Training Report

DET PARTICULARS	
Employee ID	**927456**
Employee Name	**Meet Dave**
GET batch	**ER&D/GET**
GET Track	**ER&D/GEN/JUL 23**

Table of Content

MODULE 1: Domain Training

1.1 Introduction to the Automobile:

- Chassis: Introduction, Exhaust, Fuel system
- Chassis Axle.
- Steering Wheels
- Suspension, Brakes & latest Trends in Chassis
- Seating, Safety systems and Automotive materials
- Body Engineering (Interiors, Exteriors)
- Automotive Regulations, Virtual Validation and Testing
- Powertrain

1.2 Introduction to Electric Vehicle.

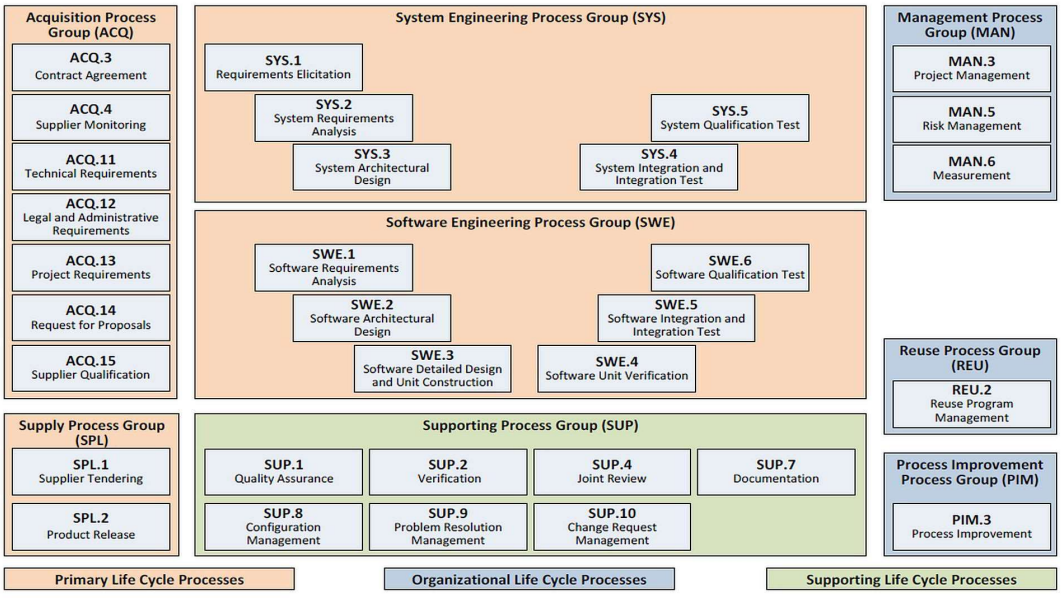
- Motor, Motor Controller
- Battery Pack, BMS, Charging System
- Regenerative Braking, Selection of EV components.
- Latest Trends in EV

MODULE 2: ASPICE

2.1 Basic Overview

- Automotive SPICE **(ASPICE)** is a framework for improving software development processes in the automotive industry, focusing on safety, quality, and efficiency.
- The V-Model is a parallel software development and testing methodology that matches each development phase with a corresponding testing phase.
- It is a software quality assessment and improvement tool for the automotive industry.
- Together, ASPICE and the V-Model play a crucial role in enhancing software quality, safety, and compliance in the automotive industry.

2.2 Automotive SPICE process reference mode:



In this model we basically focus on Automotive SPICE process reference mode which is categories between Three main parts

- Primary Life Cycle Processes
- Organizational Life Cycle Processes

- Supporting Life Cycle Process

2.3 Process capability levels and process attributes:

- **Capability levels:** A capability level is a set of process attribute(s) that work together to provide a major enhancement in the capability to perform a process. There are five capability levels (level 0 – level 5) and have their own process attributes.
- **Process Attributes (PA):** Process attributes are features of a process that can be evaluated on a scale of achievement, providing a measure of the capability of the process.

