web app for editing a video.

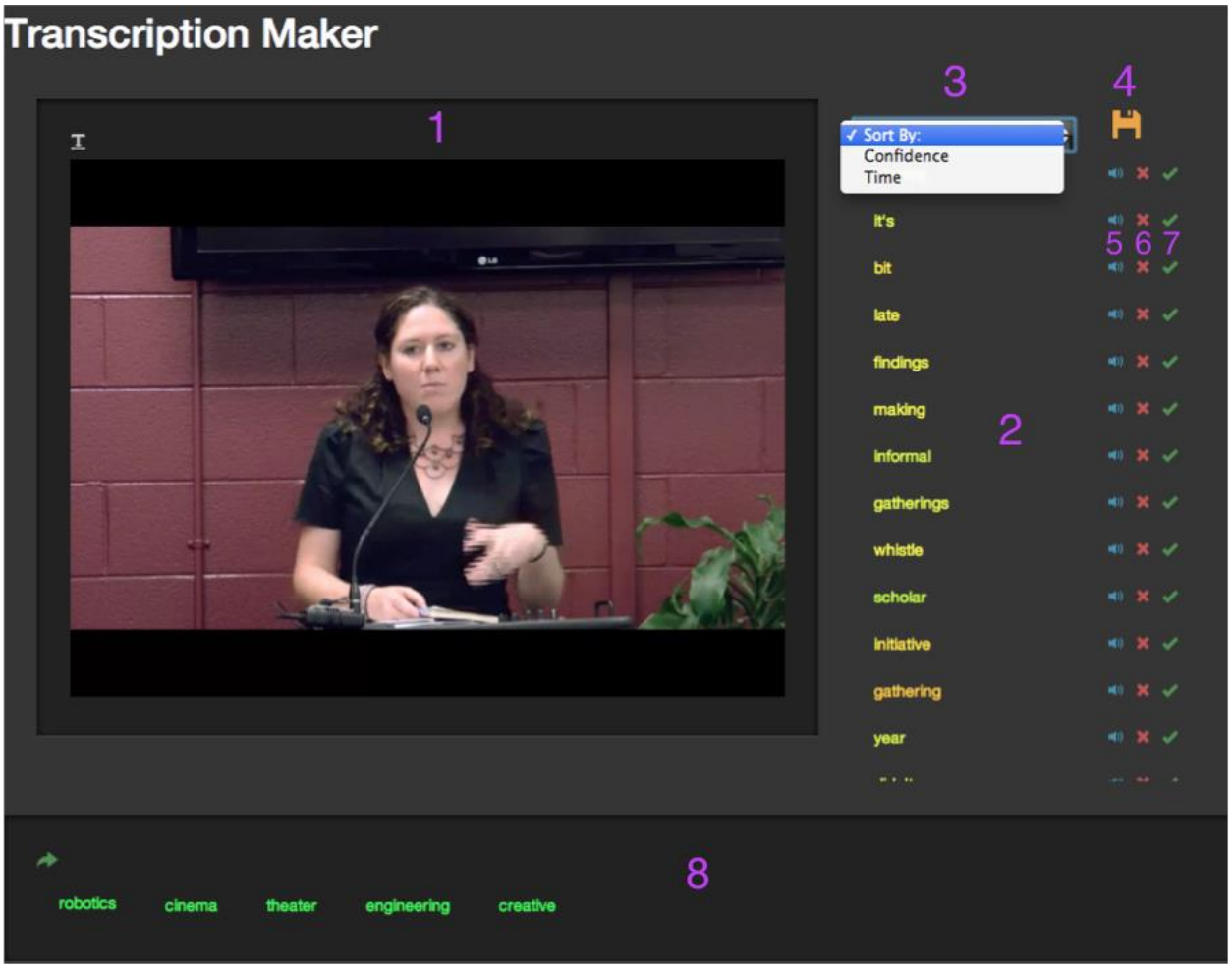


Figure 2.7: Layout of editing a video transcription in the web app

I will describe the functionality of the web app by sections.

Section 1
This section in Figure 2.7 displays the actual video.

Section 2
This section displays all of the words in the transcription, along with ways to manipulate the words and the video. That functionality will be discussed in more detail in the following sections. If you notice, you can see that the words are different colors. The colors represent the confidence level for that word. Adobe Premiere's speech analysis provides a confidence level for each word. The more green a word is, the more confident Adobe is that it transcribed that word correctly. The more red a word is, the less confident Adobe is that it transcribed that word correctly.

Section 3
This section is a menu that allows the user to sort the words based on either time or confidence level. If the user sorts by time, the words will appear in the order that they are in the video. If the user sorts by confidence level, the words will appear in the order of least confident to most confident.

