Department of Computer Engineering

Faculty of engineering and technology

University of Buea

Département de Génie Informatique

Faculté de Génie et de Technologie

Université de Buea

COURSE CODE: CEF 440
COURSE TITLE: INTERNET PROGRAMMIMG AND MOBILE PROGRAMMING

TASK THREE: REQUIREMENT ANALYSIS ON A CAR FAULT MANAGEMENT SYSTEM

Presented by:
GROUP 16

NAME	MATRICULE
BELLAH LOVETTE MANYI	FE22A172
TABOT ALISTAR GIFT	FE22A301
NGEIMASHUNG NJUZE RAVINE	FE22A260
ANNE BENITA NKENGFUA	FE22A149
ETA INDIRA	FE22A209

Course instructor:

Dr Nkemeni Valery

MAY 2025

ABSTRACT

This report presents a comprehensive requirement analysis for the development of an automotive diagnostic system that leverages embedded sensors and an Engine Control Unit (ECU) to detect, interpret, and log vehicle faults. The purpose of the requirement analysis phase is to ensure that all system expectations are clearly defined, feasible, and aligned with stakeholder needs. Through stakeholder engagement, internal reviews, and validation sessions, both functional and non-functional requirements were identified, evaluated, and classified. The analysis also involved detecting ambiguities, dependencies, and potential inconsistencies within the gathered requirements. The final output of this phase is a validated Software Requirements Specification (SRS) that serves as a foundation for system design and development.

TABLE OF CONTENT

INTRODUCTION

In the evolving landscape of automotive technology, the need for intelligent and responsive diagnostic systems has become critical for vehicle maintenance and safety. The proposed project seeks to develop a robust vehicle fault diagnostic system using embedded sensor networks and ECU communication protocols. This system will automatically monitor engine parameters, detect abnormalities, generate diagnostic trouble codes (DTCs), and present fault information to the user through a dashboard interface.

The objective of the requirement analysis phase is to bridge the gap between what stakeholders need and what the technical team will build. This involves gathering, analyzing, prioritizing, and validating all requirements related to the system's performance, functionality, and usability. By ensuring clarity, completeness, and technical feasibility of each requirement, the project aims to reduce the risks of rework, cost overruns, and stakeholder dissatisfaction.

This report documents the activities undertaken during the requirement analysis phase, including requirement elicitation, classification (functional vs. non-functional), dependency identification, and validation with stakeholders. It concludes with the development of a formally agreed-upon Software Requirement Specification (SRS) that will serve as the basis for the subsequent design phase.

LITERATURE REVIEW

The process of requirement analysis is a foundational activity in software engineering that directly affects the success of a software system. It ensures that the final system meets user needs and functions correctly within its intended environment. This section reviews key literature on best practices in requirements engineering, particularly in the context of embedded systems and automotive diagnostics.

1. **Requirement Analysis in Software Engineering**
   Requirement analysis involves identifying what a system should do (functional requirements) and how it should perform (non-functional requirements). According to Sommerville (2016), effective requirements analysis improves project outcomes by reducing errors, managing scope, and aligning expectations early in the software development lifecycle.

   Studies by Kotonya and Sommerville (1998) emphasize that completeness, clarity, feasibility, and validation are critical dimensions of high-quality requirements. Ambiguity, in particular, is a leading cause of system failure and is best addressed through stakeholder engagement and iterative refinement.

2. **Requirement Gathering and Stakeholder Involvement**
   Stakeholder involvement has been widely researched as essential for accurate requirement capture. Wiegers and Beatty (2013) explain that involving a diverse set of stakeholders ensures comprehensive coverage and early resolution of inconsistencies. In the context of automotive systems, stakeholders may include technicians, engineers, regulators, and end users.

   Research from the IEEE Software Engineering Body of Knowledge (SWEBOK) indicates that requirement elicitation techniques like interviews, questionnaires, observation, and document analysis are suitable for domains with both technical and user-facing components—such as diagnostic systems in vehicles.

3. **Requirement Classification: Functional vs. Non-Functional**
   Glinz (2007) outlines that functional requirement define system behavior, while non-functional requirements constrain how the system performs. In automotive diagnostics, for example:

   - **Functional**: Log sensor data, generate fault codes, display faults.
   - **Non-functional**: Real-time performance, durability under harsh conditions, compliance with OBD-II.

