SE 3XA3: Software Requirements Specification

Lines Per Minute (lpm)

Team #16, Lines Per Minute (lpm)
Jay Mody - modyj - 400195508
Jessica Lim - limj31 - 400173669
Maanav Dalal - dalalm1 - 400178115

March 30, 2021

Table 1: Revision History

Date	**Developer(s)**	**Change**
February 6, 2021	Jay/Jessica/Maanav	Initial document write-up.
February 8, 2021	Jay/Jessica/Maanav	Document completed & submitted.
March 25, 2021	Jay/Jessica/Maanav	Edit SRS Document for R1.

Contents

List of Tables

List of Figures

This document describes the requirements for lpm (Lines Per Minute). The template for the Software Requirements Specification (SRS) is a subset of the Volere template (Robertson and Robertson, 2012).

1 Project Drivers

1.1 The Purpose of the Project

Typing speed is an increasingly important skill in the digital world. Being a fast typist reduces the friction between transferring thoughts from your mind to your screen. While there are many tools that allow individuals to practice their typing speed for normal text passages, there are few tools to do the same with code. Code, unlike text passages, has a higher frequency of symbols, brackets, and numbers. There are many websites and packages that exist to improve typing speed in typical circumstances, however very few are targeted at the intricacies of coding programs (such as curly braces, camelCase, and semicolons).

The goal of the lpm package is to provide a typing tool specialized for code, in a place where you can often find programmers, the command line.

1.2 The Stakeholders

1.2.1 The Client

In this case the client is the group consisting of the TAs as well as Professor Bokhari - the staff of SFWR ENG 3XA3.

1.2.2 The Customers

The customers for lpm are developers or otherwise individuals interested in improving their typing speed in programming contexts. These customers are not paying for the package, as it is open sourced, however they will be using the product, and are deemed customers.

1.2.3 Other Stakeholders

Given that it is an open source project, other stakeholders include anyone on the internet who views our project after it is made. This is relevant to three main parties:

1. People who look at the source code after our development process for Software Development Principles (i.e. future 3XA3 students)

2. People who look at the source code after our development process to make a similar or related Python package (in the same way we are using wpm as a foundation to build lpm).

3. The owners and developers of the open source code that lpm sources for it's code snippet data.

1.3 Mandated Constraints

- **Solution constraints**: At completion, the produced system shall be a Python package, and available for download through the pip package manager.
- **Input constraints**: The only device used for input is a keyboard, given that it is a command line application.
- **Space constraints**: The lpm python package shall be no larger than 25 MB of space on the computer's hard drive (using similar Python packages as comparison).
- **Budget constraints**: The lpm package shall cost exactly $0.00, as all of the developers are unpaid.
- **Schedule constraints**: The lpm package shall be completed on or before April 12th, in accordance with the 3XA3 deadline for the project.

1.4 Naming Conventions and Terminology

- The terms "**terminal**" and "**command line**" are used interchangeably.
- **Command line interface** (abbreviated as **CLI**) is an interface for an application provided through the command line.
- **Command line application**: An application that is delivered using a CLI
- **Typing interface**: The interface that users practice their typing.
- **FR**: Functional requirement.
- **NFR**: Non-functional requirement.
- The terms "**system**" and "**application**", and "**product**" all refer to the lpm application this document specifies.
- **pip**: A package manager for python (**?**).
- **PyPI**: The python package index, which serves python packages through the pip tool (**?**).

1.5 Relevant Facts and Assumptions

- The user should be comfortable using the command line.
- The user shall be able to read basic English.
- The user shall be running any operating system that can run a Python environment, and running either Python2 or Python3.
- The final package will be named lpm assuming that name is available in PyPI (as of the time of writing, it is currently available for use, but has not been claimed).

2 Functional Requirements

2.1 The Scope of the Work and the Product

Lines Per Minute is a command-line application written in python that's lets individuals practice their typing speed for code. The application is delivered through PyPI as a pip install-able package, and is strictly interfaced via the command line.

2.1.1 The Context of the Work

Lines Per Minute is created as the final project for the course SE3XA3 Fall 2021 at McMaster University. The final project for SE3XA3 involves taking a current open source project, and going through the entire software design process to recreate the project with additional features or fixes that improve upon the original project. The project inspiration for lpm is the wpm python package (**?**).

