POLITECNICO DI TORINO

Master's Degree

in Mechatronic Engineering

Master's Degree Thesis

STUDY AND IMPLEMENTATION OF A SOFTWARE PROTOTYPE FOR RELIABILITY EVALUATION OF VEHICULAR ELECTRONIC SYSTEMS

Supervisor

Prof. Giorgio Bruno

Candidate

Antonio Padula

Company tutor

Eng. Raimundo Marcio Pontes

April 2022

Abstract

The automotive industry is constantly evolving due to growing technology. Modern knowledge in systems control and embedded software architectures has made it possible to equip the latest commercial vehicles with advanced electronic systems, to offer a constantly connected driving experience and actively guarantee the safety of drivers and passengers. Certainly, one of the main objectives to be achieved when it comes to innovating a vehicle equipped with advanced systems is to provide a reliable and efficient solution, as complex integration through multiple on-board systems is required. Indeed, the high complexity of a vehicle's electronic architecture requires a time-consuming and expensive simulation phase for component verification and validation. To this purpose, the thesis proposes the implementation of a software prototype capable of faithfully reproducing the behaviour of the commercial vehicle infotainment system. The prototype emulates it in terms of logical architecture, user interface and telematics applications, taking into consideration the challenging scenario of creating a communication interface between the telematics Electronic Control Unit (ECU) and a personal computer. With this, the prototype provides an easily accessible testing environment for this case of integration of vehicular electronic units. It took some preliminary steps to arrive at this implementation. Initially there is an in-depth study of the vehicle, to understand the functional aspects of the electronic architecture and communication between the systems, up to the introduction of software errors and an initial classification of the problems encountered. Subsequently, an exemplary case of integration between the various vehicular electronic systems, which required various technical revisions, is sought. Then the history of the software issues encountered and the related actions taken is created. Considering that the validation operations are quite complex and repetitive, the main objective of the software prototype is to simplify the testing of the telematics electronic control unit in the software integration with the infotainment system. The concerned ECU was designed to manage the telematics and connectivity functionalities, making the contents displayed on the infotainment system available. Automated testing sessions for component validation are performed using an innovative approach. The instructions for the test are inserted into a JSON file, using a template created specifically for the prototype. Then, through an iterative process, the instructions are converted into actions to be performed on the control unit. The received response is processed through an algorithm that compares the expected behaviour with the one obtained. Consequently, once an appropriate percentage of matching parameters is obtained, the software configuration is assumed to be acceptable and the test is passed. The entire application was developed with the .NET platform, following the MVVM (Model-View-ViewModel) architectural pattern to improve the reusability and maintainability of the code. The programming language used is C-sharp (C#).

Acknowledgements

I would like to express my appreciation to my supervisor Prof. Giorgio Bruno for guiding me throughout the preparation of the final thesis, giving me valuable support and suggestions to improve the writing and its content.

I would also like to make a special thanks to Eng. Raimundo Marcio Pontes with deepest gratitude. His constant support and critical remarks helped me to ensure continuous motivation in this thesis.

I also extend my thanks to all the members of the Iveco Telematics team, especially to Eng. Ivano Lo Iacono, with whom I have had the pleasure of working together in recent months and who have provided me with all the support necessary to complete an ambitious project like this.

The biggest thank certainly goes to my family, to whom I owe all my achievements.

To my parents, Paola and Gaetano, to whom I have always looked with great admiration as I find them as a huge example to follow. Their teachings and their support in my decisions have constantly encouraged me to do more and to improve in every aspect of life.

To my brothers, Vincenzo and Giovanni, with whom we always support each other even if we are located in different places. I am grateful for all the moments spent together; we form the best team.

Finally, I thank immensely my girlfriend and personal happiness, Camilla. She believed in me even when myself didn't, always staying by my side and encouraging me in every moment.

Table of Contents

Chapter 1

Introduction

1.1 Purpose of the Thesis project

Product differentiation makes it difficult to integrate the subsystems that make up the vehicle's hardware and software architecture.

It is possible to define a reliability index not only on testing but also on the way in which the software was developed, on the type of hardware, on the type of operating system or on many other factors, including critical issues in terms of security, cyber security and many surrounding factors.

The thesis is oriented towards a study and research in the field based on the reported vehicle faults and on how to properly verify and validate the software, in order to increase its reliability, through the creation of a software prototype in which it can be highlighted any system criticalities or failures related to the integrated software of an Electronic Control Unit (ECU) and, in this case, the specific event or factor that causes it.

As of today, the testing procedures of vehicular ECUs are often slow and repetitive and they can require a lot of time and technical reviews from the testing expert; thanks to modern knowledge in mechatronics engineering field, it is possible to make the testing and validation of electronic components easier and more efficient.

1.2 IVECO and the Reliability Engineering sector

IVECO (Industrial Vehicles Corporation) is a transport vehicle manufacturing company founded in Turin in 1975. It designs, manufactures and markets a wide range of light, medium and heavy vehicles; they can offer a span of vehicles that goes from commercial vehicles to quarry/building site vehicles, city and intercity buses and special vehicles for applications such as firefighting, off-road missions, Defence and civil protection.

Figure 1.1: IVECO company logo.

Recently IVECO has completed its spin-off from parent CNH Industrial, forming the new IVECO Group.

This new operational and financial brand incorporates all the brands and industrial realities that refer to on-road mobility and that have their roots in the history and traditions of IVECO.

Under the new logo, that can be seen in Figure 1.2, the range of protagonists of this new chapter for special road transport is shown. In the middle, there are the leading companies of the group: Iveco, which produces light, medium and heavy commercial vehicles, and FPT Industrial, which produces engines and advanced propulsion systems for the agricultural, construction, marine and commercial vehicle industries.

Figure 1.2: IVECO Group logo and all its brands.

IVECO and CNH Industrial who have always shared a vision of quality and sustainability together, now will be better placed to exploit their full potential in terms of financial support, value generation and sustainability commitments. Thanks to greater management focus, the new on-road vehicles group will be able to accelerate their innovation, think faster on their strategic plans and be an active participant in the industry consolidation.

Speaking about sustainability, IVECO is the only manufacturer to offer ecological diesel and natural gas engines across its range; the company launched their first commercial vehicle powered by natural gas in 1996. But the vehicle range offered does not end there, the company, following the industry trend and the sustainability principles already mentioned, intends to bring the first fully electric IVECO truck to its price list. When it comes to sustainable mobility, actually there is no alternative solution designed to fit all forms of transport and applications. IVECO produces vehicles with alternative drive systems to try to solve this need, by providing in addition to the propulsion systems described above a Daily model of the light range equipped with an electric motor and also a Eurocargo model of the medium range equipped with a hybrid engine.

Figure 1.3: IVECO Daily, for over 40 years one of the most successful vehicles of the company.

But the desire to innovate has always been present in this company and by analysing the history of the products offered by the company in recent years, it is possible to see how the range of commercial vehicles has been enriched with

modern and technological features and systems that allow a new world of integrated services and solutions.

It is important to highlight the involvement of the company into the Agile Methodology and its philosophical and efficient approach to project management; this method provides to the companies major flexibility and a better way to respond to the always changing industry scenario. This topic will be discussed in detail later.

For example, among the best-selling models produced by IVECO we find the trucks available in the IVECO WAY Range, the “customer-centric” way, as Iveco said. With this name, the company wants to focus the attention on their approach that fully exploits the possibilities offered by connectivity and being “always on” with the customer, included in these vehicles.

Figure 1.4: IVECO S-Way, the 100% connected truck

It is the case of IVECO ON, the latest pack of services that offers connected and integrated systems and transport solutions to their customers. It enables the driver to be always aware of the vehicle’s and driver’s main performance parameters, such as fuel consumption, and to achieve reduction in Total Cost of Ownership (TCO). In Figure 1.4 it can be seen the IVECO Stralis S-WAY model of the heavy range, the absolute protagonist of this thesis project precisely for its innovative equipment and complete connectivity functionalities.

With this entire offering, IVECO addresses the key trends driving the automotive industry that saw a big jump up-wards in the product innovation cycle, decreasing the gap separating the vehicle and the services around it, making connected trucks and keeping skilled professional drivers possible, increasing awareness on-road and fulfilling the demanding requirements in sustainability.

1.2.1 Reliability Engineering

Today, when speaking about a very reliable vehicle we suddenly think about a vehicle that always fulfils its function and under any operating conditions.

It is interesting to note how over the years the definition of "reliability" has grown and changed in meaning. In the 1940's the term was all encompassing and a cure-all for whatever ailed the piece of equipment was, "Make it more reliable." With the passing of time the definition took on many new implications and became more quantitative. Splinter words such as “maintainability”, “availability” and "product assurance" filled the voids that reliability could not cover. [1]

The term "reliability" can therefore have more meanings if different types of people are taken into consideration. For example, an academician refers to “the ability for the part or assembly or system to fulfil its function under specified operating conditions for a specified period of time under specified environment, without failure” for what regards to reliability, while maintainability is “the probability that, when maintenance action is initiated under stated conditions, a failed system will be restored to operable conditions within a specified total time”.

For the average customer, reliability and maintainability often have the same meaning; the relevant aspect is that the vehicle is equipped with safety systems in accordance with the law and that guarantees reliable travel on the road. The purchaser has confidence in it.

To a manufacturer, reliability is not only a unique definition; it is a company-wide program.

In Iveco, the Reliability sector is included in the company's industrial Quality area; it includes all the activities related to the maintainability, availability of the vehicle and also plays fundamental roles for the aspects inherent to the quality of the vehicle and its efficiency.

1.3 Involvement of the Agile Methodology in the Thesis project

The ever increasing need to manage diversified activities of human life, through technologically advanced electronic devices, requires continuous improvements and greater guarantees of the software that is used in them.

In the perimeter of software development, there has never been any shyness in introducing new methodologies. In fact, over the past 25 years, several different approaches to software development have been introduced, of which only a few have survived to be used today.

Software vendors are required to improve the quality of the software provided and reduce the overall cost. [2]

The business management models have had to adapt to the new “turbulent” production conditions, characterized by very high competition and requests for greater productivity, speed and quality. A context that has led to the change in the nature of the projects to be managed, for which the principles of “traditional” management are no longer enough.

In order to help PM (Product Managers) and the company organization in the management of "complex projects", advanced project management methodologies and techniques have been introduced. [3]

In recent years, a wide range of technologies have developed to make software more reliable.

While there is no agreement on what the concept of "agile" actually refers to, it has attracted a lot of interest among professionals and lately also in academia.

In the following sections, we will discuss some important proposals regarding changes in software lifecycle management and their effect on the delivery of any software product. [2]

Precisely for this reason, I was immediately able to see why the interest in these new paradigms in software development is present in a large global company such as IVECO.

For the purposes of the thesis work to be carried out in the company, I had the opportunity to deepen and study the Agile methodology as well as to be able to apply it together with the Telematics team with which I developed a software prototype for the verification and validation of the telematics' electronic control unit capable of automating the testing process; this will be discusses in Chapter 4.

It should be added that the Agile methodology is a grouping of methods; each illustrates important new proposals related to changes in software lifecycle management and their effect on the delivery of any software product.

These methods are not based on the classic and linear design approach but take advantage of the possibility of creating a project divided into phases, each called "sprint". For each sprint a goal is agreed, such as the implementation of a new functionality, which is agreed together with the customer, who is shown the progress of the project up to that point.

A system based on iterative processes is created, a fundamental concept for the Agile method, which allows you to easily make changes to the project, reduce production costs, avoid wasting resources and unnecessary time; this allows, eventually, to prevent a project failure by acting at the right moment.

Specifically, the method adopted in the company in the Telematics department for project management and also adopted for the development of this thesis is the Scrum model.

Scrum is the most widespread Agile method, particularly suitable for complex and innovative projects. It is a framework, or a logical structure or architecture on which a software development can be implemented using a set of practices which, dividing the project management process into different sprints, coordinates the development of the product with the customer's requests.

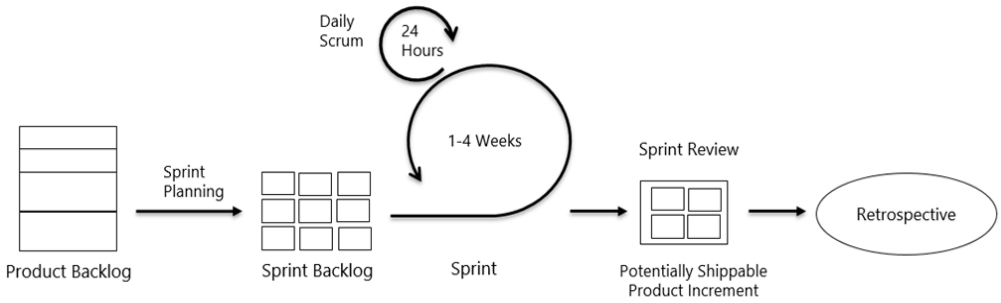


Figure 1.5: The various phases of a project management according to Scrum.

During the journey of this activity in the company it was impossible not to notice the effective involvement of my business within the development team, thanks to the various support meetings that the Scrum method provides. Among these I report, for example, the brief daily meeting in which the whole team is updated and each member is called to discuss what has been done compared to the previous day and what is expected to be done for the next day. This meeting becomes extremely useful especially in the case of critical issues or blocking problems, where the support of the team can make the difference to unblock the situation.

Chapter 2

Preliminary activity in software reliability

2.1 Study of the heavy vehicle range

The purpose of the first phase of the thesis activity, held at Iveco S.p.A. in Turin, was to deepen and study a series of methodologies and tools that are useful for understanding the electronic components and systems with which I was going to work during the internship.

In this chapter, beginning with a presentation of all the commercial vehicles of Iveco and of the milestones the company achieved by developing technologies that turned out to be pioneers in the industry, the concept of a connected vehicle is introduced. This concept is an essential point in the digital revolution that has revolutionized the automotive industry.

Iveco with its “Iveco On” services exploits the maximum potential obtainable from integrating connectivity functionalities in a truck, providing its customers with a new ecosystem of connected and always operational services. The concept of remote diagnosis and monitoring is introduced by enhancing what can be obtained from the implementation of telematic devices on commercial vehicles.

All this is made possible by the trend that the last twenty years has seen affecting the automotive industry: digital transformation and the rise of embedded electronic systems. The vehicles consist for the most part of electronic equipment that command and manage the activation of switches, advanced driver support systems (ADAS), engine management and control, and so on, transmitting a multitude of messages and signals via highly efficient and reliable CAN communication lines. Indeed nowadays it is enough to consider that the majority of support requests received are inherent to failures related to vehicle

electronics, as a consequence of the ever increasing presence of these systems on board.

It is therefore necessary to understand how the integration between so complex systems works. To this purpose, some of the functional electronic diagrams will be analysed below. These schemes highlight the work of optimizing the electronic architecture, that is made possible by means of the multiplex system.

The research experience of software issues related to Electronic Control Units, (in short ECUs) within the company is then described to distinguish the actions and technical reviews that are carried out during the product industrialization phase and those instead used for post-marketing support of the product, with customer reports of vehicle breakdowns and anomalies. There will be a specific focus on the OPL document used for the management and resolution of electronic systems anomalies related to cases of software errors that specifically require the involvement of multiple figures (such as manufacturers and suppliers).

Finally, to complete the discussion about vehicular diagnostic analysis and data acquisition from the electronic systems, the instrumentation used for measuring error parameters and for vehicles monitoring is presented.

2.1.1 Commercial vehicle models and their missions

The heavy range models represent the flagships of the Iveco brand; Stralis with its most recent on-road version, the S-WAY, guarantees the perfect balance between advanced vehicle technology and an efficient and powerful series of engines produced by FPT Industrial, giving higher class fuel economy.

With all its vehicles, Iveco aims to guarantee high-level functional and technical specifications; for this reason, the commercial vehicles fleet includes a wide range of configurations and models suitable for any need and mission.

All Iveco commercial vehicles will be listed and described below; its comprehensive offering starts from the light range 2.8-ton vans for urban use to the 44-ton heavy range vehicles for long distances and heavy loads use.

All these vehicles provide customized transport solutions for their customers, who have a wide choice of configurations based on cargo and load requirements. Furthermore, as mentioned in the introduction to Iveco commercial vehicles in Chapter 1, the fleet is completed by vehicles equipped with alternative propulsion systems; these models are available for specific missions and can be equipped by alternative Natural Gas, Hybrid and Electric tractions.

- **STRALIS**

  Powerful and reliable, the new Stralis on-road range offers engines with exceptional fuel efficiency, cutting-edge technologies and advanced connectivity systems.

  This model, being a 100% connected truck, is the spearhead in terms of technological innovation.

  Its advanced connectivity features can keep drivers in constant communication with the fleet manager, Iveco service specialists and the dealer network. This allows, in addition to constant monitoring of road safety for those on board the vehicle, also a remote maintenance and diagnosis service; a detailed description of these functions will be provided in the paragraph dedicated to the connected vehicles.

  Due to the technological nature of this vehicle, all the studies and researches discussed later will refer specifically to the Iveco heavy range.

Figure 2.1: The Stralis of Iveco heavy range.

- **EUROCARGO**

  The Eurocargo model represents Iveco medium range. Sustainable, efficient and versatile, this is the perfect truck for urban missions and municipal services for medium-sized transport.

  This vehicle allows greater driving comfort during urban missions thanks to the reduced width of the cabin, the greater steering angle and the better turning circle compared to a model in the heavy range.

Figure 2.2: The Eurocargo of Iveco medium range.

- **DAILY**

  The Daily model represents Iveco light range and it is the longest-produced vehicle of the company. Boasting over 40 years of success, the "Van of the Year 2018" is a true champion of sales and innovation. With this vehicle, Iveco was the first in the industry to introduce a truck-derived chassis with rear traction and independent front suspension. This configuration allows heavy load transport maintaining a compact and versatile vehicle body, as well as advanced systems such as Electronic Stability Program (ESP) to regulate vehicle handling during steering and braking maneuvers.

Figure 2.3: The Daily of Iveco light range.

2.1.2 Connected vehicles

In introducing the Iveco commercial vehicles fleet, it has been highlighted the important presence of technologic implementations on board the vehicles. Innovation in the production of smart vehicles is growing fast in both the automotive and transport industry. Indeed, with the modern knowledge and available technologies in the electronic and mechanical field, it is possible to develop multiple functions that extend the operations done by all the vehicle components: braking system control, engine management control, automated gearbox, automatic suspensions, telematics, etc. In this way, the experience of driving the vehicle acquires a much more efficient and safe meaning.

Among all the electronic systems, the vehicles are equipped with a telematics ECU that allows wireless data transmission and communication: it is therefore appropriate to define the “connected vehicle”.

The technology of connected vehicles is functional to the supply of digital content and services, to the transmission of telemetry data and to remote monitoring or managing of on-board systems.

These vehicles, capable of exchanging information via a wireless network, are equipped with cutting-edge technological systems such as built-in cellular modem, multifunctional infotainment and telemetry sensors capable of interfacing with other vehicles or infrastructures.

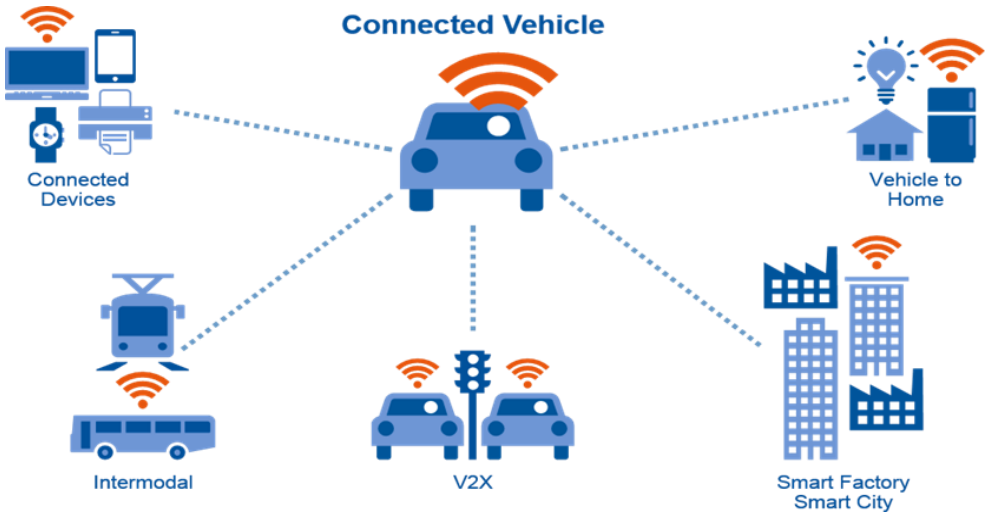


Figure 2.4: Interaction of a connected vehicle with other systems. [4]

The modern truck fleet could unlock huge efficiency gains by adopting connected vehicles and procedures. Once a fleet might only have monitored immediate data points such as vehicle GPS data, for instance, but an increase in sensor technology and cellular bandwidth is opening the door for more granular data management centred on maximising vehicle uptime.

It is critical that fleets begin digitising their operations today. The e-commerce boom accelerated by COVID-19 shows little signs of slowing and, as customer expectations on delivery times ramp up, intelligent data management will be vital in helping fleets meet ongoing demand.

The promise of capturing vehicle data and turning it into profitable new services and business models has driven significant investments in the commercial vehicle telematics space by truck manufacturers and suppliers alike.

Automotive telematics systems have become permanent presence in the industry since the early 1990s, when General Motors company released the OnStar system; the first ever unique combination of Global Positioning Satellites (GPS) and cellular module technology to offer an always-on real operator instant communication in case of any kind of emergency.

Nowadays, telematics and embedded systems in automotive industry are evolved and they are surely more powerful than that. They allow manufacturers to collect

data also from event data loggers, communication systems and security systems as well as data from other vehicle systems, such as mechanical components.

In fact, recent technological advancements and the push towards a fully driverless vehicles scenario are incentivizing the implementation of these systems. Truck manufacturers must therefore invest in their innovation sector and think about how they can design these systems and integrate them into new vehicles.

As expected, with the intensification of a methodological approach to the new connected systems in vehicles, new opportunities have arisen in the field of diagnostic analysis and real-time vehicle monitoring.

In fact, having integrated a telematics ECU that constantly exchanges data via wireless network, it is possible to acquire parameters relating to the functional status of all vehicle components in order to prevent any breakdowns and the accidental consequences that they could cause.

This component integrated with a cellular modem allows the driver to have an overview of all the safety and assistance services available; among these, for example, the Iveco RAS (Remote Assistance Service) system which allows a quick analysis of the errors/failures in the electronic systems even remotely. Thus, it is possible for the maintenance technician to provide immediate assistance to the driver and evaluate whether it is necessary to bring the vehicle at maintenance for a complete check-up.

