Requirements Document for OBD-II Insight Bot (Group 18)

Gleb Suchkov Kalmykov (hcwm13), Muhammad Ariz Muhammad Akmal (pgbh35), Kai Chung Matthew Chu (vbzc55), Jaden Zhen Han Yuen (cwms51), Fred Andrew Viner (whhx62)

Date: 18th November 2025

Document Version: 3.1.0

1 – Introduction

1.1 - Overview and Justification

With the constant rise of innovative technologies, modern vehicles begin to generate a vast amount of diagnostic data via their OBD-II (On-Board Diagnostic, Version II) systems. A complex system that gives rise to many questions and issues for vehicle owners. Our aim is to close this diagnostic divide, the gap between the complexity of vehicle health information and user comprehension.

IBM has commissioned us to create "OBD InsightBot", a conversational AI chatbot that is designed to assist in reading and analysing OBD-II logs. The primary client contact for this project is John McNamara, Product Owner at IBM.

The main goal for this project is to develop a system that can correctly parse OBD-II logs into readable data, extract the key metrics such as engine RPM, coolant temperature, and fault codes, and then generate natural human-like language explanations for any user queries.

This document is divided into 3 main sections. Section 1 (Introduction), split into 1.1 Overview and Justification, 1.2 Project Scope, 1.3 System Description; discusses the overall outlines of the system along with the problems that the solution aims to solve. Section 2 (Behavioural Requirements) split into 2.1 how the specification was arrived at, and 2.2 Behavioural specifications which discuss the core features through user stories. Section 3 (Project Management) covers the identified risks, development methodology employed and the project timeline.

1.2 - Project Scope

Currently, 76% of UK motorists claim to have no idea what to do if their car breaks down, with 54% relying on professional assistance when it comes down to any vehicle maintenance problem. With the automative diagnostic software market being valued at $6.2 billion in 2024, IBM will be able to expand its market share by simplifying and globalising the diagnostic software to the average consumer.

OBD InsightBot is a local application that will accept the OBD-II logs as input and will prompt the user to ask questions regarding any concerns with their vehicle, answering any questions and follow-ups about the conditions that it can reliably identify, in a natural and conversational tone, helping the consumer make informed maintenance decisions.

In alignment with IBM's objectives our key goals for the project are accessibility, create a simple, easy to use user interface that feels intuitive to use; and reliability, to have our users feel confident that they have correctly identified the issue and ease their worries.

1.3 - System Description

We are to create a stand-alone chatbot application that can run on a Windows based operating system. After launching the software, the user will be prompted to register for an account (if they do not have any existing data) and then sign-in before being able to access the chatbots features. The user can then start a new chat (with the requirement of uploading a valid OBD-II log) and will then be redirected to the main chat area. Once the logs have been uploaded, key-metrics will be parsed in the background, such as “Coolant Temperature” or “Vehicle Speed” and stored within the program’s memory.

After the OBD-II log has been parsed successfully, the user can then ask questions regarding the data uploaded, via the text chat interface. Our system will utilise IBM Granite models to create a RAG (Retrieval-Augmented Generation) pipeline such that questions can be answered by the model utilising the parsed OBD-II data knowledge base. Once a response has been generated using the natural language generation capabilities of IBM Granite, it is fed back to the text chat interface. The user then has the option to ask follow-up questions if they deem the response to not be appropriate. Additionally, the user will also be able to ask their questions via a dictation feature, or have their responses read back to them aloud via text-to-speech.

Finally, all the user's chats can be accessed via a submenu; opening, renaming, deleting or optionally exporting them to a `.txt` file.