While the wpm package strictly focuses on English text passages, lpm focuses on code snippets. While tools like typing.io exist, the creators felt that a completely free and non-online solution should exist. As such, wpm provides a perfect starting point for creating a lightweight and free alternative to practicing typing code that can be delivered via the command line. The project was started in January of 2021, and is expected to finish in April of 2021.

2.1.2 Work Partitioning

Event Number	Event Name	Input	Output
1	Starting the Typing Interface	Keyboard	Typing Interface
2	Next Code Snippet	Keyboard	Typing Interface
3	Prev Code Snippet	Keyboard	Typing Interface
4	Start the Timer	Keyboard	Typing Interface
5	Stop the Timer	Keyboard	Typing Interface
6	Exit Typing Interface	Keyboard	Return to Command Line
7	Show Stats	Keyboard	Command Line Output
8	Change Settings	Keyboard	Command Line Output
9	Show Help Menu	Keyboard	Command Line Output

Table 2: Work Partitioning Events

Event Number	Summary
1	Using the CLI, the user can start the typing interface with python code snippets via "lpm python". The user can also specify multiple languages to select code snippets from by adding additional arguments "lpm python java javascript".
2	While in the typing interface, the user can use the right arrow key to skip the current snippet and load the next one in the order.
3	While in the typing interface, the user can use the left arrow key to backtrack to load previous code snippet in the order.
4	While in typing interface, the user can start the timer by inputting a keystroke (which will be used as the first input character).
5	While the timer has been started, the user can choose to stop the timer with teh escape key. The timer may be started again using event 4.
6	While in browsing mode (not typing) ~~in the typing interface~~, the user can press the escape key to quit the typing interface and return to the command line.
7	Using the CLI, the user can show their lifetime stats as command line output via "lpm –stats".
8	Using the CLI, the user can open and modify the json file with the settings via "lpm –settings".
9	Using the CLI, the user can show a help menu via "lpm –help".

Table 3: Work Partitioning Events Summaries

2.1.3 Individual Product Use Cases

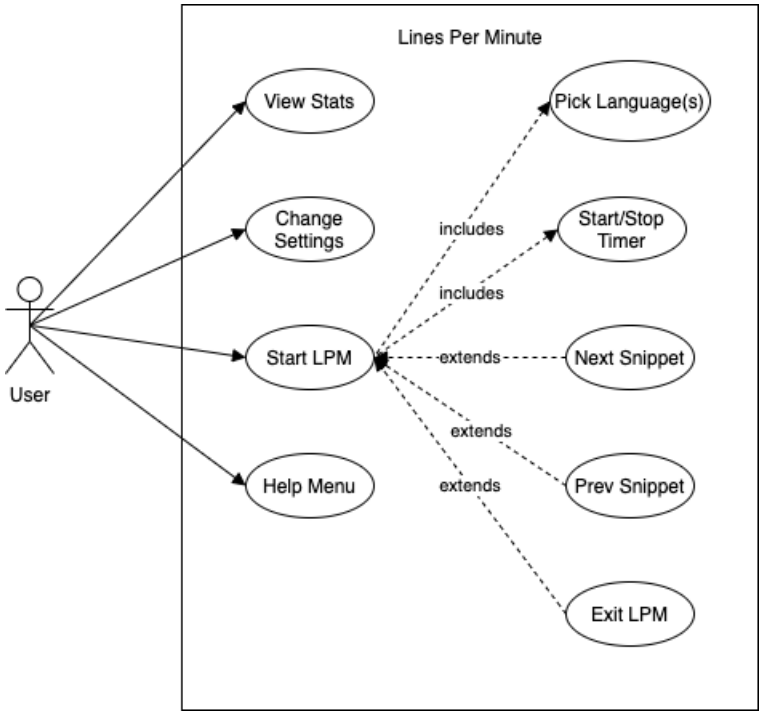

Figure 1: Use Case Diagram for lpm

Use Case 1:	**View Stats**
Related Requirements	FR1, FR4, FR5, FR25, FR26, FR27, FR28
Initiating Actor	User
Actor's Goal	To display user's typing speed statistics on terminal
Participating Actor	lpm system, User
Precondition	lpm is installed on commandline
Postcondition	User typing speed statistics will be displayed on screen
Flow of events	1. User opens terminal 2. User types 'lpm –stats' into the terminal 3. lpm, wpm and other stats are displayed through the terminal 3.1 Alt: If no stats are available, warning is displayed

Table 4: Use Case 1

Use Case 2:	**Change Settings**
Related Requirements	FR16, FR17, FR18, FR19
Initiating Actor	User
Actor's Goal	Change the settings of the lpm settings
Participating Actor	lpm system, User
Precondition	lpm is installed on commandline
Postcondition	lpm main screen settings will be changed to new settings
Flow of events	1. User opens terminal 2. User types 'lpm –settings' into the terminal 3. Settings file will open directly on the terminal 4. User can type into the settings to change settings 5. User saves settings file 6. Settings will be changed to new settings 6.1 Alt: User is given warning for invalid settings. Settings are rolled back to previous settings.

