Software Requirements Specification

for

Ubuntu Touch Audio Mixer (working name)

Prepared by:

Steven Lares

Contents

Chapter 1

Introduction

The aim of this document is to provide in-depth insight of the Ubuntu Touch Audio Mixer software by defining the problem statement in detail.

It describes expected capabilities from end users while defining high-level product features.

This document was built using references from:

1. https://www.overleaf.com/latex/templates/cse355-software-requirements-specification-layout/pvjpzxthtngc
2. https://www.perforce.com/blog/alm/how-write-software-requirements-specification-srs-document
3. https://www.geeksforgeeks.org/software-engineering-quality-characteristics-of-a-good-srs/
4. https://www.geeksforgeeks.org/software-engineering-classification-of-software-requirements/?ref=lbp
5. utdallas.edu -SRS4.0 doc
6. https://ieeexplore.ieee.org/document/278253

It was compiled through the LaTeX template found in

https://www.overleaf.com/latex/templates/cse355-software-requirements-specification-layout/pvjpzxthtngc

It uses Prototype Outline 1 for SRS Section 3 from the IEEE link

1.1 Purpose

This document is intended to be read by volunteer developers and testers, as well as anyone else curious enough to learn more about the project.

Admittedly, this SRS is more developer/tester oriented since it relies on git version control to track its development. Git is used instead of maintaining a version history table within the SRS itself; the latter of which is typical for SRS documents created in Microsoft Word or SRS-creation software.

Therefore, learning how to compare git commits and branches will help with understanding the SRS development over time.

NOTE 1: This document is meant to aid a first official release. It will likely not be maintained once the project has it's official release, and will stay for archival purposes.

NOTE 2: This document is also a loose adaptation of an SRS. As such, it contains design decisions, doesn't make mention of financials, and has some code practices outlined.

1.2 Product Scope

The app will be downloaded by the user through Ubuntu Touch's Open Store. It will not be available elsewhere, other than the GitHub repo which hosts source code and possibly test builds of the app.

It is intended to be used in two major ways:

1. It can be used as a portable external audio mixer.
   (a) This means that the app will act as a bridge between:
      i. An external audio playback / microphone device feeding audio input into the phone running the app.
      ii. An external audio playback device receiving mixed audio output from the phone running the app.
   (b) This is the main strength of this app for the following reasons:
      i. As of 10/21/2022, there are no Android or iOS alternatives that provide this functionality.
      ii. PulseEffects does provide similar functionality, however there is a major limitation: You must run a server on an Linux machine, and have another Linux machine (with the same package installed) connect to this machine in order to receive the mixed audio output
2. It can also be used as an internal audio mixer.
   (a) This means that the app will be able to mix audio across the system.
   (b) This functionality is comparable to other apps on the Android and iOS stores, usually with the terms "equalizer", "EQ", "Mixer" in their name.
   (c) This functionality leads this app into having desktop counterparts:
      i. EQualizerAPO for Windows (https://sourceforge.net/projects/equalizerapo/)
      ii. PulseEffects for Linux (https://github.com/wwmm/easyeffects).
   (d) Why use this Ubuntu Touch implementation instead of Android or iOS? Well:
      i. This app will behave closer to its desktop counterparts in that it will not contain ads, subscriptions, and other scummy money-grubbing schemes. If donations are added to support the app, they will stay on the official Open Store page and out of the user's way.
      ii. The app is open source, and is not a mysterious black box.
      iii. Will be implemented using as much native Ubuntu Touch and Linux functionality as possible.
      iv. Will allow the user to branch audio streams, do processing on those branched off streams, and then finally converge them.
      v. The app will allow you to order the processing modules in the order that you want. This allows for more robust audio processing than currently available alternatives.
      vi. It will also be written with a faster and more memory efficient programming language compared to Java (Android) or Swift (iOS). This is especially important as real time audio mixing can be a CPU and memory intensive process.
   (e) **NOTE:** This is also the "fall-back" in case functionality #1 is not possible in the UT environment. However, it will still retain its strength over Android / iOS counterparts.

NOTE: The above list may be revised over time depending on how closely initial requirements can be fulfilled in the UT environment.

1.3 Risk Definition and Management

At this early stage within the project, there are no risks. This may change as the project matures.

Chapter 2

Development Practices

2.1 Code Style

https://mitcommlab.mit.edu/broad/commkit/coding-and-comment-style/

2.2 Test Style

Will implement a classical unit testing and test driven development style (TDD). I will use the book "Unit Testing Principles, Practices, and Patterns" by Vladimir Khorikov as a guide.

TDD will be implemented using concepts from parts 1 and 2 of the book for unit testing. Part 3 is for integration testing.

By combining TDD with this SRS, I will be able to better outline units of behavior and shared dependencies.