Alternative solutions:

- **OBDAI**

  Released January 2025. More fleshed out system than our project scope. The app is used alongside a physical scanner that when inserted into the OBD-II port of a vehicle transmits **live** data to mobile app or desktop application. Features include:
  - Explains fault codes and shows how to fix them, either with a written guide or link to a video demonstration.
  - Provides real-time monitoring of vehicle statistics such as “Engine RPM”.
  - Highlights predicted maintenance required based on current data.
  - Hardware is required; no option to upload log file.
- **ChatGPT, Google Gemini, etc,**

  OBD-II data can be uploaded to chatbots like ChatGPT which then utilise LLMs (Large Language Models) to interpret the data and provide answers based on the data provided. The features of ChatGPT (for the most part) match the features included in our system, so why not just that instead of our application? Well:
  - Privacy concerns: data uploaded to LLMs can be used to train and improve the model further.
  - No guarantee that the response will be based on facts. There are many examples of ChatGPT providing incorrect responses.
  - Possible not as accurate as our proposed RAG pipeline (though testing to verify this after the solution has been built will be required)

Integration requirements

This project does not require any specific system to be implemented with; however, our client Mr. McNamara suggested a possible final integration with TORCS (The Open Race Car Simulator). The nature of this integration has not yet been defined; however, it would involve using TORCS to help visualise fault codes present in the log files to give the target user (inexperienced) a better idea of the under-lying issues with their vehicle.

2 – Behavioural Requirements

2.1 - Requirements elicitation

Mr. McNamara kindly provided us with an office hour every week where he was available to answer any questions. Although the project brief provided was very thorough, these regular meetings with our client proved vital to the requirements development process.

Below is what we considered to be the most important questions and the responses provided:

- What do you define as the key metrics of the OBD-II logs? - Any metric that could prove useful in providing information about the vehicle to the user.
- Is the chatbot expected to understand manufacturer-specific fault codes? - It falls to the decision of the group and what we think we can achieve within the time frame.
- Are there any additional features that you would like to be implemented that were not mentioned in the brief? - Yes, it would be great to have a dictation feature, such that the user could ask questions about the logs using their voice. Similarly, a text-to-speech system that reads chatbot responses aloud would be nice. Additionally, there should be a way to separate chats based on specific vehicle and/or person using the application.

As a group, we made decisions based on the added information we were given. We decided against manufacturer-specific fault codes; research into the topic revealed different vehicle manufacturers can use the same code to identify completely different faults, as well as many of these faults being locked behind proprietary software used by OBD-II diagnostic tools in merchant garages.

Initially, the user stories we wrote were far too vague, coming up with stories such as "As a user, I want to diagnose problems with my vehicle so that I can fix them" and "As a user, I want to see how important the issues with my vehicle are, so I know whether to be concerned". Switching to the **INVEST** criteria, we dissected these initial ideas, asking what it means for there to be a "problem" with a vehicle and to define different levels of "concern".

We then used a Behaviour-Driven Development (BDD) approach to convert the refined user stories into features in Gherkin pseudocode. As a group, we broke down each feature into different scenarios utilising the `Given-When-Then` syntax. In the beginning, we found writing the `When` part for each scenario the toughest, as we had to get in the mindset of generalising our condition rather than writing about the specific implementation. For example, for **BR2.1**, the first condition was "`When the user clicks the "New Chat" button`", which was too specific. Reviewing the Gherkin docs and examples helped us understand and re-write.

We presented these features and key scenarios to our client during the office hours to validate that our understanding of the requirements provided to us were correct before we started development.

2.2 – Behavioural Requirements

BR1 - As an average user I want to have an account so that I can access the application, organise chats between different vehicles.

```
Feature: Account Management

    # BR1.1
    Scenario: User creates an account
        Given the user is on the sign-up page
        When the user enters an account name and password
        Then the user has an account

    # BR1.2
    Scenario: User logs-in to their account
        Given the user is on the log-in page
        And they have a valid, registered account
        When the user enters their valid username and password
        Then the user should be granted access to the application

    # BR1.3
    Scenario: User logs-out of their account
        Given the user is currently signed into their account
        When the user presses the sign-out button
        Then the user is signed out and redirected to the log-in
        page

    # BR1.4
    Scenario: User deletes their account
        Given the user has confirmed they want their account
        removed
        When the user has pressed a button to confirm deletion
        Then user's information is deleted from local database
```

This feature is designed to meet the client's criteria of being able to organise the chats of different vehicles. With an account, it will also be much easier to return to the application after an extended period, as all the relevant chat history will be available in one place.