It should be denoted that with the increase of smart roads and smart cities, smart trucks are a necessity; here also lies the big potential of the connected trucks. With these arrangements a lot of space for improvements is given first of all to fleets management, being able to manage and plan their cargo missions more effectively.

With a unique platform that allows to monitor real-time import container data like vessel and terminal information, actual location of the truck, expected arrival at destination, holds and more, it is possible to completely eliminate those inefficiencies present in management of commercial transport. The managers

would also be notified of potential problems in order to be able to propose a resolutive action plan well in advance, avoiding costs for unnecessary delays.

To this purpose, the solution proposed by Iveco is the ecosystem of services included in “Iveco On”, which is composed by five main services outlined in Figure 2.5. These are: Iveco On Fleet, Iveco On Uptime, Iveco On Care, Iveco On Maintenance & Repair, Iveco On Parts.

Figure 2.5: The offer of services and solutions in IVECO ON.

Iveco On is a flexible and customizable package, available for the heavy Way range and the light Daily range, in which there are connected services to maximize the efficiency and profitability of vehicles, up to a series of aftermarket solutions for vehicle maintenance.

Iveco vehicles equipped with these services provide a more productive work environment as they allow you to manage infotainment, navigation, driving support tools and advanced fleet management services in a simple and integrated way.

Among the connected functions designed to improve the driving experience and the transport productivity, there are:

- A fleet management tool, capable of automatically calculating the best route to travel based on the mass and size of the vehicle and optimizing the routes based on the processing of the orders. The service also allows you to receive advice on a quarterly basis from a TCO (Total Cost of Ownership) expert aimed at how to reduce fuel consumption on the basis of data acquired electronically;

Figure 2.6: Iveco On Fleet Management vehicle monitoring.

- The DSE (Driving Style Evaluation) system, an algorithm which by processing the data acquired from the engine, the vehicle and via GPS, allows the driver to have an evaluation of driving performance and to effectively calibrate fuel consumption; by continuously exchanging data via the Internet, the Iveco On system produces a smart report on the performance of vehicles and drivers;

Figure 2.7: Indices for the evaluation of the DSE.

Figure 2.8: DSE user interface.

- The DAS (Driver Attention Support) system, a safety feature that protects the driver from fatigue and sudden falling asleep at the wheel by suggesting when to pay more attention to driving or when it is time to take a break; using the data connection network, it constantly exchanges information making it possible to remotely monitor and to finally create the Safe Driving Report with the assessment of the safety level on board.

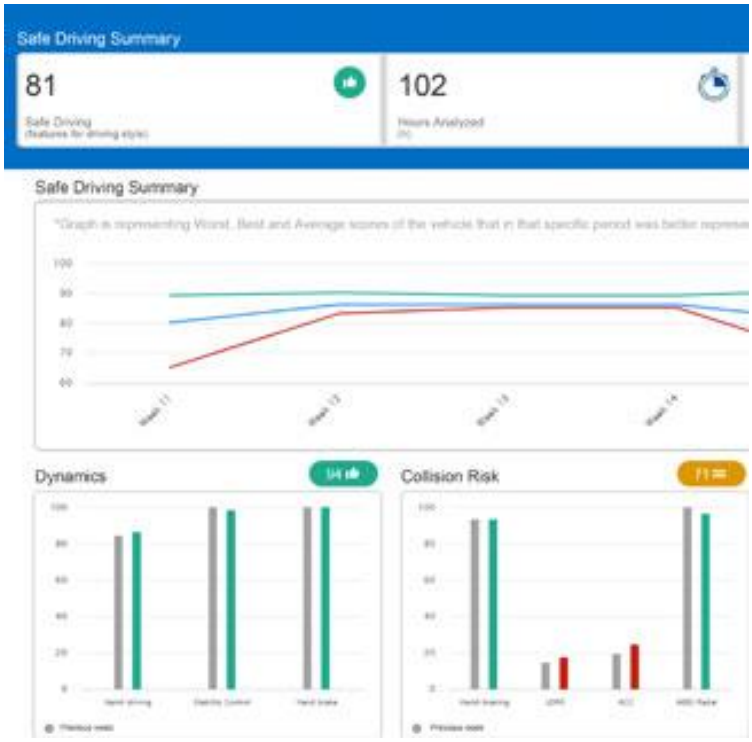


Figure 2.9: Safe Driving Report is done based on the driver's driving style.

Furthermore by fully exploiting the potential of connectivity, Iveco provides its Customer Service remote support 7 days a week and 24 hours a day. In this way, the support is able to respond instantly to any request for assistance made by a driver on board the vehicle.

The scenario created by choosing the implementation of connectivity functionalities on vehicles is full of advantages for all interested parties. With connected vehicles the costs of operations are reduced by giving, at the same time, an innovative and always active assistance and support service to the drivers. They can count on an increasingly reliable and safe vehicle for their commercial transport missions.

2.1.3 The role of automotive electronics in commercial vehicles

The commercial vehicles market is constantly evolving it comes to technology development about automotive electronics. Vehicles have rapidly transformed

to create a new way of imagining the functioning and the applications of the vehicle itself thanks to the development of embedded electronic systems.

For over a decade now, the first electronic systems applied to the automotive field, namely starter motors and batteries, have been consolidated as a standard. Over the years, the presence of electronics in a vehicle has followed a growing trend, developing various applications for engine management, vehicle ignition and shutdown, radio and multimedia functions, telematics units for connectivity, advanced infotainment systems, electronically controlled transmission, on-board climate control, active driving safety systems, smart navigation, adaptive cruise control and others.

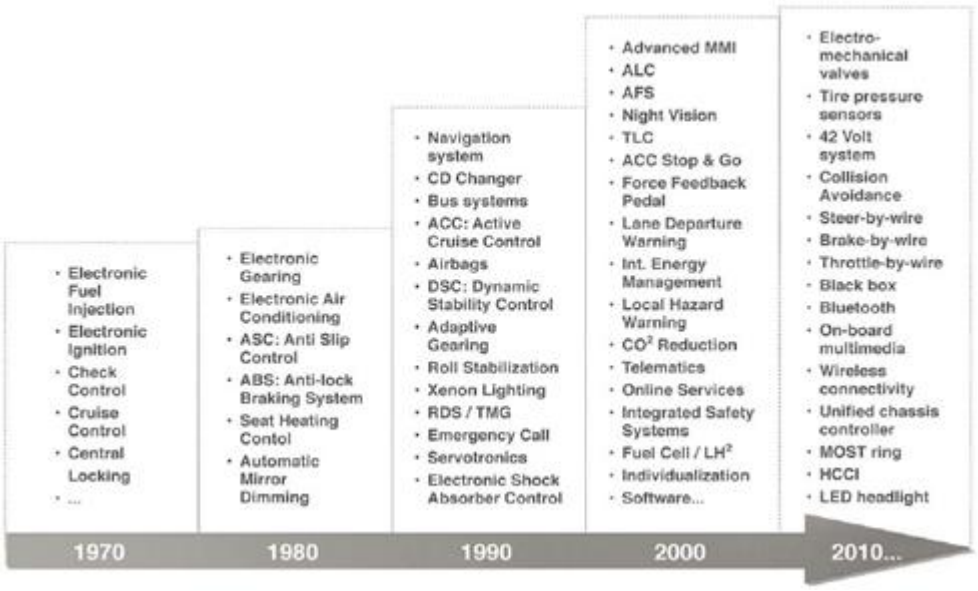


Figure 2.10: Roadmap of the evolution of vehicle electronics.

Focusing on safety features: it is a high priority in commercial vehicles. Thinking of an active system that constantly monitors the navigation of the fleet, reducing the risk of accidents, is enough to understand the enormous potential of these applications.

Going further, drivers need technologies in their vehicles to reduce fatigue and to prevent the accident of falling asleep to the wheel. With the development of a telematics system electronically connected to sensors that acquire data from

vehicle components (such as the engine, the acceleration given by pressing the pedal or sudden movements of the steering wheel) it is possible to detect these extreme conditions for driving and prevent accidental events by instantly reporting the fleet manager and alerting the driver on board.

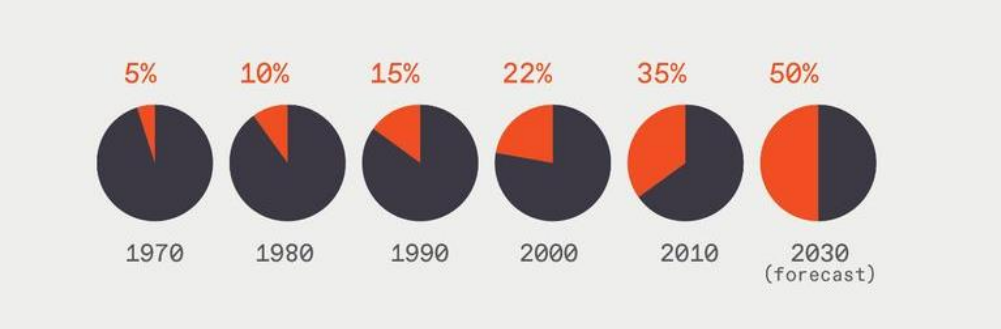


Figure 2.11: Electronics system as percent of total car cost [5]

While ten years ago only premium vehicles contained multiple microprocessor-based Electronic Control Units (ECUs) networked and engineered inside the body of the vehicle, that could process million lines of code, today also low-end vehicles are equipped with powerful ECUs that can manage software advanced technologies like Advanced Driver Assistance Systems (ADAS), adaptive cruise control or automatic emergency braking. These are becoming standard.

2.1.4 The Multiplex system and CAN BUS connections

Following the continuous functional and managerial innovations of the vehicle components, the Multiplex system has been designed to simplify the wide complexity of the electronic system and to improve its quality and reliability.

The Multiplex is a multifunction system made up of many electronic components interacting with each other and that are continuously monitored. The purpose of this system is the adoption of digital electronic technology components that allow to significantly simplify and optimize the entire system

connections. In fact, the Multiplex is composed of a series of electronic control units connected to each other by means of CAN lines.

The CAN acronym stands for “Controller Area Network”; it indicates a network protocol, created by Bosch GmbH company in the early ‘80s, that allows the communication among different electronic control units inside the vehicle. Nowadays, it is a standard for the automotive industry.

The reason of its success can be summarized in relevant technologic improvements:

- Fast response times while communicating;
- Simplicity and flexibility of connections management;
- High reliability when there are signals disturbances.

Communication on the CAN network occurs through two or more nodes which can be represented by a simple I/O (input/output) device or by an integrated computer with CAN transmission capability. Through special interfaces, the node also allows to communicate on the network with external computers through an Ethernet-over-USB connection in order to manage messages or, perhaps, carry out an analysis of the vehicle embedded systems to find any faults.

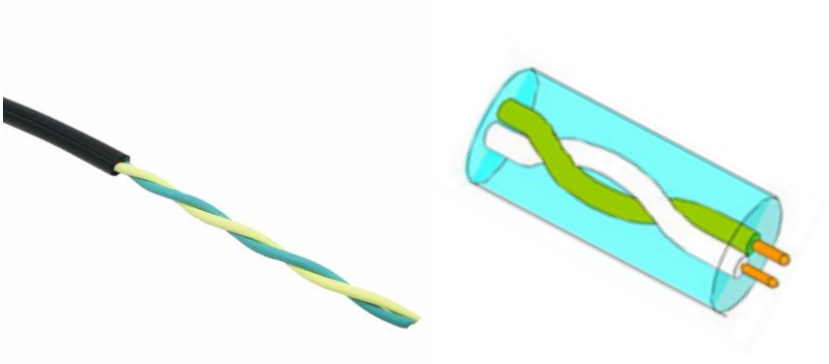


Figure 2.12: CAN line configuration.

The nodes are put into communication through a serial communication channel called BUS (Binary Unit System) composed of several cables intertwined with each other, as in Figure 2.12, often shielded by a sheath. For this reason, in fact, the use of connections via CAN BUS lines allows to limit the disturbances

caused by electromagnetic fields. These lines are an excellent solution for applications that require a high level of reliability.

The CAN BUS lines allow to manage the bit rate and, therefore, the transmission speed while being able to reach extremely fast response times. These are much more convenient for the ECUs that manage main vehicle components since they need quick communication channels, such as for the activation of the braking system in the event of sudden braking.

Before the use of the multiplex system, the management of the connections between electronic switch and mechanical actuator was very complex and often this was the cause of breakdowns and malfunctions.

With the flexibility and efficiency made available by the CAN lines, it was necessary to implement this technology in order to simplify the vehicular electronic architecture. Finally, it could be proposed the implementation of the multiplex system for connecting the ECUs.

In Figure 2.13 it is possible to see the multiplex system functional scheme: the command from the switches is received from the first ECU, which is processed and then transmitted via the CAN line to the second ECU, which processes the input received and activates the actuators. All this takes place using a single transmission point, thus reducing the complexity of traditional electrical management which required multiple connections to reach each system.

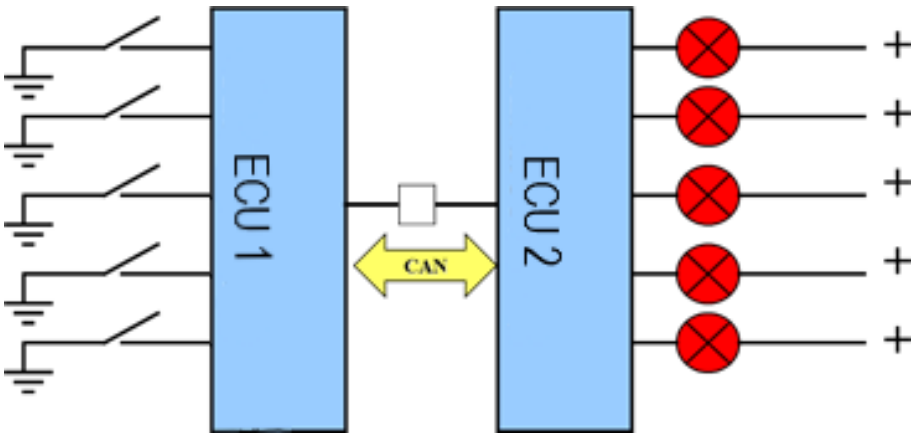


Figure 2.13: Multiplex system scheme. [6]

Subsequently, the main vehicular electronic components and their functionalities will be discussed.

2.2 Study of the electronic architecture of the vehicles

At the beginning of the thesis work in Iveco, it was decided to first focus the attention and deeply study the electronic systems available on the Stralis model of the heavy range in order to fully understand the functions of the ECUs and their integration.

In the previous paragraphs the multiplex system and the CAN BUS serial connections that allowed a better electronic management of the vehicle have been introduced. This made it possible to implement new functions while decreasing the number of cables, fuses and relays in the electronic architecture.

On this network travel signal messages that communicate various information, such as the operation or status of a component, the presence of a fault, the specific command to activate a warning light, the digital elements to show on a display and many others.

However, in order for these messages to be processed by all the interoperating systems, they have to circulate in a correct "format". So, it is necessary to use electronic control units to convert these commands into actuations. The messages can be acquired by the equipment (such as sensors, switches, buttons) into digital signals.

The number of electronic control units available on the heavy range vehicles is considerably high because of the rapid adoption of these systems seen in the industry. Subsequently, some ECUs that guarantee the operation of main vehicle components and the CAN lines involved will be examined.

2.2.1 Electronic functional diagrams

The primary objective of this study was to acquire a systemic vision of the electrical and electronic systems of the trucks. For this purpose, a special attention to the electronic functional diagrams is needed.

These documents, provided by the company, are detailed reports of all the technical aspects that are involved in the electronic control units of the vehicles. More specifically, in systems engineering, a functional block diagram describes the functions and interrelationships of a system using a graphical language.

Therefore, it is very useful to better understand the function of the system under investigation and the input and output variables that are involved, that are: the signals to be processed by each electronic unit, the data inputs acquired by sensors or the commands to send to actuators.

These lead to a complete technical specification of the electronic systems and their software integration with the other vehicle systems.

In the network of the multiplex system of the Iveco Stralis (Model Year 2019) the dialogue of vehicular electronic systems, such as the engine ECU, on-board multimedia systems, telematics applications, electronically operated equipment (wipers, windows, lights and so on) takes place via the following main CAN lines:

- VDB – Vehicle Data Bus;
- BCB – Body Control Bus;
- ECB – Engine Control Bus;
- IB – Infotainment Data Bus;
- TB – Telematics Data Bus. [7]

Each of these connection lines contributes to the transmission of the signals to the specific ECU. In particular, it is noted that the transmission speed of the information contained in the signals is the same in all the above CAN lines except for the VDB (Vehicle Data Bus) line.

The VDB, in fact, has a bitrate of 500 kBit/s that is exactly double compared to the other lines, which transmit at a standard speed of 250 kBit/s. This is because the information transmitted in VDB converges in many interconnected units among which we find systems that require a very fast response time and greater security and fidelity in data transmission, such as those related to movement and dynamics of the vehicle.

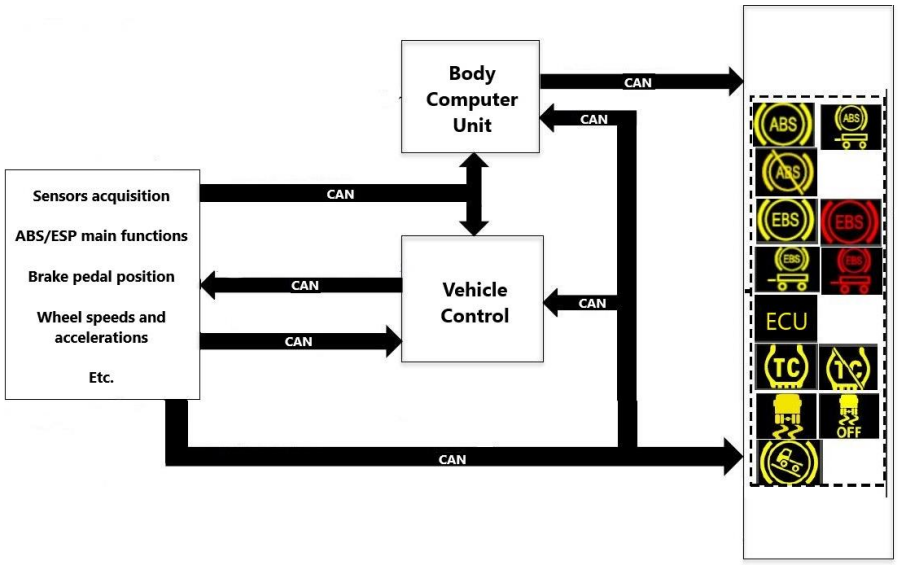


Figure 2.14: Brake management electronic functional diagram. [7]

In Figure 2.14 it is possible to see an example of systems interconnected with each other by the fast VDB line; this is the electronic functional scheme for vehicle brake management system. The use of this line here is necessary as it must transmit a multitude of vehicle data instantly, such as: vehicle speed, steering angle, engine torque, front and rear axle speed, working status of advanced driver-assistance systems (ADAS) like EBS and ABS, brake and accelerator pedal position, and so on.

Figure 2.15 below shows the functional diagram with which the adaptive system for cruise control was designed. This advanced feature allows the driver to be able to set a fixed cruising speed for long distances thanks to the use of multiple advanced systems, including positioning sensors and radar devices for detecting vehicles along the trajectory. Also here there is an extensive use of the VDB (Vehicle Data Bus) fast transmission line.

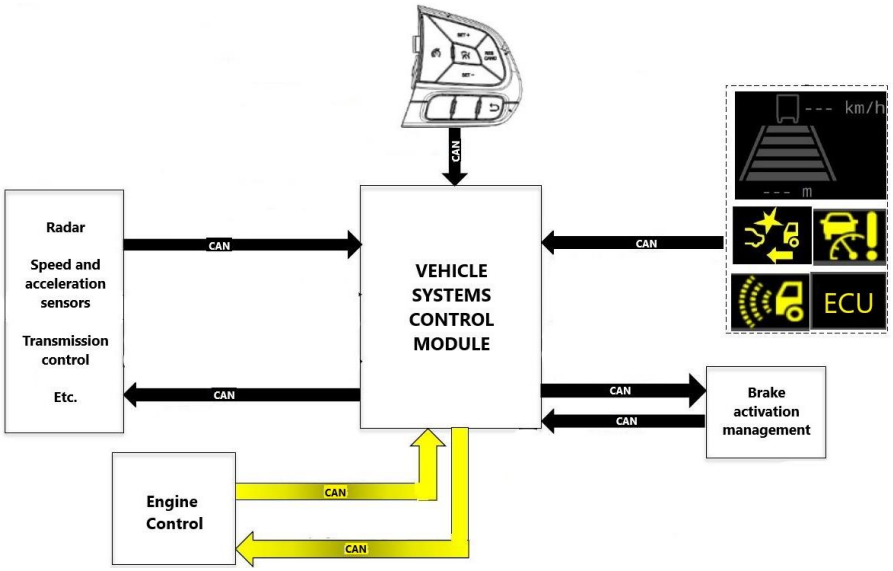


Figure 2.15: Adaptive Cruise Control Management electronic functional diagram. [7]

Finally, Figure 2.16 shows the electronic functional diagram of the telematic system for evaluating driving performance and fuel consumption, the Drive Style Management. This system, thanks to a combination of data packets transmitted via wireless, allows the analysis of vehicle parameters remotely, making its functionalities constantly connected with Iveco technical support.

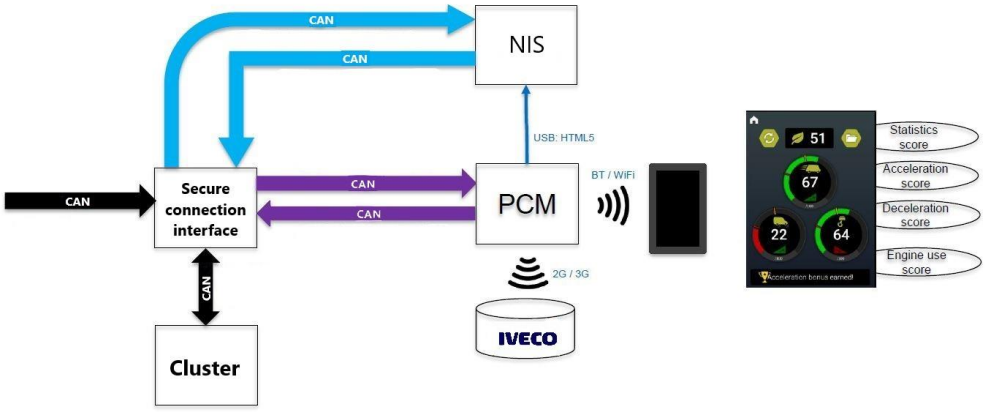


Figure 2.16: Drive Style Management electronic functional diagram. [7]

In this schematic it can be seen a particular electronic control unit, namely the PCM (Processing and Communication Module), which will be the subject of chapters 3 and 4 as it is the absolute protagonist of the internship activity carried

out in Iveco. This electronic control unit allows the Drive Style Management system to send and receive the score indices to the on-board infotainment system, to the smartphone through a specific app and to the back-end through different communication channels (USB, Bluetooth, Wi-Fi, 2G/3G cellular networks).

