**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Currently Shaun is using outdated software when fault testing and CAN Reading, and this is due to the price of the programs available.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The system that Shaun currently uses is an old outdated and simple piece of software.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 thing 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 at a later date, 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.

**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 also 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.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **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, aswell 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 character | Varchar | AWD | Must be selected from the |
| Make | MakeID | Provides the specific ID of the make/  Manufacurers | 8 digits | Int | 33 | Must be unique and only consist of numbers. Must not be null. Automatically increments if all data is unique |