This feature is a **MUST** have; it is a core requirement of our system to allow users to create accounts to help better distinguish between different vehicles. This would not be possible if not for an account management system.

The Scenario "User logs-in to their account" (BR1.2) is central to the main purpose of the application, enabling a returning user to access their specific data without having to filter through irrelevant chats.

BR2 - As an average user I want to create a new chat so that I can upload my OBD-II logs and ask questions about it.

```
 Feature: New Chat Creation with Log Upload

    # BR2.1
    Scenario: User uploads a valid log file
       Given the user logged-in and on the home page
       When the user starts a new chat
       And they choose a valid OBD-II log file in ".csv" format
       And they attempt to upload it
       Then the file should be uploaded and processed
       And the user should be redirected to the new chat screen
       with the provided context of the uploaded log

    # BR2.2
    Scenario: User uploads an invalid file type
       Given the user is logged in
       And they are starting a new chat
       When the user selects a non ".csv" file
       And they attempt to upload it
       Then the file should be rejected by the system
       And the user should be prompted to upload a valid log file

    # BR2.3
    Scenario: User uploads a valid file type with bad data
       Given the user is on the file upload menu
       When they select a file of a supported type but contains no
       OBD-II data
       And they attempt to upload it
       Then the system should process the file and fail to find
       any valid data
       And the user should be prompted to upload a valid log file

    # BR2.4
    Scenario: User is logged out and tries to start a chat
       Given the user is not authenticated with an account
       When the user attempts to start a new chat
       Then they should be redirected to the log-in page
```

Therefore, this feature is a **MUST** have and is core to the clients demands. Our project brief made it clear that the system should be able to process static OBD-II log files. It is vital that the features functions as expected and can handle any erroneous data provided.

The Scenarios BR2.2 and BR2.3 are equally as important as the accepting scenario. The client made clear that the user of this software may not be a technical expert and thus potentially could upload the wrong file. These scenarios ensure a graceful failure.

BR3 - As an average user I want to be able to manage my chats so that I can keep track of the most relevant ones.

```
 Feature: Chat history

    Background:
       Given the user is logged in to a valid account
       And the user has had previous chats with the bot

    # BR3.1
    Scenario: User views chat history
       Given the user is on the chat history page
       When the user clicks on corresponding dates
       Then chat logs on that day is shown

    # BR3.2
    Scenario: User deletes chat history
       Given the user has chosen a selection of chat to be deleted
       When the user has confirmed they want their selection of
       chat deleted
       Then chat logs in the chosen selection are deleted

    # BR3.3
    Scenario: User renames chat log
       Given the user is on the chat history page
       When the user right clicks a chat log
       And clicks the "rename" button
       And enters a name
       Then the chat log is renamed

    # BR3.4
    Scenario: User exports chat log
       Given the user is on the chat history page
       When the user right clicks a chat log
       And clicks the "export" button
       And choose a format to export as
       Then the chat log is downloaded in that format
```

This feature is a **MUST** have. This enables a user to look at the vehicle's problems that occurred in the past and organising them. Having this feature is also vital due to its ability to store multiple vehicles' data at a time, which allows a user to keep track of these problems simultaneously.

The scenario BR3.1 is especially important for showing the main purpose of this feature, as it allows the user to see the previous chats with their data with ease whenever and wherever, allowing for quick retrieval of data.

