Software Requirements Specification

for

File watcher

Version 1.0 approved

Prepared by Liam Wilkenson,

Deepika Chinnathambi

<organization>

February 4,2025

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The File Watcher Software is designed to monitor file system changes in real time. It detects file modifications, creations, and deletions in specified directories and logs these changes into an SQLite database. This system is useful for tracking changes in sensitive folders, automating backup processes, or monitoring unauthorized file access.

1.2 Scope

The software will be developed using Python and will run on multiple operating systems. It will support:

- Real-time monitoring of directories
- Logging of file changes into an SQLite database
- Notification or alert system for specific file modifications (optional for extra credit)
- A simple command-line or graphical interface to view logs (optional for extra credit)

1.3 Definitions, Acronyms, and Abbreviations

- **SRS**: Software Requirements Specification
- **SQLite**: A lightweight, embedded database for storing logs
- **GUI**: Graphical User Interface
- **CLI**: Command-Line Interface

1.4 References

- SQLite official documentation
- Python watchdog library

1.5 Overview

This document describes the functional and non-functional requirements of the File Watcher Software. It outlines the system's capabilities, design constraints, and quality attributes.

2. Overall Description

2.1 Product Perspective

The File Watcher Software functions as an independent monitoring tool. It can be integrated into security systems, backup solutions, or DevOps workflows.

2.2 Product Features

- Detects file modifications, creations, and deletions.
- Logs file changes to an SQLite database.
- *(Optional)* Sends notifications for specific file events.
- *(Optional)* Provides a user-friendly interface for viewing logs.

2.3 User Classes and Characteristics

- **Administrators**: Configure monitoring paths and view logs.
- **Developers**: Integrate the software into existing workflows.
- **Security Teams**: Use logs to detect unauthorized file access.

2.4 Operating Environment

- OS: Windows, macOS, Linux
- Python 3.x
- Required Libraries: watchdog, sqlite3, logging

2.5 Design and Implementation Constraints

- Must efficiently handle large directories without significant performance overhead.
- SQLite database size should be optimized to prevent excessive storage use.
- Limited GUI support (if implemented, it should be lightweight).

2.6 User Documentation

A user guide will be provided explaining:

- Installation steps
- Configuration options
- How to view logs and interpret results

2.7 Assumptions and Dependencies

- Users have Python installed.
- Users have the required permissions to monitor specified directories.
- The software depends on third-party libraries like watchdog and sqlite3.

---

3. Specific Requirements

3.1 Functional Requirements

- Detect and log file changes (create, modify, delete).
- Store logs with timestamps and file paths in SQLite.
- Provide a method to retrieve logs.
- *(Optional)* Alert users when specified files change.

3.2 Performance Requirements

- Monitor at least 100,000 files without excessive CPU/memory usage.
- Log file changes with minimal delay (<1 second for small files).

3.3 Security Requirements

- Ensure that logs are not accessible by unauthorized users.
- Protect against malicious file modifications (if applicable).
- If flagged as potentially dangerous behavior, alert security team (email) within 60s.

3.4 Software Quality Attributes

- **Reliability**: Should run continuously without crashing.
- **Usability**: CLI should be user-friendly; GUI (if included) should be simple.
- **Maintainability**: Code should be modular and well-documented.
- **Scalability**: Should support monitoring multiple directories.

---

4. External Interface Requirements

4.1 User Interfaces

- CLI with commands to start/stop monitoring and view logs.
- *(Optional)* Simple GUI for viewing logs.

4.2 Hardware Interfaces

- No specialized hardware required.

4.3 Software Interfaces

- Python libraries: watchdog, sqlite3, logging.
- SQLite database for log storage.

4.4 Communication Interfaces

- If network notifications are added, software may require internet access.

---

5. Other Non-functional Requirements

5.1 Performance

- Should not consume more than 5% CPU usage under normal operation.

5.2 Security

- Ensure logs are encrypted (if applicable).

5.3 Maintainability

- Code should be well-structured for future enhancements.

5.4 Portability

- Should run on Windows, macOS, and Linux without modification.

6. Other Requirements

- The SQLite database should support indexing for efficient querying.
- The system should provide logging mechanisms to troubleshoot failures.
- Future expansion should allow integration with cloud storage for backup.

Appendix A: Glossary

- File System Monitoring: The process of tracking file changes in a directory.
- Event Listener: A software component that listens for file system changes.
- Log File: A file where monitored events are recorded.
- Real-time Monitoring: Continuous tracking of file events without delay.

Appendix B: Analysis Models

- Class Diagram: A UML class diagram illustrating key components such as the file watcher, event handler, and database logger.
- State-Transition Diagram: A representation of how the system moves between different monitoring states (Idle, Active, Error Handling).
- Entity-Relationship Diagram (ERD): Showcasing database schema and relationships between tables (e.g., Events, Files, Users).

Appendix C: Issues List

- Pending Decisions:
  - Whether a GUI should be included in the base implementation.
  - The frequency of database cleanup to avoid excessive storage use.
- Open Issues:
  - Determining optimal performance tuning for large-scale directory monitoring.
  - Handling edge cases like symbolic links and network file systems.
  - Implementing user authentication for secure log access.