Christopher Hansen

Professor Batchelder

11/16/2020

Software Requirement Specification Document for Toastmaster's Toolbox System

Table of Contents

Preface

The software requirements specification document serves to provide information pertaining to the requirements engineering, system architecture, system modeling, and system design of the Toastmaster's Toolbox system. The material focus of this document is for the current developer and any future developer(s) to understand what has been or will be implemented for this system. This is the second version of the software requirements specification document, and incorporates details of the current system, as well as corrections made to the document from the original revision.

Introduction

This system will serve as an aid in Toastmasters presentations. The system is called a Toastmasters Toolbox and will fulfill many of the roles in a traditional Toastmasters meeting. These roles include facial expression analysis, speech disfluency feedback, timing cue, provide feedback on the speaker's rate of speech, and will provide a report to the user at the end of the speech. This system will be composed of a software system that runs on a single computer, or PC. The system aims at helping the user improve their public speaking abilities in a stand-alone manner. This is in line with the intention of Toastmasters.

Glossary

1. API: Application Programming Interface; computing interface which defines interactions between multiple software intermediaries.
2. Emulation: reproduction of the function or action of a different computer, software system, etc.
3. Graphical User Interface (GUI): a visual way of interacting with a computer using items such as windows, icons, and menus, used by most modern operating systems.
4. Linux: open-source operating system.
5. Lubuntu: lightweight distribution of Ubuntu, a Linux operating system.
6. Module: a python file that contains a set of variables, functions, and/or classes.
7. Package: a namespace that can contain multiple packages and modules.
8. PC: Personal Computer
9. Processor: an electrical unit that executes instructions and computations
10. Real-time: the actual time during which a process or event occurs.
11. Software Architecture: fundamental structure of a software system
12. Virtual Machine: An emulation of a computer system.

User Requirements Definition

This system will fulfill multiple roles that are typically performed by a human observer during a Toastmasters speech.

Functional Requirements

1. The system shall perform a real-time facial expression analysis of the current speaker.
2. The system shall provide speech disfluency feedback to the current speaker.
3. The system shall display timing cues to the current speaker that indicate how long the speaker has spoken.
4. The system shall provide a report of the overall performance of the speaker once the speech is done.
5. The system will provide real-time feedback on the rate of speech of the current speaker.

Non-Functional Requirements

1. The system shall be composed of software within a single PC as well as one external device for the Ah-Counter.
2. The system should perform in real-time.
3. The system will utilize a virtual machine running a Lubuntu distribution of Linux hosted on a Windows 10 operating system.
4. The software will be developed in a Linux environment running within the virtual machine.
5. The system will utilize a Graphical User Interface to allow the users to provide input and observe outputs from the system.

