**COIMBATORE INSTITUTE OF TECHNOLOGY**

**DEPARTMENT OF AI AND DS**

**HOME APPLIANCE CONTROL SYSTEM**

**SOFTWARE REQUIREMENTS SPECIFICATION**

*DATE:*23-08-2025

*DOCUMENT ID:*

*S.KANISHA 2403717624322023*

*S.PRATHEESHA 2403717624322037*

*G.PRIYA 2403717624322038*

*M.SARANYAA 2403717624322048*

*S.KESHIKA 2503717624322301*

**CONTENTS:**

1. **Introduction**  
     1.1 Document Purpose  
     1.2 About the Project  
     1.3 Document Scope  
     1.4 Terminology Used  
     1.5 Related Documents  
     1.6 Document Overview
2. **Product Overview**  
     2.1 Introduction  
     2.2 Business Services Supported  
     2.3 Product Characteristics  
       2.3.1 User Interface  
       2.3.2 Operating Environment  
       2.3.3 Hardware Interfaces  
     2.4 User Characteristics  
       2.4.1 General User Characteristics  
       2.4.2 Admin Characteristics  
       2.4.3 Maintenance Personnel Characteristics  
     2.5 General Constraints  
     2.6 Priority of Requirements
3. **Functional Requirements**  
     3.1 HACS Authentication System  
     3.2 HACS Appliance Controller List  
     3.3 HACS Appliance Operation Control  
     3.4 Functional Requirements for Specific Appliances
4. **User Interface Requirements**  
     4.1 Look and Feel Requirements  
     4.2 Usability Requirements
5. **Non-Functional Requirements**  
     5.1 Performance Requirements  
     5.2 Reliability Requirements  
     5.3 Security Requirements  
     5.4 Maintainability Requirements  
     5.5 Scalability Requirements  
     5.6 Portability Requirements
6. **Operating Environment**  
     6.1 Hardware  
     6.2 Software  
     6.3 Printer  
     6.4 External Data Storage
7. **Acceptance Criteria**
8. **References**
9. **Glossary**

[**GLOSSARY** **14**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250009)

[**INDEX** **15**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250008)

[***PURPOSE OF APPENDICES*** **15**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250007)

[**APPENDIX A: EXTERNAL CONTEXT MODEL** **16**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250006)

[**APPENDIX B: INTERNAL CONTEXT MODEL** **16**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250005)

[**APPENDIX C: USE CASE ANALYSIS NOTES** **16**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250004)

[**APPENDIX D: DEVELOPMENT PROCESS** **16**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250003)

[**APPENDIX E: CONTRIBUTIONS** **17**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250002)

[**APPENDIX F: MEETING AGENDAS/MINUTES** **17**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250001)

[**APPENDIX G: PROBLEM INVESTIGATION REPORTS** **18**](file:///C:\Users\prabh\AppData\Local\Microsoft\Windows\INetCache\IE\YQQFG714\SE%5b1%5d.docx#_TOC_250000)

1. INTRODUCTION

1.1 PURPOSE

The purpose of this Software Requirements Specification (SRS) is to describe in detail the design, development, and testing of the Smart Vacuum Cleaner module in the Home Appliance Control System (HACS). This document serves as a foundation for developers, testers, project managers, and other stakeholders to understand the complete system.

The Smart Vacuum Cleaner was chosen because it is a relatively advanced household appliance compared to devices such as light bulbs or fans. While a bulb only requires ON/OFF control and a fan might require speed control, the vacuum cleaner demonstrates a richer set of functionalities. It can be started, stopped, paused, resumed, scheduled, cancelled, and it can generate automatic alerts based on internal sensor readings. The complexity of the vacuum cleaner makes it an ideal appliance to demonstrate the principles of software engineering, requirements gathering, system design, and project management.

Another purpose of this document is to capture the contributions of all team members. Unlike an individual project, this work reflects the effort of a team of five members, each holding a specific role. This division of responsibility makes the project more realistic, as in industry, software projects are never developed by a single person. The Project Manager, Requirements Engineer, System Designer, Developer, and Tester all had their own tasks, and this SRS reflects how their contributions combined to create a functional and reliable solution.

This document is not just a list of requirements. It is a complete explanation of the system, written in such a way that even a non-technical reader can understand the goals, scope, functions, and importance of the Smart Vacuum Cleaner within HACS. For technical members of the team, it serves as a guide and reference document to ensure consistency across design, implementation, and testing.

1.2 SCOPE

The scope of the Home Appliance Control System (HACS) is to centralize the management of household devices. Instead of users manually interacting with each appliance, HACS allows all devices to be accessed and controlled from a single interface. This could be a mobile application, a web-based control panel, or even voice assistants.

The Smart Vacuum Cleaner module within HACS is an excellent representation of the system’s capabilities. It is designed to provide:

Remote control: Users can operate the appliance without physical presence.

Scheduling: Cleaning tasks can be automated at specific times, improving user convenience.

Operational control: The appliance can be paused, resumed, or cancelled mid-operation.

Alerts and notifications: The system provides real-time alerts such as low battery and full dustbin, enabling users to take necessary actions.