Attribute ID	Process Attributes
Level 0: Incomplete process	
Level 1: Performed process	
PA 1.1	*Process performance process attribute*
Level 2: Managed process	
PA 2.1	*Performance management process attribute*
PA 2.2	*Work product management process attribute*
Level 3: Established process	
PA 3.1	*Process definition process attribute*
PA 3.2	*Process deployment process attribute*
Level 4: Predictable process	
PA 4.1	*Quantitative analysis process attribute*
PA 4.2	*Quantitative control process attribute*
Level 5: Innovating process	
PA 5.1	*Process innovation process attribute*
PA 5.2	*Process innovation implementation process attribute*

2.3 Process attribute rating:

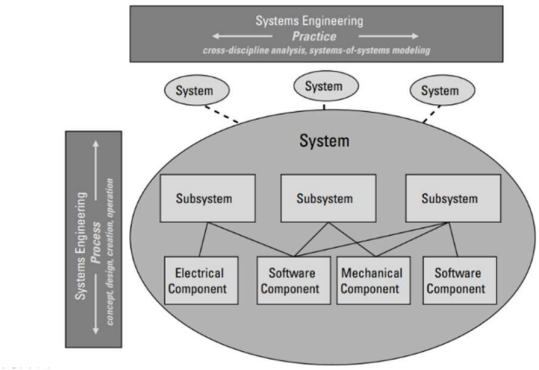
A process attribute rating is a judgement of the degree of achievement of the process attribute for the assessed process.

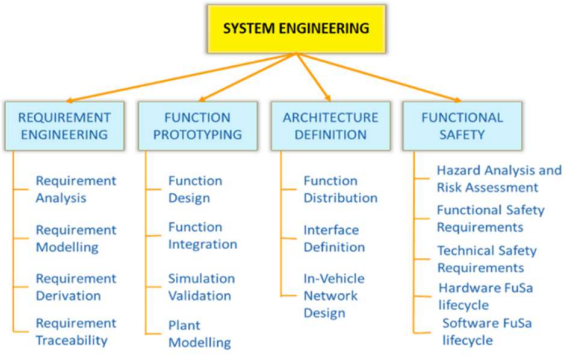
- N -Not achieved 0 to ≤ 15% achievement
- P -Partially achieved >5% to ≤ 50% achievement
- L -Largely achieved >50% to ≤ 85% achievement
- F Fully achieved > 85% to ≤ 100% achievement

MODULE 3: REQUIREMENT GATHERING

3.1 System Engineering:

System is a group of interacting elements (or subsystems) having an internal structure which links them into a unified whole. System Engineering is an Interdisciplinary approach/Process to creating large, complex systems that meet a defined set of business and technical requirements.





In system engineering basically focus on two System requirement:

- **Sys.1 (Requirement Elicitation)**
- **Sys.2 (System Requirement Elicitation)**

3.2 Software Engineering

In software engineering we focus on software requirement:

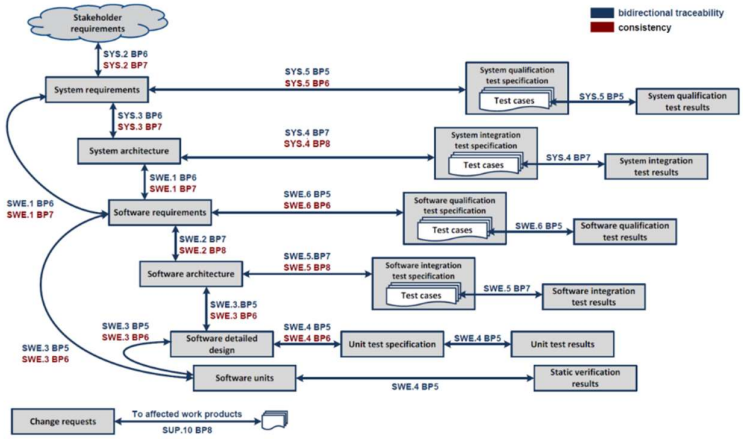
- **SWE.1 (Software requirement Analysis)**

3.3 Rules

- Avoid ambiguity.
- Avoid making multiple requirements.
- Avoid building let-out clauses.
- Avoid designing the system.
- Identify the type of user who needs the requirement (Object).

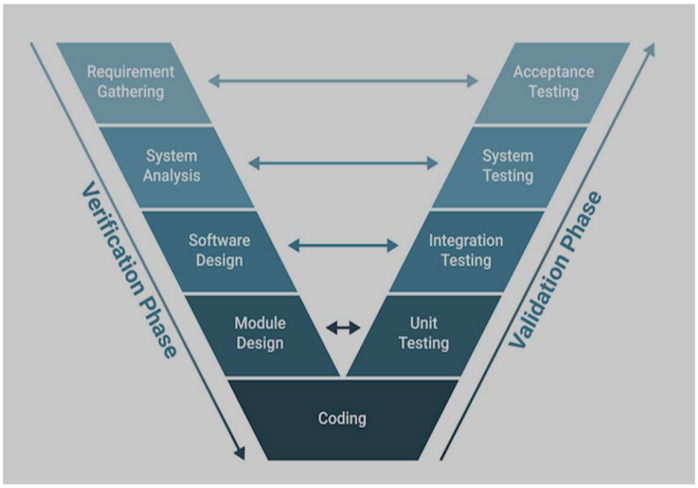
3.4 Traceability

Ensuring that each requirement is traceable to corresponding validation and verification activities on the right side of the "V," guaranteeing thorough testing and alignment with stakeholder needs.