Use Case Diagram

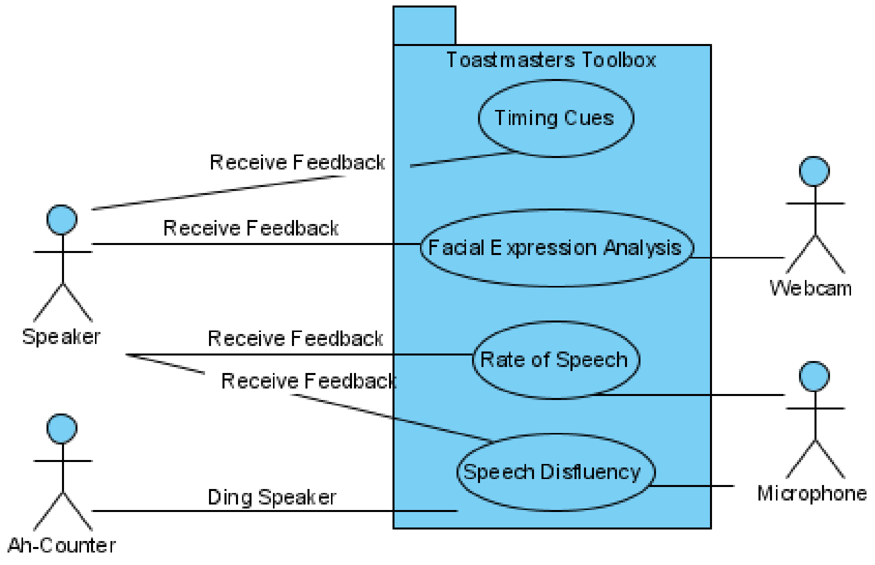

Figure URD-1

System Architecture

The architecture of the system can be classified as a client-server layered architecture. The system will be composed of four main modules. Each of the modules are contained within their own respective file. In total, there are four files for each of the four modules. There are two higher level modules for the two different GUIs for the server side of the system and the client side of the system. These are called the *Server and Host Graphical User Interface Module* and the *Client Ah-Counter Graphical User Interface Module*. These will allow the users to interact with the system through an intuitive interface using a variety of methods provided by the GUIs. The server and client will each have a lower-level, logic, and event driven module as well that interact with each other and their respective GUIs. These are called the *Server and Host Application Logic Module* and the *Client Ah-Counter Application Logic Module*. These modules will utilize, and therefore have access to even lower-level, open-sourced modules that are contained within external libraries that are included in the files. All included modules are imported at the beginning of each file for their respective module. See the System Models section for a graphical representation of the overview for the system architecture, Figure SM-3.

Server and Host Application Logic Module

The Server and Host Application Logic Module provides most of the functionality of the Toastmaster’s Toolbox system. This module is contained within the python file *Final_Project.py* and is the most processor intensive module. This module can be broken down into 7 main sections. These sections provide the classes and methods that are used to fulfill a large portion of the functional and non-functional requirements of the system. These 7 sections are composed of 4 threads that run simultaneously, as well as 3 additional sections that are comprised of methods that are grouped according to their functionality. These sections are as follows: Facial Expression

Recognition Thread, Web Server Hosting Thread, Speech Recognition Thread, Timer Thread, File Input/Output (I/O) Methods, Reporting Methods, and Generic Application Methods.

Facial Expression Recognition Thread

This thread utilizes the fer (Facial Expression Recognition) and Video modules from the external FER package. This is an open-source software that utilizes a neural-engine to determine what emotions a user is exhibiting by acquiring video from the webcam and processing the feed to determine the facial expressions. This thread continuously loops and analyzes the expressions by frame from the webcam. This thread also maintains output to the Server and Host GUI to update the speaker on what emotion they are exhibiting, the ‘magnitude’ of the emotion, calculates and outputs the FPS (Frames-per-Second) counter to the Server and Host GUI, and also allows the user to mirror the video feed based on if a QRadioButton widget is activated on not on the Server and Host GUI. This thread begins running at the launch of the program and will terminate once a user stops the speech. The thread can resume only if a user begins a speech again, after stopping it.

Web Server Hosting Thread

This thread utilizes the flask module from the external open-source Flask package. The thread is simple, simply hosting a simple web server. This section of code also routes to and runs methods based on requests from the client. These methods include ways to update the GUI with the current Ah-Count and play an audio file when the ah-counter dings the speaker, and also allows the ah-counter to change the background color of the Ah-Counter label on both Server and Client GUI Modules. This thread begins running at the launch of the program and will terminate once a user stops the speech. The thread can resume only if a user begins a speech again, after stopping it.

Speech Recognition Thread

This thread utilizes the pyaudio module from the PyAudio package, as well as the speech_recognition module from the Speech Recognition package. This thread continuously loops, gathering audio data from the microphone by utilizing the pyaudio module and its functions, sends this audio data to Google's speech recognition API (application programming interface) using the speech_recognition module, calculates the average words-per-minute, and then outputs this to the Server and Host GUI Module. This thread also outputs a string of the recognized audio to the Server and Host GUI, as well as the number of words that it recognized. If this thread does not recognize any audio, it will output a corresponding message. This thread begins running at the launch of the program and will terminate once a user stops the speech. The thread can resume only if a user begins a speech again, after stopping it.