Table 5: Use Case 2

2.2 Functional Requirements

2.2.1 Command Line Interface

FR1: The CLI shall be avaliable via the command "lpm" inside any directory in a given terminal session (assuming the application has been added to the user's PATH).
FR2: The CLI shall provide an option to start the typing interface.
FR3: Upon launch, the CLI shall accept flags that allow the user to select one or more languages for the typing interface.
FR4: The CLI shall provide an option to view a help menu.
FR5: The CLI shall provide an option to view lifetime typing statistics.
FR6: The CLI shall provide an option to change the settings of the program.

2.2.2 Typing Editor

FR7: On startup, the typing interface will randomly generate a code snippet from the specified languages.
FR8: The typing interface shall display the current code snippet to the user.
FR9: The typing interface shall display the link ~~author and title of the project~~ where a code snippet was sourced.
~~**FR10**: The typing interface shall display the session statistics.~~
FR11: The typing interface shall display the current code snippet statistics.
FR12: The typing interface shall be able to skip ahead to the next code snippet in the list.
FR13: The typing interface shall be able to backtrack to the previous code snippet in the list.
FR14: The typing interface shall start the timer and start reading the user's input if they press a non arrow key (the key that is pressed will be considered the first input).
FR15: The user shall be able to stop the timer when typing a snippet.
FR16: The code snippet shall be shown to the user using the TEXT_COLOR font color.
FR17: If the user types a correct output for a character, that character shall be indicated as correct with the CORRECT_COLOR font color.
FR18: As the user types an incorrect output for a character, that character shall be indicated with the INCORRECT_COLOR font color.
FR19: The user shall be able to delete previously typed characters in which case the characters that were deleted are returned to their original TEXT_COLOR font color.
FR20: The user shall be able to exit the typing interface.

2.2.3 Code Snippets

FR21: The code snippets shall be shorter than MAX_LENGTH ~~MAX_SNIPPET_LENGTH~~ lines.
FR22: A line for a code snippet shall not exceed MAX_COLS ~~80~~ characters.
FR23: Code snippets should be made available for Python, Java, and JavaScript.
FR24: Each language shall have a minimum of MIN_NUM_SNIPPETS ~~20~~ code snippets.

2.2.4 Statistics

FR25: The application shall track the user's characters per minute speed during a given code snippet, ~~typing session,~~ and the entire lifetime history of the application.

FR26: The application shall track the user's words per minute speed during a given code snippet, ~~typing session,~~ and the entire lifetime history of the application.
FR27: The application shall track the user's lines per minute speed during a given code snippet, ~~typing session,~~ and the entire lifetime history of the application.
FR28: The application shall track the error rate during a given code snippet, ~~typing session,~~ and the lifetime entire history of the application. The error rate can be defined as the ratio between correctly typed characters and all characters that were typed.

3 Non-functional Requirements

3.1 Look and Feel Requirements

NFR1: The typing interface shall respond to a user input within ~~0.1~~10ms.
NFR2: The user interface should be visible ~~in both light and dark terminal backgrounds~~ regardless of the user's terminal theme.
NFR3: The provided theme shall be easy on the eyes and follow the WCAG AA or AAA specification in terms of colour choice.
NFR4: The code snippets chosen should be diverse, and representative of the languages' syntax.
NFR5: The entirety of the user interface should fit within a terminal window sized ~~640x480 pixels or larger, and scale up according on the current terminal window size~~ greater than MAX_COLS in width and MAX_LINES in length.

3.2 Usability and Humanity Requirements

NFR6: The system's typing interface, as well as its cursor indicator, should be intuitive.
NFR7: The system shall be easy to use for anyone with basic knowledge of the console.
NFR8: The instructions will be easily comprehensible by anyone with basic understanding of English
NFR9: The user will only require the keys on a typical 60% keyboard to correctly type all given code.