It should be noted that many of these electronic systems are reached by several CAN lines: this is highlighted in the graph in Figure 2.17, which shows the complexity of providing the vehicle's digital functionalities that constantly need to communicate with multiple channels and to receive more data at the same time from several interconnected lines. It is shown the relevance of the VDB line, that contributes to the transmission of many systems data involved in main vehicular functions which require speed and operational fidelity, as mentioned previously.

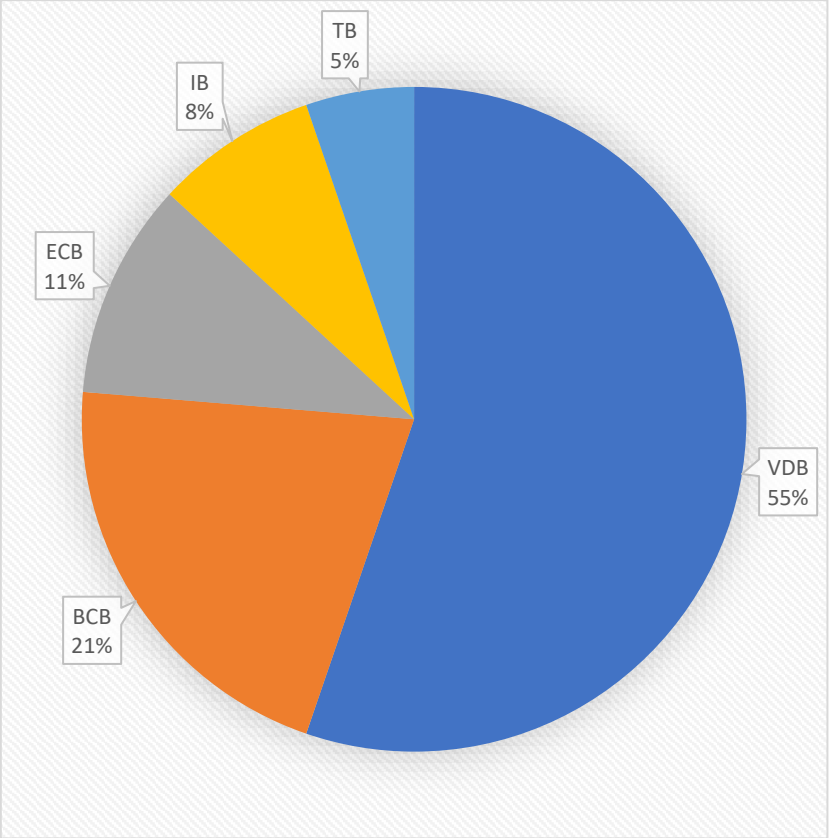


Figure 2.17: Percentage of ECUs connected by each CAN line out of the total available.

2.3 Analysis and classification of vehicular software issues

Moving from the idea to the finished product involves the fulfilment of a series of consecutive phases that are essential to make the new vehicle ready for marketing and for the operations it will be used for.

After the necessary initial analysis and requirements specification phase of the idea, we move on to what is called the vehicle industrialization phase. This involves the creation of technical drawings, the design of the vehicle, the simulations and tests for the evaluation of the performances of each vehicle component and system, as well as the reliability over time. [8]

This phase is extremely important to detect any design problems and, if necessary, to make the necessary changes at a much lower cost than applying them later. In addition to a cost factor, the advantages offered by an accurate industrialization process are many, whether we are dealing with full designed vehicles, single components or other products, namely:

- Performances optimization;
- Improvement of the degree of reliability;
- Better control over environmental sustainability and End-Of-Life (EOL) predictions;
- Apply more effective design and functionality changes. [8]

The path taken in Iveco for the development of this thesis is set purely with a view to vehicles software reliability, this in order to study and analyse the methodologies and practical approaches used in Iveco to improve their vehicles.

It is necessary to distinguish the reliability work done by Iveco during the pre-marketing period, therefore during the design and industrialization of the vehicle, from the post-marketing one, therefore during the use of the vehicle by the customer. Below it will be described the in-depth analysis made regarding the industrialization phase while in the next chapter the actions taken during the

post-marketing phase, dealing with the customers support claims, will be considered.

This preliminary research activity points to the creation of a classification in which we want to highlight all the issues affecting the reliability of the software, considering that the reliability related to the vehicle's hardware components is now very mature. The interest in the reliability of the software integrated into the vehicle has become increasingly evident since, in the last twenty years, there has been an increase in the presence of electronic components driven by software. Hence, therefore, the need to investigate specifically all the aspects inherent to the reliability of the software, that is how reliable the software that is used in the individual electronic control units and in the integration with other systems and subsystems is.

2.3.1 Focus on software errors

To understand the process of improving the quality of the software embedded into the vehicle ECUs, it is first necessary to focus the attention on what these issues actually are.

The discipline of Software Engineering provides various approaches to software development consisting in systematic operations of management, maintenance and improvement of the software and, because of this, it could be very easy to come across terminologies that are similar to each other but with very different meanings.

Software error defines the misunderstanding or misconception of the software developer, where the developer can be a software engineer, a programmer, a data analyst or a tester. A software error makes the system behave differently as it was expected by the design requirements. The reasons for these errors can be many where the most common are badly defined or misunderstood requirements and syntactic or semantic defects in the logic. On the other side, the outcome of a software error is a fault.

There are different types of faults, which can be divided according to different categories. Some of them are permanent, which means once occurred the software will be unable to recover and will fail. Some of them are temporary, where the software is not working properly for some time, often in seconds or minutes, or the faults even are only instantaneous, but after that it resumes as nothing happened. Some faults occur frequently and can be easily predicted, and/or removed. Moreover, there are faults called sporadic which are of particular interest as it is not possible to predict in any way their nature and behaviour. Sporadic faults may only appear once and never reappear again or appear at irregular intervals. This makes them an element to be constantly sought with extreme attention and, if possible, to intervene and correct in the shortest time. [8]

2.3.2 Classification of software failures with Open Points List

The Open Points List (OPL) documents are used to report and keep track of the actions taken to solve issues during the industrialization phase of the vehicle. It was useful, during this preliminary research phase, to deepen their structure and the methodology with which they are used. These were used to identify software errors, detected in the ECUs and in their integration, that have been the cause of software unreliability issues at the vehicular level.

A generic OPL file is similar to a common spreadsheet in which several figures involved in the industrialization of the vehicle participate: if, for example, the component on which the problem was detected comes from an external supplier, here will be reported all the technical interventions that the latter has carried out in order to resolve it. The history of the interventions carried out is shown in chronological order, accompanying the description with an expected date on which the feedback from the company that initially requested the fix will be provided. [8]

The history of the interventions carried out is certainly helpful to understand the effective approach to use if a similar issue arises again.

In addition to this information, each OP (Open Point) includes the device or electronic system concerned, an indication of the main software functionality affected by the anomaly, a brief description of the issue with its current state and its degree of severity.

These last two parameters require specific in-depth analysis for their understanding, as their values express at a glance important information for the effective reading of the document:

- **Status description**

  This parameter allows to quickly identify which of the points in the list is still subject to evaluation (OPEN), which has been resolved (CLOSED) and which has been annulled (CANCELED).

  The interpretation is made more immediate thanks to the use of specific colours for each box of the status.

  Specifically, the OPEN status can be accompanied by another indication, the sub status. This provides further information about the progress of that point, such as if the issue is awaiting a response from the supplier or if the manufacturer has to validate a fix received from the supplier and to confirm a solution.

- **Severity level**

  As its name suggests, this parameter defines the priority assigned to the open point based on the severity of the issue; here too, as for the description of the status, the interpretation is facilitated by specific colours that recall the assigned severity and by numerical parameters in order of priority.

  This parameter can take on five values starting from level 5, the lowest score, which is assigned to issues with a very low level of severity as can be the case of a non-blocking anomaly that simply requires an update of the software configuration, up to get to level 1, which is the highest score

and is assigned to blocking issues with a very high level of severity, such as in the case of systematic errors that make the embedded system behave differently from what is defined. [8]

The OPL documents that have been studied for the purposes of this discussion, therefore, reported a series of cases with issues relating to the vehicles being designed and manufactured. Subsequently, during the research of the software issues, the exemplary case of software integration between two important vehicular electronic components emerged, which will be discussed in chapter 3. In chapter 4 will be described the project phase of the thesis, with which the purpose was to create a software prototype able to reproduce the scenario during the actual use of these two systems by the customers; thus carrying out experiences and issues from the post-industrialization phase of the vehicle.

2.3.3 Instrumentation for diagnosis and their application on vehicles

To conclude the discussion of the study and in-depth analysis carried out during this preliminary phase of the thesis activity, it is necessary to describe the instrumentation used by the Iveco technicians for the diagnosis and analysis of vehicle data. These acquired parameters are used by the Iveco Quality sector to improve the reliability of the vehicles and are also present in the OPL documents described previously.

During the first period of the internship, there was the opportunity to use and have a look at the instrumentation for the vehicle diagnosis at the Iveco Maintenance and Diagnosis Centre; here standard procedures are carried out for the vehicle maintenance, the improvement of the platform and for the measurement of performance parameters in order to improve the fleet.

These tools, mainly used for vehicles data monitoring both via wired and wireless remote connection, allow the acquisition of some vehicle parameters useful for the evaluation of many aspects. Through this instrumentation it is

possible to carry out maintenance operations to ensure continuous improvement of the vehicle on the market.

The tools in question, for which the company has provided a dedicated day of demonstration sessions on the vehicles, are the following:

- Telemaco
- E.A.SY
- UDT System

The ECUs that make up the electronic structure of the vehicle are equipped with advanced features for self-diagnosis. In fact, with the use of some of the tools listed above, it is possible to read the faults memory of the electronic control units and acquire all the failed parameters. A DTC (Diagnostic Trouble Code) is used to identify the anomaly which, as specified in the SAE J1939 protocol that defines in Europe the data structure of the messages sent on the CAN communication lines and encodes the parameters, indicates the specific error code.

Figure 2.18 schematically shows the structure of the DTC that is composed of four elements:

- Suspect Parameter Number (SPN), indicates on which component or system the anomaly occurred;
- Failure Mode Identifier (FMI), indicates what was the cause of the fault;
- Conversion Method (CM), used for the correct conversion of the SPN code;
- Occurrence Count (OC), indicates how many times the anomaly has occurred.

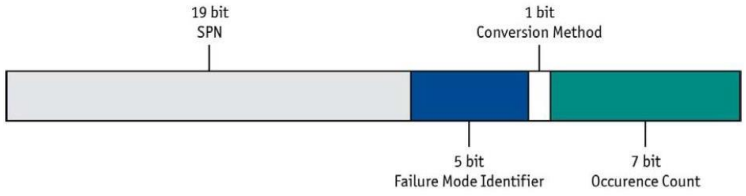


Figure 2.18: Diagnostic Trouble Code. [9]

The parameters are read via TELEMACO by directly connecting the diagnostic tool to the OBD (On Board Diagnostics) port inside the vehicle, and then acquired via a wired ethernet connection to an external notebook PC; the main advantage in using this tool is the continuous acquisition of data since it is connected to the vehicle for the entire diagnostic session.

The E.A.SY (Electronic Advanced System) device allows the diagnosis of the vehicle parameters transmitted via CAN lines. The acquisition takes place through an integrated software, called Silver Scan Tool, installed on the Windows operating system. This tool sees its main utility when quick acquisitions are needed or that do not require a particular operating environment, as it is a compact and shock-resistant diagnostic tool.

Figure 2.19: E.A.SY device for the diagnosis of Iveco vehicles electronic components.

Unlike Telemaco, which allows to acquire collected data already translated from the standard transmission binary code into readable and understandable parameters by the operator, E.a.sy requires a connection interface that performs this task.

The UDT device, the most recent, was released by Iveco together with the release of the heavy range Stralis model. With this one it is possible to directly send signals and/or messages via the CAN line to the vehicle's electronic devices. Once connected, it is possible to select a specific ECU or a specific path in the CAN lines to carry out tests on several interconnected ECUs; after carrying out the test, the system provides all the technical information necessary for the maintenance and verification of the affected components, which can be for example the DTC codes of the error messages encountered.

Figure 2.20: A truck during a diagnostic session.

Chapter 3

Research and analysis of the exemplary case in the Telematics sector

In the second chapter, the various types of errors and software anomalies with the associated severity have been described in detail. During the internship at Iveco, it was possible to get to know and listen to the experience of many people who were able to work personally on important projects for the innovation of the fleet of commercial vehicles offered by the company. Particularly relevant was a detail regarding software errors, or commonly called "bugs" in computer science: nowadays it is very rare to find this type of errors in vehicles; they are practically non-existent. This gave particular emphasis during the research phase of an exemplary case of issue related to the software used in the electronic components of the vehicles.

It should be noted that for most of these software anomalies, if any, it is very likely that they are software faults; these are due to hardware errors and/or failures that lead to different behaviours of the ECU than originally specified during the design phase. This is the case for example of a sensor which, not working as it should, acquires incorrect data, consequently leading to incorrect processing by the software.

The main objective of this thesis is to analyse, verify and validate the embedded software of the ECUs in order to improve the reliability; to this end, in-depth research of all the software integration cases was done in the Telematics sector.

With this research, the aim was to find an issue that took a lot of resources and time to be understood and solved, in order to analyse and learn an effective

methodology to be applied in new software implementations and developments that require integration with electronic components of the vehicle.

An exemplary case that definitely caught the attention during the research in the database of support claims from customers is the one regarding the actions taken in order to correctly and reliably integrate the New Infotainment System, available on the whole range of Iveco vehicles and which is called more simply with its acronym NIS, with the telematics electronic control unit PCM (Processing and Communication Module).

3.1 Study of the case: The integration between NIS and PCM

As seen in the previous chapters, Iveco vehicles are equipped with systems that allow the receipt of wireless data which thus enable an ecosystem of services that can be managed remotely and are always connected.

The heart of the communications that take place between interconnected electronic systems is called “Processing and Communication Module”, shortened as PCM.

This ECU was developed by the Iveco Telematics team, with whom I had the opportunity to work together for the implementation project of a software prototype, described in chapter 4, and to learn the work methodologies and effective approaches for the software development. Thanks to this activity I was able to improve my technical skills, by sharing work plans and carrying out a full project up to its completion entirely using the Agile Scrum methodology.

The goal of the PCM is to improve the driving experience for the driver by providing features such as advanced assisted driving, smart evaluation of driving performance, management of on-board multimedia systems, data exchange for applications that require wireless connection. [10]

Among the on-board electronic instruments connected to this electronic control unit we find the New Infotainment System, shortened as NIS. As the name

implies, Iveco's infotainment system is all that allows the driver to interface with the vehicle and get useful information. Here we find multimedia applications, radio functionality, smart navigation integrated with fleet management, “Iveco No Stop” app to request support at any time of the day and specific information panels dedicated to fuel consumption and the driver's driving performance evaluations.

The operation of each application is due to the software integration of these two systems mentioned above. In fact, each window or panel shown on the NIS is processed and transmitted by the PCM. This occurs via a network protocol over which data packets are exchanged.

3.1.1 Introduction to PCM and NIS

The PCM (Processing and Communication Module) telematics unit is based on a software architecture managed by a custom Linux OS. The basic software of the PCM bundle, the Telematics applications and the framework, that is the layer with all the available services, are produced by different external suppliers together with the hardware. Iveco, therefore, provides the software integration of all of them into the unit. [10]

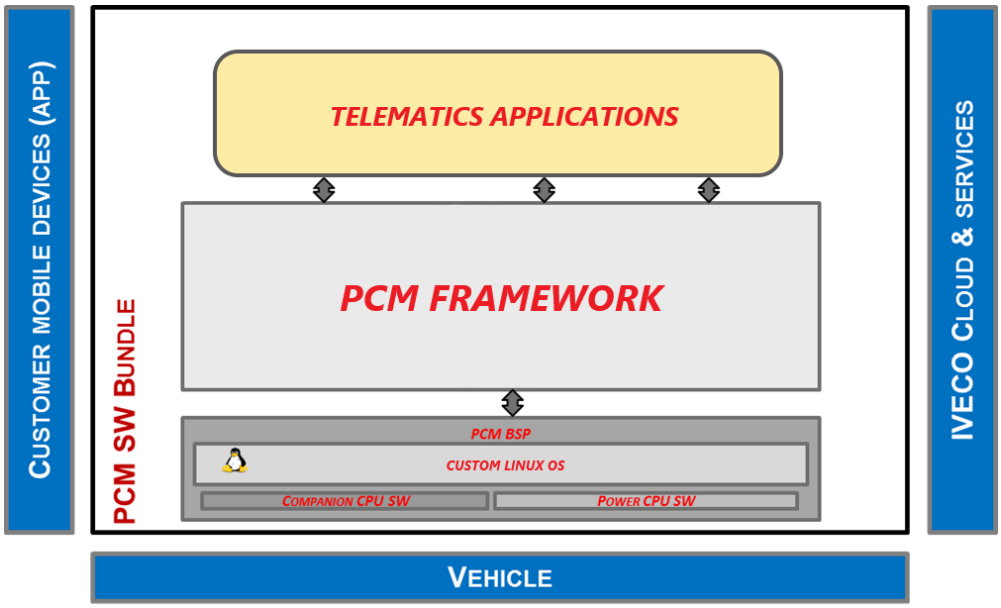


Figure 3.1: PCM software bundle architecture. [10]

Within the proprietary framework we find a series of services that allow to:

- Control any application, service and device, collectively referred as modules, in the system;
- Manage the underlying vehicle CAN Data Bus through a dedicated interface, providing a homogeneous and agnostic view over the vehicular information by mapping digital signals and messages to logical ones;
- Access the hardware through a Bus Gateway from the OS interface layer. [10]

Furthermore, the PCM is designed to guarantee the transmission of data through the implementation of servers used for specific functions such as:

- Interface for diagnostic purposes split into two parts: one is toward the CAN line and can be used by testing devices to make requests into the client protocol, the other one is for applications so that they can make requests and obtain diagnostic data too;
- Enable applications to register itself against one or more URLs and receive the requests directed to such URLs via the related connector. Applications can generate and expose HTML web pages relying on a web server that provides the client with static resources and dynamically generated data, separating the UI management from the server logic;
- Update the applications and all the other software packages by means of a dedicated server that is able to control and monitor all the runtime characteristics and system resources related to the software bundle;
- Exchange messages with other applications or servers invoking the appropriate methods on the related library;
- Validation of new software bundles integrity by signing bundle manifest file with certificates. [10]

The PCM electronic control unit can communicate with the other integrated systems through different channels, based on the required functionality, which are: CAN lines, USB, Wi-Fi, Bluetooth and 2G/3G/4G cellular networks. Figure 3.2 shows the main connections involving on-board and off-board systems with which the ECU exchanges data. It must be emphasized that all connections are privacy-compliant, i.e. they comply with the cybersecurity of the data transmitted. Every signal is processed according to specific protocols for user data protection and to prevent any external malicious attack or unauthorized access to vehicle systems.

Among these interconnected systems there is the NIS (New Infotainment System), a system that is responsible for all the interactions that the driver has with the vehicle's functionalities. This component is manufactured by an external supplier and consists of physical buttons (two knobs and two keys) and a touch-enabled display.

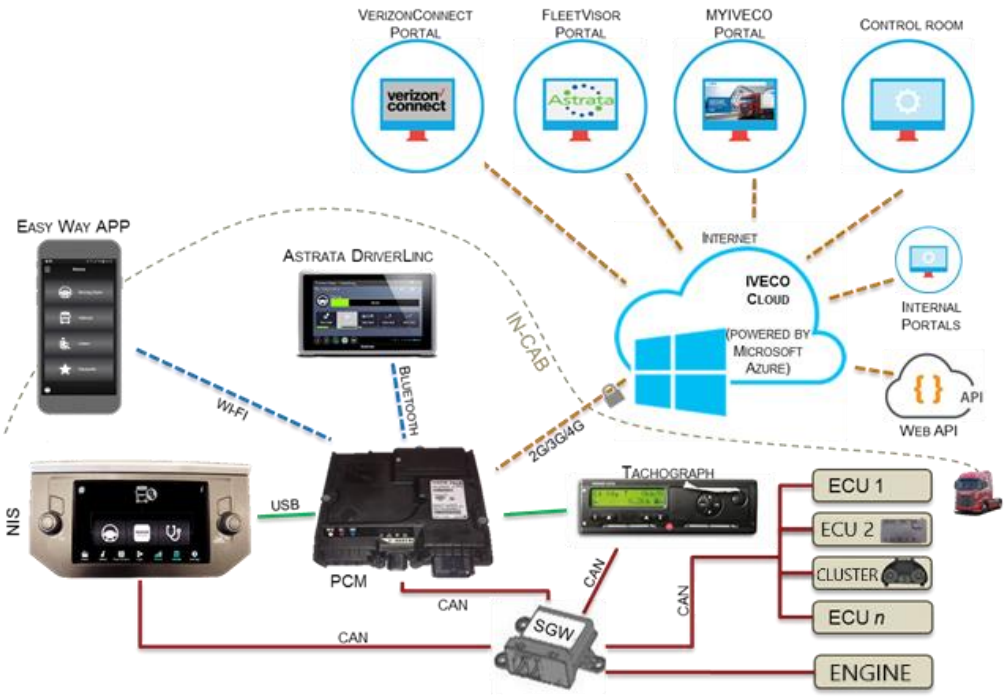


Figure 3.2: PCM connections with other systems. [10]

The NIS is equipped on board the vehicles and offers, integrated with the PCM, an intuitive UI (User Interface) with various functions and applications, navigable through the user's touch inputs. At the bottom of the interface there is a bar that shows the various applications tabs: there are the standard functions for listening to radio channels and music, the access to the rear camera, direct connection to the cellular network and to the services of the “Iveco Easy Way” smartphone app, the “Vehicle” panel and a special section for configuring user settings.