   Proper classification ensures that both operational features and quality attributes are addressed during development.

4. **Software Requirement Specification (SRS)**

IEEE Standard 830-1998 specifies the structure and contents of a Software Requirements Specification document. An SRS serves as a contract between users and developers. It improves communication and serves as a basis for system design, testing, and maintenance.

In automotive diagnostic systems, the SRS should clearly define the timing constraints (e.g., “data must be logged within 2 seconds”), safety requirements, and interoperability with protocols like CAN and ISO 9141.

5. Requirement Validation Techniques

Validation ensures the documented requirements reflect user needs and are technically viable. According to Davis (1993), common validation methods include requirements reviews, prototyping, and use case modeling.

In embedded diagnostic systems, stakeholder reviews and simulations using vehicle test data are practical for verifying requirement correctness. Peer reviews can identify inconsistencies or incompleteness early, saving significant downstream effort.

6. Automotive Diagnostic Standards and Practices

Modern automotive systems follow standardized communication protocols such as OBD-II and CAN bus, which dictate how diagnostic data is retrieved and interpreted. Literature on these protocols (Bosch, 2004) highlights the importance of system compliance with regulatory standards for interoperability and reliability.

These standards also influence non-functional requirements related to latency, logging frequency, and fault classification mechanisms.

METHODOLOGY

A. Review and analyze the requirement gathered

In requirement analysis refers to whether all the necessary and relevant requirements for a system

i. **Completeness**:

Suggested Improvements:

1. How sound detection is done

Missing**:**

No details about how engine sounds are captured.

Solution:

- Use a built-in microphone on the user's mobile device to record engine sounds.
- Then, apply machine learning algorithms to compare the sound with a database of known faults.

Update Requirement:

Users can either manually upload sound recordings via the app or enable automatic real-time monitoring using car-installed microphones.

2. How users report or confirm faults

Missing:

No clear process for confirming or submitting detected issues.

Solution:

- Include a fault report form in the app that auto-fills detected issues but allows user confirmation before sending.
- Add a confirmation pop-up after fault detection asking: Do you want to report this to a mechanic?

Update Requirement:

Users will receive fault notifications and can confirm or cancel the alert before it’s sent to a mechanic.

3. Criteria for what qualifies as a fault

Missing:
No rule or system to define a fault.

Solution:

- Create a fault classification model trained on different engine sounds: normal, warning, and critical.
- Use thresholds like decibel levels, frequency patterns, or anomalies compared to a baseline.

Update Requirement:
A machine learning model will identify faults based on pre-labeled audio datasets that define what constitutes a fault.

3. Security, user authentication, or access roles

Missing:
No plan for who can access what, or how users log in securely.

Solution:

- Add user accounts with roles: Vehicle Owner, Mechanic, Admin.
- Use secure login with OTP or two-factor authentication.
- Apply data encryption for sound files and reports.

Update Requirement:
The system will include secure login with role-based access. Only verified users can upload or view car data. Mechanics will access only the faults assigned to them.

ii. Clarity:

This is how easy it is to understand the requirements by developers, designers, testers, and stakeholders

Suggested improvements:

1. How the system distinguishes between normal and faulty sounds

Missing:
No clarity on how sound types are classified.

Solution:

- Implement a machine learning classification model (e.g., SVM, CNN) trained on audio recordings labeled as “normal” or “faulty.”

- Use audio features like pitch, frequency, and duration to compare against a known database of faults.
- Document the process clearly in the system design or user guide.

Update Requirement:
The system uses AI to compare user-recorded engine sounds with a pre-trained model based on labeled sound samples, identifying fault types through pattern recognition.

2. Mechanism for sound comparison and alerting

Missing:
It's not defined how the app decides when to alert or notify.

Solution:
- Add logic that triggers alerts only when certain thresholds are crossed — like decibel limits or pattern matches.
- Clarify the backend processing steps: record → compare → alert → confirm → notify.
- Use flowcharts or UML diagrams in the documentation.

Update requirement:
When a sound is recorded, it's processed by a comparison engine that checks for matches in the fault sound database. If a match with critical probability (>80%) is found, an alert is sent.