Section 4
This section displays a Save button. When the user is finished making changes to the transcription, whether it is completely finished or just finished for that work session, he or she can click the save button to update the changes in the database. If the user wants to edit the transcription of a video in multiple sessions, then he or she can simply save at the end of each session and pick up where they left off for the next one.

Section 5
This section displays an audio button for a particular word. When the user clicks the audio button, the snippet that corresponds to that word will be played in the video. That way the user can verify that the transcribed word is correct. If it is correct, nothing needs to be done. If it is not correct, the user can simply double click the word and edit it, as seen in Figure 2.8 below. Once the user is finished editing a word, he or she can press enter.

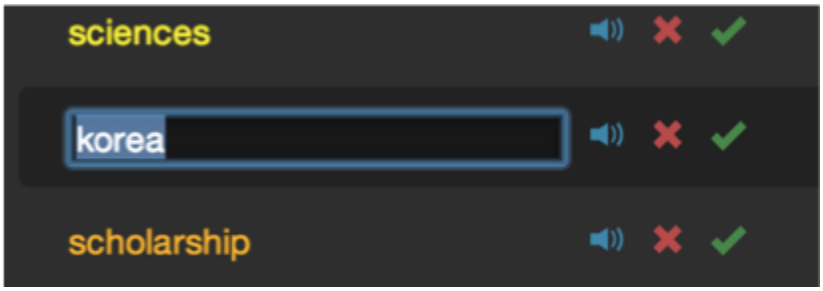


Figure 2.8: Editing a word in the web app

Section 6

This section displays a button to delete a word. Perhaps that snippet of the video is just background noise or a filler word. In that case the user can simply delete the word altogether.

Section 7

This section displays a button to add a word to the list of keywords, which is displayed in section 8. The purpose of this functionality will be discussed more in-depth in the next section.

Section 8

This section displays a collection of keywords that are added by the user. The purpose of this functionality is to use the keywords as tags for the specified video in the VTechWorks video repository. This will essentially make the video more searchable. Once the user is finished selecting and adding keywords, he or she can view the raw text by clicking the green arrow in Section 8. From there, the keywords can be copied and pasted wherever the tags for each video in VTechWorks are stored. Figure 2.9 below shows the raw text of the keywords. To go back to the regular view of the keywords, the user can simply click the green back arrow.