The scope also covers the software engineering lifecycle applied in this project:

Requirements analysis: Collecting and documenting what the system must do.

System design: Creating diagrams and models that show how the system will function.

Implementation: Building the features and ensuring they work correctly.

Testing: Verifying that the system meets all requirements.

Project management: Ensuring timely delivery, risk management, and effective teamwork.

This wide scope ensures that the project does not only produce a working system but also demonstrates the academic and practical aspects of developing a software system.

1.3 DEFINITIONS, ACRONYMS AND ABBREVIATIONS

HACS: Home Appliance Control System, the central platform controlling all devices.

SRS: Software Requirements Specification, a document that defines system requirements.

FR: Functional Requirement, describing what the system must do.

NFR: Non-Functional Requirement, describing qualities such as performance and reliability.

DFD: Data Flow Diagram, used to represent flow of information.

UI: User Interface, the part of the system through which users interact.

These terms will be used throughout this document.

1.4 REFERENCES

1. IEEE Standard 830-1998: Guidelines for writing software requirements.

2. Real-world smart vacuum cleaner manuals (iRobot Roomba, Xiaomi, etc.).

3. Academic papers on home automation and IoT-based systems.

4. Notes from project team meetings and brainstorming sessions.

1.5 OVERVIEW

This document is organized into multiple sections:

Section 2 describes the overall system and its context.

Section 3 details the system requirements, including functional and non-functional needs.

Section 4 describes the system design with diagrams and explanations.

Section 5 explains implementation details, contributed by the Developer.

Section 6 focuses on testing approaches and test cases, contributed by the Tester.

Section 7 presents project management contributions such as planning and risk handling.

Section 8 concludes with reflections and future scope.

The structure ensures that each team member’s contribution is visible and connected to the overall system.

2. OVERALL DESCRIPTION

2.1 PRODUCT PERSPECTIVE

The Smart Vacuum Cleaner is a subsystem of the Home Appliance Control System (HACS). HACS acts as the central hub through which all appliances in the household can be monitored and controlled. While devices such as fans and lights may offer only simple ON/OFF control, the Smart Vacuum Cleaner represents a more advanced appliance with multiple modes, real-time feedback, and automated scheduling capabilities.

In terms of architecture, the Smart Vacuum Cleaner does not exist in isolation. It communicates with the HACS server, which acts as an intermediary between the user and the appliance. The user interacts with HACS using a mobile phone or web interface, and HACS translates these commands into instructions that the vacuum cleaner can understand. Similarly, the vacuum cleaner sends responses or alerts back to HACS, which are then presented to the user.

The Smart Vacuum Cleaner relies on multiple sensors such as:

Battery sensor: Detects when the battery is low.

Dustbin sensor: Detects when the dustbin is full.

Obstacle sensor: Detects obstacles in cleaning path (for advanced versions).

By integrating these sensors with HACS, the user is provided with a seamless experience that extends beyond simple control. The Smart Vacuum Cleaner therefore acts as both a client (receiving commands) and a server (sending information back).

2.2 PRODUCT FUNCTIONS

The Smart Vacuum Cleaner provides the following functions, each of which is essential for user convenience and automation:

1. Start Cleaning

The user can start the vacuum cleaner from the HACS interface. This command initializes the cleaning process and the vacuum begins moving according to its programmed patterns.

2. Stop Cleaning

When the user chooses to stop, the vacuum halts all operations immediately. This is important in cases where cleaning is no longer required or unexpected guests arrive.

3. Scheduling

Scheduling is one of the most important functions. A user can set a specific time for the vacuum to start cleaning, such as 9 AM daily. The system stores the schedule in its database and automatically triggers the vacuum at the right time.

4. Pause and Resume

Sometimes, cleaning may need to be temporarily paused, for example, if a child is playing in the room. The pause function stops cleaning without cancelling it. Resume continues the operation from where it stopped.

5. Cancel Operation

Cancelling is different from stopping or pausing. When cancelled, the current task is abandoned completely, and the system resets to its default state.

6. Low Battery Alerts

When the vacuum’s battery drops below a threshold (for example, 20%), the system sends a notification to the user. The vacuum may also return to its charging dock automatically.

7. Dustbin Full Alerts

When the dustbin is full, cleaning cannot continue. The system alerts the user so that the bin can be emptied.

Each of these functions contributes to making the Smart Vacuum Cleaner practical, autonomous, and user-friendly.

2.3 USER CHARACTERISTICS

The Smart Vacuum Cleaner is expected to be used by a wide variety of users. Therefore, the system design must take into account different levels of technical expertise:

General Users: These are homeowners who may not have technical knowledge. They require simple buttons such as “Start”, “Stop”, and “Schedule”. The system should provide clear feedback such as “Vacuum started successfully” or “Battery is low”.

Advanced Users: These users may want more control, such as configuring schedules for different days or setting advanced cleaning modes.

Administrators: In some households, one person may manage the system settings. This user may configure default schedules, update appliance software, or connect new devices.

The system must be designed for ease of use, with minimal training required. The UI should avoid technical jargon and instead use clear icons, labels, and prompts.