Unfortunately, I think TDD (through unit testing) can only be implemented to a limited degree.

This project is integration test heavy, not unit test heavy. This is because this project has many shared dependencies, and the domain model is fairly simple. Therefore, the TDD unit testing book might have limited applicability.

For code that can be unit tested, use the "Unit testing" book as a guide for good code design. These tests should actually be in the code, not the SRS. For that reason, I have revised the old functional requirements away from its original unit test design.

- Link on TDD through classical unit testing:
  - https://khalilstemmler.com/articles/test-driven-development/introduction-to-tdd/#Classic-Inside-OutChicago-a
  - Note that this type of TDD (inside out, classical style) will entail applying it to functionality that is specific to the domain model and mocks shared dependencies. Refer to page 37 of unit testing book for details.

Chapter 3

Ubuntu Touch Notes

This chapter contains additional notes on items specific to development in the UT environment. There is some decision making that is listed here.

General docs: https://docs.ubports.com/en/latest/

3.1 License

Choose license from the following list.

1. https://www.gnu.org/licenses/gpl-3.0.en.html
   (a) As far as I can tell, this is the only one with a "viral" approach to software licensing. That is, if you use a GPL componeent in your software project, your entire software project must be GPL.
   (b) Will go with this, since I plan to make this an open source, non commerical project (but might open donations in the future)
   (c) Most open source linux projects tend to be GPL anyways, and since it is viral it makes sense to just pick this to begin with.
2. https://opensource.org/licenses/MIT
3. https://opensource.org/licenses/BSD-3-Clause
4. https://opensource.org/licenses/ISC
5. https://www.apache.org/licenses/LICENSE-2.0

Resources:

1. https://www.quora.com/What-are-the-four-licenses-MIT-Apache-GPL-and-Creative-Commons-How-are-they-different
2. https://www.quora.com/Which-license-should-you-use-on-Github-for-open-source-projects
3. https://www.gnu.org/licenses/license-recommendations.html
4. https://www.gnu.org/licenses/why-not-lgpl.html

3.2 IDE and Debugger Choice

- Use QtCreator for IDE since it has debugging and testing:
  - https://docs.ubports.com/en/latest/appdev/code-editor.html
- Use either C++ or RUST. Use UT Specific libraries when possible:
  - https://docs.ubports.com/en/latest/appdev/nativeapp/index.html

3.3 App Suspension

Apps are suspended whenever not in the foreground, or when the device is locked. When an app is suspended, it cannot receive location data. For this reason, apps will not be able to track your location whenever they are not in use or the device is locked.

1. https://docs.ubports.com/en/latest/userguide/dailyuse/location.html
2. https://docs.ubports.com/_/downloads/ro/latest/pdf/

NOTE: May need the user to use UT Tweak to remove app suspension for full usage. However, there may be a way around it.

3.4 Misc Links

Other guides that may be helpful in the future:

1. https://docs.ubports.com/en/latest/appdev/guides/index.html
2. https://phone.docs.ubuntu.com/en/platform/
3. https://phone.docs.ubuntu.com/en/platform/guides/app-confinement
4. https://wiki.ubuntu.com/SecurityTeam/Specifications/ApplicationConfinement
5. https://phone.docs.ubuntu.com/en/platform/guides/lets-talk-about-performance

Chapter 4

Shared Dependencies

This is the chapter that focuses heavily on shared dependencies.

What are shared dependencies? To paraphrase the book "Unit Testing Principles, Practices, and Patterns" shared dependencies are both the external and out-of-memory dependencies that the project will have. This definition can also include internal dependencies, such as file I/O.

As it stands, this project is heavy on shared dependencies.

Since I will follow the classical approach to TDD, I need to know these shared dependencies so that my unit tests can mock these out.

4.1 Internal Dependencies

This section outlines the internal dependencies that the project will have. This includes file I/O, telephone API's, audio stream API's, etc. I think this will mostly be functionality that is already implemented within Ubuntu TOuch.

TODO: Focus on domain model / UI mockup / functional requirements first. Then, work on outlining the internal libraries / frameworks / tools that UT has implemented that's for functionality.

TODO: Find out where Audio streams are stored / called in Ubuntu Touch. Any limits?

TODO: How to do file I/O?

4.2 Shared Dependencies

Below are the results of learning how to import external libraries and Linux packages into the project. At the end of this section, I have included a table of candidate open source project that I will likely use for audio processing.

Note: The table may change as I learn more while working on the project. I am even open to the idea of combining several open source packages if it is necessary.