The "Vehicle" panel, shown in Figure 3.3, is where most of the main telematics applications reside. Here there is access to: monitoring for remote vehicle diagnosis (RAS), forwarding of non-stop Iveco support requests (ANS), evaluation of driving style and fuel consumption (DSE), access to vehicle information such as the VIN code (Vehicle Identification Number) and licenses, etc.

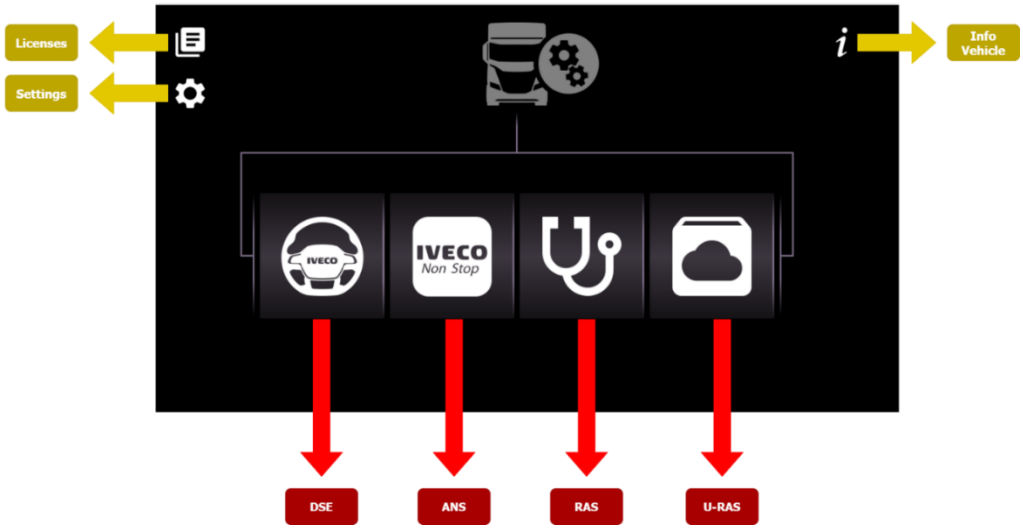


Figure 3.3: Vehicle tab homepage with its applications. [10]

From here the discussion is focused on the thesis study case, which analyses the software integration of the PCM electronic control unit with the NIS infotainment system.

3.1.2 Analysis of their integration

These two systems are designed to provide information regarding different aspects and functions of the vehicle only if the user makes specific selections within the NIS menus and submenus or if there are starting events such as the insertion of the key. Their activity must take place simultaneously according to functional logics.

Let's focus now on the "Vehicle" tab, introduced previously and that from now on will be called VehicleApp for technical simplicity, as it plays a fundamental role in describing the integration method developed by Iveco's Telematics sector.

The VehicleApp application includes core features such as:

- Provide a browsable HTML home page for all PCM applications;
- Host graphical resources for all PCM applications;
- Collect information and dispatch them on-board and off-board. [10]

Each application redirects, with each request made by the user, to a specific web-app accessible as a graphic resource via an http address. This solution was chosen because it offers several advantages, including the possibility of making changes to the software architecture without going to operate on the NIS system and the displayed graphic interface; consequently, there is an effective reduction in maintenance costs and times. Below, images of some applications available within VehicleApp tab are shown:

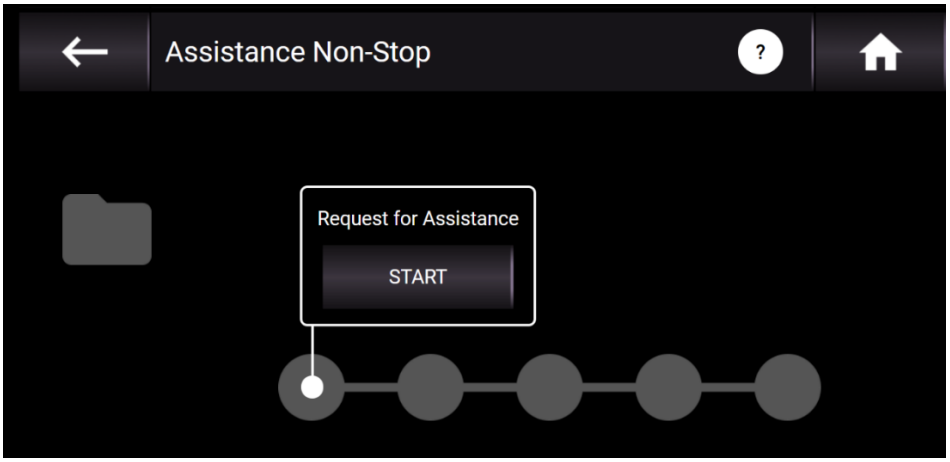


Figure 3.4: VehicleApp: Assistance Non-Stop service. [10]

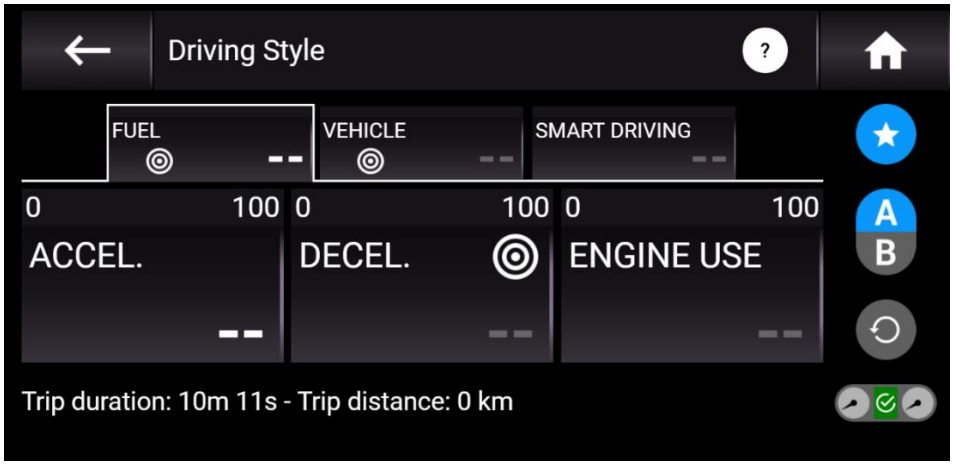


Figure 3.5: VehicleApp: Evaluation of fuel consumption. [10]

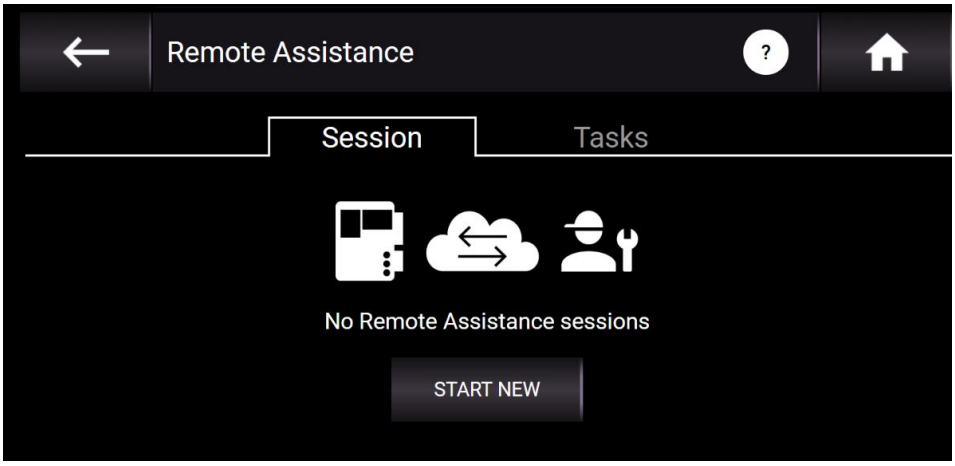


Figure 3.6: VehicleApp: Remote assistance and ECUs monitoring. [10]

NIS is requested to grant the connection via any available USB high speed port toward the external Telematics unit (PCM). The system has to be able to interface via software to the PCM, in order to act as browser for Telematics Applications without system re-design. [11]

Interface to Telematics-based Applications requires the following statements:

- PCM and NIS will be connected to Secure Gateway (SGW) by two separate buses, to separate the “trusted” from “untrusted” signals. This fulfils the cyber security requirement;

- An USB connection will be available between PCM and NIS, using Ethernet-over-USB protocol;
- Telematics-based applications contain the whole application logics, including the graphical resources of the model view part;
- NIS-based web-view applications are almost viewers of the Telematics applications: they don't include logics, but just a HTML5 browser fully integrated with the visualization of Telematics unit applications;
- Telematics-based applications will be displayed in the foreground, according to a separate main logic, into NIS launcher; their accessibility will be managed via the “ping” parameter. [11]

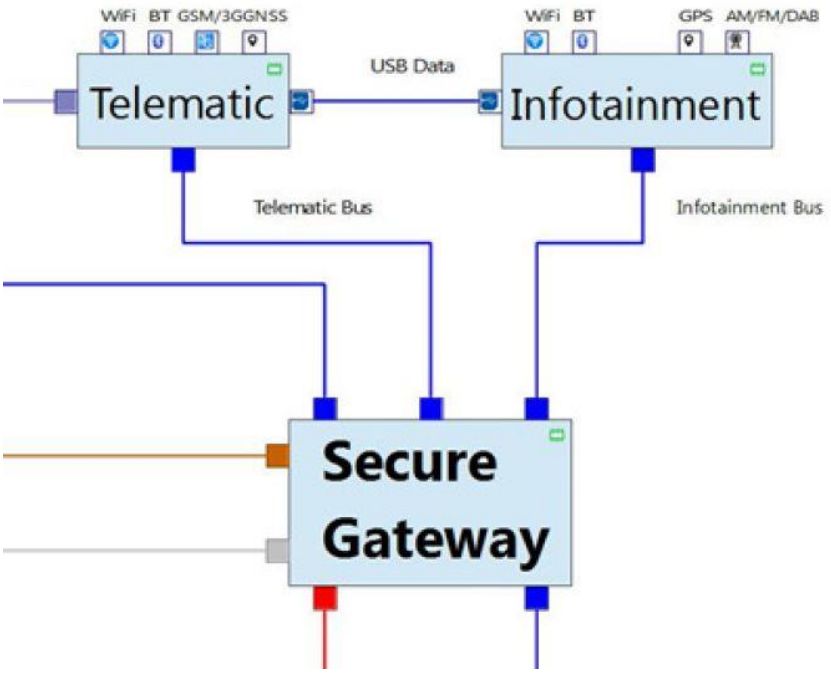


Figure 3.7: Scheme of the Telematics unit (PCM) and Infotainment system (NIS) interconnected through the Secure Gateway (SGW). [11]

As can be seen in the scheme in Figure 3.8, the web-server is the central point of data exchange between the Telematics applications and the NIS web browser. In fact, this is the PCM module that offers and processes the contents to be shown on the NIS in a totally independent way from this system which, as said before,

is produced from an external supplier; the web-server verifies that the Telematics applications have the resources available to serve the request on the display. [11] In order for the requested application page (coming from the PCM telematics unit) to be displayed correctly on the NIS launcher, however, the "ping" logical parameter must be able to complete the path highlighted in purple in the diagram: when a request is made on the infotainment (which corresponds to the user's selection on the launcher), the telematics unit is activated by receiving the request for a new page to dispose on the NIS display.

When the PCM receives a request of new foreground visualization, the NIS will evaluate if it is possible to serve it: to do this, a positive response from the "ping" parameter is needed. Therefore, the following conditions must be met in order to show the new page in the foreground:

- Requested Telematics application has the resources available to send back to the launcher through the web-server;
- Ping monitoring task has access to ExComGw module, that is available and informs that the server is reachable and the request can be fulfilled (in other words, the server has not triggered any http error). [11]

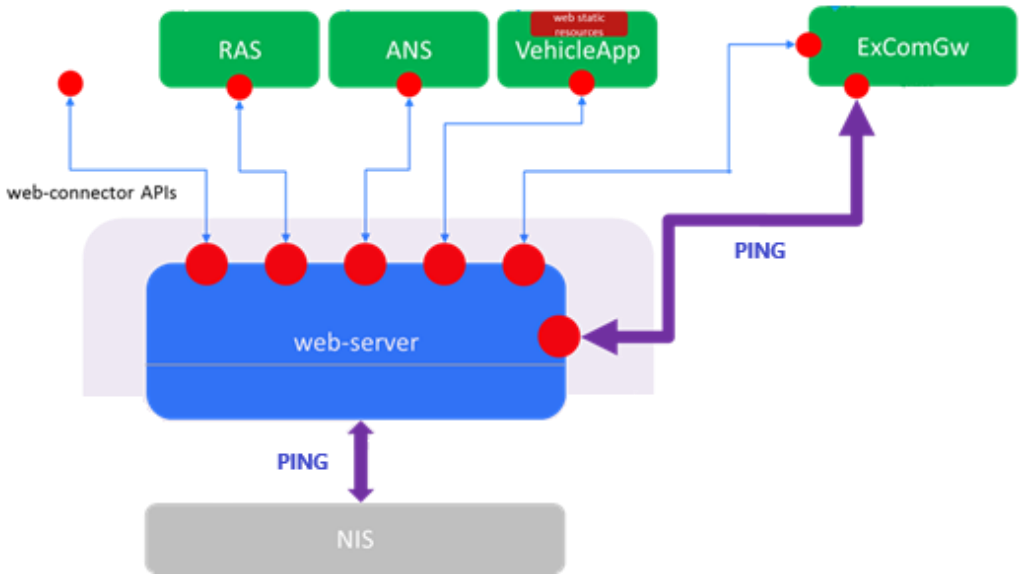


Figure 3.8: Scheme of the software architecture for the management of Telematics applications through the ping monitoring task. [10]

The design of this integrated operation has seen significant revisions during the post-industrialization phase of the vehicles, as several anomalies have been reported by customers who used its functionalities. Precisely for this reason, it was interesting to retrace the interventions carried out in the resolution of the issues raised and in the validation of all the changes made; the following discussion, therefore, describes the research of the issues that occurred in the implementation of the software architecture between NIS and PCM, by reporting the history of the technical interventions taken.

3.2 Creation of the history of issues encountered and related actions taken

To fully understand the nature of this software integration between two electronic components that perform main functions for on-board telematics, it was necessary to search for everything that has been done during the development. These includes all the operations done starting from the first phase of product design up to the improvements made, gradually, as the customers reported any fault.

Once again, the intrinsic importance in this study case should be highlighted, since it is not only a case of integration between two very complex and different systems, in which two completely different software platforms are managed. Above all, also, it required a considerable use of resources and time as it was necessary to communicate effectively with external suppliers, to try to align two separate workflows in the best possible way. This required more effort, during development, by the engineers involved.

The use of the Agile approach in project management helps to simplify the communication between teams and the operations of design and development, including testing and software release, precisely in the projects in which more figures are involved. As already mentioned previously, this approach involves different methodologies: among these, there is “DevOps”.

3.2.1 Focus on DevOps methodology

DevOps methodology is used in software development and it emphasizes collaboration and communication between developers and IT operations professionals, as the name suggests. In fact, it comes from the combination of the two words "Development" and "Operations".

With DevOps, it can be immediately gained a competitive advantage in terms of faster deliveries of better software and continuous innovation; this allows to effectively check the reliability of the software produced.

In traditional organizations the functions "Development", i.e. developers and software engineers, and "Operations", i.e. IT professionals (system engineers, verification and validation engineers), are very often distinct. In short, it can be said that the former deal with code generation and development while the latter deal with the correct integration and release into production.

As these functions have different objectives, there is a risk that some aspects of the workflow may conflict, inevitably leading to slowdowns in production releases and throughout the business.

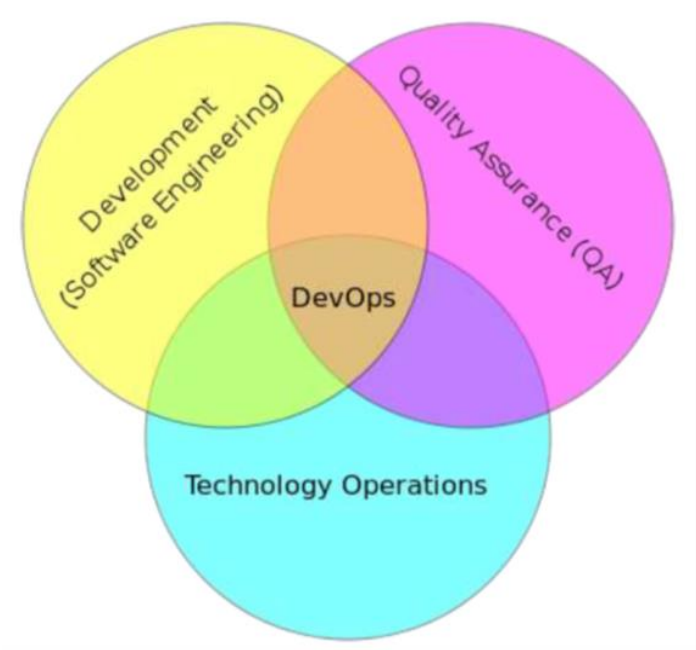


Figure 3.9: DevOps means collaboration among different functions.

The DevOps movement has produced, and continues to produce, numerous principles and tools that can be adopted by organizations of all sizes. All these principles have given rise to a working methodology that aims to improve the way the business provides value to its customers, suppliers and partners. [12]

The advantages of adopting the DevOps methodology are summarised below:

- Improvement of the quality of the code, products and services (fewer failures, higher success rate of changes, etc.);
- Increased effectiveness (for example more time spent on activities that create added value, greater product value for the customer);
- Time-to-market improvement;
- Better IT alignment and business responsiveness;
- Faster, smaller and more frequent releases of code;
- Less waste of resources and fewer anomalies;
- Improvement of productivity, customer satisfaction and staff satisfaction;
- Lower long-term costs. [12]

The opportunity offered to me by the company to be able to study, research and deepen a relevant case, such as the one subject of this thesis, has seen an increase in productivity and better management of time and resources by using the DevOps methodology first and the Scrum methodology later, during development of the software prototype.

Furthermore, by learning these methodologies I was able to further enrich my personal experience and work skills regarding the development of an industrial product according to an Agile approach.

3.2.2 The research of the issues in the company databases

DevOps methodology was first necessary for carrying out the consequent research activity, that is the research of the issues concerning the development of the software integration of the PCM telematics unit and of the NIS on-board infotainment system.

To properly release a PCM software bundle a platform provided by Microsoft is used, that allows to effectively manage all the phases that make up the development cycle of a software release. This is Microsoft Azure DevOps, which is a Software as a service (SaaS) platform that provides an end-to-end DevOps toolchain for developing and deploying software. The platform offers different tools: software version analytics and insights, reporting and feedback through teams capabilites, specifications and requirements management, project management tools, automated builds, test and management of new software functionalities. It basically covers the entire application lifecycle.

In the Figure 3.10 it can be seen the diagram of a pipeline with integration and continuous release of software, managed using the tools offered by Microsoft Azure DevOps platform.

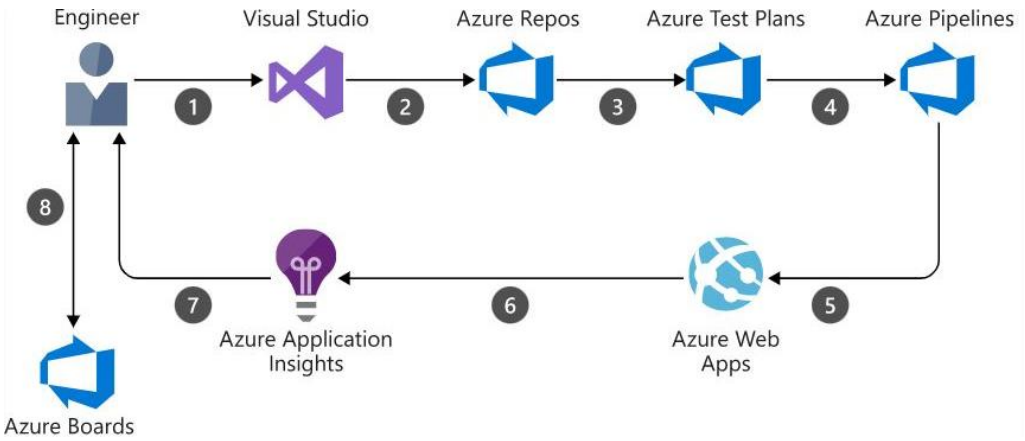


Figure 3.10: Software development cycle using Azure DevOps tools.

Obviously, therefore, if two distinct figures are working on a software release or, as in the thesis software integration study case, two different companies, it is clear that an enormous advantage is obtained by using this platform for the combined management of two workflows.

Here, therefore, at this point of the thesis activity in Iveco, it was necessary to request the access to the Microsoft Azure DevOps server used by the Telematics team to communicate and receive feedback with the external suppliers about the development of the PCM unit software bundles.

After obtaining access from the company to the internal Azure DevOps server a work team for the in-depth research was created, together with me and my company tutor Eng. Pontes. The support team designated for this activity consisted of three people, who gave their contribution to design and develop the software prototype implemented in the project phase of the thesis (which will be discussed in chapter 4). Eng. Lo Iacono has been designated to support me specifically during the development of the software project, as he has participated in the development of the PCM-NIS software integration.

Within the Azure databases, the research was set up in such a way as to focus the results obtained according to our needs: therefore, using keywords and special search filters, the platform provided all the tickets regarding software bugs, errors and faults found on the NIS system.

A total of 73 elements were found in the CLOSED and/or RESOLVED status: the results indicating the OPEN status were deliberately discarded, as the objective of the research was the reconstruction of the actions and technical interventions carried out to correctly fix the NIS system software issues. It should be noted, however, that some open cases have also been found, in fact, even if with lower probability, issues continue to arise.

For the complete reconstruction of the history of issues encountered, the cases found in an OPL file (Open Points List, the function and use of this document is discussed in chapter 2) must also be added to the list.

Analysing the description of the problem, the effect observed by the user, the reproducibility and the type of solution (where present), these issues where then grouped by common aspects in order to build classes. Some of the discovered bugs have been excluded as they were not inherent to the analysed case, leaving therefore 33 exposed cases. These cases were then grouped by "cause" of the problem, finding 8 categories and then by "final effect" of the problem, finding 11 categories.