3. How long the time window is to contact a mechanic

Missing:
Users don't know how much time they have before action is taken.

solution:
- Define a default time range (e.g., 15 minutes) which can be configured by the user.
- Add a countdown timer in the app interface.
- Mention fallback action (e.g., auto-escalate if no response).

Update requirement:
Users must respond to fault alerts within 15 minutes; if no action is taken, the app will automatically notify the nearest available mechanic.

iii. Technical Feasibility

It means checking whether it's possible to build the system with the available technology, skills, and resources.

Feasible Components:

1. Audio recognition requires training with a wide dataset of fault/noise types since the system needs a comprehensive and diverse dataset of various car engine sounds to accurately classify faults. Without a good training set, the system could produce false positives or miss some faults.

Missing**:**
How does the system comprehend diverse dataset of various car engine sounds

Solution:

- Expand the dataset by collecting more real-world recordings of both normal and faulty engine sounds. Work with car repair shops, automotive manufacturers, or use crowdsourcing to gather data.
- Collaborate with automotive engineers or companies that specialize in vehicle diagnostics to create a robust dataset.
- Enhance the model with continuous learning: Allow the system to learn from new sounds over time and improve its accuracy as more data is collected.

Updated requirement:
Ensure that the system is trained with a large, diverse dataset of engine sounds, constantly updated through user input and collaboration with industry experts.

2. Hardware dependency (e.g., microphones) may raise cost or compatibility issues
Relying on external microphones or create compatibility issues for users with different devices.

Missing**:** How to solve compatibility issues

Solution:
Optimize for device compatibility: Ensure the system works on a variety of mobile devices (iOS, Android) and does not require expensive hardware.

Updated requirement:
Ensure the system supports sound recording from smartphones, but provide optional affordable hardware upgrades for users seeking better accuracy.

3. Reliable internet connectivity may be needed for cloud-based analysis. So the system may require constant internet connectivity for cloud processing, which can be problematic for users in areas with poor internet access.

Missing**:** How to implement the offline functionality for users who are offline

Solution**:**

- Implement offline functionality: Allow the system to record and process sounds locally on the device. Once the internet is available, sync the data with the cloud.
- Cloud-based caching: Temporary storage of data on the device, which can be uploaded once connectivity is restored.
- Optimize the cloud service to minimize data transfer and processing times, ensuring a smooth experience even with occasional connectivity issues.

Update requirement:
Incorporate offline capabilities and data caching to ensure that users can record and analyze sounds without constant internet access. Once online, the app will sync with the cloud.

4. Starting with a small range of fault types and expanding gradually. It's not practical to detect all types of car faults at once, especially in early development stages.

Misssing**:** How to manage the scalabilty of the car faults been added to the app

Solution:

- Begin with commonly known faults (like engine knocking, belt squealing, or misfiring sounds).
- Build a modular system that allows adding more fault types over time without redesigning the whole app.
- Collect user feedback and sound samples to improve the model and expand its recognition abilities in future updates.

Requirement update:
To manage complexity, the system will first recognize a limited set of frequent car faults, then gradually integrate more based on feedback and data.

5. Real-time alerts and notifications can be implemented, delivering instant feedback to users when a fault is detected is key, but it must be timely and accurate.

Missing**:** How to manage the Real-time alerts and notifications

Solution**:**

- Use push notifications on the mobile app to alert users the moment a fault is detected.
- Ensure notifications include clear action steps, like "Sound detected: possible engine knock. Contact a mechanic?".
- Integrate with the mechanic response system so that users can quickly request help.

Update requirement**:**
Real-time alerts will use push notifications to inform users immediately about possible faults, with quick access to contact mechanics.

6. Databases can store sound files and user data effectively storing a large amount of audio data and user information securely and efficiently.

Missing: How to store large amount of audio data and user information securely and efficiently

Solution:

- Use a cloud-based database (like Firebase or AWS) to handle large audio files and user activity logs.
- Implement compression techniques to reduce file size without losing audio quality.
- Ensure data security by applying encryption, user authentication, and access control.

