Software Requirements Specification
for
File System Watcher

Version 1.0 Approved

Prepared by Weilan Liang, Aden Abdulahi, and Dereje Teshager

University of Washington

February 03, 2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
File System Watcher	January 31, 2025,	Initial Draft	1.0 draft 1 Approved

1. Introduction

1.1 Purpose

The purpose of this SRS project document is to build a window-based program that allows the user/s to monitor or watch events on files with specific extension types. The information collected will be displayed on the window as file system events occur. The project will be implemented by a team of collaborators. The product will be a python program that allows the users to monitor any un-intended/intended activity/activities on specified file/files. As this is the first draft of the SRS, in the future it may include revisions or release number.

1.2 Document Conventions

This SRS project document will consist of six sections: the introduction, overall description, system features, external interface requirements, other non-functional requirements, and other requirements. The main task in this project will be to develop a menu strip with keyboards, and shortcuts to each menu and its options. Buttons on a button strip/tool
Bar that shortcuts to menu options, and a list box to display information about files that are being watched. Additionally, the project will have some instructions so that the user can choose which type of file extension to watch. Any additional buttons deemed necessary will be included in later revisions.

1.3 Intended Audience and Reading Suggestions

This project will be beneficial to any user, either developer, project manager, marketing staff, users, testers, and documentation writers to ensure, and know who accessed and what type of file had been accessed based on predetermined list of target files.

1.4 Project Scope

The scope of the SRS is:

- *To build software that could watch assigned files, and users will have secured and safe products.*
- *To produce reports, of who accessed and what file had been accessed*

- *To build user friendly file watcher software.*

1.5 References

Not applicable now, but we may update it later.

2. Overall Description

2.1 Product Perspective

The file watcher project specified in this SRS is small scale project and is part of the TCSS504 course. The objective is to ensure that our team members gain the knowledge on how to Ticker, modularization, SQLite and any other programing components required to ensure a functional program.

2.2 Product Features

File watcher project will contain the following buttons:

- *directory to monitor with start and stop buttons*
- *Query or write by extension*
- *Database*
- *File system watcher events*
- *(once a button is clicked the other button will look like a busy (un-clickable).*

2.3 User Classes and Characteristics

Though this program is suitable for a variety of users, but it is more appropriate to:

- *Teachers: To monitor students watching lecture videos, assignments, and assigned readings.*
- *Safety and Security personnel: to monitor who accessed and what they accessed*
- *Business owners: To know what customers are looking for and who are they (age group)*
- *Program designers and software developers: To ensure the safety of their products*

2.4 Operating Environment

The file watcher project program will operate in windows, mac, or Linux etc, and can be installed on laptops, mobile phones and iPad. The program is built using python.

2.5 Design and Implementation Constraints

The design will be presented using UML as a draft and as we are building the program from scratch, it will take some time to present the full design implementation.

2.6 User Documentation

User Manual: A comprehensive guide that provides step-by-step instructions on installing, configuring, and using the File Watcher program. It will include descriptions of key features, use case scenarios, and troubleshooting tips.

2.7 Assumptions and Dependencies

Assumptions:

- *A-1: The system will be installed on a supported operating system machine (e.g., Windows, macOS, or Linux).*
- *A-2: The program will have access to the file system where it is intended to monitor (with necessary read/write permissions).*
- *A-3: End users are assumed to have basic computer literacy and knowledge of how to use the file system, but no advanced programming knowledge.*

Dependencies:

- *D-1: The file watcher will depend on the file system's API to detect file changes.*
- *D-2: The program uses Python Watchdog for file monitoring.*
-

3. System Features

3.1 Directory Monitoring Feature

3.1.1 Description and Priority

Description:
This feature continuously monitors a user-specified directory (and optionally subdirectories) for file system events including creation, deletion, modification, and movement of files. It leverages the watchdog library to capture events in real time.

Priority:
High

3.1.2 Stimulus/Response Sequences

Stimulus: The user inputs a directory path and clicks “Start”

Response: The system validates the path and initiates the watchdog observer on the directory.

Stimulus: A file is modified, created, deleted, or moved.

Response: The system captures the event and passes it to the logging module for recording in the database.

3.1.3 Functional Requirements

REQ-1: The system shall allow the user to input a valid directory path for monitoring.

REQ-2: The system shall monitor the specified directory for file creation, deletion, modification, and movement events.
REQ-3: The system shall use the watchdog library to detect and capture file events in real time.
REQ-4: The system shall allow the user to specify an extension filter to restrict monitoring to files matching a given pattern (if provided).
REQ-5: The system shall offer an option (via a checkbox) to include directories in the monitoring process.
REQ-6: The system shall display error messages for invalid directory paths or inaccessible directories.

3.2 Event Logging and Database Feature

3.2.1 Description and priority

Description:
This feature logs all captured file system events into an SQLite database. Each event record will include details such as the event type, file path, timestamp, and any added metadata required for future queries or audits.