Among these anomalies, some have been reported several times, highlighting the repetitiveness of the errors in question. Others, on the other hand, totally fall

within the definition of intermittent error (which has been described in chapter 2), that is, it occurs once or at irregular intervals, making the understanding of these issues and their resolution extremely complicated.

All the data collected during the research were then reported in an Excel spreadsheet, attaching detailed graphs with percentages showing the breakdown of the issues according to the classification. In this way, it has been created a further product useful for understanding the resolution procedures and the effects observed that were reported the most.

To conclude the reconstruction of the history of the issues inherent in the PCM-NIS software integration case, it was organized a meeting with a person who gave his availability to provide access to an additional database. This is called Asist and it is used among the Iveco facilities globally; in this database the research was unsuccessful as any report inherent to the under-investigation case was find.

Subsequently, the main issues highlighted by this research undertaken in the company's internal databases are reported and described.

3.3 Significant issues encountered

When dealing with the design of a complex integration such as the one regarding the two electronic systems, PCM (Processing and Communication Module) and NIS (New Infotainment System), it very often requires more efforts in the post product release phase rather than in the preliminary design phase.

As mentioned previously, for the management of an entire software development project, specific platforms are therefore used that provides servers located on cloud infrastructures and advanced tools for the combined management of multiple workflows. Each of these can include code development, building of software packages, various tests on the concerned unit and on the integration, release in production, taking charge of the claims on anomalies and failures. In this scenario it must also be considered that for each workflow it is important to

respect the planned schedules to ensure a continuous release of new software. Often, therefore, issues that may lie at the base of this pyramid of chained tasks, during the development of the code, there is time to make the needed changes and avoid blocking issues or sporadic errors after the release.

In the case of the PCM-NIS software integration various issues arose which mostly affected the applications made available by the PCM on the NIS display, in particular the VehicleApp, but also communication issues involving the ExComGW module; it is responsible for giving feedback about server connection status of the PCM via the network presence parameter (i.e. ping, defined previously).

3.3.1 Analysis and identification: the approach used

The approach used for tracking anomalies within databases can be summarized as follows:

1. Advanced search configuration focused on "SW bugs - SW errors" category;
2. Display of all points relating to the NIS system, filtering the search contents using selected keywords;
3. Unbundling of the list of points obtained on the basis of the status label. Only points in the CLOSED and RESOLVED status are taken into consideration;
4. Grouping of issues according to the effect generated;
5. Analysis of each individual issue using the description of the initial report and of the interventions carried out until resolution (where present);
6. Detection and definition of the different causes that led to the same effect.

In this way it was possible to optimize the identification and the understanding of the causes that led to a consistent number of closed and resolved points, trying to create groupings and redirecting each problem to its solution.

The ticketing system, with which communications take place in the Azure DevOps database, provides for each reported claim:

- The name and surname of the claimant;
- A unique identification code of the ticket;
- The vehicle electronic system affected by the fault/anomaly;
- The version of the software bundle installed at the time of the failure;
- The reproducibility of the problem (if frequent or intermittent);
- Any devices with which the monitoring was carried out;
- The logs with the data collected by the vehicle over time by the self-diagnosis functions available in the Iveco vehicle's electronic control units;
- The logs with the data collected by the maintenance worker who carried out the diagnosis (where available).

Although the availability of this information was necessary to understand the reported claim and the issue encountered and allowed the groupings to be created based on the reported effect, it was certainly not sufficient to make a hypothesis of the cause of the issue itself: an adequate technical reasoning on every single issue, using the notions studied and learned about the PCM and NIS systems, has made it possible to group all the points also based on the cause of the issue.

Through this approach, it was possible to exclude most of the points that were not relevant to the integration case but which perhaps only concerned the NIS infotainment system. For example, the following report clearly refers to an anomaly affecting the NIS system and the Iveco Easy Way App (EWA):

> “ **55848** [SYS_T] Misalignment between NIS and EWA on media section
>
> **...**
>
> When cycling on/off, NIS/media turns ON but the EWA/media section remains disabled ”

Therefore, not reporting software anomalies detected in the integration of NIS and PCM systems, this and tickets with similar claims were excluded from the identification, narrowing the search field and the total number of points to be identified.

Many reports have been found regarding issues on the visualization of the applications user interface in the NIS display for, perhaps, an incorrect initialization of the PCM when the vehicle is turned on: the most striking cases and which have had many technical interventions are those involving the graphical elements called "spinners".

These graphical elements are the solution of the developers of the NIS software architecture to "mask" all the loading times of a new interface/menu or when the vehicle is switched on, as the graphic resources have not yet been made available by the PCM. A “spinner” is also shown in the case of a disconnection of the web-server or the communication channels unavailability between the two systems. The functioning of these two systems must go hand in hand, as described in the in-depth paragraph of their integration above; the “spinners” will also be discussed and further explored in chapter 4 as, for the development of the software simulator of the NIS system, they have been added to faithfully recreate the behaviour of the infotainment.

These "spinners" have been the subject of various issues highlighting, in many cases, design flaws in the VehicleApp application. In the ticket below there is a significant technical intervention that was decisive:

“ **69071** [NIS] Circular spinner - Horizontal spinner - White page

Steps

1. First test:

PCM on suspend, then wake up of the PCM from EWA ->

NIS was ok for few minutes then a Horizontal spinner appears on NIS. Checking EWA

Vehicle info page is in loading. Recovery: restart VehicleApp

2. Second test:

PCM up & running (due to the opening of the car) then put web-server & VehicleApp in debug. After this reboot P&CM ->

After reboot Horizontal spinner appears on NIS and Vehicle info page on EWA is in loading.

Two recoveries have been tried: 1) Restart web-server -> nothing happened. 2) Restart VehicleApp -> the system restarted working well

...

The issue has been observed also during intensive test taking on vehicles with last software bundle… In order to avoid the NIS stuck over a spinner, VehicleApp has been removed from suspend on RAM so at resume (after PCM suspend mode) VehicleApp boots each time without causing any CUSTOMER experience interruption on NIS. ”

In this ticket there is evidence of some issues which, as will be seen in the classification below, have been reported frequently highlighting their repetitive behaviour. The PCM unit manages the start-up of all the applications available on the NIS display on the basis of the assigned priority in the design phase. All the modules that participate in the connectivity and in the communication interfaces of the two systems are therefore activated one by one respecting the priority assigned to each NIS functionality: this priority list generally sees different “run levels” where the activation of key functions for the safety of the driver and the vehicle cover the highest levels and are activated first while the remaining functions, such as those offered by the VehicleApp application, are put on hold until their activation level is reached.

For this reason, there have been issues such as the one described in the above ticket, where the VehicleApp application remained blocked on "horizontal spinner” despite the fact that the PCM was in the state of suspension; this graphic element, in fact, should only be shown whenever the VehicleApp is not active but the PCM unit is still ready to provide the necessary http resources. The driver in this situation could not do anything.

So, two were the solutions: VehicleApp was moved to the right run level assigning it a higher priority, in order to avoid that the PCM unit is in the "suspend to run" state when the VehicleApp functions are requested, and the

"circular spinner" was introduced, a graphic element that suggests a loading in progress every time the activation from the suspend state of the PCM is expected.

All categories of significant issues that are the result of this analysis and identification will now be classified.

3.3.2 Classification by cause and final effect

From the research of the issues inherent to the case of software integration between the PCM electronic control unit and NIS system within the company databases, a total of 73 exposed cases have been found. All these cases have been solved or closed, as they were not blocking or indirectly solved.

After the screening of the issues concerning only one of the two systems or not inherent to telematics applications, 33 cases remained of which an in-depth analysis and identification was made for each following the approach described above.

From here, having these data, it was possible to see the integration case with more clarity on its complexity. A lot of work has been done to improve the design of these components and achieve an excellent level of integration. This leads to a reflection: innovating is not easy but, starting with the right idea and effectively organizing time and resources, it is possible to make constant improvements, in order to provide the customer with cutting-edge technologies that give the security of being able to start the vehicle and have everything under control.

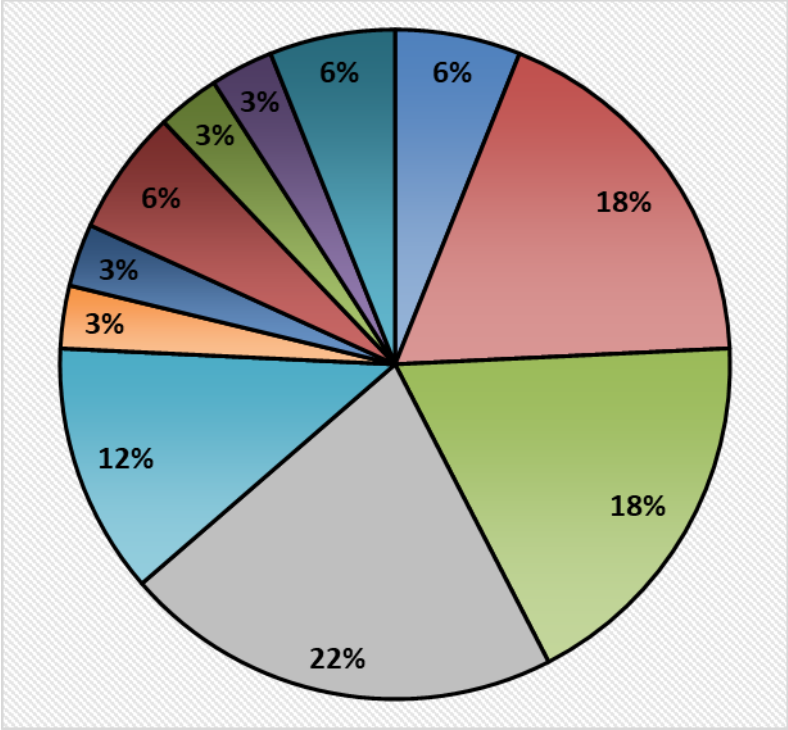
All 33 issues resulting from the research, analysis and identification activity were classified according to two criteria:

- Final effect generated, obtaining the eleven categories in Table 3.1;
- Triggering cause and applied solution, obtaining the eight types in Table 3.2.

The data collected are represented by a qualitative graph with the percentages of finding of each class with respect to the total number of cases found.

Table 3.1: Classification of PCM-NIS software integration issues by final effect.

CATEGORIES OF FINAL EFFECTS	CASES FOUND
Horizontal Spinner	2
Circular Spinner	6
Infinite Spinner	6
White Page	7
Black Page	4
VehicleApp running false	1
Wrong visualization of NIS page	1
Web Server Error	2
Comunication Error	1
High CPU Consumption	1
NIS Blocked	2

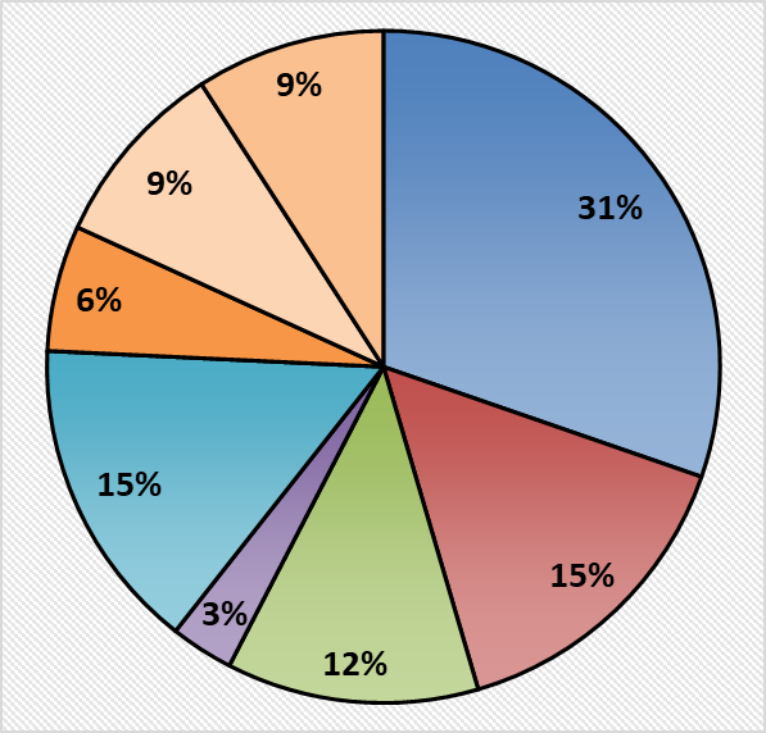


Total Tickets Number: 33

Total Final Effects Number: 11

Table 3.2: Classification of PCM-NIS software integration issues by cause.

TYPES OF CAUSES	SOLUTIONS	CASES FOUND
Framework issue	LiFT Release	10
VehicleApp application issue	SW Release	5
Application configuration issue	Bundle Release	4
ExComGW application issue	SW Release	1
Issue no longer encountered	Resolved indirectly	5
Issue not resolved (Borderline case)	-	2
Issue on NIS	SW Release	3
Issue not resolved on NIS (Borderline case)	-	3



Total Tickets Number: 33

Total Causes Number: 8

Chapter 4

Implementation of the software prototype

After an initial phase of study and in-depth analysis of the entire Iveco vehicle range, the connected functions and the cutting-edge systems that make up the electronic architecture, the work in the company continued with the research of software issues. For each of these issues, an attempt was made to reconstruct the resolution context and the components involved.

From a careful analysis of various claims regarding faults in the Iveco heavy range electronic control units the significant issues were then highlighted, namely those that had blocking effects for customers who use the vehicles.

By creating the groupings of the signals sought, two vehicle electronic components particularly caught the attention, which with their integration provide main functions for on-board telematics and connectivity: the NIS infotainment system and the PCM electronic control unit.

This case of software integration between two different electronic systems that perform completely different functions was of greatest interest for the Telematics sector of Iveco. In fact, these are two components that enable main functions for the driver and for which various technical interventions were required to ensure that everything works properly. This aspect was highlighted by the classification of the software issues inherent to the exemplary case, which shows the frequency of encountering the major issues.

With this, the classification reported in the previous chapter represents the culmination of the analytical phase of the thesis activity and the beginning of the experimental project phase which this chapter will deal with.

4.1 Description of the project

The idea of creating a simulator of the NIS infotainment system arose from the need to simplify the testing and validation operations currently carried out on this component. It has been seen, in fact, how the software integration between NIS and PCM required a lot of time and resources to be implemented at a high level of design.

Although the two electronic systems in question are extremely different from each other due to the functions they must perform, they have the common goal of providing information.

Specifically, this information can have different priorities assigned. Telematics applications must be able to constantly receive data from several main vehicle systems, to provide information that gives the user safety while traveling on the road. It goes without saying, therefore, that guaranteeing the reliability of the components involved is of high importance.

Many of the software issues encountered in this exemplary software integration case affected the way the information was shown on the infotainment display via the telematics control unit. For example, these were blocking errors that prevented customers from obtaining relevant information on their driving status or even unable to access the remote assistance offered by Iveco.

Iveco employees had to manage and find the most suitable solution to correct all the points that the customer found, as the software did not comply with what was expected. The two systems were not well synchronized with each other, so further checks were always needed to study, deepen, analyze and improve their software integration.

What has also made the integration between the two systems more complex is certainly the participation of two different companies in the project. In fact, the PCM control unit is produced by Iveco S.p.A. while the NIS infotainment system is supplied by an external partner.

With these premises, the opportunity offered me by Iveco was born. I was asked to design a NIS simulator with the support of a team made up of engineers in the Telematics sector. In this project, the first goal to be achieved was to reproduce the behavior of the NIS system faithfully. Subsequently, to experiment the reproduction of the issues occurred in the past, detected by the claims analysis, for the functional validation of this software prototype.

Therefore, after this application calibration phase, it was possible to develop an innovative functionality: automated testing. Through the automated tests, the goal is to allow to detect many software issues in the shortest time and further simplify the testing and validation operations. The automated procedures avoid long and repetitive manual interventions for the verification and analysis of the component.

The main purpose of this implementation is to bring out as soon as possible software issues related to the visualization and processing of contents between the two integrated electronic systems, which would otherwise lead to malfunctions. By providing the ability to emulate the external partner's component on a personal computer, there is no need to connect hardware to the test bench. In this way, the testing and validation process of the Iveco telematic control unit is faster and more accessible.

4.2 Technical preparation for project development

To start the development of the NIS simulator, a work plan was first set up, in which the various phases of the project were defined. Throughout this part of the internship I was able to use the fundamental concepts of the Agile Scrum methodology which provides for targeted meetings according to portions of work called "sprints".

The advantage of this software development methodology is the total absence of downtime. In fact, having attended Scrum meetings daily of the Telematics team, I was able to avoid slowdowns for the entire duration of the project as I was provided with adequate support for its development.

The first day of work on the project was essential to properly set up the work. We then organized a preliminary meeting in the company to define the technical details of the activity.

During this meeting the PCM electronic control unit and the NIS system, their communication interface and all that has been done in the past to improve the implementation of the two electronic systems were defined in detail. I was also provided with exhaustive documentation to further investigate the two undisputed protagonists of this thesis (described in chapter 3).

After the necessary analysis of the internal documentation, I was then offered the experimental activity object of this implementation: reconstructing the NIS-PCM integration through software simulation.

Assuming that I am particularly inclined and attracted by the Software Engineering field, I welcomed the proposal with particular interest, even though I was not completely confident in my technical preparation. This feeling was one of the most important personal motivations throughout the activity, as the desire to understand new development techniques and to innovate has always pushed me to act.

4.2.1 Technologies used

The entire application was developed with the C-sharp programming language (C#). It is a flexible programming language that developers can use for virtually every type of software or application development and testing purposes. For the purpose of developing a desktop application specifically for Windows, this language is widely used and specifically tailored to the architecture of Microsoft's OS. [13]

Versatility might be the most salient feature of C#, but there are plenty of other advantages for anyone who works with it. Some of the most important ones include:

- **Faster development time**

  C# has several features that allow developers to code faster than with other languages such as statically typed and ease-to-read language.

- **High scalability**

  Engineers can quickly make adjustments and build on top of any C# program to expand its functionality and support more users.

- **Object-oriented**

  Being object-oriented allows C# to be highly efficient and extremely flexible, all of which makes development easier and less resource-intensive.

- **Gentle learning curve**

  C# shares many of the same features and overall approach to programming as other popular languages, making it easier to understand.

- **Vast support community**

  C# is one of the most widely used languages in the world, which means that there are many sources available for documentation and support. [13]

This programming language, therefore, has all the features necessary to meet the requirements defined for the development of the software prototype. Being an object-oriented type of language and its familiarity with the C language (that I have already used) allowed me to manage code writing effectively despite having no previous experience with C#.

As for the development environment, the choice fell on Microsoft Visual Studio. This hugely popular Integrated Development Environment (IDE) is the perfect companion to the C# programming language, as they are both created by Microsoft.

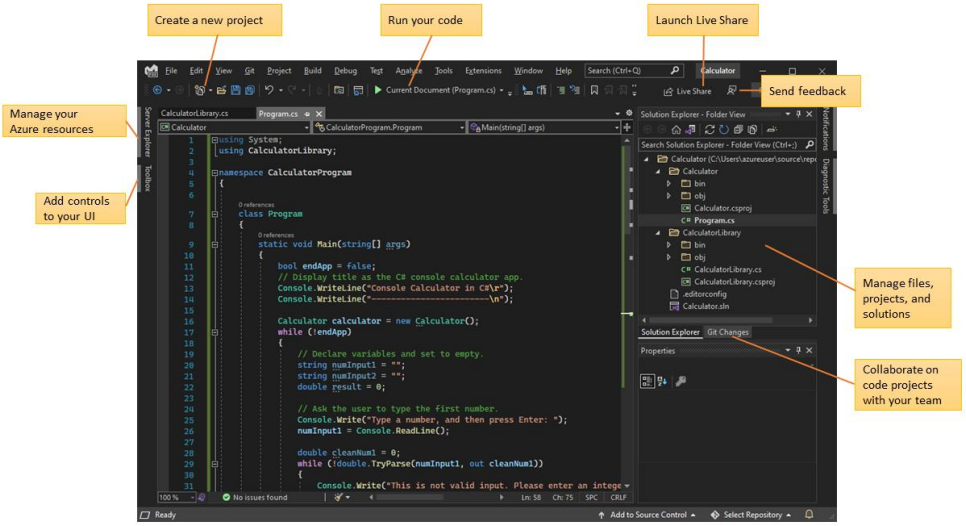


Figure 4.1: The Visual Studio IDE, with callouts indicating the location of key features and functionality. [14]

Visual Studio includes features that have improved productivity during software development, making coding more effective and faster. Most of these features had great impact on my productivity during the project phase. Some of the most popular are:

- **Squiggles and Quick Actions**

  Squiggles are wavy underlines that alert you to errors or potential problems in your code as you type. These visual clues help you fix problems immediately, without waiting to discover errors during build or runtime. If you hover over a squiggle, you see more information about the error. A lightbulb might also appear in the left margin showing Quick Actions you can take to fix the error.

- **Refactoring**

  Refactoring includes operations such as intelligent renaming of variables, extracting one or more lines of code into a new method, and changing the order of method parameters.

- **IntelliSense**

  IntelliSense is a set of features that display information about your code directly in the editor and, in some cases, write small bits of code for you. It's like having basic documentation inline in the editor, so you don't have to look up type information elsewhere.

- **Call Hierarchy**

  The Call Hierarchy window shows the methods that call a selected method. This information can be useful when you're thinking about changing or removing the method, or when you're trying to track down a bug.

- **CodeLens**

  CodeLens helps you find code references, code changes, linked bugs, work items, code reviews, and unit tests, without leaving the editor.

- **Go To Definition**

  The Go To Definition feature takes you directly to the location of a function or type definition. [14]

Among the main features of Visual Studio we find the availability of the .NET platform, fully integrated with the writing of code in C# language.

As a developer platform, .NET includes libraries, reusable components, programming languages and a variety of tools that help engineers develop, compile and deploy modern software for virtually any use case.