BR4 - As an average user I want to ask general questions via a text interface about my vehicle (such as a summary of engine status) so that I can gain insights into its condition.

```
Feature: General Vehicle Status Queries

    Background:
        Given the user is logged in to a valid account
        And the user has provided a valid OBD-II log file
        And the system has parsed key metrics from the data
        And the log file does not contain any fault codes

    # BR4.1
    Scenario: User asks for a summary when all metrics are at a
    normal level
        Given relevant metrics such as "Engine RPM" are at their
        expected levels
        When the user asks for an overview of their vehicle health
        Then the system will utilise IBM Granite to generate a
        natural language response
        And the response should convey that the vehicle appears
        "healthy"
        And the response should contain relevant information like
        current engine status

    # BR4.2
    Scenario: User asks for a summary when one or more metrics are
    abnormal
        Given that a parsed metric is flagged as being different
        than usual like, "Engine Coolant Temperature" is high
        When the user asks for a summary of their vehicle health
        Then the system will use IBM Granite to create a natural
        language response
        And the response should make clear that despite the lack of
        fault codes, these certain metrics are abnormal
        And recommendations should be provided to both maintain
        human safety and suggest ways to "fix" the vehicle

    # BR4.3
    Scenario: User asks a question about something that is not
    contained within the log
        Given that the log file does not contain information about
        a specific metric, like "Climate Control"
        When the user asks a question regarding something that
        isn't covered in the OBD-II data
        Then the system will use IBM Granite to generate a natural
        language response
        And the response should make clear that information
        regarding that metric was not found within the log
        And it should explain that the chatbot can only provide
        insights into present data
```

The implementation of this feature is an absolute core to the system's functionality, as allowing users to enquire the chatbot on anything regarding their current vehicle status is in tune with the client's expectations of the project as per the initial project outline document.

This feature is a **MUST** have; playing the role in making sure users have a base form of communication (via text interface) which ties in neatly with features BR2 and BR3 as well.

The scenario BR4.2 is a good example of usability, as it helps users to take immediate action and schedule maintenances for their vehicles for longevity concerns.

BR5 - As an average user I want vehicle fault codes to be explained to me so that I can diagnose any issues with my vehicle.

```
 Feature: Fault Code Explanation

    Background:
       Given the user is logged in to a valid account
       And the user has provided a valid OBD-II log file
       And the system has successfully parsed fault codes from the
       log

    # BR5.1
    Scenario: User asks to explain a specific, generic fault code
       Given the log contains a certain fault code, like "P0169"
       When the user asks to explain a certain fault code
       Then the system will use an IBM Granite model to generate a
       natural language response
       And the response should define the code, i.e. "P0169 –
       Incorrect Fuel Composition"

    # BR5.2
    Scenario: User asks for a general summary of all generic fault
    codes
       Given the log contains multiple different fault codes
       When the user asks a question like, "What is the problem
       with my vehicle?"
       Then the system utilises an IBM Granite model to create a
       natural language response
       And the response should list and define all fault codes

    # BR5.3
    Scenario: User asks for an explanation of a manufacturer-
    specific fault code
       Given the log contains a non-generic fault code
       When the user asks for an explanation of said fault
       Then the system uses an IBM Granite model to generate a
       natural language response
       And the response should explain that only generic OBD-II
       fault codes are supported

    # BR5.4
    Scenario: User asks about fault code when none exist in the log
       Given the log does not contain any fault codes
       When the user asks any question regarding them
       Then the system uses an IBM Granite model to generate a
       natural language response
```

This feature is a **MUST** have, as fault codes provide insights to the status of the vehicle, which is the main purpose of the application. This feature is added because the client states that most target users do not know a lot about cars, hence they do not know what the fault code would indicate. Scenario BR5.1 and BR5.2 are great examples, as it really provides deeper insights towards the vehicle's conditions.