Timer Thread

This thread utilizes the internal time module. This thread begins once the user starts the speech via the Server and Host GUI. The thread begins processing by receiving values for different speech timer settings from the Server and Host GUI, and then begins a timer. Through each loop, the thread determines if the speaker is within certain thresholds that were specified earlier and outputs a background color to the Server and Host GUI that represents the timer flags. The color corresponds to the threshold that has been met or surpassed. This thread ends once the speaker reaches the end of the allotted speaking time.

File I/O Methods

This section is a grouping of two methods that can be called to either save the generated report to a .txt file or import a report from an existing .txt file. The .txt file that is saved is always "Toastmaster Report.txt". If this file already exists, it will be overwritten. The import function will import a file under the same “Toastmaster Report.txt” name and output the report to the Server and Host GUI.

Reporting Methods

These methods provide ways to generate the report after the speaker is done and output it to the Server and Host GUI, navigate to the report page on the GUI, or to cancel the report, which just navigates to the previous page of the Server and Host GUI.

Generic Application Methods

This section contains methods to begin and stop the speech, to terminate all running threads, to set the speech settings once the user clicks the ‘Enter’ button on the first page of the Server and Host GUI, and a function to quit the application when user exits the program.

Server and Host Graphical User Interface Module

This is the highest-level module that allows the speaker to interact with the system. It allows the speaker to provide inputs in the form of button or key presses and provides outputs to the user that it receives from the lower-level Server and Host Application Logic Module. This module is composed of a single pyQt5 file called *Final_Project.ui*.

Client Ah-Counter Application Logic Module

This module is a lower-level module that takes in inputs from the Client Ah-Counter GUI and provides outputs to the Client Ah-Counter GUI as well as the Server and Host Application Logic Module via a network. This module is not processor intensive and runs one thread that accesses the hosted web server, ran by the Server and Host Application Logic Module. The module also contains functions that run when triggered by the event of the Ah-Counter clicking one of the buttons. These functions execute simple logic functionality such as incrementing or decrementing the ah counter and then request to post to an address of the webserver.

Client Ah-Counter Graphical User Interface Module

This is the highest-level module that allows the Ah-Counter to interact with the system. It allows the Ah-Counter to provide inputs in the form of button presses and provides outputs to the user that it receives from the lower-level Client Ah-Counter Application Logic Module. This module is composed of a single pyQt5 file called *flask_client.ui*.

System Requirements Specification

Functional Requirements

1. The system shall output the real-time video of the speaker in the GUI. This is to mimic the function of a mirror.
2. The system shall analyze the expressions of the speaker using the footage captured by the webcam.
3. The system shall determine the mood of the speaker based on the expressions it analyzes.

Non-functional Requirements

1. The capture, analysis of expressions, and providing feedback shall all occur in real-time.
2. The system shall capture the audio of the speaker's speech via a microphone.
3. The system shall parse through the speech in real-time.
4. The system shall output an audio signal and graphical signal through the GUIs when the speaker uses a filler word such as "uh" or "um."
5. The Ah-Counter shall use a device connected to the internet that allows them to ding the speaker via a button when the speaker uses a filler word.
6. The Ah-Counter's device shall communicate with the system via a webserver.
7. The webserver shall be hosted on the speaker's machine.
8. The system shall keep track of how many times the Ah-Counter dings the speaker.
9. The system shall keep track of the number of filler words that it detects.
10. The system shall output a graphical signal that has a color that corresponds to the appropriate time for the cue.
    - 10.1 There shall be three different cues corresponding to three different times.
    - 10.2 There shall be 3 different colors that correspond to the three different cues.
11. The user shall be able to configure the three different times that correspond to the cues.