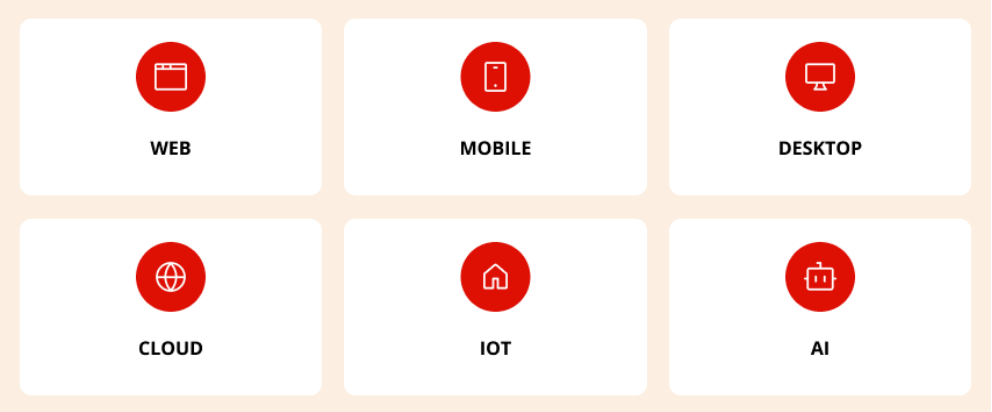


Figure 4.2: The various application targets of .NET platform. [15]

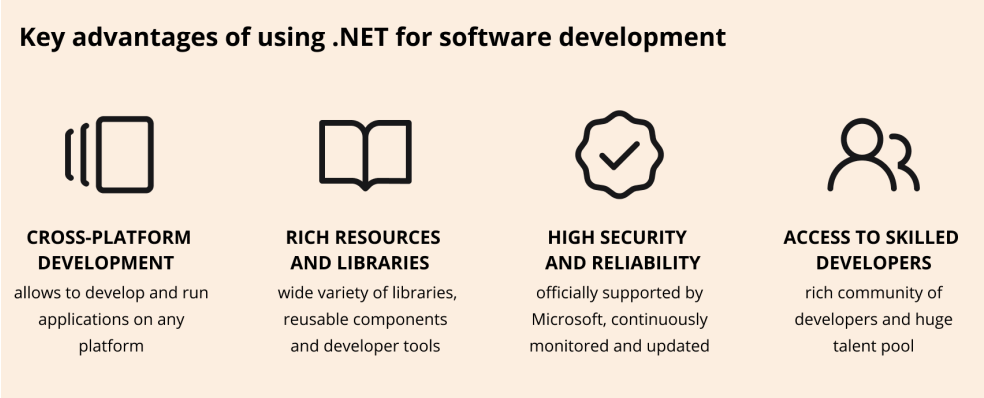


Figure 4.3: Key advantages of using .NET platform. [15]

This particular platform includes the .NET Framework for software development, which allows to build and run web and desktop applications on Windows. The ability to manage desktop applications with integrated web browsers was fundamental for the creation of the NIS simulator, as it enabled key PCM functions availability. The versatility of this platform in extending the basic functionalities available was fundamental to solve various blocking issues during the integration of the NIS simulator with the PCM control unit. In the following discussions, all the difficulties and problems encountered during development will be highlighted as well as all the resolving methods that have proved effective.

4.2.2 The MVVM architectural pattern

The newly introduced technologies are functional to modern software development, providing countless tools for simplified and immediate design. Managing software applications involves several factors. The more the functionalities implemented increase, the more the resources to be managed considerably increase the number of lines of code. Adopting an organized method of software development can be very helpful.

There are many architectural patterns to organize the workflow of designers and software engineers. Certainly, among the most popular there is the Model-View-ViewModel (MVVM) architecture.

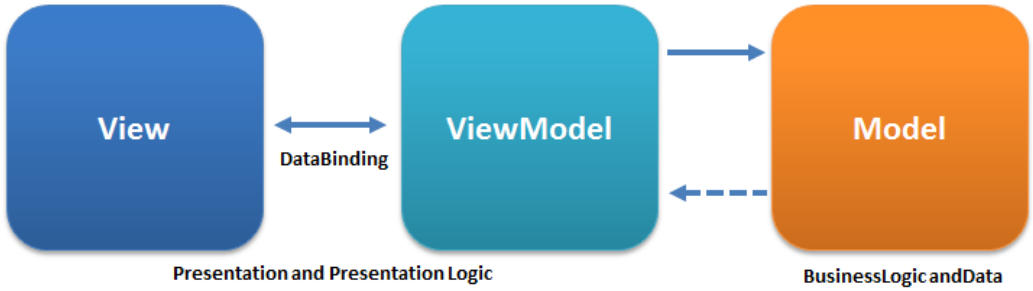


Figure 4.4: MVVM software architecture. [16]

The MVVM pattern structures the software as in Figure 4.4, using the following components:

- **Model**

  The Model is where the business logic of the application and the domain model reside to communicate with other objects of various entities. The model code contains various methods to execute and is designed independently of the specific UI platform. The software prototype model is managed by the CoreLogic class code, entirely written in C#. Here is where the network communication with the external electronic control unit takes place.

- **View**

  The View is responsible for defining the structure, layout and appearance of what the user sees on the screen. In the software prototype it is defined as MainWindow, designed exclusively with XAML markup language.

- **ViewModel**

  The ViewModel is the intermediary between the view and the model. He is responsible for managing the logic of the view. Here the methods of the model classes are invoked, supplying the data obtained from the model to the specific view that requested the data. In the software prototype it is defined as a class written in C#. [16]

The main benefit of implementing the software prototype using the MVVM pattern is the improvement of the application testability due to the clear separation between the application logic and the user interface. Furthermore, it increases the opportunities for reusability of developed technologies and written code.

It will be now discussed the design of the NIS simulator, realized with the use of these software technologies and development methodologies learned. The actions taken to integrate the software prototype with the PCM control unit will then be described. Finally, we will deal with the implementation of the innovative functionality of automated testing for the verification and validation of the component.

4.3 Design of the NIS simulator

The software prototype is intended to faithfully emulate the behaviour of the infotainment system. Reaching this goal has seen several stages of development, in accordance with the Scrum methodology which involves organizing the

development phase using iterative processes, dividing all the work into many portions.

The first step was to create the architecture of the prototype running on a Personal Computer (PC), developing the graphic interface and the interaction logic for each element. The second step was then to implement the main logic for processing the signals received by the PCM.

The infotainment of Iveco vehicles includes various menus and submenus to access the various multimedia functions, remote support services, vehicle information and telematics applications for driving assistance. All the elements shown on the NIS display are actually web pages carefully processed by the PCM control unit, which must be available at each new input given by the user.

Since this is a software integration between two electronic systems that constantly exchange information via network protocol, those who designed these systems had to take into account the possibility in which, for any reason, the PCM is not available or is inactive. During the phase of defining the functional specifications, therefore, the developers of the NIS have devised a solution that exploits a logical flow of interaction with a status signal.

The complexity of this behavior was evident during the first phase of the software prototype, as it took several interventions to reproduce it using only the personal computer.

4.3.1 Creating the user interface

There are very popular design templates that can often help software development but managing and considering a large number of graphics can still be tricky. This difficulty is usually due to design practices that require a lot of effort and a lot of written code to effectively integrate the UI (User Interface) part with the core logic part of the application.

Visual Studio provides access to an entire ecosystem of libraries and tools that simplify the development of the desktop application. Specifically, what was

definitely useful for creating the simulator user interface is the Windows Presentation Foundation (WPF) platform. This platform exploits the main features of the Model-View-ViewModel (MVVM) architectural pattern and is currently one of the most used.

As the whole application takes shape and the number of functions grows considerably, the number of lines of code and parts interacting with each other increases. It is therefore essential to adopt a design method organized according to architectural patterns. The WPF architecture was used for the development of the NIS simulator.

An essential feature of this platform is data binding: UI elements can be easily chained to methods to execute, written in the code behind in C#.

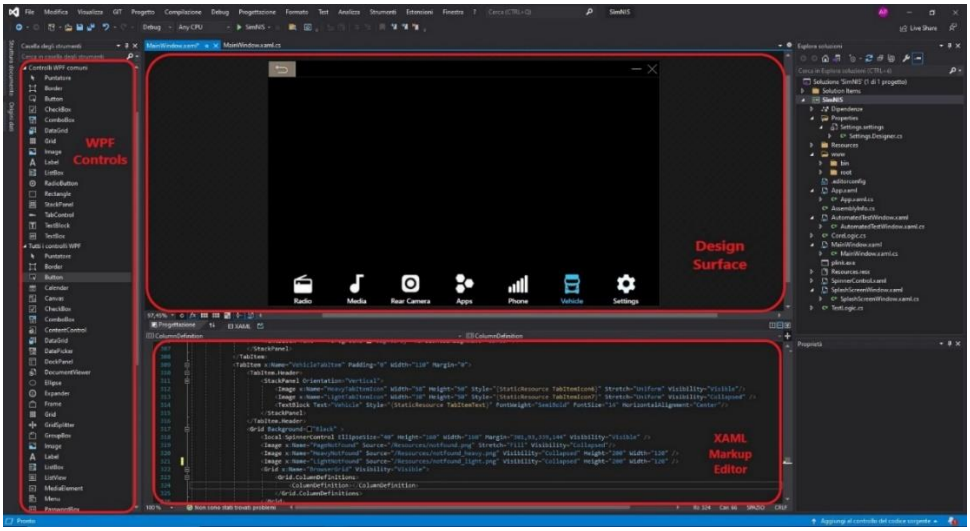


Figure 4.5: Design area for the simulator's user inteface.

The UI design interface in Visual Studio is configured as in Figure 4.5. The creation of the graphic elements is done in a very intuitive way, by selecting and dragging the desired control in the design surface from the column on the left.

The controls of WPF are different and each has a specific function: we can find from the simple Button to more advanced controls such as the TabControl. With this control it was possible to faithfully reproduce the interface for navigation on

NIS system, consisting of the various tabs of the available functions (Radio, Media, Camera, Apps, Phone, Vehicle, Settings).

Each control, once added into the design surface, creates its own XAML markup in the editor below. XAML (short for eXtensible Application Markup Language) is the language used to describe the characteristics of the UI elements of applications based on the WPF platform.

Taking advantage of the data binding offered by this platform, each property that makes up the markup code of the elements is accessible and editable from the code behind. Here the potential offered by this technology is highlighted which, in full compliance with the MVVM architectural pattern, allows separating the code of the view of the model from that which manages the functional logic of each element. The advantage is the total reusability of the code by multiple elements or the easy reconfigurability of portions of code for any implementation of new features, without having to worry about the graphical interface.

Moving now specifically to the Vehicle tab, this is where the user sees everything that the PCM control unit processes on behalf of the NIS; this is the VehicleApp, already mentioned previously. Through this tab it is possible to view the NIS homepage.

The NIS simulator uses an implementation logic based on a "ping", that is a parameter that communicates the current status of the PCM: the telematics ECU, in fact, is designed to activate only when a new request is received from the infotainment system or by other interconnected systems. As when starting the vehicle after turning the key in the pawl, the PCM processes the requested contents following the order of priority assigned to the functionality to be disposed. This means that until the PCM is available, no content can be viewed on the NIS.

To overcome this, the NIS displays particular graphic elements that suggest a loading in progress to the user. These elements are shown only when necessary,

i.e. when the “ping” parameter returns the unavailability value for a certain period of time.

Below is shown the XAML code that manages the properties of the UI elements to display in the Vehicle tab:

```
<local:SpinnerControl EllipseSize="40" Height="160" Width="160"
Margin="301,93,339,144" Visibility="Visible"/>
<Image x:Name="PageNotFound" Source="/Resources/notfound.png"
Stretch="Fill" Visibility="Collapsed"/>
<Image x:Name="HeavyNotFound" Source="/Resources/notfound_heavy.png"
Visibility="Collapsed" Height="200" Width="120"/>
<Image x:Name="LightNotFound" Source="/Resources/notfound_light.png"
Visibility="Collapsed" Height="200" Width="120"/>
<Grid x:Name="BrowserGrid" Visibility="Visible">
    <Grid.ColumnDefinitions>
        <ColumnDefinition></ColumnDefinition>
    </Grid.ColumnDefinitions>
</Grid>
```

The "Visibility" property is managed by the code in C# and enables/disables the displaying of the graphic elements to be shown during loading times or when the PCM is not available.

The Circular Spinner, shown in Figure 4.6, has been implemented using a User Control, a tool available in the WPF platform, and displays the animation shown by the NIS infotainment system during loading times or when the PCM is not available.

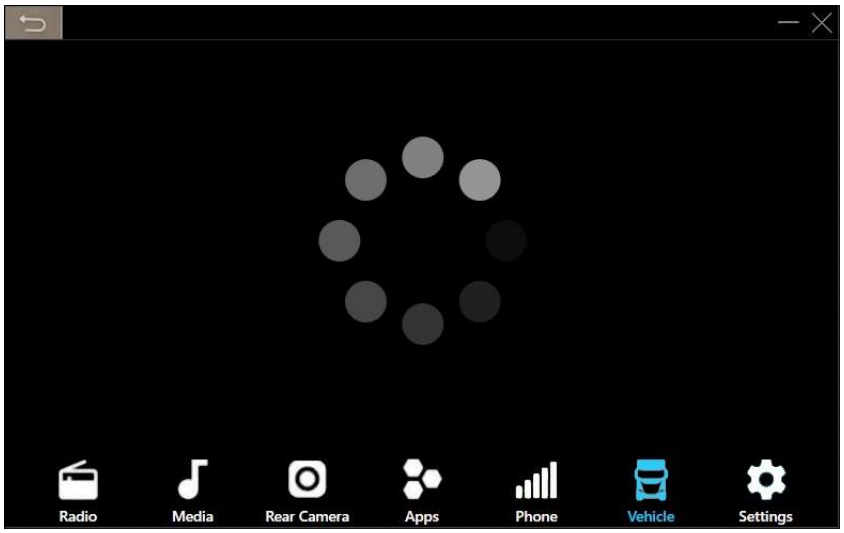


Figure 4.6: Circular Spinner in the NIS simulator.

The Error Page, in Figure 4.7, is shown after a prolonged period of unavailability of the PCM, for example in the case of malfunctions or system blocks. To recreate this page, showing a combination of images provided by the Telematics team was enough. The page adjusts its graphical contents to show the vehicle currently configured in the simulator.

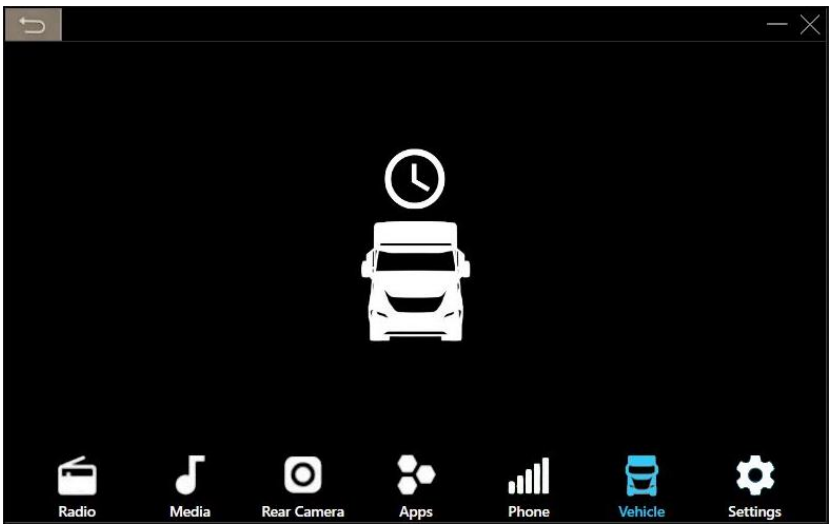


Figure 4.7: Error Page in the NIS simulator.

Finally, a splash screen has been added showing the Iveco logo, to be shown when the application is started. This graphic control element consists of a window that masks the application as long as it is being loaded. It allows the completion of the initialization operations in the background and avoids giving access to the user interface when all the functions are not yet available.

Figure 4.8: Splash screen shown at NIS simulator startup.

4.3.2 Implementation of the logic flow

For the correct functioning of the software integration between NIS and PCM, the developers implemented a communication interface based on a logic flow.

The PCM control unit processes the contents shown on the display of the NIS infotainment system but, when this does not happen, feedback between the software architectures of the two completely different components must be guaranteed.

Introduced previously, the “ping” network presence parameter acts as a communication bridge between the two systems. Its value is critical to the functioning of all software integration.

This complex communication between two vehicular electronic systems took a lot of work to find the correct implementation. Most of the problems encountered are due to delays in the processing of content by the PCM or due to incorrect configuration of the VehicleApp application. The final effect most reported by customers using the in-vehicle features refers to infotainment systems stuck on white pages.

From the in-depth study of the resolutive interventions applied, it is clear that adopting a configuration of the VehicleApp specifically designed to interact with the PCM has solved most of the problems.

The effective solution was to make the NIS system show the requested pages only when the PCM control unit is available and has processed the content. If there is no availability on the part of the PCM, the application shows screens that inform the user to wait. In this way, the PCM control unit is guaranteed time to process the response and the NIS system is still operational for the user.

The implementation of the logic flow inside the simulator has seen a large part of the first phase of the project. With it, it was possible to enable the core part of the software architecture of the prototype.

I was provided with internal documents with the functional specifications of the PCM control unit to deepen the logical scheme of interaction with the NIS

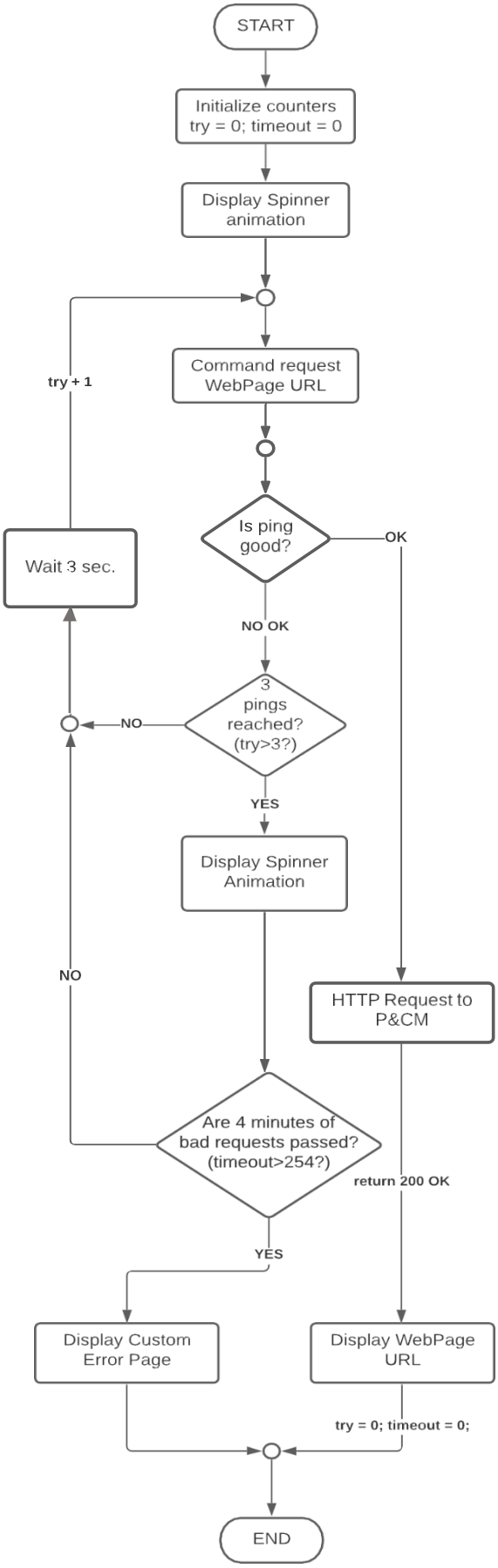
system. Here I was able to fully understand the solution adopted and then develop the logical scheme valid for my prototype. The scheme shows the path of the ping to check the status of the PCM and the web page to be displayed. The C# written code for the implementation of the logic flow in the software prototype and the scheme in question are shown below.

```
while (currentStatus != Status.Idle)
{
    HttpWebRequest req = (HttpWebRequest)WebRequest.Create(address);
    req.AllowAutoRedirect = false;
    req.Timeout = 500;

    try
    {
       using (HttpWebResponse response = (HttpWebResponse)req.GetResponse())
        {
            startTime = DateTimeOffset.Now.ToUnixTimeSeconds();
            request = 0;

            if (currentStatus != Status.Success)
            {
                Debug.WriteLine("\nSUCCESS.");
                OnPingSuccess();
            }
        }
    }
    catch (Exception ex)
    {
        Debug.WriteLine("(" + request + ") NO SUCCESS...");
        if (currentStatus != Status.Timeout)
        {
           if (currentStatus != Status.Error)
           {
              if (request > 2)
              {
                 OnSpinnerLaunch();
                 currentStatus = Status.Error;
              }
           }
           else
           {
               currentTime = DateTimeOffset.Now.ToUnixTimeSeconds();
               if (currentTime - startTime > timeout)
               {
                  OnErrorTimeout();
               }
           }
        }
        request++;
    }
    Thread.Sleep(3000);
}
```

Figure 4.9: Scheme for PCM-NIS interaction logic.



To check the availability of the web page to the PCM a new ping is created every three seconds, that is an http request to the "…/ConnectionStatus" address of the ExComGW. This module of the PCM signals the activity status of the control unit.

If a successful response is received, then it means that the ECU is active and available to send the page on the infotainment display. At this point the application shows the NIS homepage sent by the PCM.

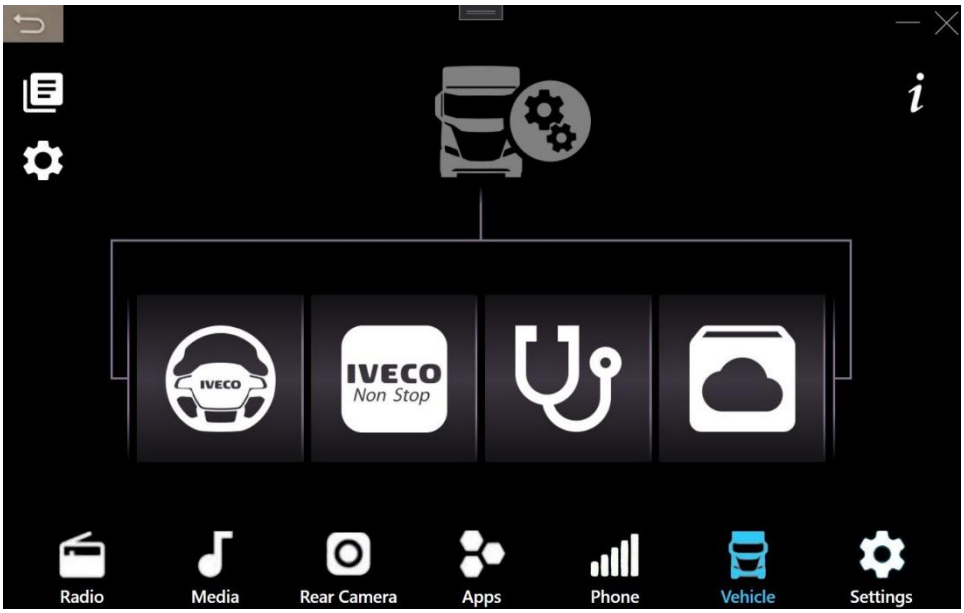


Figure 4.10: VehicleApp homepage displayed in the NIS simulator.