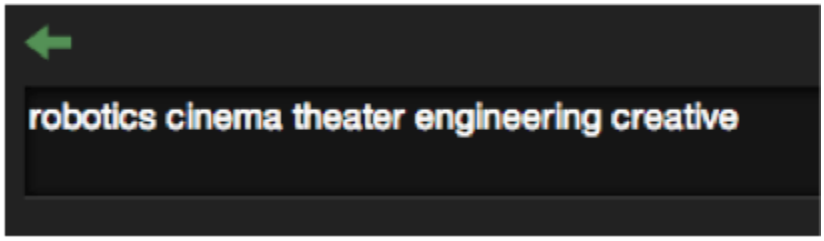


Figure 2.9: Raw text of the keywords

3. Developers' Manual

CMAP

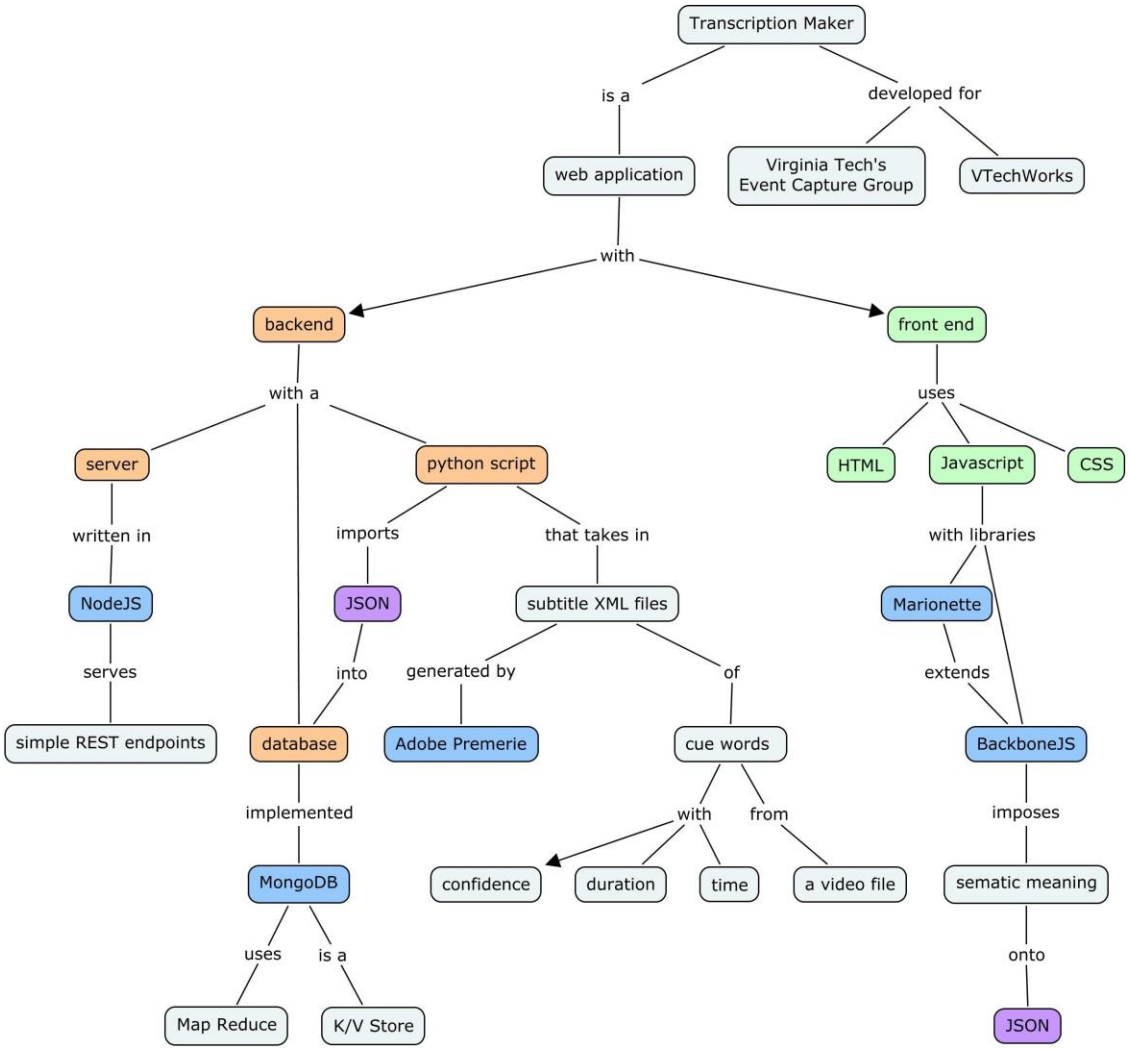


Figure 3.1: CMAP giving a project overview

Requisites Skills and Resources

This project is a web application. In order to be a competent developer on this project you should be somewhat proficient in the following languages and frameworks:

- Python
- Javascript
  - Backbone documentation
  - Marionette documentation
  - Node documentation download
  - ExpressJS documentation

- JSON
- MongoDB documentation download
- HTML / CSS

Furthermore you should have a cursory knowledge of

- Adobe Premiere CS6
- XML
- Database Operations
- The Model View Controller Paradigm
- REST
- JQuery

Python is largely used for the conversion of XML files that Adobe Premiere spits out and the eventual import into the MongoDB database. Javascript is the language of choice for the project luckily it's widely used and easy to pick up so the support is outstanding.

MongoDB is a lightweight Key Value store that is used to store the various cues generated by Adobe Premiere. It interacts with the ExpressJS server, which serves up simple endpoints displaying the data from Adobe Premiere in JSON.

The Frontend is largely coded using Marionette. Marionette is a Model View Controller framework written in Javascript that provides semantic meaning to JSON models, and allows quick and easy DOM interaction.

Initial Setup

MongoDB Database

Downloading

MongoDB can be easily set up by navigating to their website. Simply download their package and install as desired. From there it's simply a matter of using the executables in their bin directory.

Configuration

In order to run the database you'll will need to run the `mongod` executable and specify a particular database path for it to store its files. A sample setup is displayed below.

```
mongod --dbpath /path/to/database
```

This will need to be kept consistent and not change, throughout the during of the app.

You can confirm the database is up and running by navigating to http://localhost:27017/ on your machine or trying to use the `mongo` process.

Importing the Test Data

Our program will watch for the changes on the video collection. We've provided the test data we used in this program.

It can easily be imported after ensuring the `mongod` process is running by running the following command from the root of the project directory.

```
mongoimport --db library --collection videos ./testData.json
```

Python Watch Script

Installation of Packages

The python watch file has a few packages that need to be installed before work can begin. This can be done using the popular package installer, pip.

On an UNIX machine the commands are as follows:

```
pip install watchdog            // used for file watching
pip install bs4                 // used for XML parsing
pip install ntlk                // used for stopword filtering
pip install pymongo             // used for insertion into mongodb
```

Download NTLK Stopwords

Follow the instructions here to download the NTLK corpus of stop words.