Update requirement**:**

A secure, scalable cloud database will be used to store recorded car sounds and user data, with encryption and access control measures.

iv. Dependency Relationships

Is simply one part of the system relies on another to work properly.
These are the most important parts of the car fault detection system that rely on one another to work correctly:

1. Sound Detection and Database:
The system depends on accurate detection of engine sounds, which must be recorded and stored properly in the database. If recording fails or is missing, fault detection cannot happen.

2. Sound Analysis and Trained Model:
The app depends on a machine learning model or algorithm trained with sound data to decide whether a sound is normal or faulty. If the model isn't accurate, it could lead to false alerts or missed faults.

3. User Alerts and Detection Accuracy:
Notifications to users depend on the system correctly identifying faults. A weak analysis means users won't trust the alerts.

4. User and Mechanic:
The response mechanism depends on users getting alerts and then mechanics being available to respond in time. If either side fails, the system doesn't fulfill its purpose.

5. Authentication System and User & Mechanic Access:
Secure access to the app depends on a working login/authentication system, which defines user roles (e.g., car owner vs. mechanic). Without it, data privacy and trust are compromised.

6. Database Management and System Performance:
The database must efficiently store and retrieve sound files, user details, and fault logs.
Poorly managed databases can lead to slow performance or data loss.

7. Database and User:
The user interacts with the database indirectly through the system's interface. When a user uploads a car sound, the system stores that sound in the database.
Similarly, if a user reports a fault or checks past notifications, the data is read from or written to the database.
The user relies on the database to store, retrieve, and manage all information related to car faults, sound recordings, alerts, and mechanic responses. Without the database, the user's data wouldn't be saved or available for later use.

8. Admin and User:
The admin is responsible for managing the system and overseeing user activity. This includes creating new user accounts, assigning roles (like regular user or mechanic), and setting system permissions or rules.
The user, on the other hand, depends on the admin to access and use the system. For example, a user cannot sign in or use the features of the system unless the admin has approved or registered them. The admin also monitors system usage and can respond to issues

B. IDENTIFY INCONSISTENCIES, AMBIGUITIES & MISSING INFORMATION,2. DETAILED IDENTIFICATION OF CATEGORY

Category	Original Requirement	Requirement Description	Suggested Resolution
Ambiguity	High diagnostic accuracy under NFR1	No target accuracy percentage—"high" is subjective.	Define a measurable target (e.g. "achieve ≥ 90 % detection accuracy for dashboard lights and ≥ 85 % for engine-sound classifications").
Ambiguity	Quick responses (within seconds)	"Within seconds" could mean anything from 1 s to 10 s; different features have different complexity.	Specify maximum per-feature (e.g., light-scan ≤ 2 s, sound-analysis ≤ 4 s on average hardware).
Ambiguity	Timely maintenance reminders	How and when reminders are triggered—by date, fault severity?	Clarify reminder logic (e.g., "push reminder when a high-urgency fault is detected, or every 6 months or after a particular distance covered").
Missing	Model update / retraining mechanism	No detail on how on-device ML models will be updated to accommodate new fault patterns.	Specify server-side model retraining pipeline and in-app update process

Missing	Data management & storage	Unclear where and how user data (images, audio clips, usage logs) are stored, how long retained, and how users can delete it.	Define data-retention policy, local cache limits
Missing	User authentication & roles	The requirements don't state if users must create an account, sign in, or if there are different user roles (e.g., admin vs. standard user).	Add authentication FR (e.g., "FR-Auth: Users may register/login via emai)l
Missing	Error handling & fallback UI	No description of how the app behaves if recognition fails (e.g., no light detected, sound too noisy, offline without models).	Specify fallback flows (e.g., "Show 'retry' prompt if camera scan fails; in offline mode, warn user when analysis may be outdated; log errors for diagnostics").
Inconsistency	Mechanic Locator described as "optional" but survey says "potentially increasing user trust and app utility"	If it's truly optional, should it appear in the requirement document? Survey feedback suggests users value it nearly as much as repair guidance.	Decide placement: either include it on the document with minimal feature set (show static mechanic list).

Inconsistency	Video tutorials ("YouTube or embedded")	Embedding YouTube content may conflict with offline requirement; unclear whether videos are downloadable.	Clarify: "Embedded videos require online mode and will not be available offline; provide text-based repair summaries for offline use."