For the management of the element to be shown in the VehicleApp, whether it is the homepage or the circular spinner or the error page, the application uses a method that records the events arriving from the core logic and carries out the operations based on the event received.

Having never used this programming technique before, it was useful to deepen it through the official documentation available online. To keep the implementation of the logic separated from the code that manages the graphical interface, the use of C# events and delegates was essential.

In case of an unsuccessful response, the logic catches the exception and restarts by sending a new ping request.

The "request" counter variable counts the unsuccessful http requests made and if the following thresholds are reached without receiving a successful response, the application shows the graphic control elements:

- After 3 unsuccessful requests (equal to 9 seconds) ➔ **Activation of circular spinner**
- After 240 seconds from the first unsuccessful request ➔ **Displaying of error page**

With this core logic, the prototype perfectly emulates all the operating conditions of the real NIS system when selecting a content on the display.

In the event of an error condition (therefore after 240 seconds of unsuccessful requests) the user can generally restart the logical flow from the beginning by simply pressing the "Back" button on the NIS hardware keypad. This possibility has also been integrated in the simulator by means of the button visible at the top left in Figure 4.10, which resets all the counters and restarts the CoreLogic.

Having kept the CoreLogic code separated from all the rest of the code makes this prototype potentially exploitable also by future new logic implementations.

4.3.3 Application configuration and PC interaction

The software prototype was developed using the Model-View-ViewModel software architectural pattern, which made it possible to implement different configurations in the simulator. Simply using the same methods of CoreLogic with different parameters sent by the model or even different graphics elements.

Initially, the configurable parameters were designed in the Settings tab, visible in Figure 4.11.

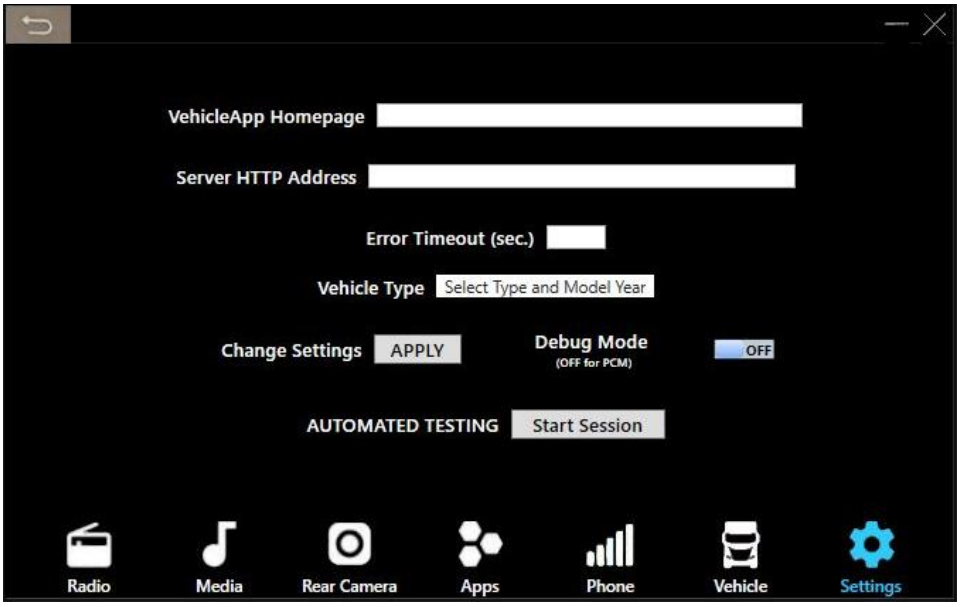


Figure 4.11: Settings panel in the software prototype.

The application parameters that can be modified are the following:

- **VehicleApp Homepage**

  The address of the homepage to show in Vehicle tab. Making this parameter editable was necessary for testing and debugging purposes. The application has default URLs on which it is configured according to the mode of the simulator (if connected to the PCM or in debug for functional testing when not on the test bench).

- **HTTP Server Address**

  The address to which CoreLogic makes http requests to know when PCM is available for displaying web pages in the Vehicle tab. It was useful to make this parameter also modifiable in order to use the simulator in different work conditions.

- **Error Timeout**

  This parameter manages the threshold for unsuccesful http requests beyond which the Error Page is shown in the Vehicle tab. Requires the

entry of a numeric value that corresponds to the time limit in seconds. By default, it is set to 240 seconds (per NIS system specification).

- **Vehicle Type**
  Through this setting there is the possibility to change the vehicle of the simulated infotainment system. The configurations of the Iveco Light and Heavy range have been added; by changing this setting the graphic resources of the selected vehicle are applied. Furthermore, it allows to modify the operating CoreLogic applied to the model, guaranteeing the possible use of a new logic implementation in the simulator.

- **Debug Mode**
  This switch allows you to select the working mode of the simulator. When deactivated, it configures all parameters in order to integrate the prototype with the PCM control unit. If the Debug mode is activated, the prototype creates a background process which configures a local web-server. The http resources provided by the web-server allow to emulate the behavior of the NIS-PCM software integration only within the personal computer.

- **Automated Testing**
  This button initiates an automated testing session, an innovative functionality developed within the software prototype. It will be discussed in detail later.

Introducing the switch for usage mode selection was essential for the development of the software prototype. In fact, due to the global pandemic taking place during the development of this thesis, I have not always had the opportunity to access the company headquarters.

The idea of implementing this alternative use of the simulator, therefore, arises precisely from the need to carry on the activity even though I could not carry out

unit testing of the prototype with the PCM control unit at the test bench. This implementation was very effective.

To run the simulator in Debug mode, the application logic starts a background process that creates a local web-server on the IP address of the personal computer. The web-server is activated using command line parameters sent through “Plink”: this small tool is the companion command-line utility for the popular “PuTTY” SSH client. It allows you to create SSH process logins for the execution of automated Linux commands from Windows. The created web-server contains all the html resources useful to recreate the NIS homepage. This is shown in the Vehicle tab by automatically setting the correct parameters, i.e. URL of the local web-server and IP address of the personal computer, in the Settings panel.

Finally, the operations that the application performs at each start have been developed to exactly reproduce the behavior of the NIS system. Basically, each time the application is closed, it saves the applied configuration in memory. At the next start, the application keeps all the settings used in the last run of the simulator and places the user interface on the last tab displayed. In addition, the last selected usage mode (PCM or Debug) is activated.

4.4 The integration with the PCM

The development of the NIS simulator required several design phases to arrive at a functional prototype. The main objective designated at the start of the activity was to propose an implementation of a software prototype that emulated the behavior of the NIS infotainment system. Once this is achieved, develop an innovative functionality that leverages the prototype to simplify the verification and validation of the PCM electronic control unit.

To achieve this last objective, however, the fundamental requirement was to faithfully reproduce every characteristic of the NIS infotainment system: its general behavior, each window displayed, the activation times of the control

graphic elements and the logical solutions implemented. Everything in the simulator had to be implemented the same way.

Only after achieving this fidelity in the simulator it was possible to proceed with the creation of the automated testing functionality.

4.4.1 Test bench configuration

The internship activity included two days a week for the meetings at the company headquarters with my tutor and the Telematics engineer assigned to me to support the development of the prototype, in accordance with the restrictions in force due to the pandemic situation. These days were essential to carry out the prototype checks on a weekly basis, as the functionalities of the application were added and/or modified.

For this purpose, an electronic test bench was set up at the company headquarters to work with. Figure 4.12 shows a photo of the installation.

This electronic bench reproduces the entire dashboard of Iveco commercial vehicles including: at the top the Instrument Cluster, at the center the NIS infotainment system and at the bottom various interconnected control units, including the PCM telematics control unit. With this instrumentation it was possible to verify each time the new functionalities developed in the software prototype.

The integration with my personal computer took place by connecting the PCM control unit via an Ethernet-over-USB connector.

Its usefulness has been seen above all after having concluded the initial design phase of the software prototype. That is when it was necessary to carry out calibration tests with the real NIS system, to reproduce as closely as possible its behavior in the simulator and to test the software integration with the PCM control unit.

Figure 4.12: The electronic bench used for the calibration tests.

4.4.2 The calibration tests

As previously mentioned, various technical interventions and revisions were required to achieve the main objective of the software prototype. The NIS system has to display a lot of contents and information since it is designed to show a particular window to the user at any time.

To exactly reproduce its behaviour, therefore, I have analysed in detail the NIS system when it is connected to the PCM control unit.

The PCM is designed with a Linux-based software architecture; this allows to easily send commands via SSH client. The following examples are some of the available commands:

mctl stop ExComGW ;

mctl start ExComGW ;

mctl restart VehicleApp .

The first calibration test was done to verify that the thresholds for the activation of the graphic control elements in the simulator (i.e. Homepage, Circular Spinner and Error Page) matched the real ones of the NIS.

Therefore, I have analysed the behaviour of the NIS when the ExComGW, the module responsible for the availability of the network together with the web-server, is deactivated. The functioning of the application logic with the status parameter “ping” is based entirely on the availability or not of this module.

This first test showed full correspondence with what happens inside the simulator in reference to the activation times of the graphic control elements. However, an inconsistency was found when starting the application: the Circular Spinner did not respect the activations sent by logic. The problem was therefore resolved, which resided in a variable that did not correctly trace the operating status of the NIS in the core logic.

During the tests for the calibration of the simulator other issues emerged that differentiated the behavior of the prototype from that of the real NIS system. These were mainly graphical glitches (i.e. white flash when switching UI elements, small delay when loading UI elements) which required better optimization of the model for the management of the graphical interface.

It took several attempts to understand the nature of these issues. Eventually, it turned out that the cause of the glitches was the implemented web browser. This component, which makes interactive pages available in the Vehicle tab, has seen several changes to be optimized to the fullest.

Microsoft's Edge technology-based WebView browser, initially implemented to display homepage content, has proven to be limited in some functionality. For the sole function of loading web pages this component worked properly, which is why it took some time to understand that this component had to be replaced with another solution.

Thanks to constant support from the designated team, I learned of an alternative browser that is fully compatible with the .NET development platform. CefSharp is a web browser based on Chromium technology, much more efficient than WebView. With its implementation all the issues related to graphical glitches have been solved.

This browser has also allowed the design of an important functionality to conclude the development of the NIS simulator: navigation within the menus and submenus of the pages shown in the VehicleApp.

4.4.3 Emulation of navigation with touch gestures

When using the NIS infotainment system in commercial vehicles, the interaction with it takes place via touch gestures on the display. The user can navigate within the menus and submenus available in the PCM software framework using this type of inputs.

In the development of the simulator we wanted to reproduce this functionality to achieve the complete emulation of the real NIS system. After having correctly implemented the visualization of the pages, it was necessary to make accessible every resource (information, applications, etc.) sent by the PCM.

The software architecture of the PCM was developed thinking about its use via a touch device. The NIS, in fact, through its touch display sends the selections on the view made by the user with the finger. Just like a smartphone.

The simulator, on the other hand, having been developed for use on a personal computer, requires inputs via mouse clicks. The goal of this implementation, therefore, was to devise a functional conversion method for mouse inputs. Were

it not for the particular functioning of the NIS-PCM software integration, this functionality would not have required special efforts to be implemented.

As has already been said several times previously, all that is shown on the NIS system are web pages processed by the PCM telematics control unit and then made available to the user through an integrated browser. This means that as all components of the UI, being part of the prototype model, were accessible from the code behind easily, this could not be applied to the contents shown in the browser.

The window displayed in the VehicleApp is basically an external rendering of html resources shown through a client, the browser. Practically the view of the contents takes place on the NIS but the current resources (the information of each element, including the interaction area for selecting it) reside on the PCM.

The solution devised involves the use of the web browser introduced earlier, CefSharp. This implementation is an easy way to embed a full-featured standards-compliant web browser into a C# and .NET application. [17] CefSharp has several browser controls for applications developed with WPF including the OffScreen version, a headless browser specifically designed for automation projects. The developer tools provided by this browser, based on Chromium technology, make it extremely versatile for various applications.

The development of the navigation functionality through touch gestures within the prototype took place as follows:

a. **CefSharp Browser implementation in the Vehicle panel**
   This control replaced the previous integrated browser, Microsoft's WebView. Two instances of the browser are required for the touch functionality: standard CefSharp for WPF platform, which displays web pages, and an OffScreen version which runs automated processes in the background.

b. Browser initialization with custom settings

To enable interaction with touch gestures, the two instances of the browser were initialized with UserAgent set on mobile device and the Remote Debug function set to a specific IP address. With the latter a debug session is created on the http address of the homepage to be displayed, enabling functions available only to developers on the html source code.

```
public MainWindow()
{
     InitializeComponent();
     AppWindow = this;

     CefSharp.Wpf.CefSettings settings = new CefSharp.Wpf.CefSettings()
     {
         CachePath =
Path.Combine(Environment.GetFolderPath(Environment.SpecialFolder.LocalApp
licationData), "CefSharp\\Cache"),
         UserAgent = "Mozilla/5.0 (Linux; U; Android 4.4.2; en-us; SCH-
I535 Build/KOT49H) AppleWebKit/534.30 (KHTML, like Gecko) Version/4.0
Mobile Safari/534.30",
         RemoteDebuggingPort = 8088
     };
     Cef.Initialize(settings);

     InitializeBrowser();
}
```

c. HTTP request to the VehicleApp homepage

Through the main CefSharp browser the web pages arriving from the PCM must be shown, while the secondary browser performs the conversion of the inputs in real time. The process begins with the http request to the PCM of the VehicleApp homepage, to be made available on the primary browser. When the main frame of the primary browser is loaded, the Remote Debugging session is requested.

```
private void Browser_FrameLoadEnd(object sender, FrameLoadEndEventArgs e)
{
    if (execOn == "PCM")
    {
        Dispatcher.BeginInvoke((Action)(() =>
        {
            browser.SetZoomLevel(-2.58);
            debugBrowser.Load("localhost:8088");
        }));
    }
}
```

d. Html source code retrieval via OffScreen browser

Via the OffScreen browser, not visible on the screen, automated procedures are carried out in the background. Here, taking advantage of the CefSharp developer features, a Remote Debugging session is created. All the addresses available in the two instances of the browser are then displayed, including the homepage of the VehicleApp previously requested. The cybersecurity protocol requires that each address available for debugging at each session has a different identification fragment in the URL. The entire html source code of the page is then retrieved and saved in a string.

```
private void DebugBrowser_FrameLoadEnd(object sender,
FrameLoadEndEventArgs e)
{
    Dispatcher.BeginInvoke((Action)(async () =>
    {
        string html = await debugBrowser.GetSourceAsync();
        TextReader tr = new StringReader(html);
        DoHtmlParse(tr);
    }));
}
```

e. Creation of the debug URL through html parsing

Once the source code has been obtained, its content is analyzed through an additional library specific for the manipulation of html code. Therefore, the property containing the identifying fragment of the URL of the Remote Debugging instance is extracted from the source code. Finally, through the following method the string containing the specific URL for enabling the remote debugging session on the secondary browser is constructed. For the correct implementation of this part of the code, the importance of using the “try-catch” technique should be noted. An exception occurred punctually during the html parsing of the page in the secondary browser, blocking the execution. Handling the exception with “try-catch” allowed to avoid the block. The Remote Debugging session even without requesting a new URL remained active, thus requiring a single initialization.

```
public void DoHtmlParse(TextReader html)
{
    HtmlDocument htmlDoc = new HtmlDocument();
    htmlDoc.Load(html);
    try
    {
        string result =
htmlDoc.DocumentNode.SelectSingleNode("//body/div[2]/p/a").GetAttributeVa
lue("href", "");
        StringBuilder sb = new StringBuilder("localhost:8088");
        string debugUrl = sb.Append(result).ToString();
                new CefSharp.OffScreen.ChromiumWebBrowser(debugUrl);

                debugBrowserActive = true;
    }
    catch (Exception)
    {
        if (debugBrowserActive == false)
        {
            Debug.WriteLine("No remote session.\n");
        }
    }
    SetPage(false, true, false);
}
```

f. Homepage display with navigation through touch gestures enabled

VehicleApp is then shown on the primary browser. Navigation via touch is enabled throughout the use of the application, thanks to the secondary browser that works constantly in the background. The whole process is done only once, at the first http request to the PCM and it happens instantly.

4.5 Test automation using the software prototype

The idea of designing a PCM control unit testing and validation system within the software prototype arose from the need to make these operations simpler. As seen throughout this experience in Iveco, in fact, the development of new functions for the two integrated systems promptly required multiple technical interventions.

Much of the complexity came from the fact that the software integration case involves two completely different electronic components and that, above all, they are manufactured by two different companies. This resulted in the extension

of intervention times: they had to wait for the feedback from both parties for each technical revision and the validation of the components.

With the implementation of a software prototype capable of evaluating specific areas and/or parameters in their software integration, an effective alternative solution is proposed.

4.5.1 Development of the functionality

For the development of this functionality, the requirements to be met were first defined. Through the use of scripts, that are text files containing the instructions of the test to be performed, the prototype would have to read and "translate" them in order to perform actions on the PCM and/or on the NIS simulator. This, in order to evaluate certain factors useful for component validation, such as: correct initialization of the application logic following software restarts, execution times of functionalities, availability of applications and user interface components.

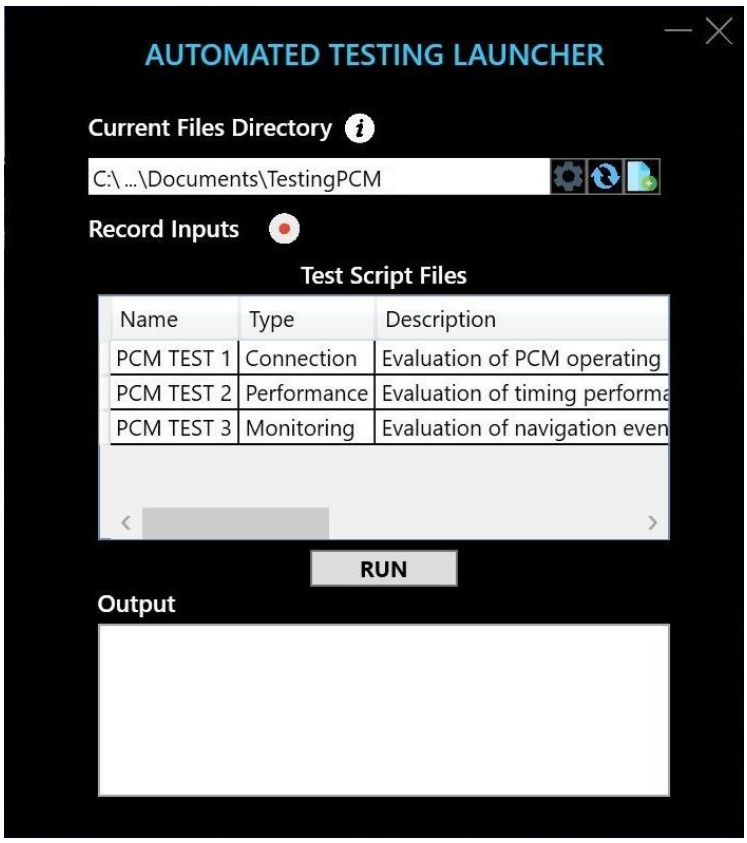


Figure 4.13: Automated testing launcher.

The automated tests are implemented in an additional panel accessible from the simulator Settings tab, visible in Figure 4.13 above.

From this interface the directory for the files useful for the testing session is automatically created. If there are scripts used previously, the application automatically sets the specific directory when the session is started.

Using the buttons displayed on the right of "Current Files Directory" it is possible to: set a new directory on the PC from which load the scripts, refresh the contents of the directory and add a new script to the current directory.

All the scripts present in the selected directory are displayed in the Data Grid below, providing the following information:

- **Name**
  The name of the file loaded from the set directory, that is, the script with the instructions for the test case.

- **Type**
  Indicates the type of test case, giving a quick indication of the type of validation carried out.

- **Description**
  Brief description of the test case indicating the specific functionality or component analysed by the test.

By selecting the header of each column the scripts can be sorted based on the column property. In this way it is possible to see all the available test cases relating, for example, to a specific Type.

This information is taken directly from the content of each scripts which is compiled according to a template created specifically for the tests. The scripts are text files in JSON format, a standard used in the company for various activities in the Telematics sector. A JSON file is a text file that stores simple data structures and objects in the JavaScript Object Notation (JSON) format,

which is a standard data exchange format. Based on plain text, it is easy to understand for both humans and computers. Inside each script, in addition to the information described above, there are the instructions for performing the test. Some examples of scripts with JSON files are shown below.

1. ***Script to verify the correct reconnection of the PCM following a restart of ExComGW***

```
   {
      "tstInfos": [
  {
    "name": "JSON_TEST_V1",
    "type": "Connection",
    "description": "Server restart"
  }
],
"actions": [
        {
    "id": "StopExComGW",
    "cmdMsg": "mctl stop ExComGW",
    "check": 16
  },
  {
    "id": "StartExComGW",
    "cmdMsg": "mctl start ExComGW",
    "check": 8
  }
]
   }
```

2. ***Script for testing the operation of the VehicleApp application following a restart of the PCM***

```
{
   "tstInfos": [
      {
  "name": "JSON_TEST_V2",
  "type": "Application",
  "description": "System reboot"
      }
   ],
   "actions": [
      {
  "id": "SetVehicleTab",
  "setMsg": "vehicleapp"
      },
      {
  "id": "RebootPCM",
```

```
    "cmdMsg": "reboot"
        },
        {
    "id": "SpinnerActive",
    "check": 16
        },
        {
    "id": "PingSuccess",
    "check": 8
        }
     ]
}
```

Each instruction can contain specific actions to be performed based on what is needed to automate in the test case: command lines to be sent to the PCM for the activation/deactivation or the restart of its software architecture's modules, automatic inputs to be performed on the user interface of the NIS simulator or set the check of specific events arriving from the PCM during the software integration with the NIS simulator.

In the automated testing launcher there is also a specific button to enable a new recording of click inputs: it has been implemented an advanced method that gets the user mouse click selection on the browser and retrieves the specific selected element http resources. This feature is useful to create an automated navigation inside the VehicleApp homepage and applications. Once selected the desired inputs path, the application gets the information of all the selected elements (such as the specific URL) and creates automatically the click actions template to put inside the JSON test script.