3.5 Validation and Verification:

Conducting validation activities to confirm that the software meets stakeholder intentions and verification activities to ensure adherence to documented requirements.



4.1 Overview:

1. **Overview of Automotive Trends:** An exploration of current and emerging trends shaping the automotive industry, from electrification and connectivity to autonomous driving and sustainability.

2. **Overview of Embedded Electronics:** An examination of embedded electronic systems in vehicles, highlighting their role in controlling various functions and enhancing vehicle performance.

3. **Automotive Electronics and its Requirements:** An insight into the specific demands and standards governing electronics in automobiles, focusing on safety, reliability, and functionality.

4. **Typical Development Cycle for the Electronic Control Unit (ECU):** A glimpse into the stages involved in designing, developing, and testing Electronic Control Units (ECUs) that govern vehicle operations.

5. **Basic Electric Vehicle Information and Trend:** An introduction to electric vehicles (EVs), including their key components, environmental benefits, and their growing presence in the automotive market.

6. **Application with Examples:** Real-world applications and examples illustrating how embedded electronics and automotive trends come together in innovations like advanced driver assistance systems (ADAS) or infotainment systems.

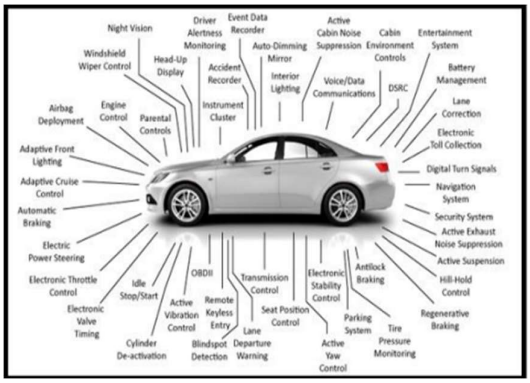


Fig 4.1 High End Passenger Vehicle: Sensors Image

Fig 4.2 High End Commercial Vehicle Image

4.2 EV Classification:

- Electric Vehicle
- Hybrid Vehicle
  - Series-Parallel Hybrid
  - Series Hybrid
  - Parallel Hybrid

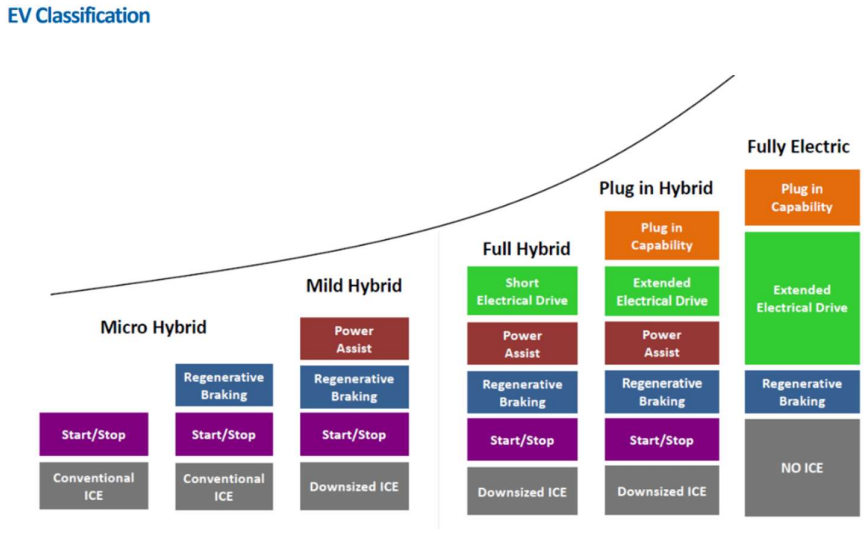


Fig 4.3: EV-Classification

5.1 Stages:

- **Product Strategy and Planning Phase:** The initial stage of defining product goals, market research, and strategic planning for a new product.
- **Concept Evolution Phase:** The process of refining and evolving initial product concepts based on market feedback and feasibility assessments.
- **Concept Development Phase:** Transforming refined concepts into detailed product specifications and design plans.
- **Product Design and Development Phase:** The actual creation and engineering of the product, encompassing design, prototyping, and development activities.
- **Design Validation and Product ionization Phase:** Ensuring that the product design meets quality and performance standards and preparing for mass production.
- **Pre-production Phase:** Preparing the production line, tools, and processes for mass manufacturing, addressing any remaining issues before full-scale production.
- **Ramp-up Phase:** Gradually increasing production volume to reach full-scale manufacturing capacity, ensuring quality control and efficiency.