2.4 CONSTRAINTS

The system must operate within certain limitations:

1. Connectivity Constraint

The Smart Vacuum Cleaner must remain connected to HACS for remote operation. If Wi-Fi or Bluetooth is disconnected, commands will not reach the device.

2. Performance Constraint

The system must respond to commands within 3 seconds. Any longer delay may frustrate users.

3. Scheduling Accuracy

Scheduled cleaning must be accurate to the minute. A cleaning task set for 9:00 AM should not start at 9:05 AM.

4. Hardware Constraint

The vacuum cleaner hardware must include sensors for dustbin status and battery level.

5. Power Constraint

The vacuum requires adequate charging. If the battery is too low, scheduled tasks cannot run.

2.5 ASSUMPTIONSAND DEPENDENCIES

Assumptions:

Users have access to smartphones or panels to operate the system.

The vacuum cleaner supports integration with HACS.

Sensors are reliable and calibrated.

Dependencies:

The system depends on stable Wi-Fi connectivity.

It depends on external servers (if cloud storage is used).

It depends on battery charging stations being available.

3. SYSTEM FEATURES(REQUIREMENTS ENGINEER CONTRIBUTION)

The Requirements Engineer is responsible for identifying, analyzing, and documenting the needs of the system. In the case of the Smart Vacuum Cleaner within HACS, this means carefully defining what the system must do (functional requirements), how well it must do it (non-functional requirements), and the different scenarios in which users will interact with it (use cases).

The main objective of this section is to ensure that there is no ambiguity in the system definition. Every function and behavior should be documented so that designers, developers, and testers all have a common understanding.

3.1 FUNCTIONAL REQUIREMENTS

Functional requirements are the backbone of the system. They describe what the system should do in terms of features and services. Below are the functional requirements of the Smart Vacuum Cleaner, along with detailed explanations.

FR1: Start the Vacuum Cleaner

The system shall allow the user to start the vacuum cleaner.

Explanation: This is the most basic requirement. The user, from the HACS interface, presses a “Start” button. The system then sends a command to the vacuum cleaner. Once the vacuum receives the signal, it activates its motors and begins cleaning.

Importance: This requirement is mandatory, as it represents the fundamental purpose of the appliance. Without it, the system would have no practical use.

FR2: Stop the Vacuum Cleaner

The system shall allow the user to stop the vacuum cleaner.

Explanation: At any point during operation, the user may decide to stop the vacuum. This might be because the cleaning task has been completed, or the user requires silence. Once the “Stop” button is pressed, the vacuum halts immediately.

Importance: This ensures user control and safety. A cleaning device must always allow the user to stop it quickly.

FR3: Scheduling of Cleaning Tasks

The system shall allow users to schedule cleaning operations at specific times.

Explanation: This feature is critical for convenience. A busy user may want the vacuum to run at 9 AM every day, or only on weekends. The schedule is saved in the HACS database and automatically triggers the vacuum.

Example: A working professional may schedule cleaning at 10 AM while they are at the office. By the time they return home, the cleaning is done.

Importance: Scheduling demonstrates automation and is a key feature distinguishing smart devices from traditional appliances.

FR4: Pause Operation

The system shall allow the user to pause cleaning operations.

Explanation: Sometimes, the user may not want to stop the entire task but may need a short break. For example, if a baby is sleeping or someone is on an important phone call. The pause feature halts cleaning temporarily.

Importance: Provides flexibility and convenience without discarding the task.

FR5: Resume Operation

The system shall allow the user to resume cleaning after it has been paused.

Explanation: When the pause condition ends, the user can resume the operation. The vacuum will continue from the point where it left off, instead of starting over.

Importance: Enhances efficiency and reduces unnecessary rework.

FR6: Cancel Operation

The system shall allow the user to cancel cleaning operations.

Explanation: Cancelling is different from pausing or stopping. Cancel means the task is discarded. If a scheduled operation was running, cancelling it will delete it and reset the vacuum to idle state.

Example: A user scheduled cleaning for 10 AM, but an unexpected meeting is at home. They cancel the operation so the vacuum does not start at all.

FR7: Low Battery Alerts

The system shall send an alert when the vacuum’s battery is low.

Explanation: If the battery drops below a threshold (e.g., 20%), the system sends a message: “Battery Low. Please charge.” In some designs, the vacuum automatically returns to the charging dock.

Importance: Prevents sudden shutdowns and ensures smooth operation.

FR8: Dustbin Full Alerts

The system shall notify the user when the dustbin is full.

Explanation: A full dustbin prevents further cleaning. The system must detect this condition and alert the user: “Dustbin Full. Please empty.”

Importance: Maintains cleaning effectiveness and prevents overload on the device.

3.2 NON FUNCTIONAL REQUIREMENTS

Non-functional requirements describe how the system should behave in terms of performance, usability, security, and reliability.

NFR1: Performance Requirement

The system must respond to user commands within 3 seconds.

This ensures that users do not experience delays. For instance, if the user presses “Stop,” the vacuum should not take 10 seconds to halt.