BR6 - As an average user who sometimes prefers speaking, I want to dictate my queries so that I can chat hands-free and faster than typing.

```
 Feature: Speech-to-text Dictation

    Background:
       Given the user is logged in to a valid account
       And the user has started a new chat or continuing a
       previous one
       And the user has granted microphone permission
       And the user has provided a valid OBD-II log file
       And the system has parsed key metrics from the data

    # BR6.1
    Scenario: User starts and completes a basic voice message
       Given the user clicks the "Dictate" microphone button
       When the user speaks
       Then the IBM Granite SST model converts it to transcript
       And when the user presses "Stop"
       Then the system displays the transcript in the text bar
       without auto-sending

    # BR6.2
    Scenario: User continues dictation from the caret position
       Given the text bar already contains text and the caret is
       mid-sentence
       When the user starts dictation and speaks
       Then the transcript is inserted at the caret position
       And existing text after the caret remains unchanged

    # BR6.3
    Scenario: User stops speaking and does not press "Stop"
       Given the user is in "Dictate" mode
       When there is continuous silence for N seconds (e.g., 3s)
       Then the system auto-stops dictation
       And the system displays the last complete sentence to the
       text bar

    # BR6.4
    Scenario: User denies microphone access
       Given the user clicks the "Dictation" button and permission
       is denied by the OS
       When the dictation fails to start
       Then the system shows a clear, actionable error
       And the system provides instructions to enable mic access
       in settings
```

This feature is a **SHOULD** have, as it matches the needs of the client having a dictation feature using Speech-to-text. Although this feature is a standalone and is separate from the "voice mode" feature.

Scenario BR6.2 is a helpful instance as users who generate better ideas when speaking can pause their text queries and continue it by speaking to the chatbot instead.

BR7 - As a hands-free user, I want to converse with the chatbot by voice so that I can speak and hear responses naturally while I drive and am curious about my vehicle's conditions.

```
Feature: Voice Conversation Mode

    Background:
        Given the user is logged in to a valid account
        And the user has started a new chat or continuing a
        previous one
        And the user has granted microphone permission
        And the user has provided a valid OBD-II log file
        And the system has parsed key metrics from the data

    # BR7.1
    Scenario: User starts a voice session and get a spoken reply
        Given the user taps the "Voice Mode" button
        When the user speaks a prompt
        Then the IBM Granite TTS Model transcribes the speech and
        generates a response
        And streams an audible TTS reply to the user

    # BR7.2
    Scenario: User converses with natural turn-taking with
    endpoints
        Given the user is speaking
        When the user stops for the configured pause duration
        (e.g., 3s)
        Then the system detects end pointing
        And the system begins generating and speaking the reply
        without additional user action

    # BR7.3
    Scenario: User uses a wake word to activate the chatbot
        Given the user activates wake word mode
        When the user says the wake word (e.g., "Hey InsightBot")
        Then the system enters listening state
        And the user can start a chat hands-free while driving
```

This feature encapsulates the client's vision of having a driver of a vehicle being able to ask questions about the data without the need to type any questions in more of an old-fashioned vehicle way, with a laptop in the back. Stating that the user can have full back and forth conversations while being able to focus on the road and without a need to type anything.

The Voice Mode is a **SHOULD** have; it replicates the application's core BR4 feature but is aimed towards catering to the various users who prefer to communicate and have conversations in a linguistic fashion rather than through a text-reply interface.

The BR7.2 scenario really captures the essence of how natural conversations are carried about, making each chat very engaging and mindful.

BR8 - As an average user I want the responses to my questions to be categorised so that I can prioritise critical information.

```
 Feature: Danger level

    Background:
       Given the user is in a chat
       And the user has provided a valid OBD-II log file
       And the system has parsed key metrics from the data
       And the system has generated a response

    # BR8.1
    Scenario: Information is categorised as critical
       Given the information indicates that the vehicle has a
       severe issue that requires immediate action
       When the chat bot prints its response
       Then the information is highlighted in red

    # BR8.2
    Scenario: Information is categorised as potential danger
       Given the information indicates that the vehicle has a
       moderate issue that requires attention
       When the chat bot prints its response
       Then the information is highlighted in amber

    # BR8.3
    Scenario: Information is categorised as harmless
       Given the information indicates that the vehicle has a mild
       issue that is not emergent
       When the chat bot prints its response
       Then the information is highlighted in green
```

This feature is designed to allow the user to quickly identify the importance of information given by the chat bot. This feature reflects a request from the client, where he suggested the application to be able to categorise information based on how "dangerous" each part of the information could be.