Configuration

The python script `watchForNew.py` will need to be edited to suite the configuration of the filesystem the server is running on. As such there are two variables at the top of the script that need to be changed.

```
PREMERIE_PATH: "" // this is the path to the directory in which the log file
                     for Adobe Premiere's Media Encoder is kept
XML_PATH: "" // this is the path you which for the XML files to be moved to and
                renamed.
```

To find the premiere log's file location open Adobe Media Encoder and click on a completed action's status

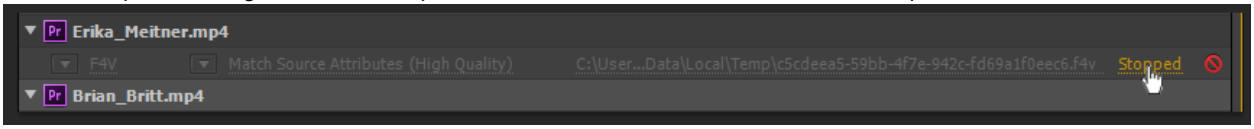


Figure 3.2: Opening the log file in Adobe Media Encoder

This will open the file in a text editor

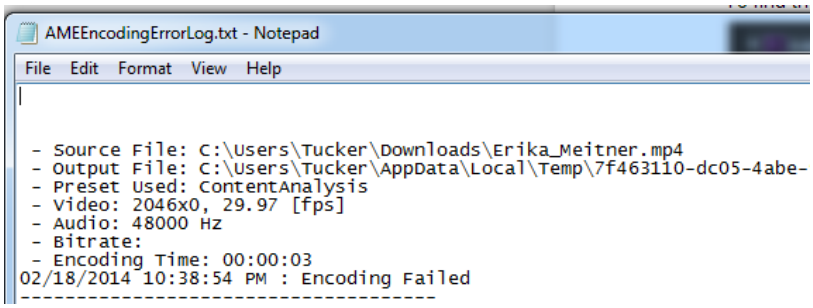
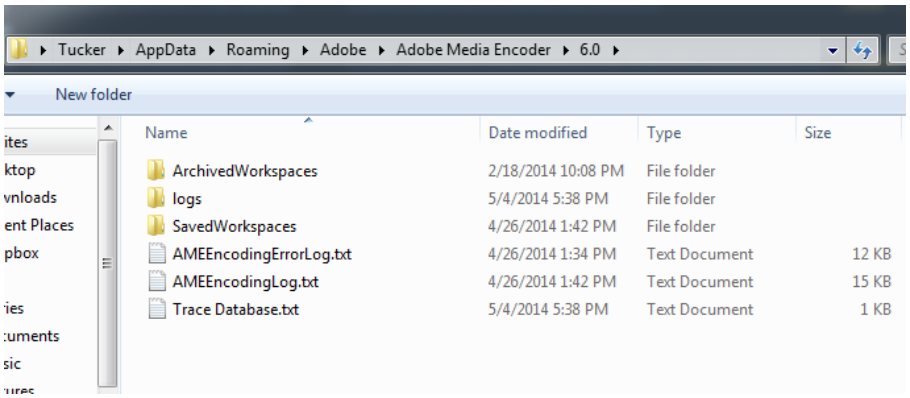


Figure 3.3: Log file as show in Notepad



Simply go to save-as to find the path

Figure 3.4: Directory where the Adobe Media Encoder log file is saved

This is the path that PREMERIE_PATH needs to be set to.

Node Front End

Install Dependencies

Assuming node is installed it should have also come with the package manager, npm; however, if it is not installed you may want to use it.

All of the NPM dependecies are stored in the package.json file and simply running

```
npm install
```

Will resolve all the dependencies and install the necessary packages.

Running the Server

Assuming you already have the the mongod process running. The server can be ran using

```
node server.js
```

After which a server will be running at localhost:4000.

Project Inventory

See **Figure 3.7** for the complete project inventory tree.

Root Files

`watchForNew.py:` The python script responsible for watching Adobe Premerie's log file and importing all subtitle files generated into the database.

`server.js:` A simple node server responsible for serving the applications end points and file systems.

`package.json:` A list of dependencies required by the NodeJS server, use `npm install` to install them.

`README.md:` A readme file containing a cursory version of what was explained in the Initial Set Up Section

`testData.json:` The test data we used for the database, can be imported following the instructions in Importing the Test Data.