Priority:
High

3.2.2 Stimulus/Response Sequences

Stimulus: A file system event is detected by the monitoring feature.
Response: The system logs the event into the SQLite database.

Stimulus: The user clicks the “Query” button to review logged events.
Response: The system retrieves and displays the relevant records from the database.

3.2.3 Functional Requirements

REQ-7: The system shall create and maintain an SQLite database to store file system event records.
REQ-8: the system shall record each event with a unique identifier, timestamp, event type, and file path.
REQ-9: The system shall provide a “Clear DB” functionality to delete existing records from the database.
REQ-10: The system shall allow the user to select a custom database file location through the GUI.
REQ-11: The system shall support querying logged events by file extension, event type, or time range.
REQ-12: The system shall handle database errors gracefully and inform the user of any issues.

3.3 User Interface and Control Feature

3.3.1 Description and Priority

Description:

This feature provides an intuitive graphical user interface (GUI) built with Tkinter. It enables the user to configure monitoring settings, start/stop the file system watcher, specify the database location, and manage logging options.

Priority:

Medium

3.3.2 Stimulus/Response Sequences

Stimulus: The user enters parameters (e.g., directory path, file extenstion, database path) and clicks "Start"

Response: The system initiates the monitoring process and updates the UI to reflect the active status (By grading out the Start button).

Stimulus: The user interacts with control buttons ("Stop", "Browse", "Write", "Clear DB", "Query").

Response: The system executes the corresponding operations and provides real-time feedback or error messages as necessary.

3.3.3 Functional Requirements

REQ-13: The system shall provide text fields for entering the directory path and database location.

REQ-14: The system shall offer comboboxes for selecting file extensions and filtering event types.

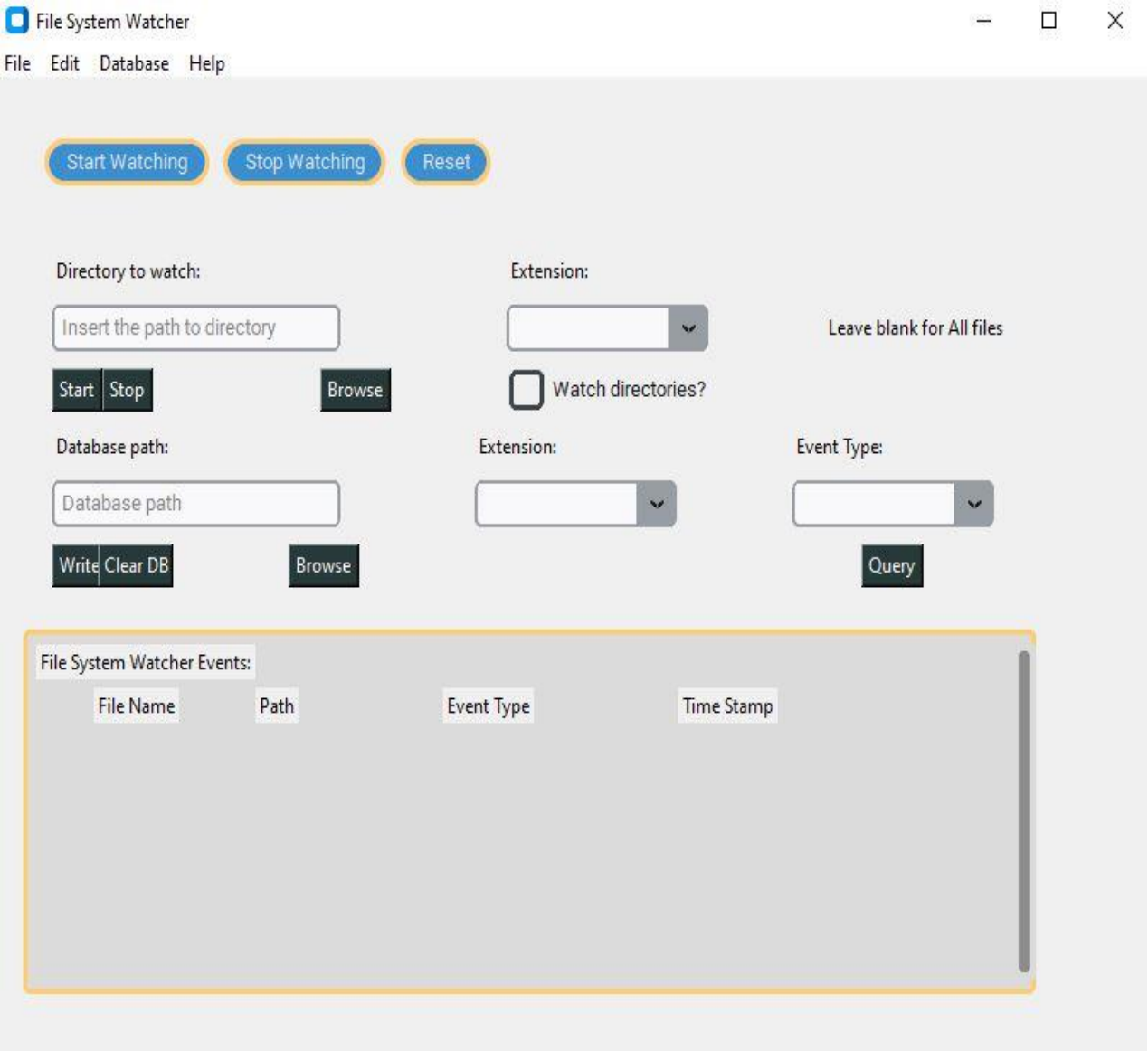
REQ-15: The system shall include “Start” and “Stop” buttons to control the monitoring process.

