Amrita School of Computing

23CSE311 Software Engineering

Lab Worksheet 2:

Software Requirements Specification

Date:

CO2	Apply requirement engineering principles to analyze, model, and validate requirements for effective software solutions.

Objective:

To clearly document all functional and non-functional requirements of the system, ensuring a shared understanding among stakeholders and serving as a reference for design, development, testing, and maintenance.

Team Information:

Team Member Name	Student ID	Role in Project
Ajith Ashok	AM.SC.U4CSE23204	UI Development
Arjun Rajesh	AM.SC.U4CSE23208	Team Lead (Feature Implementation & Software Integration)
Yadhu Vipin	AM.SC.U4CSE23270	Co-Lead (Feature Implementation & Software Integration)

1. Introduction

This document provides a detailed Software Requirements Specification (SRS) for the **WinKnight** project. WinKnight is a C# Windows application designed as an automated, self-healing system recovery and optimization tool.

1.1 Purpose

The purpose of WinKnight is to provide average Windows PC users with a single, automated tool to maintain system stability and performance. It aims to proactively prevent system issues by creating restore points, automatically diagnose and repair problems using built-in Windows utilities, optimize the system by cleaning caches, and give users control over Windows Updates. The system connects a user-friendly interface with powerful backend system commands, making advanced PC maintenance accessible to non-technical users.

System Functionalities:

- Automate the creation of system restore points before critical events.
- Automatically diagnose system file integrity and perform repairs.
- Allow users to clean system cache and temporary files with one click.
- Provide a simple interface to pause, resume, and manage Windows Updates.
- Display overall system health and scan reports on a central dashboard.

1.2 Document Conventions

- **SFC – System File Checker**
- **DISM – Deployment Image Servicing and Management**
- **WPF – Windows Presentation Foundation**
- **WinUI – Windows UI Library**
- **API – Application Programming Interface**
- **UI – User Interface**
- **OS – Operating System**
- **WMI – Windows Management Instrumentation**

1.3 Intended Audience and Reading Suggestions

- **Development Team** – To understand the system's requirements, constraints, and features for implementation.
- **QA Testers** – To create test cases for validating that the application meets the specified functional and non-functional requirements.
- **Project Stakeholders (e.g., IT Support Teams)** – To understand the full scope and capabilities of the software as a diagnostic and maintenance tool.

1.4 Product Scope

WinKnight is a desktop application for the Windows operating system that will:

- Automate the creation of system restore points before critical events (**RestoreGuard**).
- Proactively monitor the system for errors and automatically run Windows repair tools like SFC and DISM (**SelfHeal**).
- Clean unnecessary system caches and temporary files to optimize performance (**CacheCleaner**).
- Provide users with direct control to pause, resume, and schedule Windows Updates (**UpdateStopper**).
- Present all functionality within a single, graphical user interface that displays system status and reports.

1.5 References

- Microsoft Windows API Documentation.
- Microsoft .NET Framework Documentation.
- Official documentation for Windows command-line utilities (SFC.exe, DISM.exe).

2. Overall Description

This section provides a general overview of the software's functionality.

2.1 Product Perspective

WinKnight is a standalone, self-contained desktop application designed for Windows 10 and 11. It serves as an intelligent front-end that automates and simplifies the use of existing, powerful Windows system utilities. It does not replace tools like System Restore, SFC, or DISM, but rather integrates them into a proactive, automated maintenance cycle, making them accessible and useful for the average user.

2.2 Product Features

- **RestoreGuard:** Proactively creates system restore points on a schedule or before critical events like Windows Updates to ensure a safe recovery point.
- **SelfHeal:** Automatically scans the Windows Event Log for critical errors and runs diagnostics using SFC and DISM to check for and repair system file corruption.
- **CacheCleaner:** Frees up disk space and can improve system responsiveness by clearing temporary files from user and system cache folders.
- **UpdateStopper:** Grants users control over the Windows Update service, allowing them to temporarily disable automatic updates to prevent unwanted restarts or potential stability issues.
- **Unified Dashboard:** A central UI that allows users to initiate manual scans, view the status of automated tasks, and read simple, clear reports on the system's health.

2.3 User Classes and Characteristics

- **General Windows Users:** The **primary users** of the application. They typically have basic computer skills but lack the technical expertise for advanced system maintenance. They require a simple, intuitive, and automated "one-click" solution to keep their PCs running smoothly.
- **IT Support Technicians:** A **secondary class of users**. They are technically proficient and can use WinKnight as a rapid first-response tool for diagnosing and resolving common system issues, saving time on manual command-line operations.

2.4 Operating Environment

- **Operating System:** Windows 10 / Windows 11
- **Framework:** .NET Framework
- **Development Environment:** Visual Studio 2022+
- **Hardware:** A standard PC capable of running Windows 10 or 11.