5.2 Gateway

- **Product Strategy Review:** Evaluating and revising the overall product strategy to align with market dynamics and company objectives.
- **Concept Selection:** Choosing the most promising product concept from a pool of options based on criteria like feasibility, market demand, and profitability.
- **Concept and Project Approval:** Gaining organizational consensus and approval for the chosen product concept and the initiation of the associated project.
- **Project Release:** Launching the project, mobilizing resources, and commencing the execution phase to bring the product to market.

6.1 Embedded System:

An embedded system is a combination of hardware and software designed to perform a dedicated function or set of functions within a larger system, often with real-time constraints.

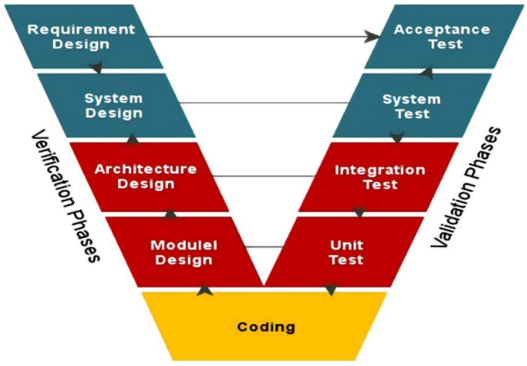
Examples:

- Consumer electronics (e.g., smartphones, smart TVs, digital cameras).
- Automotive systems (e.g., engine control units, infotainment systems, ABS).
- Home appliances (e.g., washing machines, microwave ovens, thermostats).
- Industrial machines and automation (e.g., PLCs, robotics).
- Medical devices (e.g., pacemakers, infusion pumps, medical imaging equipment).

6.2 V – Model:

A software development and testing methodology that emphasizes the relationship between development phases and their corresponding testing phases.
It has main two Phases:

1) Verification Phase
2) Validation Phase



6.3 Introduction of Bare Metal Programming:

The practice of programming hardware directly, without an operating system, commonly used in embedded systems development.

6.4 Types of Testing - Alpha, Beta:

Alpha testing involves internal testing of a software product, while beta testing involves external testing with a select group of users to gather feedback and identify issues before a wider release.

- Application of embedded system

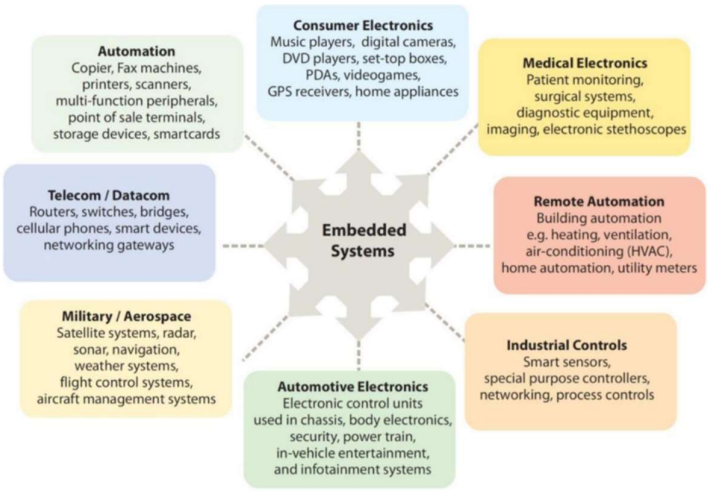
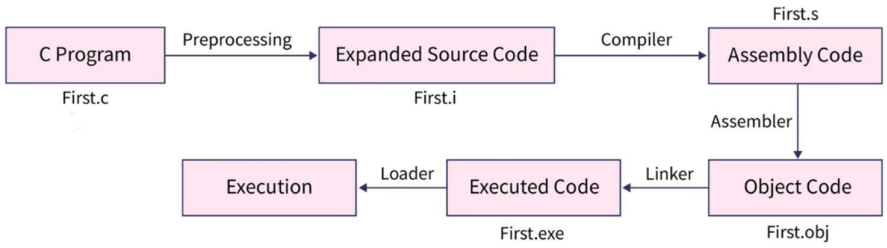