NFR2: Usability Requirement

The system should have a simple, intuitive user interface.

Buttons should be clearly labeled (Start, Stop, Pause).

Icons should represent operations (a pause icon, a dustbin icon).

NFR3: Reliability Requirement

The system should work 99% of the time without failure.

Scheduled tasks must run at the correct time.

Alerts must always be generated when necessary.

NFR4: Scalability Requirement

The system should support future appliances.

While this project focuses on the Smart Vacuum Cleaner, the same platform should be extendable to fans, lights, air conditioners, etc.

NFR5: Security Requirement

The system must ensure that only authorized users can operate the vacuum cleaner.

Unauthorized access could cause safety issues or inconvenience.

3.3 USE CASE DESCRIPTIONS

Use cases describe real-world scenarios of how users interact with the system.

Use Case 1: Schedule Cleaning

Actors: User, System

Precondition: User is logged into HACS.

Steps:

1. User selects vacuum cleaner.

2. User chooses “Schedule” option.

3. User enters time (e.g., 9:00 AM).

4. System saves schedule.

5. At 9:00 AM, vacuum starts automatically.

Postcondition: The vacuum completes the scheduled cleaning task.

Exceptions: If the battery is low, the schedule is delayed until after charging.

Use Case 2: Pause and Resume

Actors: User, System

Precondition: Vacuum is running.

Steps:

1. User presses “Pause”.

2. Vacuum halts immediately.

3. Later, user presses “Resume”.

4. Vacuum continues from the same position.

Postcondition: The cleaning task is completed.

Use Case 3: Low Battery Alert

Actors: System, User

Precondition: Vacuum battery < 20%.

Steps:

1. Vacuum detects low battery.

2. Sends signal to HACS.

3. HACS shows alert to user.

Postcondition: User charges the vacuum, or vacuum docks automatically.

Use Case 4: Dustbin Full Alert

Actors: System, User

Precondition: Dustbin capacity reaches maximum.

Steps:

1. Sensor detects dustbin is full.

2. Vacuum stops cleaning.

3. HACS sends notification: “Dustbin Full.”

Postcondition: User empties dustbin before restarting.

Use Case 5: Cancel Operation

Actors: User, System

Precondition: A scheduled or running task exists.

Steps:

1. User selects “Cancel”.

2. System aborts task.

3. Vacuum stops immediately.

Postcondition: Operation is completely removed.

3.4 EXTENDED EXPLANATION OF USE CASES

The Requirements Engineer emphasized that use cases help bridge the gap between user expectations and system behavior. For example, the “Schedule Cleaning” use case illustrates how automation reduces manual effort, while “Pause/Resume” reflects real-world flexibility. Alerts ensure the appliance is safe and effective.

Each use case was carefully analyzed to identify exceptions (such as low battery during scheduled operation), ensuring that the system does not fail in unexpected situations.

4. SYSTEM DESIGN (SYSTEM DESIGNER CONTRIBUTION)

The system design process translates the documented requirements into a clear technical structure that can be implemented by developers. While requirements describe what the system should do, the design explains how the system will achieve it.

In the case of the Smart Vacuum Cleaner within HACS, the System Designer created models, diagrams, and explanations that describe the overall architecture, interactions between components, and flow of data.

The design is critical because it provides a blueprint. Without proper design, developers may misinterpret requirements, leading to errors or inconsistencies.

4.1 SYSTEM ARCHITECTURE

The Smart Vacuum Cleaner system architecture is divided into three layers:

1. User Interface Layer

This layer is the entry point for the user.

It includes a mobile app or control panel where the user can press buttons like Start, Stop, Schedule, etc.

The UI is designed to be simple and intuitive, even for non-technical users.

2. Control Layer (HACS Server)

This is the brain of the system.

It receives commands from the user, processes them, and sends them to the vacuum cleaner.

It also collects data from the vacuum (like battery level, dustbin status) and forwards alerts to the user.

3. Appliance Layer (Smart Vacuum Cleaner)

This is the actual device that executes commands.

It includes motors, wheels, sensors, and embedded software.

It responds to commands such as start, pause, cancel, and resume.

The three layers work together to ensure smooth functioning. If one layer fails (e.g., loss of connectivity between Control Layer and Appliance Layer), the system cannot function correctly.

4.2 CONTEXT DIAGRAM DESCRIPTION

The context diagram is a high-level view of the system showing how external entities (users, environment) interact with the system.

The User provides input (Start, Stop, Schedule) through the HACS interface.

The HACS Server processes this input and communicates with the Smart Vacuum Cleaner.

The Vacuum Cleaner executes tasks and returns status updates.

The System sends feedback to the User in the form of notifications (alerts, confirmations).

In narrative terms, the context diagram shows that the user does not directly control the vacuum but interacts through HACS, which ensures a standardized and secure process.

4.3 DATA FLOW DIAGRAM (DFD)

The Data Flow Diagram illustrates how data moves within the system.

Level 0 (High Level):

User → HACS → Smart Vacuum Cleaner.

Vacuum Cleaner → HACS → User.

Level 1 (Detailed):

User initiates a command (e.g., Schedule).