2.5 Design and Implementation Constraints

- The application must be developed in C# using either WPF or WinUI 3 for the user interface.
- The application requires administrator privileges to access system-level APIs and execute repair tools.
- The functionality is dependent on the behavior and availability of built-in Windows command-line utilities (SFC, DISM) and APIs (WMI).
- The initial version must be developed within the project timeline of approximately 11 weeks.

2.6 Assumptions and Dependencies

- Assumption: Users will have administrator rights on their machine to install and run the application.
- Assumption: The target Windows OS has functional, non-corrupted versions of SFC.exe and DISM.exe.
- Dependency: The application is dependent on the .NET Framework being installed on the user's machine.
- Dependency: The stability and functionality of the tool are dependent on the consistency of the Windows APIs and command-line interfaces across different Windows updates.

---

3. Specific Requirements

This section defines the detailed requirements for WinKnight.

3.1 Functional Requirements

FR-1: RestoreGuard Module

- FR-1.1: The system shall be able to create a system restore point manually when triggered by the user from the UI.
- FR-1.2: The system shall be configurable to automatically create a system restore point on a user-defined schedule (e.g., weekly).
- FR-1.3: The system shall monitor for and attempt to create a restore point immediately before a Windows Update is initiated.

FR-2: SelfHeal Module

- FR-2.1: The system shall scan the Windows Event Log for critical system errors.
- FR-2.2: Upon detecting critical errors or a user trigger, the system shall execute the System File Checker (sfc /scannow) to verify the integrity of system files.
- FR-2.3: The system shall execute DISM (/Online /Cleanup-Image /CheckHealth) to check the health of the Windows component store.
- FR-2.4: If SFC or DISM detects corruption, the system shall automatically attempt a repair using the appropriate commands (e.g., DISM /Online /Cleanup-Image /RestoreHealth).

FR-3: CacheCleaner Module

- FR-3.1: The system shall identify and calculate the space used by temporary files in standard locations (%temp%, C:\Windows\Temp, C:\Windows\Prefetch).
- FR-3.2: The system shall allow the user to clear these caches with a single click.

FR-4: UpdateStopper Module

- FR-4.1: The system must provide a clear UI option to enable or disable the Windows Update service.
- FR-4.2: The system must allow users to disable updates for a scheduled duration (e.g., 7 days, 30 days), after which updates will be automatically re-enabled.
- FR-4.3: The UI must clearly display the current status of the Windows Update service (e.g., "Active," "Paused until [Date]").

FR-5: User Interface

- FR-5.1: The UI shall have a main dashboard displaying the overall system health status.
- FR-5.2: The UI shall provide a button to trigger a full manual scan (SelfHeal and CacheCleaner).
- FR-5.3: The system shall display the live status/progress of any ongoing scan or repair operation.
- FR-5.4: Upon completion, the system shall present a simple, non-technical report summarizing the actions taken and the results.

3.2 External Interface Requirements

- User Interfaces: The application will feature a graphical user interface (GUI) built with WPF or WinUI 3. The design will be intuitive, requiring minimal user interaction to perform core tasks.
- Hardware Interfaces: The application has no direct hardware interfaces but will interact indirectly with the system's storage drive to read/write files and create restore points.

- **Software Interfaces:**
    - Windows OS: Interfaces with the Windows Management Instrumentation (WMI) to manage system restore points and the Windows Update service.
    - Windows Event Log: Interfaces with the Event Log API to read system error data.
    - Command-Line Tools: Interfaces with SFC.exe and DISM.exe via the System.Diagnostics.Process class to execute scans and repairs.

3.3 Performance Requirements

- PR-1: Background monitoring processes should consume minimal system resources (e.g., < 5% CPU on an idle system).
- PR-2: An active scan or repair process should not render the user's system unusable.
- PR-3: The UI must remain responsive during backend operations.

3.4 Security Requirements

- SEC-1: The application must run with administrator privileges, and it must properly request these privileges upon launch using a UAC prompt.
- SEC-2: The application must not modify system files or settings outside of its documented scope (SFC/DISM repairs, cache cleaning, Windows Update service).
- SEC-3: The UpdateStopper module must warn users about the potential security risks of disabling updates.

3.5 Software Quality Attributes

- Reliability: The application must execute repair operations robustly. Failed operations should be handled gracefully and reported to the user without causing system instability.
- Usability: The UI must be clean, simple, and easy for non-technical users to understand. Reports and statuses should avoid technical jargon.
- Maintainability: The software will be architected into distinct modules (RestoreGuard, SelfHeal, etc.) to facilitate easier updates, bug fixes, and feature additions.

3.6 Other Non-Functional Requirements

- Compatibility: The application must be fully functional on standard installations of Windows 10 and Windows 11.
- Safety: Critical operations, such as running a repair, must include safeguards and clear logging to mitigate the risk of unintended data loss.

Instructor’s Signature & Comments (For Lab Use):