REQ-16: The system shall include “Browse” buttons for directory and database selection using file dialog.

REQ-17: The system shall display real-time status updates and error messages in a designated area of the GUI.

4. External Interface Requirements

4.1 User Interfaces



The File System Watcher will feature a modern, user-friendly graphical user interface (GUI) built using Tkinter, ttk and customtkinter to conform to contemporary desktop design standards. Key characteristics include:

- *Consistent Layout: The application window will be divided into clearly labeled frames (e.g., Menu, Watch Settings, Database) with standardized padding, fonts, and colors.*

- *Standard Controls: The GUI will include buttons, text entry fields, comboboxes, and checkboxes, all following established conventions (e.g., “Start”, “Stop”, “Browse”, and “Query” buttons).*

- *Screen Layout Constraints: The interface will be designed to be responsive, with controls adjusting to window re sizing. Standard error messages and help prompts will appear as pop-up dialog.*

- *Keyboard Shortcuts: Frequently used functions (such as starting/stopping monitoring or opening file dialogs) may have keyboard shortcuts to enhance usability.*

4.2 Hardware Interfaces

This application is designed to run on standard desktop and laptop computers. It interfaces with:

- *Storage Devices: Interface with local hard drives, SSDs, and network-attached storage devices through the operating system’s file system APIs.*

- *Input Devices: Standard keyboard and mouse for user interaction.*

- *No Specialized Hardware: There are no requirements for dedicated hardware interfaces beyond what is available on typical consumer systems.*

4.3 Software Interfaces

The File System Watcher will integrate with several software components:

- *Operating System: Compatible with Windows, macOS, and major Linux distributions.*

- *Python 3.x Environment: Requires a Python 3.x runtime with Tkinter and Watchdog installed.*

- *Tkinter and ttk: For building the graphical user interface.*

- *Watchdog Library: Used for monitoring file system events.*

- *SQLite Database: Utilizes Python's built-in sqlite3 module for data storage.*

- *APIs and Protocols: Interface with the system through standard OS APIs, Data exchanged between modules is in the form of Python objects and SQLite records.*
- *Inter-Module Communication: The GUI interacts with the monitoring module through function calls and event callbacks; logged events are communicated to the database module for storage.*

4.4 Communications Interfaces

The File System Watcher is primarily a desktop application with no mandatory external network communication. However:

- *Optional Network Features: Future release might include capabilities for remote logging or notifications, which would involve standard protocols such as HTTP or FTP.*

- *Local Inter-Process Communication: Currently, all communications occur within the local application process. Any external communications (e.g., sending email alerts) will use secure, encrypted channels and follow industry standards.*

- *Data Transfer and Synchronization: Since the application operates locally, data transfer is limited to interactions between the application and the local SQLite database. Should network functionality be added, expected data transfer rates and synchronization mechanisms will be specified in later revisions of the SRS.*

5. Other Nonfunctional Requirements

5.1 Performance Requirements

PP-1: The *system must detect file changes within 5 seconds of the event occurring.*
PP-2: The file watcher should be capable of monitoring up to 100,000 files simultaneously without significant reducing performance

5.2 Safety Requirements

SR-1 The program should not monitor, delete, or corrupt any files while monitoring. It must ensure that files being watched will not be locked or altered anytime due to program running.

5.3 Security Requirements

Not Applicable

5.4 Software Quality Attributes

Correctness: The program must detect, monitor, and report accurate changes in files it is monitoring.
Maintainability: The code must follow industry-standard practices to ensure ease of future updates and bug fixes.
Usability: The program should be easy to use and understand.

6. Other Requirements

Database Requirements: For logs or stores metadata about file changes, it should use a lightweight database such as SQLite to store all information.
Legal requirements: The program must comply with all local privacy laws.

Appendix A: Glossary

File Watcher: A program that monitors specified files or directories for changes, such as modifications, deletions, or new files.

Watch List: A list of files or directories being monitored by the File Watcher program.

Trigger Action: An action (such as sending an email, running a script, or generating a log) that occurs when a file change event is detected.

Event: A change in the file system that the File Watcher detects, such as a file being created, modified, or deleted.

Appendix B: Analysis Models

UML will be updated in later versions of the SRS.

Appendix C: Issues List

Currently not applicable, but in the future we will update the next version.