Command enters the “Task Scheduler” process in HACS.

The schedule is stored in the Database.

At the scheduled time, HACS sends command to the vacuum.

The vacuum runs the operation and returns status (success, alert).

HACS forwards this status to the user as a message.

4.4 SEQUENCE DIAGRAM DESCRIPTION

The sequence diagram represents interactions step by step in time order.

Example: Pause and Resume Operation

1. The user presses Pause in the mobile app.

2. The app sends this command to the HACS server.

3. HACS forwards the command to the vacuum.

4. The vacuum halts cleaning and sends confirmation back.

5. Later, the user presses Resume.

6. The same chain repeats in reverse order, and cleaning continues.

This sequence ensures that all steps are captured and nothing is skipped.

4.5 COMPONENT DESIGN

The system can also be broken into components, each responsible for a specific function:

1. User Command Handler: Accepts input from UI.

2. Scheduler Module: Stores and manages scheduled tasks.

3. Command Processor: Translates user commands into appliance signals.

4. Alert Handler: Detects and sends alerts to the user.

5. Database: Stores schedules, logs, and status reports.

6. Appliance Controller: Communicates directly with vacuum cleaner hardware.

4.6 ARCHITECTURAL DECISIONS

The System Designer made several important decisions:

Layered Architecture: Chosen for clarity and separation of concerns.

Client-Server Communication: Used because users cannot directly interact with the appliance.

Database Support: Added to ensure schedules and logs are stored persistently.

Alert System: Integrated to improve usability.

Alternative designs, such as peer-to-peer communication between user and vacuum, were rejected because they lacked security and scalability.

4.7 ADVANTAGES OF THE DESIGN

1. Modularity: Each component can be updated without affecting others.

2. Scalability: New appliances can be added to HACS using the same architecture.

3. Reliability: Layered design ensures failures can be isolated.

4. User Friendliness: Context diagrams and flows make the system easy to understand.

4.8 DESIGNER’S REFLECTION

From the perspective of the System Designer, the Smart Vacuum Cleaner was more challenging to design than simpler appliances. Unlike lights, which have binary states, the vacuum cleaner has multiple states: running, paused, stopped, cancelled, scheduled, and alert modes. Designing for all these possibilities required careful thought.

The use of diagrams like context, DFD, and sequence diagrams provided clarity and reduced the chances of miscommunication with developers. It also ensured that testers could design test cases properly because they knew exactly how data flows through the system.

5. IMPLEMENTATION (DEVELOPER CONTRIBUTION)

The Developer’s role in the Smart Vacuum Cleaner project was to convert the requirements and design into an actual working system. While the Requirements Engineer identified what the system must do and the System Designer explained how the system should be structured, the Developer took those documents and began implementing the features step by step.

The Developer did not work in isolation. Constant communication with the Project Manager, Requirements Engineer, and System Designer was required to ensure the implementation matched expectations. Every feature implemented was directly traceable to a requirement documented earlier in this SRS.

5.1 IMPLEMENTATION APPROACH

The implementation followed a modular approach. Instead of writing the entire system as a single block, the Developer divided the Smart Vacuum Cleaner into different modules:

1. User Command Module

Handles inputs like Start, Stop, Pause, Resume, Schedule, and Cancel.

Ensures commands from the user interface are captured correctly and passed to the control system.

2. Scheduler Module

Manages scheduled tasks.

Retrieves current time, compares it with saved schedules, and triggers operations at the right moment.

3. Operation Control Module

Handles actual running states of the vacuum: running, paused, resumed, stopped.

Ensures state transitions are smooth (e.g., moving from pause to resume).

4. Alert Module

Monitors sensors like battery and dustbin.

Generates alerts when limits are reached (low battery, dustbin full).

5. Database Module

Stores all schedules, logs, and alerts.

Ensures persistence so that schedules are not lost even if the system restarts.

This modular approach makes the system easier to maintain, test, and expand.

5.2 FEATURE-BY-FEATURE IMPLEMENTATION

5.2.1 START CLEANING

The Start feature was implemented by sending a signal from HACS to the vacuum’s motor controller.

Once the vacuum received the command, it began moving according to predefined patterns.

The Developer ensured that the Start command could only be executed when the vacuum was in Idle state.

5.2.2 STOP CLEANING

The Stop command immediately halted the vacuum cleaner’s motors.

The Developer had to ensure that all ongoing operations (such as cleaning routines) were safely terminated without causing errors.

Logs were updated to reflect that the operation ended by user request.

5.2.3 PAUSE AND RESUME

Implementing Pause and Resume was more complex than Stop.

The vacuum’s current position and state had to be stored when paused.

On resuming, the vacuum had to continue from the same position rather than starting over.

The Developer created a mechanism to capture “current state” variables and reload them on resume.

5.2.4 SCHEDULING

The Scheduling feature required interaction between the Scheduler Module and the Database Module.

When a schedule was set, the details (date, time, operation) were stored in the database.

A timer service continuously checked the current time. When it matched a scheduled entry, the command was automatically executed.

The Developer ensured error handling, e.g., if the battery was too low at scheduled time, the task was delayed and a notification sent to the user.