This feature is a **SHOULD** have. Even though the response is readable as it is, highlighting critical information enables the user to identify urgent issues with the vehicle before reading through the whole response.

Scenarios BR8.1, BR8.2 and BR8.3 are exemplary regarding allowing the user to understand what about their vehicle they should prioritise, how to compartmentalise and much more.

3 – Project Management

3.1 – Risks and Issues

	Impact	Low	Medium	High
Possibility	Probable	Small glitch which doesn’t impact core functionality	Poor Communication	Schedule Overruns
	Possible	Non-matching dependencies	Poor Quality Code	Difficulty integrating IBM Granite
	Rare	Misspelled word in release product	Slow feedback from clients	Scope Creep/Change in requirements

Risk	Prevention/mitigation
Non-matching dependencies	Before development, lay out a clear outline of which versions of dependent software to use (i.e., which version/model of IBM Granite).
Misspelled word in release product	Run manual Quality Checks (QC) by carefully reading or use a grammar-checking software
Slow feedback from clients	Utilise the scheduled Friday office hours with Mr. McNamara to get feedback.
Small glitch which does not impact core functionality	Use robust programming techniques and frequent code reviews within the group to catch bugs
Scope creep/Change in requirements	Clearly define project scope and behaviour requirement at the start of the project. Refer to this document.
Poor Quality Code	Set clear coding standards and perform frequent code reviews.
Schedule Overruns	Allocate additional time for unexpected issues (use Gantt chart in Section 3.3).
Poor Communication	Arrange team meetings at a set frequency, e.g. once per week
Difficulty integrating with IBM Granite	Complete course provided by IBM to develop understanding. Plan and allocate additional time for learning.

3.2 – Development Approach

Our team has decided to develop our application using a hybrid of both the Waterfall and Agile models. We think that the waterfall model would be the best fit for this project as it follows a sequential approach which is vital as it ensures each phase is completed before the next begins, ensuring everyone in the team is working together on the same problem. Along with the fact that the project deadlines are already decided, the waterfall approach is more than capable of handling the scheduling demands of this project.

