**NEA Analysis**

**Background to/ Identification of problem**

**Background:**

Shaun is an Auto Electrician who works for InCarTec. His work consists of anything electrical within vehicles from the stereo system to the inner ECU of the vehicle.

**Identification of Problem:**

Currently Shaun is using outdated software when fault testing and CAN Reading, and this is due to the price of the programs available. He is looking for software that is simple to understand and use while also providing him with a plethora of data that is needed to diagnose the vehicles. With cars now mostly having electronic controls everywhere, the older software is becoming harder to read effectively with the amount of data coming in. These older programs lack the ability to save profiles of different vehicles (making it quick and easy to diagnose cars of make and model that have already been worked on), filtering and focusing on specific Ids, save a log of all data coming in, and much more.

Many problems also come from the interfaces within the programs, the people operating the software aren’t always going to be the best with technology, and just want a simple to operate system that can clearly display the data that they want to see. The software should be straightforward and simple for the user rather than assuming they understand everything being presented.

**Description of the current system:**

The system that Shaun currently uses is an old outdated and simple piece of software. Currently the software only provides a way to display data coming in from the CAN bus and look back on the data stream, as well as send out data to the CAN bus, but only when manually submitted. The system is also slow to process and display data and can miss data coming in or provide invalid unreadable data. Also, the data is not stored within the system, therefore is not available to Shaun for debugging or reuse after the program is exited or a new data stream is presented.

**Available Systems:**

There are a few programs available to use but usually come with a hefty price. Most of these are heavily optimised and come with many functions, but to users like Shaun and many other people in smaller businesses, these are overkill and usually quite impractical, confusing users with data useless to them, or to many different options, making the software irritating to use.

There are other cheaper options or even free DIY options available to people, but these can also be a bit confusing to operate just due to unoptimized interfaces and functions. These also may not include harder to implement functions that may be required for common jobs, making them obsolete in industrial use.

**Research into CAN Reading and Fault Diagnosing**

**CAN Bus**

The Controller Area Network (CAN bus) is a message-based protocol that is designed to allow the Electronic Control Units (ECUs) to communicate with each other without the need of a host computer. Data is sent serially to all devices within the system but has a priority system in place that insures if two “frames” of data are sent at the same time, the one of highest priority is sent first.

The CAN bus is used in pretty much every vehicle in today’s industry due to the many benefits it provides. CAN systems are very sturdy and robust against interferences such as electric and electromagnetic disturbances, as well as being fully centralized, meaning all data is sent through one point of entry, making reading and debugging possible from a single wire, this also means that less wiring is needed, also making the costs of production reduced. The transmission of data is also very efficient as the “frames” of data are prioritized, meaning higher priority data gets to where it needs to quickly without causing any problems within the system.

Each device is called a “Node” and they all each contain their own Central Processing Unit (CPU), CAN Controller and transceiver. Data is sent onto the network, where any node which has been addressed can access it rather than it being directly sent to each node, and each node can both send and receive data, but not simultaneously.

**Fault Diagnosing**

When diagnosing electrical problems within a vehicle, the CAN bus is a very useful tool for workers. Most cars have an OBD Port which allows connection to the CAN bus, and by pretty much “plugging” into it, you gain access to all the data running around the vehicle. Most of the time, an external micro-controller is needed to read and output the data onto an external program. Usually, data is presented in HEX like this: ID: F2 9A AA 2C 36 D4 12 32. The ID is what CAN transmitter it is coming from and will always provide data for the function in that vehicle. For example, ID 201 may be the RPM, and whenever you see ID 201, it will always be related to the RPM in that vehicle. But different cars and manufacturers use different IDs for different functions. So, on another vehicle, ID 201 may be wheel speed.

Because of this, testing for faults can be very time consuming finding the correct ID for what you are trying to observe, and sometimes it may not even transmit data.

**User Identification**

The main user will be Shaun, to provide him with a personalized product for what he needs. He will have control on the system where it is safe to do so. For example, he will be able to input data into a database, but in some places from a list. But he will not be able to do things such as edit the properties of the data base or create tables and columns. This is purely to proof the system so no fault can be made.

Anybody else within the company will also be able to use the software with the same privileges as Shaun, but only within the company as the database is hosted from the Company server, requiring a password.

**User Limitations**

Shaun and most other people who will use the program aren’t always going to be skilled in ICT and Computer Technologies, therefore it may be difficult at first for them to use a new software. They also would not have experience with a system that provides data entry and may find it hard to operate.