5.2.5 CANCEL OPERATION

Cancelling was implemented as a two-step process:

1. If the task was scheduled but not started, it was removed from the database.

2. If the task was already running, the system stopped it and deleted its entry from memory.

This distinction ensured clarity between cancelling before and during execution.

5.2.6 ALERTS (BATTERY AND DUSTBIN)

Sensors were continuously monitored during operation.

If the battery dropped below 20%, a Low Battery event was triggered.

If the dustbin sensor detected maximum capacity, a Dustbin Full event was triggered.

Both events were logged in the database and notifications were sent to the user via HACS.

5.3 ERROR HANDLING

One of the Developer’s major contributions was designing error handling for unexpected conditions. Examples include:

Connectivity Lost: If Wi-Fi failed during cleaning, the vacuum automatically paused. Once the connection was restored, the user was prompted to either resume or cancel.

Battery Too Low: If the battery dropped suddenly, the vacuum returned to the charging dock automatically.

Schedule Conflict: If two schedules overlapped, the system asked the user to confirm which should run.

Error handling ensured that the system behaved safely under all conditions.

5.4 IMPLEMENTATION CHALLENGES

The Developer faced several challenges:

1. State Management

Tracking the vacuum’s states (Idle, Running, Paused, Stopped, Cancelled) was complex.

Incorrect state management could result in errors (e.g., resuming when nothing was paused).

2. Time Synchronization

Scheduling required precise synchronization with system time.

The Developer had to ensure the system time matched the appliance clock.

3. Sensor Integration

Battery and dustbin sensors provided raw data, which needed to be converted into meaningful alerts.

4. Scalability

Although implementation focused on the vacuum cleaner, the system was designed to easily integrate future appliances.

5.5 DEVELOPER’S REFLECTION

The Developer reflected that the Smart Vacuum Cleaner was one of the most challenging appliances to implement because of its multiple states and dependency on real-time data from sensors. Unlike a simple ON/OFF appliance, the vacuum required advanced logic for scheduling, pausing, resuming, and alerts.

The modular approach made implementation easier. Each feature could be developed and tested individually before being integrated into the full system. This reduced bugs and improved overall reliability.

The Developer also emphasized the importance of teamwork. Without clear requirements from the Requirements Engineer and diagrams from the System Designer, implementation would have been much more difficult. The close collaboration ensured that the system matched user needs while remaining technically feasible.

6. TESTING (TESTER CONTRIBUTION)

Testing is a critical phase of any software project. It ensures that the system functions as expected, meets user requirements, and performs reliably under different conditions. In the context of the Smart Vacuum Cleaner within HACS, testing was even more important because the appliance is not limited to simple operations. Instead, it supports advanced features such as scheduling, pausing, resuming, and generating alerts.

The Tester’s primary responsibility was to design and execute test cases, validate system requirements, identify defects, and ensure that the final product delivered to the user was of high quality. The Tester also collaborated closely with the Developer, providing feedback whenever bugs were discovered, and verifying fixes once they were applied.

6.1 TESTING OBJECTIVES

The main objectives of testing were:

1. To confirm that functional requirements were implemented correctly (e.g., the Pause feature pauses the vacuum).

2. To validate non-functional requirements such as performance (system responds within 3 seconds).

3. To identify errors, inconsistencies, or missing features at the earliest possible stage.

4. To ensure that the system behaves correctly under exceptional conditions, such as low battery or loss of connectivity.

5. To confirm that the system works smoothly for both technical and non-technical users.

6.2 TESTING STRATEGY

The Tester adopted a layered testing strategy, meaning the system was tested at multiple levels to catch defects early and to ensure coverage:

1. Unit Testing

Focused on individual modules (e.g., Scheduler Module, Alert Module).

Ensured that each module worked correctly in isolation.

2. Integration Testing

Verified that modules interacted correctly.

Example: The Scheduler Module passed tasks to the Operation Control Module without errors.

3. System Testing

Checked the entire system as a whole.

All user operations (start, stop, schedule, pause, resume, cancel, alerts) were tested together.

4. Acceptance Testing

Conducted from the perspective of end users.

Verified that the system was intuitive and met user needs.

6.3 TYPES OF TESTING APPLIED

The Tester applied multiple testing types, not just one method.

Functional Testing: Confirmed that each requirement was implemented.

Usability Testing: Ensured that the interface was easy to use.

Performance Testing: Verified that commands were executed quickly.

Stress Testing: Checked how the system behaved under extreme conditions (e.g., multiple schedules set back-to-back)

Reliability Testing: Confirmed that scheduled tasks always executed at the correct time.

Security Testing: Verified that unauthorized users could not control the appliance.

6.4 TEST CASE DESIGN

The Tester designed detailed test cases for each requirement. A sample set is provided below.