App Folder

The server will be set to serve all files in the app folder. So for instance

```
/app/images/kitten.jpg
```

Is accessible at

```
localhost:4000/images/kitten.jpg
```

Index File

The index file bares special mention because it contains all the templates for the various views. If you examine a typical template.

```
<script id="cue-edit" type="text/html">
  <a class="active">
   <input class="cue-edit" type="text" value="<%= name %>" />
   <span class="listen glyphicon glyphicon-volume-up"></span>
   <span class="remove glyphicon glyphicon-remove"></span>
   <span class="ok glyphicon glyphicon-ok"></span> 
  </a>
```

Figure 3.5: Typical HTML Template

It's clearly labeled with an ID which is used by the Marionette view as a reference. Inside the template is the HTML the defines the template.

`<%= name %>` allows you to echo the name attribute in the model the template is for.

For more information on Underscore Templating click here.

CSS

CSS is unnecessary in the grand scheme of the program, but it makes things look pretty. We are using Bootstrap to flush out our CSS, theming it with DarkStrap, and then writing our own styles as we see fit. These files can be found the the `styles/` directory. Particularly, the file we are concerned about is `styles/screen.css`

Scripts

Every file the script folder follows a particular naming scheme and file structure. First of all, they are clearly javascript files. Furthermore if you are familiar with the MVC design pattern you will see that there are set folder for set data types. Views go in the `views` folder, collections in the `collections`. We've tried to keep it as organized as possible.

```
.
|   package.json
|   README.md
|   server.js
|   watchForNew.py
|   testData.json
|
+---app
|   |   demo.html
|   |   favicon.ico
|   |   index.html
|   |
|   +---images
|   |       kitten.jpeg
|   |
|   +---scripts
|   |   |   app.js
|   |   |   config.js
|   |   |   main.js
|   |   |   require.js
|   |   |
|   |   +---collections
|   |   |       videos.js
|   |   |
|   |   +---controllers
|   |   |       index.js
|   |   |
|   |   +---layouts
|   |   |       main.js
|   |   |       transcribing.js
|   |   |
|   |   +---lib
|   |   |   |   ….                          // library files omitted
|   |   |   |
|   |   +---routers
|   |   |       index.js
|   |   |
|   |   +---templates
|   |   |   \---compiled
|   |   |           main.js
|   |   |
|   |   \---views
|   |           bank.js
|   |           cue.js
|   |           cues.js
|   |           htmlvideo.js
|   |           video.js
|   |           videos.js
|   |
|   \---styles
|       \---css
|           |   screen.css
|           |
|           +---fonts
|           |       …                       // glyphicon fonts omitted
|           |
|           \---lib
|                   bootstrap.min.css
|                   video-js.css
```

Figure 3.6: List of project inventory (some files omitted)

Overview of Code Structure

End Points

There are three main end points, we do our best to keep it as RESTful as possible.

transcription/

GET

returns a list containing the name and id of all the transcriptions currently in the database.

```
RESPONSE:
[
  {
    "_id":                                          "535becf06d4485ddd4eaca3d",
    "name":                          "Erika                          Meitner"
  }
  ...
]
```

transcription/*:id*

id is the video id

GET

returns the list of cue's associated with video.

```
RESPONSE:
[
  {
    "name":"with her",
    "time":1922390,
    "duration":170,
    "confidence":100,
    "_id":"535e807955aa08780f8a58e9"
  }
  …
]
```

PUT

Will update the list of cues for that object in the database.

```
REQUEST:
An array of JSON objects containing the cues of a speech
[
  {
    "name":"with her",
    "time":1922390,
    "duration":170,
    "confidence":100,
    "_id":"535e807955aa08780f8a58e9"
  }
  …
]
```

video/*:id*

GET

return a link to the video source.

```
RESPONSE:
{
  "_id": "535becf06d4485ddd4eaca3d",
  "videoSrc": "http://vtechworks.lib.vt..."
}
```

PUT

```
Will update the videoSrc for that object

REQUEST:
{
  "_id": "535becf06d4485ddd4eaca3d",
  "videoSrc": "http://vtechworks.lib.vt..."
}
```

Figure 3.7: Project End Points

Views and Layouts

Most of the exciting code is in the `views/` and `layouts/` folder and deserve some explanation. Some of this can be found in the Marionette documentation. Essentially layouts are compound views. In this app there are two layouts. The main layout, which is the layout you see when you first open the app.