3.3 Performance Requirements

NFR10: When the user loads the package excluding the first time loading, the time it takes for the package to be ready to accept user input shall not exceed 1 second.
NFR11: Application should be available 99.999% (5 nines) of the time. This translates to 5.26 minutes of downtime in a given year, afforded by user updates of python or lpm, as well as unforeseen circumstances in the CI/CD Pipeline.

3.4 Operational and Environmental Requirements

NFR12: The system shall work on ~~Python 2 and~~ Python 3.6 and above.
NFR13: The system shall work on Linux, macOS, and Windows operating systems.
NFR14: In terms of computer specs, the package shall run on any computer that is able to run ~~Python 2 and~~ Python 3.6+ based on their respective minimum requirements (i.e. if running on Python 3.7, the user's computer shall at least have Python 3.7's minimum requirements to be supported officially by the lpm package).

NFR15: Application should be installable via the pip package manager.

3.5 Maintainability and Support Requirements

NFR16: Application should be easy to update (via pip).
NFR17: It should be easy to add additional code snippets.

3.6 Security Requirements

NFR18: External systems shall not have access to the system.

3.7 Cultural Requirements

NFR19: Code snippets that include harmful, vulgar, controversial, political, or offensive content shall not be displayed in lpm

3.8 Legal Requirements

NFR20: Code snippets shall have a valid open source license or the authors of the code snippets have given explicit permission to the developers to be displayed in lpm.

4 Project Issues

4.1 Open Issues

- Finding open-source code snippets that can be used in our product.
- Storing large quantities of snippets in an effective and efficient manner.
- Creating an aesthetically pleasing interface for individuals with different default console settings.
- Supporting multiple programming languages.

4.2 Off-the-Shelf Solutions

- The python package wpm exists as a current command line solution for improving typing speed. However, this application only allows for typing English-language passages.
- Current solutions for improving typing speed for coding passages include typing.io. However, typing.io is a website which involves online internet connection to use.

4.3 New Problems

Not applicable to our system

4.4 Tasks

Task	Assigned to	Timeline
Requirements Specification	Jessica, Jay, Maanav	Feb 12 2021
POC - Show output on commmandline	Jessica, Jay, Maanav	Feb 22 2021
Initial Typing Functionality	Jessica, Jay, Maanav	Mar 5 2021
Test Plan	Jessica, Jay, Maanav	Mar 5 2021
Design Doc Revision	Jessica, Jay, Maanav	Mar 18 2021
Implement all Functional Requirements	Jessica, Jay, Maanav	Mar 18 2021
Fix Bugs	Jessica, Jay, Maanav	Mar 26 2021
Final Product and Presentation	Jessica, Jay, Maanav	Apr 12 2021

4.5 Migration to the New Product

- Functional requirements will each have their own separate branch and Will each be individually implemented into the application.
- Every new feature will be created on a separate branch and will include sufficient testing to ensure validity. New code will not be added to the main branch unless it has been tested. This will allow us to keep track of features.

4.6 Risks

- Console keyboard shortcuts may cause issues with our code.
- Must ensure that the program can be exited without the key stokes being confused with typing.

4.7 Costs

There should be no costs associated to this project beyond labour costs.

4.8 User Documentation and Training

- The lpm application will include a *lpm -h* command that will display a set of instructions for how to use the application, directly within the terminal.
- The instructions will also be available online through the PyPI and GitHub.

4.9 Waiting Room

- Product will be available in Java, JavaScript, and other languages.

4.10 Ideas for Solutions

- Created a command line entry-point
- Create flags that can be added to lpm command that will let one choose which language to type
- Showing updates regarding typing speed after every code passage is completed

References

James Robertson and Suzanne Robertson. *Volere Requirements Specification Template*. Atlantic Systems Guild Limited, 16 edition, 2012.

5 Appendix

5.1 Symbolic Parameters

- TEXT_COLOR = Terminal Color [252, 235] ~~#FFFFFF~~
- CORRECT_COLOR = Terminal Color [243, 235] ~~#00BFFFF~~
- INCORRECT_COLOR = Terminal Color [9, 88]~~#F9BF3B~~
- MAX_LENGTH ~~MAX_SNIPPET_LENGTH~~ = 20 ~~30~~
- MAX_COL = 80
- MIN_NUM_SNIPPETS = 10