UNIVERSITY OF BUEA

REPUBLIC OF CAMEROON

PEACE-WORK-FATHERLAND

P.O. Box 63,
Buea, South West Region
CAMEROON
Tel : (237) 3332 21 34/3332 26 90
Fax: (237) 3332 22 72

FACULTY OF ENGINEERING AND TECHNOLOGY

DEPARTMENT OF COMPUTER ENGINEERING

Task 3: Requirement Analysis Report

Project Title: Car Fault Diagnosis Mobile Application

Course Title: Internet Programming and Network Programming

Course Code: CEF 440

By Group 2:

BERINYUY CLETUS FE21A148
EFUETLATEH JOAN FE22A197
ETAPE NGABE FE22A210
ETIENDEM PEARL FE22A211
TATA GLEN FE22A309

2024/2025 Academic Year

Table of Contents

1 Introduction

This report presents the requirement analysis phase of the mobile application project aimed at diagnosing car faults through dashboard warning lights and engine sound analysis. After gathering user and stakeholder requirements in Task 2, this phase focuses on reviewing and analyzing the completeness, clarity, feasibility, and consistency of those requirements. The goal is to ensure that the requirements are well-defined, prioritized, and formally documented to serve as a blueprint for the system's design and development. This step is essential to prevent miscommunication, reduce development errors, and ensure the final product meets user needs.

2 Review and Analysis of Requirements

To ensure that the gathered requirements align with the goals of our mobile diagnostic application, we assessed them using the following criteria:

2.1 Completeness

The requirements cover all key aspects of the system, including:

- Identification of warning lights
- Analysis of engine sounds
- User feedback mechanisms
- Offline functionality for privacy
- Ease of use for non-technical users

No major functional area was found missing based on user inputs, observations, and competitive analysis. Additional minor features may be included in future iterations after initial deployment and feedback.

2.2 Clarity

The gathered requirements were clearly stated in simple terms, making them understandable for both technical and non-technical team members. Terms like "engine sound analysis" and "dashboard light detection" were specified with examples to avoid ambiguity. Some survey responses initially used vague terms, but follow-up interviews helped clarify those into actionable requirements.

2.3 Technical Feasibility

The development of the mobile car diagnostics application is technically feasible using modern cross-platform tools and libraries. The team has selected React Native for app development, offering robust support for building performant mobile apps on both Android and iOS platforms using a shared JavaScript codebase.

Key features such as camera and microphone access can be efficiently handled using libraries like react-native-camera and react-native-audio-recorder-player. Image recognition for dashboard lights will utilize on-device machine learning with models integrated using TensorFlow.js or ML Kit. Engine sound analysis will rely on digital signal processing (DSP) and pre-trained models capable of running efficiently on mobile devices.

While real-time audio processing and ensuring offline capabilities present challenges, the existing ecosystem around React Native including community support and integration tools makes the project technically achievable within the intended scope.

2.4 Dependency Relationships

The following dependencies are crucial for the successful implementation of the mobile application:

Component	Dependency Type	Description
react-native-camera	Software	Used to capture images of the dashboard for light analysis.
react-native-audio or react-native-audio-recorder-player	Software	Used to record engine sounds for diagnostic purposes.
On-device ML Models (via TensorFlow.js or ML Kit)	Software	Enables classification of images and audio without needing server access.
Mobile Sensors Access	Hardware/Permission	Required to access camera and microphone with proper permissions.
External Libraries	Software	Used for UI design, animations, form handling, and data visualization.

Where applicable, fallback mechanisms or alternate methods will be provided to handle device or permission-related constraints.

3 Identify Inconsistencies, Ambiguities, and Missing Information

During the analysis of the gathered requirements, the following issues were identified and addressed to ensure a consistent and realistic development process:

3.1 Inconsistencies

Some user feedback conflicted with each other. For instance:

- A few respondents preferred voice output for diagnostics, while others preferred only visual indicators.
- Some car owners wanted detailed technical reports, whereas others requested simplified results due to limited technical knowledge.

Resolution: We decided to include both basic and advanced views in the app, allowing users to toggle based on preference.

3.2 Ambiguities

Several responses and initial requirements contained vague terms. For example:

- "Identify all car problems using sound" was too broad and unrealistic with current hardware.
- "Offline app" was initially unclear on which functionalities must work without internet.

Resolution: Clarified that engine sound classification will focus only on specific faults (e.g., knocking, misfiring), and offline functionality will cover diagnosis and viewing results, while online mode supports updates and user feedback uploads.

3.3 Missing Information

- There was no initial mention of accessibility features, such as large text or screen reader compatibility.
- No clear specification of supported car models or engine types.
- Users did not specify how they wanted to receive fault solutions (text, image, audio).

Resolution: These gaps were filled by:

- Adding a general accessibility guideline to the UI design phase.
- Limiting the prototype to common vehicle types used locally.
- Including an option to choose the preferred format for fault solutions in the settings.

4 Prioritization of Requirements Based on Importance and Feasibility

To ensure effective planning and implementation, the identified requirements were categorized and prioritized based on two main criteria: **importance to the user** and **technical feasibility within the project timeline and available resources**. The MoSCoW prioritization method was used (Must have, Should have, Could have, Won't have for now).