C. PRIORITIZE REQUIREMENTS BASED ON IMPORTANCE AND FEASIBILITY

Applying the **MoSCoW prioritization method** to our functional requirements for the **vehicle diagnostic system**, focusing on **Importance** (user need, system value) and **Feasibility** (technical ease, cost, time).

	Requirement	**Importance**	**Feasibility**	**Priority (MoSCoW)**	**Reasoning**
1	Dashboard Light Recognition	High	High	**Must Have**	Core diagnostic functionality; technically feasible using image classification.
2	Engine Sound Analysis	High	Medium-Low	**Should Have**	Valuable feature but harder to implement due to sound pattern recognition complexity.
3	Repair Guidance & Maintenance Reminders	High	High	**Must Have**	Highly practical for users; can be implemented with rule-based systems.
4	Integrated Multi-Feature Dashboard	Medium	Medium	**Should Have**	Enhances usability; not critical but boosts user satisfaction.
5	Mechanic Locator	Medium	High	**Could Have**	Adds value but not core to diagnostics;

					easy to implement with maps API.
6	System Access (Login/Fees)	High	High	**Must Have**	Required for monetization and access control.
7	System Management via API	High	High	**Must Have**	Backbone for data operations; essential for scalability.
8	Notifications and Alerts	Medium	High	**Should Have**	Improves user engagement; technically easy via mobile push notifications.
9	Data Display & History	High	High	**Must Have**	Users must see current and past faults; improves long-term use.
10	Settings and Preferences	Medium	High	**Could Have**	Useful for personalization, but not essential to initial operation.

D. CLASSIFICATION OF REQUIREMENTS (FUNCTIONAL AND NON-FUNCTIONAL)

Functional requirements

1) **Dashboard Light Scanning**
   The app shall allow users to scan dashboard warning lights using their phone camera.
2) **Light Symbol Interpretation**
   The app shall analyze scanned symbols and display the fault meaning and suggested actions.
3) **Engine Sound Diagnosis**

   The app shall record engine sounds via the phone's microphone and analyze them for abnormal patterns (e.g., knocking, misfire, hissing).

4) **Fault Display Interface**

   The app shall display the detected faults and diagnosis results on the mobile interface.

5) **Fault History Logging**

The app shall maintain a fault history log for each user, allowing them to review past diagnostics.

6) **Maintenance Reminder System**

The app shall send maintenance reminders based on time or mileage input.

7) **Mechanic Locator**

The app shall allow users to search for nearby mechanic workshops using GPS.

8) **Repair Suggestions**

The app shall provide repair suggestions for common faults, including step-by-step guides.

9) **User Account Management**

The app shall enable user registration, login, and logout, with data secured by a backend API.

10) **Profile and Settings Management**

The app shall allow users to manage profile settings such as name, email, and language preference.

11) **Subscription and Payment**

The system shall allow in-app subscription and payment for premium features.

12) **User Notifications**

The app shall notify users via push notifications for critical faults, maintenance alerts, and login confirmations.

13) **Language Selection**

The app shall allow the user to select the app's language (e.g., English, French).

Non-functional requirements

1) **Usability**
   The app shall have an intuitive and user-friendly interface, accessible to both technical and non-technical users.

2) **Performance**

The app shall return dashboard light analysis and sound diagnostic results within 3 seconds for optimal user experience.

3) **Reliability**

   The system shall maintain at least 99.5% uptime to ensure consistent service availability

4) **Security**

   All user data (including login credentials, payment information, and diagnostic history) shall be encrypted both in transit and at rest.

5) **Scalability**

   The backend system shall support a growing number of users and shall handle up to 100,000 concurrent sessions efficiently.

6) **Compatibility**

   The app shall be compatible with Android version 8.0 and above, and iOS version 12.0 and above.

7) **Maintainability**

   The system shall follow a modular and well-documented codebase to ease future updates, debugging, and feature extensions.

8) **Localization**

   The app shall support multiple languages, including English and French, to serve a wider range of users.

9) **Data Storage**

   All user and diagnostic data shall be securely stored in a cloud-hosted database with backup and recovery features.

10) **Legal Compliance**

The system shall comply with regional and international data protection regulations such as GDPR (if used in applicable regions).