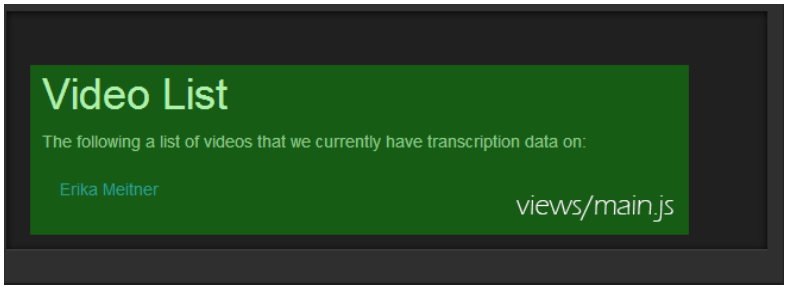


Figure 3.8: Main Layout with responsible views

The transcribing layout is shown after a video is selected.

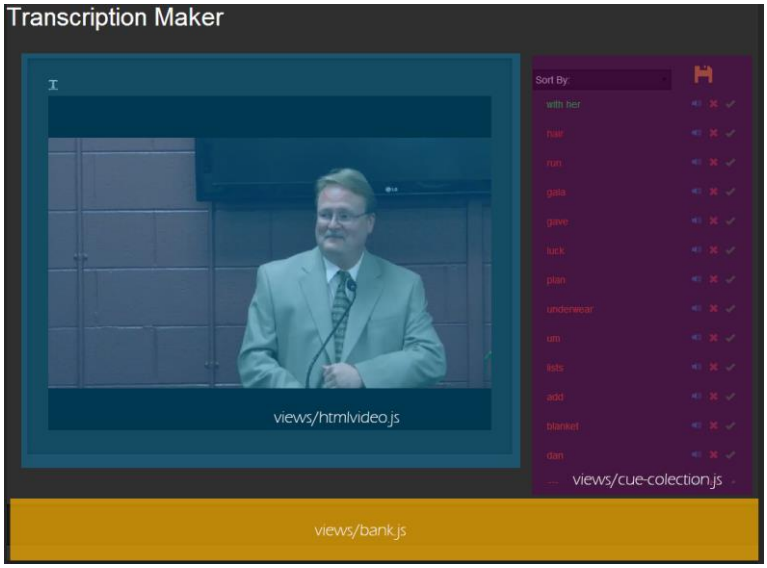


Figure 3.9: Transcribing Layout with responsible view.

Event Driven Programming

Several of the keys actions in the program are triggered by events. This is a key feature of javascript and typical of web programming. All global events are bound under the `Backbone.pubSub` module.

A summary of the events in the application are below in Table XX:

Event	Cause	Result	Triggerers	Listeners
addcue	A User clicked the check mark next to a cue	Item is added to the bank view at the bottom	views/cue.js	views/bank.js
setactive	A User clicked on a video in the main videos list	Video is set to active transcribing view is shown	views/video.js	layouts/main.js

Table 3.1: Global Events in the Transcription Maker

4. Manual Transcription Findings

In addition to the Transcription Maker, we also manually transcribed several VTech Work videos. One of our goals for this project was to compare the times of manual transcription versus automated transcription with regards to accuracy. We assumed that automated transcription would be much faster but we were unsure of how accurate the transcription would be. In order to fully test our hypothesis, several members of the group found videos (with similar times) and timed how long it took them to manually transcribe a video. Our results are shown below in **Chart 4.1:**