- NOTES:
  - You can include C++ dependencies (libraries) in your project through clickable.
  - From what I currently understand, Clickable can compile ARM software into desktop architecture as it uses a container. It may not be able to go the other way (x86 / x86_64 -> ARM)
  - Interestingly, it can run the app in a continuous integration environment due to the use of the ARM -> Desktop container. There are guides on how to do CLCI

  - The consensus on Quora seems to be that the difficulty will range from: rebuilding with an ARM compiler to rewriting from scratch. It all depends on the type of software

- Links
  - https://docs.ubports.com/en/latest/appdev/guides/dependencies.html
  - https://ubports.com/blog/ubports-news-1/post/introduction-to-clickable-147
  - https://clickable.bhdouglass.com/en/latest/
  - https://www.quora.com/How-hard-is-it-to-port-an-application-from-x86-to-ARM

- Contains the largest list of open source audio software that I could find. Look under "Software Development Libraries & APIs":
  - https://github.com/webprofusion/OpenAudio

Audio Processing Libraries / Tools:

Name	Brief Description	License Type	ARM Build	C++ library	Estimated work	Link
Faust	Functional programming language for real-time signal processing	GNU General Public License v2.0	Y, doesn't depend on external libraries so compile the generated code into ARM	Y, after generating the code	May require knowing the full extent of code functionality. That may hamper scalability, and create software design compromises. However, there is a JIT compiler that may help with this.	https://faust.gra
RustAudio	Collection of audio processing and plugin libraries for the Rust language	MIT and Apache, to name a few	Not sure, may depend on how much of the code is native Rust	N, but it is written in Rust	This is a collection of REPO's. I think cpal and dsp-chain will take care of most of the use-cases, without too much effort. Unless this can't compile for ARM (or is too slow), I might consider this a top pick, pretty straightforward.	https://github.co
DISTRHO	C++ framework for creating cross-platform audio plugins. DPF can build for LADSPA, DSSI, LV2, and VST formats.	ISC License	Most likely, seems to be mostly native code	Y	Might be a lot, doesn't hold your hand	https://github.co
JUCE	Cross-platform C++ framework for developing desktop and mobile apps and audio plugins	https://juce.com/get-juce	Natively. There are threads for workarounds. Also, most users of this software are for Windows / Mac Desktop users.	Y	The library itself seems straightforward, but lack of Linux ARM compatibility might add extra work.	https://juce.com/

Chapter 5

Functional Requirements

These are the requirements stated by the user which one can see directly in the final product In order to accurately describe the functional requirements, all scenarios must be enumerated.

https://www.geeksforgeeks.org/software-engineering-classification-of-software-requirements/?ref=lbp

Some of these will be translated to unit tests during the coding stage. That will likely be requirements that don't focus entirely on shared dependenc(ies).

TODO: These functional requirements were written down in a way that resembled unit tests. I am going to move away from this approach, and rewrite them to be closer to traditional functional requirements.

TODO: Continue adding on to this list. This will be aided through finishing up the UI mockup and having the domain model more clearly understood.

5.1 System

1. File IO
   (a) load_template
   (b) save_template
2. Device Setup Per Selected Mode
   (a) switch_between_bridge_and_internal_modes
   (b) in_bridge_mode_set_the_audio_source_input
   (c) in_bridge_mode_set_the_audio_source_output
   (d) in_internal_mode_block_the_audio_source_input
   (e) in_internal_mode_set_the_audio_source_output
3. Audio Routing
   (a) chain_audio_effects_in_series
   (b) split_audio_stream_into_two_streams
4. Audio Control
   (a) adjust_audio_effect_parameters_during_audio_stream
   (b) audio_control_modules_adjust_the_paramters_of_audio_effect

5.2 Audio Effect Modules

1. Equalization
    (a) apply_15_band_EQ_to_audio_stream
    (b) apply_30_band_EQ_to_audio_stream
    (c) apply_variable_band_EQ_to_audio_stream
2. Filtering
    (a) apply_all_pass_filter_to_audio_stream
    (b) apply_band_pass_filter_to_audio_stream
    (c) apply_low_pass_filter_to_audio_stream
    (d) apply_notch_filter_to_audio_stream
    (e) apply_high_pass_filter_to_audio_stream
    (f) apply_high_shelf_filter_to_audio_stream
    (g) apply_low_shelf_filter_to_audio_stream
    (h) apply_peaking_filter_to_audio_stream
3. Miscellaneous
    (a) apply_compressor_to_audio_stream
    (b) apply_limiter_to_audio_stream
    (c) apply_delay_to_audio_stream
    (d) apply_loudness_correction_to_audio_stream
    (e) apply_preamp_to_audio_stream

5.3 Audio Control Modules

1. LFO
2. Math calculator

Chapter 6

Non Functional Requirements

TODO: After reviewing external and internal (UT Specific) dependencies and limitations, decide if you want to include this section.