E. DEVELOP THE SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT (SRS)

Project Title: Car Fault Diagnostic Mobile Application
Version: 1.0
Date: April 30, 2025

1. Introduction

1.1 Purpose

This SRS document provides a detailed specification for a mobile application designed to help users diagnose car faults through dashboard light scanning, engine sound analysis, and other supportive features such as mechanic locator, maintenance reminders, and repair suggestions. It defines functional and non-functional requirements and serves as a foundation for development, validation, and future enhancement.

1.2 Scope

The Car Fault Diagnostic App will:

- Scan and interpret dashboard warning lights using phones camera.
- Analyze engine sounds to detect potential issues.
- Offer maintenance reminders and repair guidance.
- Provide mechanic locator features using GPS.
- Enable user account management and notifications.
- The app will be available for Android and iOS platforms.
- Operate without requiring external hardware like OBD-II scanners.
- Be user friendly and accessible to non-technical users.

1.3 Intended Audience

This document is intended for developers and engineers building a similar system, UI/UX designers, testers, project stakeholders and clients, future teams who may maintain or upgrade the app

1.4 Definitions, Acronyms, and Abbreviations

- **API**: Application Programming Interface
- **FR**: Functional Requirement
- **NFR**: Non-Functional Requirement
- **SRS**: Software Requirements Specification
- **UI**: User Interface
- **OBD-II**: On-Board Diagnostics, version 2

2. Overall Description

2.1 Product Perspective

The app functions as a standalone diagnostic assistant integrating mobile camera and microphone hardware, GPS services, and backend API support for data handling and user management.

2.2 Product Functions

- Scan dashboard lights and interpret faults
- Record and diagnose engine sounds
- Display results and fault history
- Notify users for upcoming maintenance
- Manage user profile and subscription

2.3 User Characteristics

Target users include:

- Car owners and drivers
- Vehicle maintenance professionals
- Mechanics and garages

Users are expected to have basic smartphone literacy.

2.4 Constraints

- Limited to smartphones with camera and microphone
- Requires internet access for GPS and database updates
- Must adhere to GDPR and regional data laws

3. Specific Requirements

3.1 Functional Requirements

1. **Dashboard Light Scanning**
    - Scan car dashboard lights using the camera.
2. **Light Symbol Interpretation**
    - Display fault meaning and possible actions.
3. **Engine Sound Diagnosis**
    - Record engine sounds and detect issues.
4. **Fault Display Interface**
    - Show results in an interactive format.
5. **Fault History Logging**
    - Save and retrieve past diagnostic results.
6. **Maintenance Reminder System**
    - Send notifications based on mileage/time.
7. **Mechanic Locator**
    - Display nearby garages using GPS.
8. **Repair Suggestions**
    - Offer step-by-step guidance on fixing common issues.
9. **User Account Management**
    - Support registration, login, and logout with secured API.
10. **Profile and Settings Management**
    - Allow updates to user information and preferences.
11. **Subscription and Payment**
    - Handle registration fees and renewal charges every 3 months.
12. **User Notifications**
    - Send alerts for confirmations, receipts, and maintenance.
13. **Language Selection**
    - Switch between English and French.

3.2 Non-Functional Requirements

1. **Usability**
    - Easy navigation and clean interface.
2. **Performance**
    - Return diagnostic results in under 3 seconds.
3. **Reliability**
    - Ensure 99.5% uptime.
4. **Security**

    - Encrypt all user data and transactions.
5. **Scalability**
    - Support 100,000+ users.
6. **Compatibility**
    - Android 8.0+, iOS 12.0+.
7. **Maintainability**
    - Modular codebase for easy updates.
8. **Localization**
    - Multilingual support.
9. **Data Storage**
    - Cloud-based, secure, and backed up.
10. **Legal Compliance**
    - Follow data protection standards.

4. Requirement Analysis Process

4.1 Review and Analysis

- Requirements were collected from surveys and market needs.
- Verified for clarity, completeness, feasibility, and dependencies.

4.2 Inconsistencies and Ambiguities

- Clarified vague descriptions (e.g., types of engine sounds).
- Resolved conflicts (e.g., payment models and free trial expectations).