Functional Test Cases

Test ID Requirement Input Expected Output Result

TC-01 Start Vacuum User presses “Start” Vacuum begins cleaning Pass pass

TC-02 Stop Vacuum User presses “Stop” Vacuum halts immediately Pass

TC-03 Schedule Cleaning User sets schedule for 9:00 AM Vacuum starts at 9:00 AM Pass

TC-04 Pause Operation User presses “Pause” Vacuum halts temporarily Pass

TC-05 Resume Operation User presses “Resume” Vacuum continues cleaning Pass

TC-06 Cancel Operation User presses “Cancel” Task is removed and vacuum resets Pass

TC-07 Low Battery Alert Battery < 20% Alert message “Low Battery” Pass

TC-08 Dustbin Full Dustbin sensor = Full Alert message “Dustbin Full” Pass

NON-FUNCTIONAL TEST CASES

Test ID Requirement Expected Output Result

TC-09 Response TimeUser presses “Stop” Vacuum stops within 3 seconds Pass

TC-10 UsabilityElderly user tries to schedule Interface is clear and easy Pass

TC-11 Reliability Task scheduled daily at 9 AMTask runs on time every day Pass

TC-12 Security Unauthorized user tries login Access denied Pass

6.5 EXCEPTION AND ERROR TESTING

The Tester also verified how the system behaved in exceptional conditions:

1. Low Battery During Operation

Scenario: Cleaning scheduled at 9 AM, but battery at 10%.

Expected: Operation postponed, notification sent.

Actual: System performed correctly.

2. Dustbin Full During Cleaning

Scenario: Vacuum in middle of task, dustbin full.

Expected: Operation stopped, notification sent.

Actual: Worked as expected.

3. Connectivity Lost

Scenario: Wi-Fi disconnected while vacuum was running.

Expected: Vacuum paused, resumed after reconnection.

Actual: Functionality matched expectations

4. Multiple Schedules Conflict

Scenario: User sets 2 schedules at 9 AM.

Expected: System prompts user to resolve conflict.

Actual: Worked correctly.

6.6 DEFECT REPORTING

During testing, a few defects were identified:

Issue 1: Resume command sometimes restarted from beginning instead of paused location.

Fixed by Developer after clarification.

Issue 2: Alert for full dustbin did not trigger consistently in early builds.

Fixed by recalibrating sensor.

Issue 3: Response time occasionally exceeded 3 seconds when Wi-Fi as weak.

Improved after optimizing communication.

Each defect was logged, discussed with the Developer, and resolved.

6.7 TESTER’S REFLECTION

From the Tester’s perspective, the Smart Vacuum Cleaner required more complex testing compared to simpler appliances. Unlike a light bulb (with only ON and OFF states), the vacuum required testing of multiple states, schedules, and error scenarios.

The Tester emphasized that real-world users expect reliability. If the system fails even once (e.g., scheduled task does not run), trust in the system is reduced. Hence, exhaustive testing was essential.

Another key reflection was the importance of collaboration. The Tester constantly interacted with the Requirements Engineer (to clarify expected behavior) and the Developer (to resolve bugs). This ensured the product matched both documented requirements and user expectations.

7. PROJECT MANAGEMENT (PROJECT MANAGER CONTRIBUTION)

The Project Manager plays a central role in ensuring that the Smart Vacuum Cleaner project within HACS is delivered successfully. While other team members focus on specific tasks such as requirements, design, development, and testing, the Project Manager oversees the entire lifecycle, making sure that work is aligned, deadlines are met, and communication flows smoothly among all stakeholders

The Project Manager’s responsibility is not limited to planning tasks. They also handle risk management, resolve conflicts, and act as a bridge between the development team and external stakeholders (such as clients, professors, or users).

7.1 PROJECT PLANNING

The first responsibility of the Project Manager was to create a project plan. This plan defined:

1. Scope

The project was limited to implementing one appliance (Smart Vacuum Cleaner) within HACS

Additional appliances such as lights and fans were not included, ensuring that the team could focus deeply on a single device.

2. Timeline

The project was divided into phases:

Requirements Elicitation (Week 1–2)

System Design (Week 3–4)

Implementation (Week 5–7)

Testing (Week 8–9)

Final Report and Submission (Week 10)

3. Resources

Each member was assigned a role: Requirements Engineer, System Designer, Developer, Tester.

Tools included laptops, version control software, and testing environments.

4. Deliverables

SRS Document

System Design Document

Implementation Modules

Test Reports

Final Submission Document

7.2 TASK ALLOCATION

The Project Manager clearly defined roles for each member:

Requirements Engineer gathered system requirements and created use cases.

System Designer produced architecture, diagrams, and flows.

Developer implemented each feature module.

Tester ensured system quality.

Project Manager coordinated everything and monitored progress.

By avoiding overlapping responsibilities, the Project Manager ensured efficiency and accountability.

7.3 RISK MANAGEMENT

The Project Manager also identified potential risks and created mitigation strategies.

1. Technical Risks

Risk: Sensors may fail to provide accurate data.

Mitigation: Extra testing and calibration.

2. Schedule Risks

Risk: Some tasks may take longer than expected.

Mitigation: Buffer time added to the project schedule.

3. Resource Risks

Risk: Team members may face technical issues with laptops or connectivity.

Mitigation: Backup devices and offline documentation prepared.

4. Operational Risks

Risk: Miscommunication among team members.