12. The system will display a report that contains information about the speech as soon as the speech is over.
    - 12.1 The report shall display the amount of time that the speech took.
    - 12.2 The report shall display the number of times the Ah-Counter dinged the speaker.
    - 12.3 The report shall display the most used emotion by the speaker.
    - 12.4 The report shall display the least used emotion by the speaker.
    - 12.5 The report shall display the rate of speech of the speaker.
13. The speech ends when the speaker presses a stop button on the GUI, or when the speaker runs out of time.
14. The system shall save the report to a .txt file.
15. The system's main program shall be executed in a virtual machine.
16. The virtual machine shall run Lubuntu.
17. The system shall allow an external device to connect to it via a webserver.
18. The external device shall communicate with the system via the webserver.
19. The system's latency, or the classification of real-time shall be less than 250 milliseconds.
20. The real-time feedback of the rate of speech tracker shall refresh the value every 5 seconds.
21. The virtual machine shall be hosted on a Windows 10 operating system.
22. The GUI shall output information to the speaker in real-time.
    - 22.1 The GUI shall display the number of times the Ah-counter dinged the speaker.
    - 22.2 The GUI shall display the current rate of speech.
    - 22.3 The GUI shall display the current mood it perceives the speaker to have.

23. The GUI shall have buttons to start and stop the speech.

24. The GUI shall have a window for webcam feedback.

System Models

This chapter includes graphical system models showing the relationships between the system components and the system and its environment.