4.3 Requirement Prioritization

High Priority: Dashboard scan, engine sound analysis, fault display, user registration
Medium Priority: Mechanic locator, notifications, multilingual support
Low Priority: Customization features (themes, UI layouts)

4.4 Classification

- **Functional Requirements**: FR1 – FR13
- **Non-Functional Requirements**: NFR1 – NFR10

5. Validation with Stakeholders

Stakeholders included:

- Vehicle owners
- Auto repair professionals
- Mobile app developers

Validation steps:

- Stakeholder meetings to confirm needs
- Walkthroughs of use cases and mock interfaces
- Feedback loop to adjust misunderstood or missing requirements
- Formal sign-off from clients before design began

6. Appendices

References

- Sommerville, I. (2016). Software Engineering.
- Norman, D. (2013). The Design of Everyday Things.
- Pressman, R. (2014). Software Engineering: A Practitioner's Approach.
- Chen et al. (2020). AI for Automotive Diagnostics Research.

7. Conclusion

This SRS document provides a comprehensive foundation for the development of the Car Fault Diagnostic App. It outlines all necessary functional and non-functional requirements, clearly analyzes priorities and feasibility, and incorporates feedback from stakeholders to ensure relevance and usability. The system, once implemented, will bridge the diagnostic gap for vehicle users by enabling quick, reliable, and user-friendly fault identification and resolution from a mobile device.

F. VALIDATE REQUIREMENTS WITH STAKE HOLDERS

This is to validate the software/system requirements gathered for the diagnostic system by reviewing them with key stakeholders, identifying gaps, resolving ambiguities, and ensuring mutual agreement before design and development.

1. SKATEHOLDER IDENTIFICATION

Stakeholder	**Role**	**Interest**
Car user	End user	Uses system to report car faults
Car technician	End user	Uses system to detect and diagnose faults
System engineer	Developer	Builds system logic and integration
Marketing Agent	Sponsor	Ensure product meets market needs

2. STAKEHOLDER VALIDATION SESSION

- **Date**: April 27, 2025
- **Method**: Online and onsite meeting
- **Participants**:
    - Mr. Nyanga Ngoh (Technician)
    - Miss. Kuo Kelsy (Marketing agent)
    - Mme. Fotabong Mariana

Requirement 1: Sensor Fault Detection

- Original requirement was unclear about what “faulty” meant
- Stakeholders requested thresholds and timing
- Updated to: “System detects out-of-range values (e.g., pressure > 150 psi, O2 < 10%) and logs events within 2 seconds”

Requirement 2: Diagnostic Trouble Code (DTC) Generation

- Initial requirement lacked formatting and storage clarity
- Stakeholders confirmed need for OBD-II format and timestamping
- Revised to: “System generates DTCs in P0xxx OBD-II format and stores them with timestamp in flash memory”

Requirement 3: Dashboard Fault Display

- Original requirement aimed to show all faults

- Stakeholders recommended focusing on high-priority errors only
- Revised to: “System shows only high-severity DTCs (e.g., engine, brakes) with blinking icon and fault description”

3. Feedback Resolution

Requirements were:

- Clarified for readability
- Filtered for critical importance (for dashboard)

4. Validation Checklist

Each revised requirement was:

- Correct and accurate
- Complete in terms of coverage
- Unambiguous in language
- Testable via system behavior
- Accepted by all stakeholders

5. Final Stakeholder Approval

- Mr. Nyanga Ngoh approved via a message note
- Ms. Kuo Kelsy gave verbal confirmation during meeting
- Mme. Fotabong Mariana gave verbal confirmation during meeting

CONCLUSION

The requirement analysis for the automotive diagnostic system has been successfully completed. Through stakeholder engagement and careful evaluation, we ensured all functional and non-functional needs are clearly defined, feasible, and aligned with user expectations. This analysis lays a strong foundation for system design and development, minimizing future risks and ensuring project success.

REFRENCES

1. Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson Education.
2. IEEE. (1998). *IEEE Recommended Practice for Software Requirements Specifications* (IEEE Std 830-1998).
3. Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner's Approach* (8th ed.). McGraw-Hill.
4. Ian Sommerville. (2011). *Requirements Engineering Processes and Techniques*. Addison-Wesley.
5. Interview Notes with Stakeholders (Ade, Sade, Musa) – April 2025