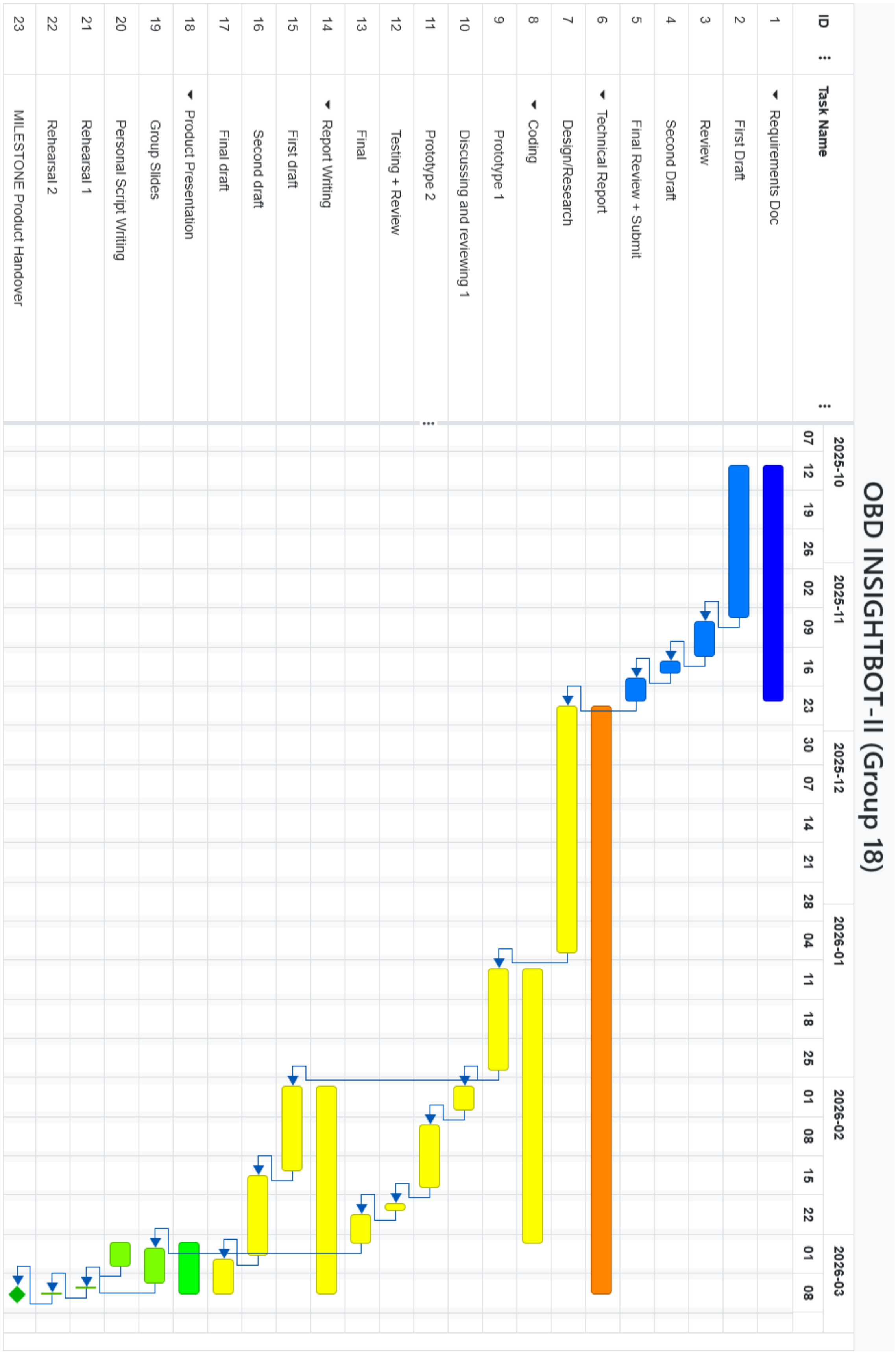
We are aware that the model can be quite rigid and not very flexible during the development process, which is why we are keeping the Agile Model in mind to be used concurrently, especially during the coding phase. Since this project requires consistent communication with the client, this model is well suited for the project. Our client has also asked we give weekly updates on our progress which directly falls under the SCRUM methodology. This suggests that the Agile model might be more suited or this project.

This approach uses pair programming which allows us to be efficient while also ensuring an equal amount of work is assigned to all members of the group. With pair programming, the person observing is also able to share his knowledge with the person who is writing the code (driver), providing instant feedback. Implementation of our project will be intensive at some points, so having someone to bounce ideas off will help to speed up the development process. We are excited to get started with this project and learn about technologies which none of us have any experience with. For that reason, pair programming is suitable to help prevent anyone from becoming overwhelmed during the development process.

We also considered some of the other Software Development Life Cycle models. In the case of the V-Model, one reason we are not using it is its lack of a clear structure. We concluded that using a simpler model combination would improve our development output. During the development, it doesn't provide clear milestones and is much harder to manage compared to other models. We also considered the spiral model, but it is much more difficult to manage and implement for our use case.

We are confident that this hybrid model is the right choice for our group and project. We are excited to learn and develop as software engineers, and we are hopeful that these methodologies can guide us to delivering a product appropriate to all the parties involved.

3.3 – Project Schedule



ID	Task Name
1	◂ Requirements Doc
2	First Draft
3	Review
4	Second Draft
5	Final Review + Submit
6	◂ Technical Report
7	Design/Research
8	◂ Coding
9	Prototype 1
10	Discussing and reviewing 1
11	Prototype 2
12	Testing + Review
13	Final
14	◂ Report Writing
15	First draft
16	Second draft
17	Final draft
18	◂ Product Presentation
19	Group Slides
20	Personal Script Writing
21	Rehearsal 1
22	Rehearsal 2
23	MILESTONE Product Handover