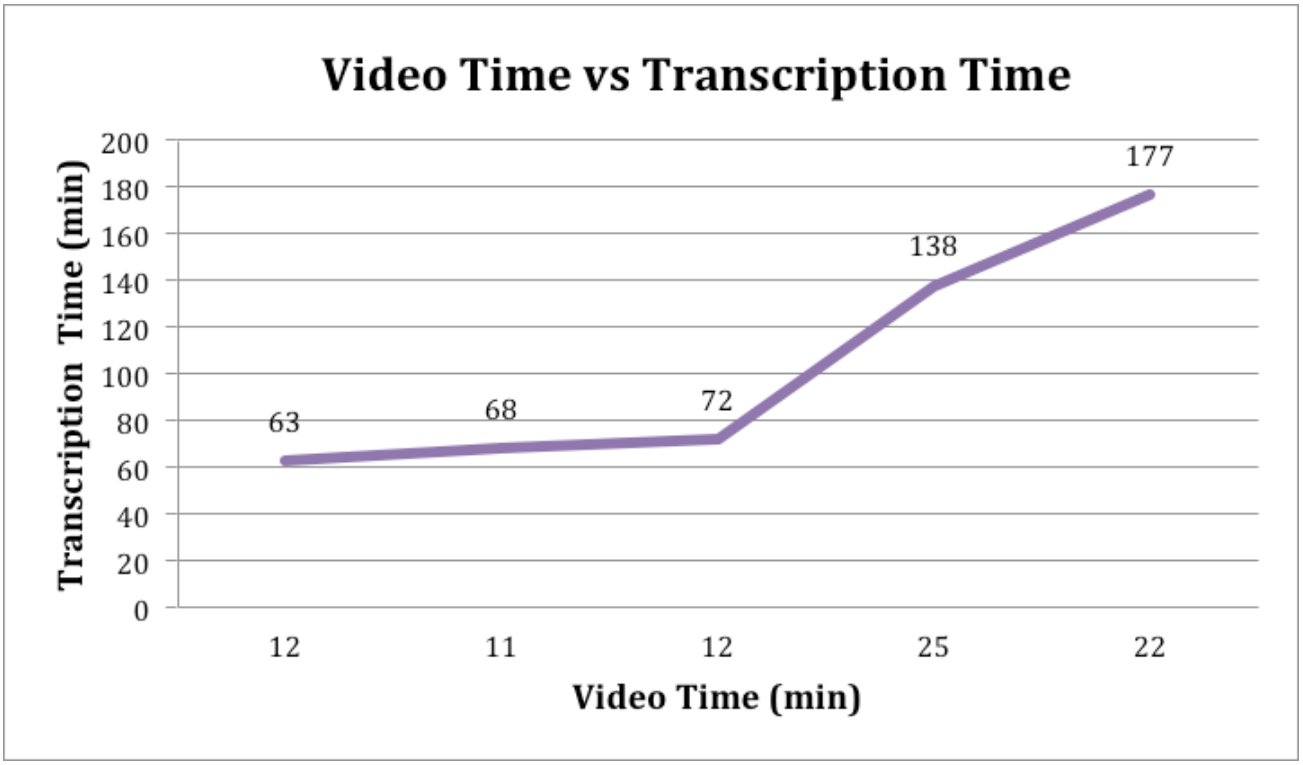


Chart 4.1: Graph displaying the manual transcription for various video lengths

As you can see, the time it took to manually transcribe a video took much longer than the actual video time. In fact, it took on average 6 times as long as the video time to manually transcribe.

Our client was very interested in why the manual transcription took so long and why the times varied for videos of similar length. We concluded that there were a number of reasons for the long transcription time. For one, it is difficult to understand a speaker with a foreign accent and this is shown in the transcription time. When the transcriber has a difficult time understanding the speaker, he/she constantly has to go back and replay the same video clip in order to understand the words. Secondly, when speakers use foreign names or foreign terms, the transcriber must replay the clip in order to understand. Often times, the transcriber still does not understand the foreign name/term and simply has to write down the syllables that he or she hears and look up the name/term when the video is over.

This is also true for subject specific terminology because the transcriber will usually not be in the same related field as the speaker. And lastly, we found transcribing difficult when a speaker on the video was not wearing a microphone. For instance, in many of the videos the audience would ask questions or a different speaker would introduce the guest speaker and would often not wear a microphone. The transcriber would have to replay the video clip multiple times in order to block out the background noise and focus on the speaker's voice. These are just some of the reasons that manually transcribing the videos took so long.

However, even though the manual transcription took so much time, we were ensuring complete accuracy. After the manual transcription, we uploaded our videos into the Transcription Maker and found that it had some of the same difficulties in understanding words. As a result, it took roughly the same time to edit the words in Transcription Maker as it did to manually transcribe the videos. So it merely becomes a choice in which method you prefer since it takes the same amount of time to ensure accurate transcription.

5. Lessons Learned

This section is a reflection on our experiences with the project.

Timeline Schedule

At the beginning of our project, we met with our client and got a very high-level overview of what needed to be accomplished. With this in mind, we set specific goals that needed to be fulfilled over the entire semester. Our schedule was as follows:

Goal	Deadline
meeting with Paul and the Event Capture Group	Feb 14
have all research completed on possible transcription tools and speech-to-text software	Feb 21
final directions selected for transcription tool and speech-to-text software	Feb 28
midterm presentations	March 4-6
initial set up of tools and early testing	March 21
rough outline of workflow completed; conversion of Adobe Premiere XML to manageable JSON complete	March 28
implementation of confidence value models and video navigation	April 4
cleanup front end and code	April 18
final outline of workflow completed; analysis of manual transcription vs automated completed; testing completed	April 25
final presentations	May 1-6
final paper due	May 6

Table 5.1: Timeline of Project Milestones

Since little research was done on speech-to-text recognition for this project, we first set off to examine what would be the best solution to meet the needs of both the system and our client. Upon further investigation, we changed the scope of our project due to the complexities of both manual and automatic speech-to-text transcription, which are described in more detail in the following section.