Lastly, the launcher provides a window to look for the outputs received during the test script runtime.

4.5.2 Processing the scripts

The scripts, using the JSON format, are easily configurable according to a data structure model built specifically to be read and processed by the implemented algorithm.

The functioning of automated tests has been implemented by writing a special class of methods in C# language called TestLogic. After selecting the test case script to execute from the launcher interface, the prototype elaborates its content and executes the test through the TestLogic methods. Taking into consideration the example script n.1 above, the automated test is performed as follows:

a. Initialization of the evaluation variables

Initially, the path of the selected file is saved in a string for later use. The variables useful for the execution of the test are then initialized: "resumeTest" changes its value in case of test failure, "nextResult" is reset and will contain the evaluation code for the PCM events, "startTime" marks the test start time in milliseconds.

```
string filePath = Settings.Default.filePath;
resumeTest = true;
nextResult = 0;
startTime = DateTimeOffset.Now.ToUnixTimeMilliseconds();
```

b. Reading and saving the instructions contained in the JSON file

The contents of the script are retrieved from the path specified above. A special library performs the deserialization of the JSON file in order to create data structures that separate the information on the type of test from the instructions with the actions to be performed. For the actions to be performed on the PCM, the algorithm creates a specific string. This string contains the extracted commands to be sent via the SSH client, the check codes to verify that the behaviour is as expected and the automated click parameters (such as the element coordinates in the browser, its specific URL and, eventually, a configurable timeout for the automated click execution).

```
public void DoJsonConversion()
{
    string fileName = Settings.Default.filePath;
    if (fileName != "null")
    {
        string jsonString = File.ReadAllText(fileName);
        Script scriptData =
JsonSerializer.Deserialize<Script>(jsonString);
        StringBuilder sb = new StringBuilder();
```

```
            foreach (var action in scriptData.actions)
            {
                string setMsg = action.setMsg;
                string cmdMsg = action.cmdMsg;
                string check = action.check.ToString();
                string sleep = action.sleep.ToString();
                string click = action.click;
                string url = action.url;
                string timeout = action.timeout.ToString();

                if (setMsg != null)
                {
                    sb.AppendLine("SET>" + setMsg);
                }
                if (cmdMsg != null)
                {
                    sb.AppendLine("CMD>" + cmdMsg);
                }
                if (check != "0")
                {
                    sb.AppendLine("CHECK>" + check);
                }
                if (sleep != "0")
                {
                    sb.AppendLine("SLEEP>" + sleep);
                }
                if (click != null)
                {
                    sb.AppendLine("CLICK>" + click);
                }
                if (timeout != "0")
                {
                    sb.AppendLine("TIMEOUT>" + timeout);
                }
                if (url != null)
                {
                    sb.AppendLine("CHECK>url>" + url);
                }
            }
            foreach (var tstInfo in scriptData.tstInfos)
            {
                tName = tstInfo.name;
                tType = tstInfo.type;
                tDescription = tstInfo.description;
            }
            cmdString = sb.ToString();
            OnStringBuild.Set();
        }
    }
```

c. Execution of the extracted actions

The instructions extracted from the JSON script are then executed based on the type of action to be performed. Different types can be performed: actions that interact with the prototype user interface to simulate touch inputs, set pauses between one action and another, change the http

address for the homepage request, send command lines to manage specific functions of the PCM or its operational status. For the commands to be sent to the PCM the following method is used, which creates an SSH process through the “Plink” client, integrated in the prototype. Here the commands for the PCM are read from the specific string created earlier and executed in the order entered.

```
public async Task EstablishConnectionToPCM(string strCmdText)
{
    string username = "root";
    string host = "192.168.249.1";
    string password = "root";

    ProcessStartInfo psi = new ProcessStartInfo()
    {
        FileName = Directory.GetCurrentDirectory() + $@"\plink.exe",
        Arguments = String.Format("-ssh {0}@{1} -pw {2}", username, host,
password),
        RedirectStandardError = true,
        RedirectStandardOutput = true,
        RedirectStandardInput = true,
        UseShellExecute = false,
        CreateNoWindow = true
    };

    Process p = Process.Start(psi);

    StreamWriter strw = p.StandardInput;

    strw.WriteLine(strCmdText);
    strw.WriteLine("exit");

    p.WaitForExit();

    if (p.HasExited)
    {
        p.Close();
        p.Dispose();
    }
}
```

d. Receiving the response and checking the PCM behaviour

After launching the commands on the PCM, the algorithm intercepts its behaviour for the required checks. To do this, it first registers for receiving the CoreLogic events and then monitors specific codes sent along with the activation of the events.

```
public void RegisterToCoreLogic(bool active)
{
    if (active)
```

```
        {
            CoreLogic.PingSuccess += OnReceivingEvent;
            CoreLogic.SpinnerLaunch += OnReceivingEvent;
            CoreLogic.ErrorTimeout += OnReceivingEvent;
        }
        else
        {
            CoreLogic.PingSuccess -= OnReceivingEvent;
            CoreLogic.SpinnerLaunch -= OnReceivingEvent;
            CoreLogic.ErrorTimeout -= OnReceivingEvent;
        }
    }
```

In the evaluation variable "nextResult" it is saved the expected code after the execution of the command on the PCM and this is then compared with the current code received by the CoreLogic events.

```
    public void OnReceivingEvent(object sender, CoreEventArgs e)
    {
        if (nextResult != 0)
        {
            if (e.Code == CoreEventArgs.SuccessCode)
            {
                currentEventMsg = EventMsg.Success;
            }
            if (e.Code == CoreEventArgs.SpinnerCode)
            {
                currentEventMsg = EventMsg.Spinner;
            }
            if (e.Code == CoreEventArgs.TimeoutCode)
            {
                currentEventMsg = EventMsg.Timeout;
            }

            if (e.Code == nextResult)
            {
                currentResultMsg = ResultMsg.OK;
                resumeTest = true;
            }
            else
            {
                currentResultMsg = ResultMsg.NOOK;
                resumeTest = false;
            }
            SendLogMsg(e.Code);
            OnEventCheck.Set();
        }
    }
```

The algorithm performs this process iteratively at each command line to be executed, verifying the behaviour obtained from time to time. If the expected codes are found for each action performed, the test is considered

passed. If, on the other hand, after the execution of a command the expected code is not found the test is stopped and is considered failed.

e. Execution of automated clicks navigation

To perform the automated navigation inside the VehicleApp homepage and the applications, the software prototype exploits a combination of monitoring tasks. First, the primary browser sends http resources at each change of the navigation address, providing the necessary information to do the check at each automated click on an element. Then, a method executes the click operation in loop over the coordinates extracted from the scripts (that have been previously recorded by means of the specific function available in the launcher). The click loop operation ends once one of these two requirements have been met: the specific clicked element's URL is received in the browser (meaning a navigation success), or the timeout interval set for the click action is elapsed (meaning that the navigation is not found).

```
private void Browser_AddressChanged(object sender,
DependencyPropertyChangedEventArgs e)
{
    if (ReceivingHttpStatusCode != null)
    {
        ReceivingHttpStatusCode(this, new HttpEventArgs() { HttpUrl =
e.NewValue.ToString() });
    }
    currentClickUrl = e.NewValue.ToString();
}
```

```
public void OnReceivingClickEvent(object sender, HttpEventArgs e)
{
    if (e.HttpUrl == nextUrl)
    {
        MainWindow.AppWindow.ReceivingHttpStatusCode -=
OnReceivingClickEvent;
        urlMatch = true;
        double timestamp = DateTimeOffset.Now.ToUnixTimeMilliseconds();
        logs.Add($"\nCLICK: X={currentClickX}, Y={currentClickY}");
        logs.Add($"URL: {nextUrl}");
        logs.Add($"EXECUTED AT: {(timestamp - startTime) / 1000} sec.");
        logs.Add("RESULT: Navigation Success.");
    }
}
```

```
public void DoClickLoop()
{
    if (currentClickTimeout != 0)
    {
        double start = DateTimeOffset.Now.ToUnixTimeMilliseconds();
        while (urlMatch == false)
        {
            MainWindow.AppWindow.MouseClick(currentClickX,
currentClickY);

            double current = DateTimeOffset.Now.ToUnixTimeMilliseconds();
            if ((current - start) > currentClickTimeout)
            {
                logs.Add($"\nCLICK: X={currentClickX},
Y={currentClickY}");
                logs.Add($"URL: {nextUrl}");
                logs.Add($"TIMEOUT AT: {(current - startTime) / 1000}
sec.");
                logs.Add("RESULT: Not Found.");
                logs.Add($"     ERROR (timeout {currentClickTimeout /
1000} sec.)     ");
                break;
            }
            Task.WaitAll(new Task[] { Task.Delay(1000) });
        }
        MainWindow.AppWindow.CheckAddress(false);
        OnClickCheck.Set();
    }
    else
    {
        while (urlMatch == false)
        {
            MainWindow.AppWindow.MouseClick(currentClickX,
currentClickY);
            Task.WaitAll(new Task[] { Task.Delay(1000) });
        }
        MainWindow.AppWindow.CheckAddress(false);
        OnClickCheck.Set();
    }
}
```

f. Final report creation

At the end of the test, a text file is released from the prototype containing all the logs collected during the test and the final result. All checks made during execution are included.

The final report also includes the test information (name, type, description) as well as the total execution time. These reports are finally saved in a specific folder in the directory selected previously.

```
public void CreateOutputFile()
{
Directory.CreateDirectory(Path.Combine(Settings.Default.myTestsPath,
"Logs"));

    string date = DateTime.Now.ToString("d");
    string time = DateTime.Now.ToString("T");
    string[] lines = logs.ToArray();
    double duration = (endTime - startTime) / 1000;

    string resultPath = Settings.Default.myTestsPath + $@"\Logs\";
    string resultFileName = String.Join("_", "PCM_Test",
DateTime.Now.ToString("dd-M-yy"), DateTime.Now.ToString("HH.mm.ss"));
    resultFileName = String.Concat(resultFileName, ".txt");
    string docPath = Path.Combine(resultPath, resultFileName);

    using (StreamWriter outputFile = new StreamWriter(docPath))
    {
        outputFile.WriteLine("DATE: " + ${date}" + " " + ${time}");
        outputFile.WriteLine("ID: " + tName);
        outputFile.WriteLine("TYPE: " + tType);
        outputFile.WriteLine("DESCRIPTION: " + tDescription);
        outputFile.WriteLine();
        outputFile.WriteLine("Total execution time: " + duration + "
(sec.)");
        outputFile.WriteLine();
        outputFile.WriteLine("TEST LOGS:");

        foreach (string line in lines)
        {
            outputFile.WriteLine(line);
        }
    }
    Process.Start("notepad.exe", docPath);

    logs.Clear();
}
```

4.5.3 Types of tests implemented

The automated testing functionality has been developed so that various types of tests can be performed by properly configuring the instructions file.

The types implemented in the prototype are three and each of them allows to verify a specific functionality or feature available in the software integration between the NIS simulator and the PCM electronic control unit. The implementation of the JSON standard for the script document structure, makes the testing configuration completely configurable to ensure different validation

targets. The types of tests implemented are the following. For each, it is provided an example of script configuration:

- **Automatic reconnection test**

  This test involves a series of command lines launched on the PCM control unit in order to evaluate its functionality following reconnections. It is possible to set the restart of a specific module of the control unit software architecture or the complete restart of the unit.

  The automatic reconnection test can be used primarily to verify that everything is working correctly following, for example, a sudden reboot. Or following changes to the framework to validate the connection functionality of the PCM.

```
{
   "tstInfos": [
     {
       "name": "PCM TEST 1",
       "type": "Connection",
       "description": "Evaluation of PCM operating conditions after a
disconnection"
     }
   ],
   "actions": [
     {
       "id": "ShowVehicle",
       "setMsg": "vehicleapp"
     },
     {
       "id": "StopExComGW",
       "cmdMsg": "mctl stop ExComGW",
       "check": 16,
       "sleep": 10
     },
     {
       "id": "StartExComGW",
       "cmdMsg": "mctl start ExComGW",
       "check": 8,
       "sleep": 5
     },
     {
       "id": "Click1",
       "click": "468,280",
       "timeout": 5,
       "url":
       "http://192.168.249.1:4040/vehicleapp/pages/index.html#/ras/",
       "sleep": 0
     }
   ]
}
```

- **Performance test**

  With this type of test it is possible to evaluate the general performance of the PCM control unit. Through the use of timestamps recorded at the beginning and at the end of the actions to be performed, it is possible to evaluate the execution time taken. This test can be combined with other types of tests to evaluate, for example, the time it takes to make a specific page available in the interface. Additionally, the test is useful for validating the PCM boot time in software integration with the NIS infotainment system.

```
{
   "tstInfos": [
     {
       "name": "PCM TEST 2",
       "type": "Performance",
       "description": "Evaluation of timing performance on RAS
application availability"
     }
   ],
   "actions": [
     {
       "id": "ShowVehicle",
       "setMsg": "vehicleapp"
     },
     {
       "id": "RebootPCM",
       "cmdMsg": "reboot",
       "sleep": 20
     },
     {
       "id": "Click1",
       "click": "468,280",
       "timeout": 0,
       "url":
       "http://192.168.249.1:4040/vehicleapp/pages/index.html#/ras/",
       "sleep": 0
     }
   ]
}
```

- **Long-running test**

  This type of test exploits the implementation of the automated touch inputs functionality on the simulator user interface. Before the test, the coordinates of the individual UI elements are recorded on the interface, selecting them in the desired order. The test then performs the same sequence of touch inputs, automating the process.

The test is useful for long-run sessions. That is, setting a certain time window in which to repeat this sequence repeatedly, in order to trace any component malfunctions. Specifically, it is possible to evaluate the reliability of the electronic control unit in processing each application of the VehicleApp, simulating a continuous sequence of requests from the user.


```
{
   "tstInfos": [
     {
       "name": "PCM TEST 3",
       "type": "Monitoring",
       "description": "Evaluation of navigation events"
     }
   ],
   "actions": [
     {
       "id": "ShowVehicle",
       "setMsg": "vehicleapp"
     },
     {
       "id": "Click1",
       "click": "170,272",
       "timeout": 5,
       "url":
       "http://192.168.249.1:4040/vehicleapp/pages/index.html#/dse/",
       "sleep": 3
     },
     {
       "id": "Click2",
       "click": "733,40",
       "timeout": 5,
       "url": "http://192.168.249.1:4040/vehicleapp/pages/index.html#/",
       "sleep": 3
     },
     {
       "id": "Click3",
       "click": "326,281",
       "timeout": 5,
       "url":
       "http://192.168.249.1:4040/vehicleapp/pages/index.html#/ans/",
       "sleep": 3
     },
     {
       "id": "Click4",
       "click": "735,40",
       "timeout": 5,
       "url": "http://192.168.249.1:4040/vehicleapp/pages/index.html#/",
       "sleep": 3
     },
     {
       "id": "Click5",
       "click": "470,280",
       "timeout": 5,
```

```
      "url":
      "http://192.168.249.1:4040/vehicleapp/pages/index.html#/ras/",
      "sleep": 3
    },
    {
      "id": "Click6",
      "click": "728,41",
      "timeout": 5,
      "url": "http://192.168.249.1:4040/vehicleapp/pages/index.html#/",
      "sleep": 3
    },
    {
      "id": "Click7",
      "click": "613,280",
      "timeout": 5,
      "url":
      "http://192.168.249.1:4040/vehicleapp/pages/index.html#/ras-
      unattended/",
      "sleep": 3
    },
    {
      "id": "Click8",
      "click": "728,40",
      "timeout": 5,
      "url": "http://192.168.249.1:4040/vehicleapp/pages/index.html#/",
      "sleep": 3
    },
    {
      "id": "Click9",
      "click": "760,47",
      "timeout": 5,
      "url":
      "http://192.168.249.1:4040/vehicleapp/pages/index.html#/vehicleapp
      /info/",
      "sleep": 3
    },
    {
      "id": "Click10",
      "click": "721,42",
      "timeout": 5,
      "url": "http://192.168.249.1:4040/vehicleapp/pages/index.html#/",
      "sleep": 3
    }
  ]
}
```

Chapter 5

Conclusions

The technology currently implemented in vehicles sees a multitude of sophisticated systems which, through the adoption of embedded software architectures, perform advanced functions: the Electronic Control Units (ECUs) and the set of interconnected devices. Today's vehicle, in fact, is mostly composed of electronic components and advanced software systems.

The initial idea for carrying out this thesis was to study, analyse and verify the reliability of the software used in the electronic devices of commercial vehicles. This point of view was the backbone of all the activities carried out at Iveco S.p.A. which, despite the particular emergency period and the closure of the workplaces due to the pandemic situation, saw the effective completion of an entire project and the achievement of the objectives set. The thesis, therefore, describes this experience, reporting entirely the three phases of work with which the activity was structured. For each of these phases, the working methods learned and the approach used in the development of innovative functionalities for the validation of electronic components in vehicles are highlighted.

The first approach was to study the full range of commercial vehicles and their electronic architecture. In this phase it was particularly useful to deepen the discussion by analysing the functional design schemes of the electronic systems and the tools used for the diagnostic analysis of vehicles. Thus, in this way, the necessary tools for understanding the software issues of these systems integrated in vehicles were learned. The next step was the in-depth research in the company's internal databases of Open Points (OP) relating to the software engineering field, obtaining the first real product useful for the thesis: the classification of software malfunctions in the various electronic components. From this first analysis it was highlighted how the complexity in making ECUs more and more reliable lies entirely in the embedded software. Most of the reported anomalies are inherent to issues related to software rather than

hardware, highlighting how the reliability of the hardware of vehicle electronic systems has already reached a certain maturity. To reach this conclusion, it was necessary to investigate the entire electronic architecture of the Stralis heavy commercial vehicle, highlighting in particular, how effective the implementation of the Multiplex system and the CAN lines was on this vehicle.

From the study carried out in this preliminary thesis phase, a case of software integration between two very different vehicular electronic components emerged: the thesis activity then continued with the analytical and research phase inherent to this exemplary case. The case of software integration between the NIS infotainment system and the PCM electronic control unit was of fundamental importance for the development of innovative telematics functions on Iveco vehicles. This exemplary case required many technical interventions to improve the software integration between the two electronic systems, therefore, research was carried out on the methodologies adopted in the past. In this way, a complete view of the actions that proved effective in the technical resolution of the issues encountered was obtained. The subsequent classification by causes and final effects of the points found, served to identify the areas of software development that were most affected by anomalies, in order to understand the nature of the error. Furthermore, with this product useful for carrying out the thesis project phase, the substantial complexities with which they had to operate in the Telematics sector during the resolution of these issues were highlighted: in particular the relevant difference between the cause and the final effect observed by the user on board the vehicle, concluding that it is not always feasible to trace the factor or part of the code affected by the anomaly based solely on the reports received. Specific tests are required in specific phases of software development and, in most cases, it is necessary to loop the testing operations on the code several times.

Finally, in the thesis the implementation of a software prototype for the evaluation of the reliability of the embedded software in the PCM electronic control unit was proposed. This ECU processes and provides all the telematics applications and functions on the NIS infotainment system; their complex software integration made it possible to conceive the activity carried out in the

project phase of the thesis. With the creation of a simulator of the NIS system, compatible with any personal computer, it was therefore possible to create an easily configurable and accessible testing and validation environment. The opportunities provided by this tool are many: first of all the versatility of navigating the contents of the PCM electronic control unit from the computer, without the need to specifically create an electronic test bench in which the NIS system is physically present. Continuing, thanks to the automated testing functionality, it has been seen how it is possible to simplify the operations for the validation of the two electronic components, through the integration of logical methods that perform targeted checks within the software architecture of the PCM electronic control unit.

Through the implementation of the MVVM architectural pattern for the software development of the prototype, the use of this solution as a substitute for the current operations for the validation of these components is further motivated. In fact, having appropriately separated the logic of each functionality from the model in the view, it is possible to add and configure new methods to automate in tests with ease. It will also be possible, should it be necessary later, to make changes only to the part of the code concerned in order to adapt the simulator to a core application logic available in a new version of the electronic control unit. Or, the reusability of the code could be exploited to extend the functional concept to further electronic components. The scenarios of use for the software prototype might be numerous, also considering the advantage that would be obtained by the reduction of the costs used for the verification and validation phase of the electronic components.

For the actual conclusion of the experimental project, a demonstration session was organized in the company which was attended by the company tutor and the members of the Telematics team. In this meeting I was able to present the main features and advantages in the implementation of the software prototype. The correct integration and execution in real time was verified using the equipment available in the electronic test bench at the company to, then, launch the automated tests on the PCM electronic control unit and review the outcome.

LIST OF FIGURES

LIST OF TABLES

BIBLIOGRAPHY

[1] Lipson Charles, Sheth Narendra J., Sheldon David B., “Reliability and maintainability in industry and the universities”, 1996

[2] Abrahamsson P., Salo O., Ronkainen L., Warsta J., “Agile Software Development Methods: Review and Analysis”, 2017 – arXiv: 1709.08439

[3] Sorge C., “Metodo Agile e Scrum: i vantaggi di applicarli all’interno dell’azienda”, 2016

[4] Hines J.F., “Forecast: Connected Car Production, Worldwide”, Gartner, 2016

[5] Charette Robert N., “How Software is eating the car”, IEEE Spectrum, 2021

[6] Iveco Group, CAN Line Evolution

[7] CNH Industrial, “Stralis MY2019 Electronic Functional Diagrams”, Internal documentation, 2018

[8] Iveco Group, Reliability Engineering Methodologies & Practical Approaches

[9] Junger Markus, “Introduction to J1939”, Vector, 2010

[10] CNH Industrial, “PCM Items Definition”, Internal documentation

[11] CNH Industrial, “NIS-PCM Core Functional Specification”, Internal documentation, 2019

[12] DevOps Methodology, QRP - url: https://www.qrpinternational.it/corsi/metodologia-devops/

[13] The C# Programming Language, BairesDev - url: https://www.bairesdev.com/technologies/csharp/

[14] Welcome to the Visual Studio IDE, Microsoft, 2022 - url: https://docs.microsoft.com/en-us/visualstudio/get-started/visual-studio-ide

[15] Digiteum Team, “Why Use .NET Platform: Advantages for Your Product Development”, 2021

[16] Wikipedia, “Model-View-ViewModel” - url: https://en.wikipedia.org/wiki/Model-view-viewmodel

[17] CefSharp Browser for .NET applications - url: https://cefsharp.github.io/