4.1 Must Have (High Importance, Hight Feasibility)

- Basic dashboard light diagnosis using camera input
- User-friendly interface for selecting car issues
- Offline access to diagnostic results
- Local storage of past diagnosis history
- Engine sound detection for selected common faults
- Simple visual indicators for alerts (e.g., red, yellow)

4.2 Should Have (High Importance, Moderate Feasibility)

- User account system for data personalization
- Integration of reverse-engineered feedback from existing diagnostic apps
- Step-by-step instructions or recommendations based on diagnosis

4.3 Could Have (Moderate Importance, Moderate/Low Feasibility)

- Voice output of diagnostic results
- Support for multiple languages
- In-app feedback system for reporting false positives
- Adaptive UI based on user preferences
- Toggle between beginner and expert views

4.4 Won't Have for Now (Low Importance, Low Feasibility

- Integration with the car's onboard diagnostics (OBD-II)
- Real-time remote assistance from professionals
- Predictive analytics based on usage patterns
- Cloud sync across multiple devices

This prioritization helps ensure that the core functionality is delivered efficiently while keeping room for enhancements as time and resources permit.

5 Classification of Requirements

In this section, the gathered requirements are classified into **Functional Requirements** and **Non-Functional Requirements** to better structure development priorities and ensure system reliability.

5.1 Functional Requirements

These define what the system should do, and the core functionalities that the app must support to meet user expectations:

1. **User Registration (Sign-Up)**
    - The system shall allow users (car owners and mechanics) to register using their name and password.
    - The system shall display appropriate error messages for invalid inputs.

2. **User Authentication (Login)**
    - The system shall allow registered users to log in using their credentials.

3. **Dashboard Indicator Scanner**
    - Allow users to scan dashboard lights using their phone camera.
    - Use computer vision to recognize and classify the lights.

4. **Engine Sound Analysis**
    - Record engine sounds through the phone's microphone.
    - Analyze sounds using AI to detect patterns linked to faults (e.g., knocking, squealing).

5. **Fault Detection and Interpretation**
    - Display the meaning of each detected symbol/sound
    - Include possible causes and the urgency of the issue

6. **Multiple Fault Detection**

- Recognize multiple warning lights in a single scan
- Provide maintenance tips.

7. **Recommendations and Repairs**
    - Offer suggested repairs or next steps.
    - Provide maintenance tips.

8. **Video Tutorials Integration**
    - Display embedded or linked YouTube videos from certified experts.

9. **Offline and Online Modes**
    - Basic features (e.g., recognition of common lights/sounds) should work offline.
    - Online mode should allow access to updated fault databases and videos.

10. **User Interface**
    - Simple and intuitive UI.
    - Allow users to navigate diagnostics and solutions easily.

11. **Mechanic Contact Retrieval**
    - The system shall retrieve a list of the nearest available mechanics based on the car's current location.

12. **History Log**
    - The app shall store previous diagnosis records for offline viewing.

13. **Survey and Feedback Form**
    - The app shall include a feedback system to gather user responses for improvement.

5.2 Non-Functional Requirements

These describe how the system should perform and the quality attributes it must meet:

1. **Performance**
    - Fast image and sound processing with minimal lag.

- Quick diagnosis within seconds.

2. **Accuracy**
    - High accuracy in recognition of warning lights and sound anomalies.
    - Continuous model training to improve predictions.

3. **Scalability**
    - Ability to expand the database of sounds, lights, and issues.

4. **Security**
    - Protect user data such as recordings and vehicle history.
    - Adhere to privacy policies and permissions.

5. **Reliability and Availability**
    - Ensure the app works under varying phone conditions (e.g., low battery, poor lighting, poor connectivity).
    - Minimal crashes or bugs.
6. **Compatibility**
    - Compatible with a wide range of Android and iOS devices.
7. **Maintainability**
    - Code should be modular and easily updatable.
    - AI models should be upgradable.
8. **Usability**
    - The app should be intuitive and easy to navigate, even for users with minimal tech skills.
    - Language support or visual aids for better understanding.

6 Development of the Software Requirements Specification (SRS)

To ensure our project is guided by a well-structured and agreed-upon set of requirements, we developed a Software Requirements Specification (SRS) document. The SRS provides a detailed and organized description of all functional and non-functional requirements identified during the requirement gathering and analysis stages.

The document includes clear definitions of the system's purpose, scope, user requirements, system features, constraints, and external interfaces. It serves as a formal agreement between stakeholders and developers and ensures all parties have a shared understanding of the system to be built.

Creating the SRS helps minimize ambiguities, identify inconsistencies, and ensure completeness and feasibility of the requirements. It also lays the groundwork for the system's design, implementation, testing, and validation phases. The SRS is maintained as a living document and will be updated as needed throughout the development lifecycle.

7 Conclusion

The requirement analysis process has helped transform user needs and collected data into a structured and actionable set of requirements. By reviewing the gathered information for completeness, clarity, feasibility, and dependencies, we were able to identify and resolve inconsistencies and ambiguities. Functional and non-functional requirements were clearly classified, prioritized, and validated with stakeholders to ensure alignment with real-world expectations. Furthermore, a formal Software Requirements Specification (SRS) has been outlined to guide the development process. This analysis serves as a critical foundation for the system design and ensures that the mobile diagnostic app is user-focused, feasible, and effective in addressing key challenges faced by drivers.