Problems and Solutions

When we first started researching manual and automatic speech-to-text transcription, we assumed that we could find a program that would transcribe speech in a sufficient manner. None of us had previous experience with the topic and assumed that it wouldn't be too difficult to find a good program, specifically because of the amount of quality programs that Virginia Tech has licenses for. However, we quickly learned that these software programs were not very accurate when trying to transcribe from speech in video to text. In fact, under many circumstances, less than 30% of the words that were transcribed were actually correct. Unfortunately, we could not find a solution to this problem by itself, as the technology just isn't as up-to-par as we'd like it to be yet.

We will now go into further detail on why automatic transcription isn't as valuable as we would want it to be. Unlike how most computer programs are coded, English can very often be ambiguous. Even to humans, one sentence could possibly be taken one way by one person, but in a completely different way by a second person. Computers cannot differentiate between these ambiguities, especially if they have to do with the tone of the speaker's voice (e.g. sarcasm). Furthermore, grammar plays a large role in helping reduce variations in the English language. Unfortunately with current technologies, computers aren't smart enough to understand the massive amounts of grammar that comes along with a seemingly endless dictionary of English words. That is to say – the complexity of our language, as of now, is not easily parsable by a computer, since the computer cannot create an equivalent vocabulary that it can easily understand. For instance, "*He went to the store to get broccoli soup and french bread."* This example shows the ambiguity in English with punctuation. Computers cannot understand whether there are two (broccoli soup/french bread) or three (broccoli/soup/french bread) items in this list. Another example that demonstrates further ambiguity is when differentiating between proper nouns and verbs: *"Pat and Sue drove the car until it got dark."* To humans, it is obvious that there are two subjects acting upon one direct object in the sentence. To a computer, it can't be sure whether there is one subject (just Pat), where *Sue* is a verb, or not. Of course we can easily differentiate, since *Sue* is capitalized, but a computer would not know that by listening to a voice. These are just a few of the endless amounts of reasons why automatic transcribing has its issues at the moment.

Videos that are transcribed aren't always perfect quality for computers to understand. The setting in which the audio is recorded has an enormous effect on how well the computer can even understand the voice of the speaker. For example, imagine how the distance between the speaker and the microphone affects the overall quality of the main speaker. If there are distractions between the two (e.g. a large audience, where coughing often occurs), the overall clarity goes down quickly. In addition, the clarity also decreases when the speaker is not using a normal tone of voice that is understood (even by humans). Examples include speaker tempo, jumping in voice, nervousness, accents, etc. Finally, natural language brings in many distractions when trying to transcribe, such as with pauses using "um" and "like."

Through our research and findings, we have concluded that there is a major problem with the current versions of automatic transcribing software. The accuracy of these programs is little to none and manual transcription is still needed to go through and correct words. As of now, our solution is the Transcription Maker, in which Adobe attempts to transcribe the video and the user has a friendly interface in order to correct the key words. We believe that the videos do not require a complete transcription because someone who is searching through the video will not care about all of the filler words. Instead, people who search through the videos only care about the key terms because that is what the video is basically about and thus sets it apart from the rest. We propose using the Transcription Maker as a tool to identify the key terms in a video. The key terms will then be used as tags in the metadata to make the video searchable.

As far as our solution to automatic transcription software, we think that the software must have a large vocabulary and understand the English language better. For instance, the software must understand that “Sue” is the name of a female and not a verb. In order to understand the difference, the software must be intelligent enough to actually process the speech as well as understand the context in which the word “Sue” is used. A large vocabulary (in addition to a dictionary) should also be built with common filler words, such as “um,” and slang words that the software can pull from and analyze the speech. We believe that automatic transcription software will not up to par until the software is able to understand and analyze the speech to near the same level that a human can interpret the meaning of the words.

6. Acknowledgements

This section lists the people that we have worked with and those that have guided us throughout this project.

We would like to thank:

Keith Gilbertson
Digital Libraries Technician
keith.gilbertson@vt.edu
(540) 231-9049
5090 Newman Library
560 Drillfield Drive
Blacksburg, VA 24060

Paul Mather
UNIX Systems Administrator
pmather@vt.edu
(540) 231-6383
2350 Torgersen Hall
620 Drillfield Drive
Blacksburg, VA 24061

Therese Walters
tkwalters@vt.edu
(540) 231-1886
203B Newman Library
560 Drillfield Drive
Blacksburg, VA 24061

Dr. Edward Fox
Professor of Computer Science
(540) 231-5113
fox@vt.edu
2160G Torgersen Hall (0106)
620 Drillfield Drive
Blacksburg, VA 24060

7. References

Adobe Systems Incorporated (2012). Adobe Premiere Pro (Version CS6). [Software]. Retrieved from http://www.adobe.com/products/premiere.html.