System Context Diagram

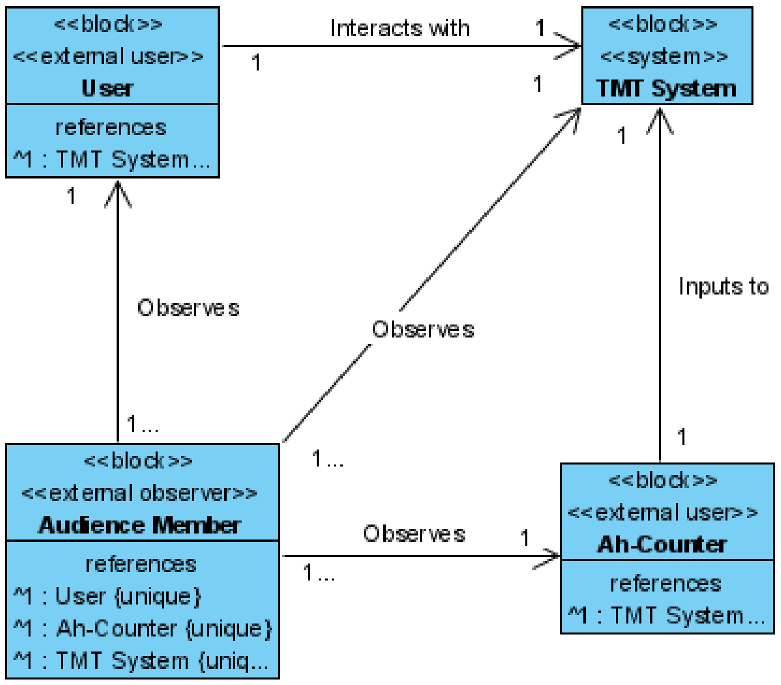

Figure SM-1

Software System Context Diagram

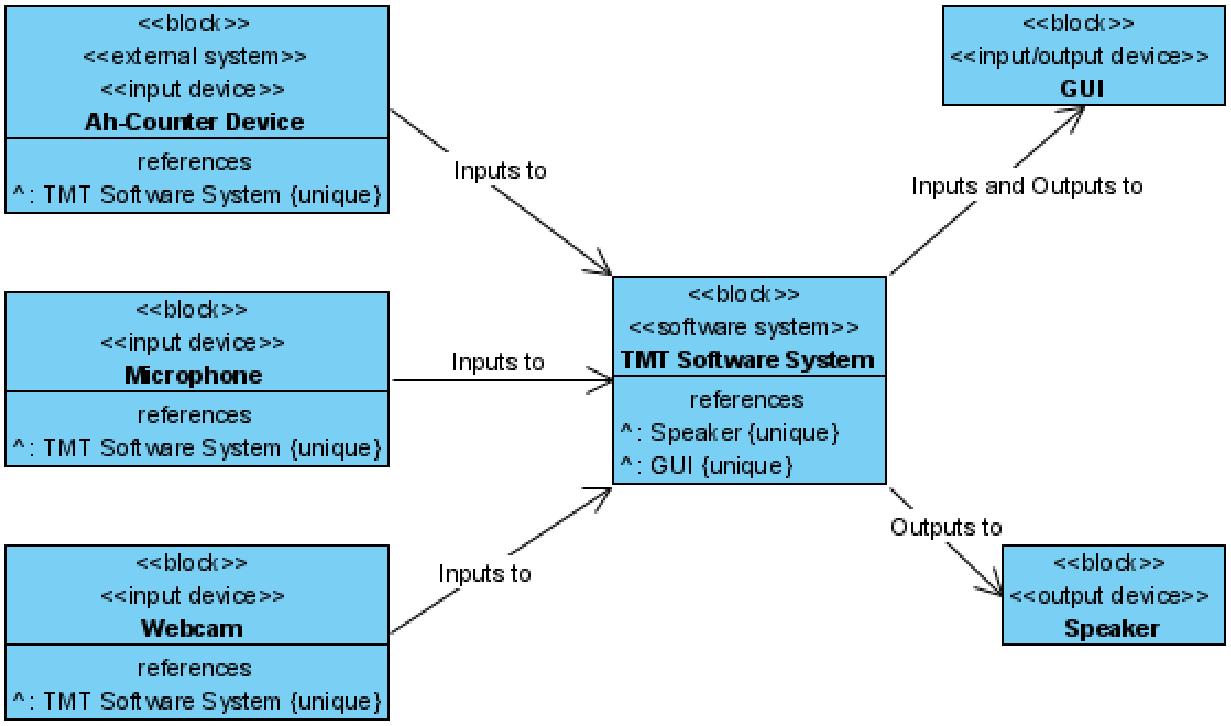

Figure SM-2

System Architecture Overview

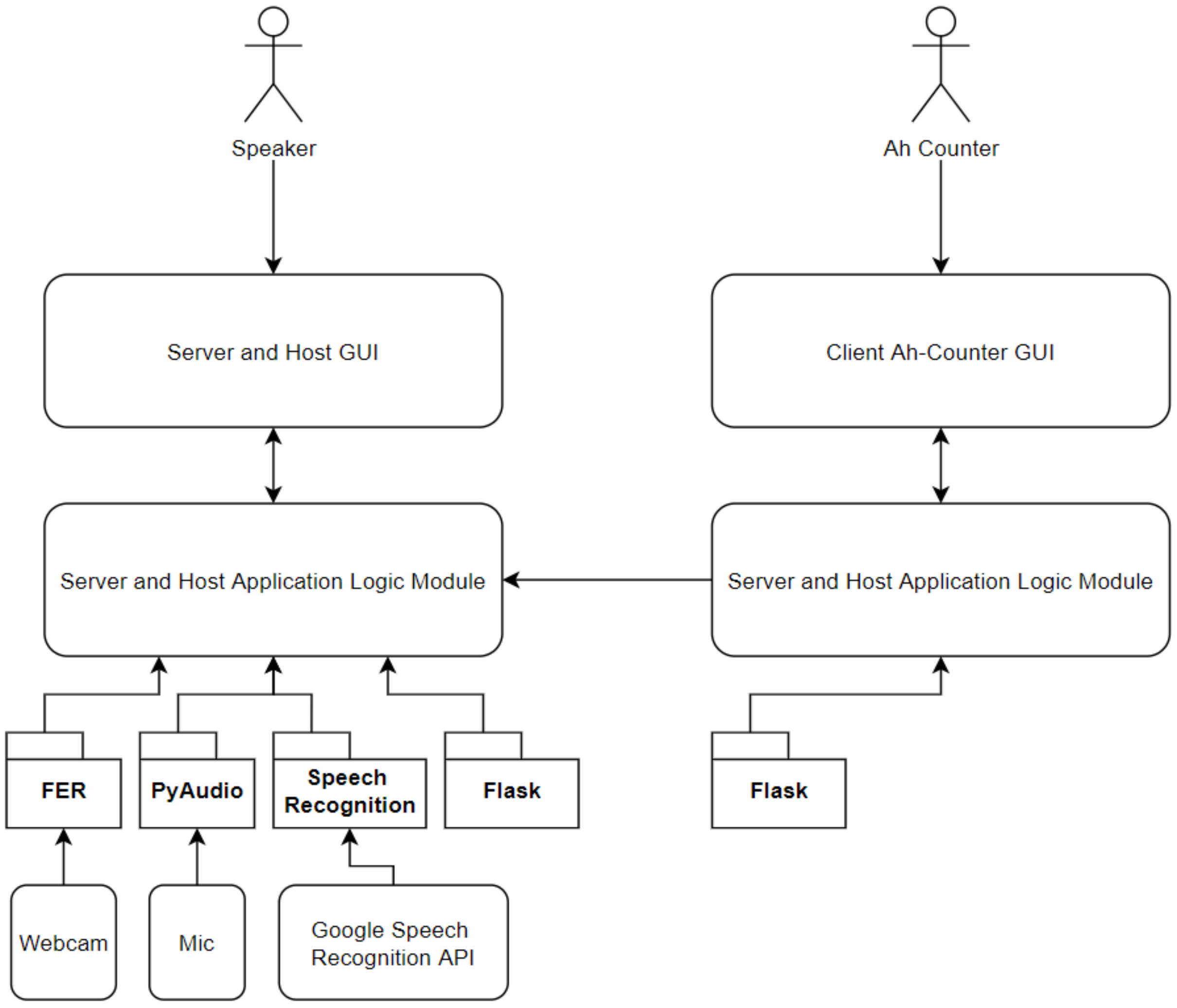
Speaker
Ah Counter
Server and Host GUI
Client Ah-Counter GUI
Server and Host Application Logic Module
Server and Host Application Logic Module
FER
PyAudio
Speech Recognition
Flask
Flask
Webcam
Mic
Google Speech Recognition API

Figure SM-3

System Evolution

This describes the fundamental assumptions on which the system is based, and any anticipated changes due to hardware evolution, changing user needs, and so on.

This system has been designed around a few hardware requirements and assumptions. I assume the speaker's computer has access to a webcam with a resolution of at least 400x300 pixels, a microphone, an internet connection, and at least two physical processing cores. The ah-counter's machine hardware requirements are even less; at least two physical processing cores and an internet connection. Both machines need two processing cores because at least one core is dedicated to the host's operating system, and the second to the virtual machine's operating system. It would be preferable if the host's machine had 4 or more cores so that they could dedicate at least 2 to the virtual machine, due to the heavy processing nature of the Server and Host Application Logic module.

Future design changes should consider limiting the number of threads required to run the software, but enough to maintain core functionality of the system. The number of threads the host machine should execute is proportional to the number of dedicated processing cores the virtual machine has access to. Higher resolution webcams are welcome, but performance can depreciate quickly depending on the resolution and processing power of the machine. If the host machine's processor can maintain required performance goals while the resolution of the webcam is increased, facial expression recognition will be improved, as well as video feed quality.

Appendix A – Hardware Requirements

Hardware requirements define the minimal configurations for the system.

1. At least two physical processing cores on server/speaker and client/ah-counter machines.
2. Server machine must have a webcam with a minimum resolution of 400x300.
3. Server machine must have access to a webcam.
4. Server and client machines must have internet connection of at least 2 Mbps download and 500 Kbps upload.

Appendix B – Diagrams

Figure URD-1: Use Case Diagram to show how users will engage in the functionality of the system.

Figure SM-1: System Context Diagram to show the boundaries and interactions of the entire system, including hardware components.

Figure SM-2: Software System Context Diagram to show the boundaries and interactions of the software system.

Figure SM-3: System Architecture Overview to show the fundamental structure of the software system.