**User Needs**

* The program should involve a simple and user-friendly interface.
* The program should be able to display data coming in and a readable manor.
* The program should include a function to allow the start and stop and pausing of data coming in so that the user can take time to go over data coming in.
* The program should be able to filter and focus certain IDs to provide a clear useful stream of data for the user.
* The program should be able to graph data coming in for the user to view.
* The program should be able to save certain data streams at the user’s desire for them to come back to and send back out.
* The program should be able to input data into a database that the user can view and edit.
* The program should be able to display the databases values that the user needs.
* The program should be able to automatically fill in fields for the user where available.
* The program should be optimized well enough to work on a low-end device effectively.

**Interview with Shaun**

1. **What system do you currently use for reading the CAN data?**

Currently we use an old version of CANalyzer, which is a popular CAN software company.

1. **What are some problems with the current system?**

Currently the system is quite outdated, so it doesn’t provide many functions. It can do pretty much the bare minimum that is needed to carry out the job. It sometimes provides errors and can be slow sometimes.

1. **What would need to be included in the software for it to perform correctly in your environment?**

I would need to be able to log data and be able to go over it later, as well as have a way to create a somewhat profile of each vehicle, so that later, if another vehicle comes in needing work done that we already have the profile created for, we can identify each ID and its use without use of trial an error. That would save a lot of time with every job. Also a way to view only certain IDs coming in, or to filter out the background data, just so that we can focus where the issue it.

1. **What do you usually want to find using the CAN analysing software?**

Usually I would want to be able to identify the steering wheel buttons, states of accessories, keypresses on the key fob, state of all lights in the car, current gear and handbrake. It is also good to be able to identify the RPM of the engine and the wheel speed etc, but these usually aren’t needed for most common jobs.

**Data Sources**

Currently the system being used does not have any way to store or provide the user with any data, therefore no data sources are needed. However, the proposed system will have many types of data to store. A database will have to be used for the large amount of data. The user will have access to input to and view the data in the database, but will be limited to what they can do, just to reduce risk of fault. This will be available in an interface for the user whenever they need. Each “session” of use the user will have to either store a new “profile” to the database or use an existing one. The interface will make it easier and simpler for the user, so that they can easily understand the data being presented and easily input data without needing to understand SQL.

|  |  |  |
| --- | --- | --- |
| **Data** | **Source** | **Destination** |
| Car Information | Stored by Shaun or other users | Database – cars table |
| Baud Rate | Inputted by Shaun or users | Database – canprofile table. Stored as a variable during session of use |
| Unique CAN IDs | Found and stored by Shaun | Database – canprofile. |

**Data Volumes**

The program will have to be able to store many different pieces of data, but they might not be very large pieces of data. Each field will be named clearly to provide ease of navigation within the database, and so that if needed, other people can understand and view it.

The data stored will be placed into tables, each linked to each other. This is important as to create a somewhat “profile” of each vehicle, this means that there is an ease of use and a large save of time for the user. But because each car is specific to their profile, this means there will be hundreds of profiles for each make, model, trim and even year of each vehicle.

This is important to Shaun, as the old system did not include any database or storage of any type to hold the data and profiles, meaning every time he had to work on a car, he would have to start from scratch trying to analyse and identify what each ID shows.

The database will be designed in a way that the user should not need to input much case and spelling sensitive data, and it will be provided for them to choose from, this is for ease for the user and to create less room for fault. But if the database doesn’t contain the required value already, there will be an option for the user to add it themselves. This means it will also be available for them to select from a list afterwards.