Mitigation: Weekly meetings and progress updates ensured alignment.

5. External Risks

Risk: Client or professor expectations may change mid-project.

Mitigation: Maintain flexible design to accommodate minor changes.

7.4 COMMUNICATION MANAGEMENT

Communication is one of the most important responsibilities of the Project Manager. The Project Manager ensured:

Weekly meetings where each member reported their progress.

Daily informal check-ins to monitor tasks.

Status reports were shared with all members.

Clear documentation of decisions to avoid misunderstandings.

The Project Manager also acted as the single point of contact with external stakeholders, ensuring that the team was not distracted by conflicting requests.

7.5 MONITORING AND CONTROL

The Project Manager used simple monitoring techniques:

Gantt Chart: Visual representation of tasks and deadlines.

Task Tracker: A shared document where each member updated progress.

Milestone Reviews: Every two weeks, progress was compared against the plan.

When delays occurred (such as in the implementation of Pause/Resume), the Project Manager helped the Developer prioritize fixes and reallocated time from less urgent tasks.

7.6 LEADERSHIP AND TEAM MOTIVATION

Beyond technical management, the Project Manager’s role included motivating the team. Members sometimes faced challenges (e.g., the Developer struggled with state management, or the Tester found too many early bugs). The Project Manager encouraged collaboration instead of blame, fostering a positive team environment.

Motivation strategies included:

Acknowledging each member’s effort during meetings.

Encouraging teamwork rather than individual competition.

Reminding the team of the end goal and the importance of their contributions.

7.7 PROJECT MANAGER’S REFLECTION

The Project Manager reflected that managing a team of five members for a software engineering project required both organization and adaptability. While technical knowledge was helpful, communication and leadership skills were equally important.

Key lessons learned included:

1. Clear Planning Saves Time: By defining roles early, confusion was avoided.

2. Flexibility is Essential: Plans often required adjustment due to unforeseen delays.

3. Teamwork Leads to Success: Each member’s contribution was vital. Without the Requirements Engineer, the system would lack clarity; without the Designer, there would be no blueprint; without the Developer, nothing would be built; without the Tester, the system would be unreliable.

4. Responsibility of Leadership: The Project Manager was accountable not only for their own tasks but also for ensuring that the entire team performed.

The Project Manager concluded that while technical knowledge is important, leadership, organization, and teamwork are equally critical for project success.

8. CONCLUSION

The development of the Smart Vacuum Cleaner module within the Home Appliance Control System (HACS) represents a significant academic and technical milestone for the team involved. This Software Requirements Specification (SRS) has documented the project comprehensively, from conceptualization through to system design, implementation, testing, and project management.

This module was deliberately chosen over simpler appliances because of its richer set of functionalities and real-time feedback mechanisms, which made it a stronger candidate for demonstrating full software development life cycle (SDLC) practices. The Smart Vacuum Cleaner supports complex operations such as task scheduling, pausing, resuming, alert handling, and status monitoring — all of which required detailed functional and non-functional analysis.

Throughout this SRS, each member of the development team contributed uniquely based on their designated role:

The Requirements Engineer ensured that every system need was accurately identified, clearly defined, and traceable to real-world user expectations.

The System Designer translated these requirements into scalable architectural diagrams and process flows, enabling clarity in development.

The Developer implemented core functionalities with an emphasis on modular design, maintainability, and accurate device control.

The Tester validated every requirement with systematic testing strategies, documenting both successes and issues that needed resolution.

The Project Manager coordinated efforts, applied risk management strategies, and ensured that team collaboration remained strong throughout the process.

From a systems perspective, the project not only showcases the control of a smart appliance but also exemplifies the principles of modularity, reusability, and scalability, which are foundational to software engineering. The design allows easy extension to other appliances in the future, such as smart fans, lights, or air conditioners.

Moreover, the SRS reflects strong academic principles by including not just technical specifications but also thoughtful reflections, stakeholder considerations, real-world constraints, and user-centered design thinking. The use of formal documentation methods (e.g., use cases, DFDs, architecture layers) has elevated the quality and professionalism of the work.

Future Scope

Although this project focused on a single device within HACS, its foundation supports expansion in several directions:

Integration of machine learning to adapt cleaning schedules based on user behavior.

Voice assistant integration (e.g., Amazon Alexa, Google Home).

Real-time location tracking and room-mapping for advanced navigation.

Power consumption optimization and eco-mode functionalities.

Centralized control dashboard for managing multiple appliances simultaneously.

These future enhancements align with the evolving trends in smart homes and Internet of Things (IoT), making the HACS platform a flexible and forward-looking solution.

Final Reflection

Ultimately, this project was more than a technical exercise — it was a demonstration of teamwork, planning, and problem-solving. By simulating an industry-like software development environment, the team not only delivered a robust software solution but also learned the importance of documentation, collaboration, and communication in the software engineering process.

This SRS stands as both a technical guide and a proof of concept for how smart home appliances can be thoughtfully designed and managed through intelligent systems.

REFERENCE:GITHUB LINK

https://github.com/kc-1106/Smart\_vacuum-Software