Preferably, the make, model and trim tables will be pretty much filled out already, allowing the user to just select from them to “build” a “car” in the car table. But the car table and CANprofile table will be filled out by the user.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Table** | **Field Name** | **Field Purpose** | **Field Size** | **Field Type** | **Example** | **Validation** |
| Cars | carID | Provides the specific vehicle an ID that can be called. | 8 digits | int | 38 | Field must be unique and only consist of numbers, must not be null. Automatically incremented if the data is unique. |
| Cars | modelID | Provides a link to the model table, as well as providing the model of the car. | 8 digits | int | 263 | Must not be null. Must be already existing. Only consist of numbers |
| Cars | makeID | Provides a link to the make table, as well as providing the make of the car. | 8 digits | Int | 104 | Must not be null. Must be already existing. Only consist of numbers |
| Cars | canID | Provides a link to the CANprofile table. Identifying the IDs needed to read the CAN data | 8 digits | int | 36 | Must not be null. Must be already existing. Only consist of numbers |
| Cars | trimID | Provides a link to the trim table, as well as providing the trim of the car. | 8 digits | Int | 111 | If not null must be already existing. Only consist of numbers |
| Cars | regYear | Provides the year of registration. Another identifier. | 4 characters | Varchar | 2008 | Must not be null. |
| Cars | Enginesize | Provides the size of the engine. Another identifier. | 1 digit 2 decimals | Decimal | 1.25 | Must not be null. |
| Cars | fuelType | Provides the type of fuel the car takes. Another identifier | 6 characters | Varchar | Petrol | Must not be null. Must be selected from the list. |
| Cars | Transmission type | Provides the type of transmission in the car. Another identifier | 9 characters | Varchar | Automatic | Must be selected from the list. |
| Cars | Drivetrain | Provides the drivetrain of the car. Another identifier | 3 characters | Varchar | AWD | Must be selected from the list. |
| Make | makeID | Provides the specific ID of the make/  Manufacturers | 8 digits | Int | 33 | Must be unique and only consist of numbers. Must not be null. Automatically increments if all data is unique |
| Make | makeName | Provides a name of manufacturer for the ID, for ease of choice for the user. | 25 characters | Varchar | Porsche | Must not be null. Must be unique. |
| Model | modelID | Provides the specific ID of the model listed for the make. Links to car table. | 8 digits | Int | 202 | Must not be null. Must only contain numbers. Must be unique. Automatically increments if all data is unique |
| Model | makeID | Provides the ID linking the Model to the Make of the car. | 8 digits | Int | 33 | Must not be Null. Must only contain numbers. Must link to an already existing MakeID. |
| Model | modelName | Provides descriptive name to the modelID. | 25 Characters | Varchar | 911 | Must not be Null. Must be unique. |
| Model | trimID | Provides an identifier and more detail to the description of the vehicle. Provides a link to trim table. | 25 characters | Varchar | 100 | Must not be Null. Must link to an already existing trim. |
| Trim | trimID | Provides a specific ID of the trim | 8 digits | Int | 100 | Must not be Null. Must be unique and can only contain numbers. Automatically increments if all data is unique |
| Trim | trimName | Provides a descriptive name for the trimID. | 20 characters | Varchar | Turbo S | Must not be null. Must be unique. |
| CANprofile | CANID | Provides a unique ID for the CAN profile | 8 digits | Int | 24435 | Automatically increases when a unique ID is created. Must only contain numbers. Must be unique |

The CANprofile table will contain more fields, which can be created by the user. This is to reduce wasted fields and space. The fields already in the table are included as Shaun has requested ease of inputting and viewing of them as they are the most common that he is looking for.

**Objectives of the System**

1. **The program should be able to read data coming from an external serial port, then display it to the interface for the user.**

**Specific Objectives:**

* 1. Read data coming from an Arduino UNO microcontroller and return CAN Data.
  2. Make sure data is readable that is coming in.
  3. Provide a preview of the data in real time within a user interface.

1. **The program should allow data manipulation for ease of analysing for the user.**

**Specific Objectives:**

* 1. The program should provide a filtering function, allowing the user to input an ID for the data stream to not include.
  2. The program should provide a focusing function, allowing the user to input an ID to be able to focus on only inputted IDs.
  3. Provide a way to pause/stop the data stream, to allow time for the user to read and analyse the data coming in.
  4. Provide a way to clear the data stream, in case the user has made any changes.

1. **The program must be able to save a text file of the current data stream when the user requires it.**

**Specific Objectives:**

* 1. There should be an option for the user in the interface that allows them to store a copy of the current data stream as a text file.
  2. There should be an option appear to allow them to choose where the file is saved.

1. **The program must be able to save data into a database and allow the user to view and input data where needed.**

**Specific Objectives:**

* 1. The user should be able to use a series of tables and lists to choose what fields they need to change and to save to them.
  2. The user should have the option to add new variables to the tables where needed but only where specified.
  3. The user should be able to view data where needed in a format easy to read.
  4. The user should be able to access existing inputted data by choosing the same “profile” as before.

1. **The program should have a graphing function that can display a chosen ID to the user in graph form.**

**Specific Objectives:**

* 1. The user should have the option to create an ongoing graph of the data coming in by inputting a single ID.
  2. .The user should be able to have access to a separate graph tab.
  3. .The graph should be easy to read and self-scaling where needed.

1. **The program should have a log in feature upon start-up to ensure only certain users have access.**

**Specific Objectives:**

* 1. There should be a login page or prompt for the user to fill out before being able to access the program