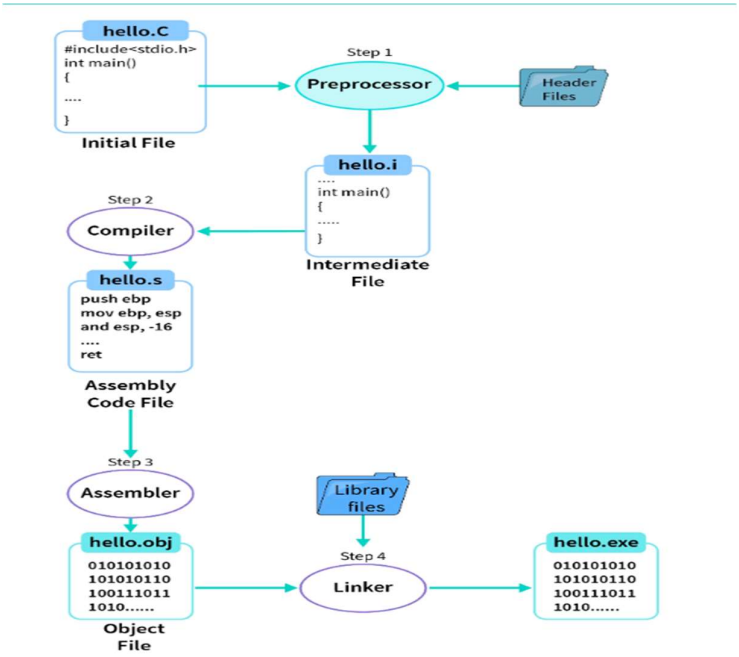
Fig 6.1: Application of embedded system

6.5 Execution Process



6.6 Execution Model For C-Program:

The compilation processes of a program refer to the sequence of steps and processes that occur when program is compiled and run. It involves several stages, from writing the source code to executing the compiled program.



MODULE 7: Embedded C

7.1 Embedded C

7.2 STM32 Board Project:

Code:

```c
/* Includes ------------------------------------------------------------------
#include "main.h"
#include <String.h>
#include "stm32f4xx_hal.h"
#include "stdio.h"
```

Here, various header files are included. These files contain definitions and functions required to interact with the STM32 microcontroller and its peripherals.

"***main.h***": The project-specific header file that likely contains some macro definitions, function prototypes, or other project-specific settings.

<***String.h***>: Standard C library header for string manipulation functions.

"***stm32f4xx_hal.h***": The main HAL header file for STM32F4 microcontrollers. It includes all necessary HAL libraries for the specified microcontroller.

"***stdio.h***": Standard I/O library header, which provides functions for formatted input and output (like ***sprintf*** and ***printf***).

```c
/* Private variables ---------------------------------------------------------
ADC_HandleTypeDef hadc1;
DMA_HandleTypeDef hdma_adc1;
I2C_HandleTypeDef hi2c1;
TIM_HandleTypeDef htim3;
UART_HandleTypeDef huart1;
UART_HandleTypeDef huart2;
```

Here, the private variables are defined. These variables are instances of various HAL structures used to interact with the different peripherals:

hadc1: ADC handle structure for ADC1.

hdma_adc1: DMA handle structure for ADC1.

hi2c1: I2C handle structure for I2C1.

htim3: Timer handle structure for TIM3.

huart1: UART handle structure for USART1.

huart2: UART handle structure for USART2.

```c
/* USER CODE BEGIN PV */
uint16_t AdcRaw[2];
uint8_t AdcConvCmplt = 0;
double Temperature;
char msg[20];
/* USER CODE END PV */
```

In the "Private variables" section, user-specific variables are declared:

AdcRaw[2]: An array to hold ADC raw data from two channels.

AdcConvCmplt: A flag to indicate whether ADC conversion is complete (initialized to 0).

Temperature: A variable to store the calculated temperature value.

msg[20]: A character array (string) to store the formatted message for UART transmission.

The ***'main'*** function is the entry point of the program, where the actual execution of the code begins.

```c
int main(void)
{
    /* MCU Configuration--------------------------------------------------------
    HAL_Init();
    SystemClock_Config();
    MX_GPIO_Init();
    MX_DMA_Init();
    MX_ADC1_Init();
    MX_TIM3_Init();
    MX_USART1_UART_Init();
    MX_USART2_UART_Init();
    MX_I2C1_Init();
```

In this part of the code, I am initializing various system components and peripherals of the microcontroller:

HAL_Init(): Initializes the HAL (Hardware Abstraction Layer) library. This function prepares the microcontroller for HAL operation, sets up the system tick timer, and performs other initialization tasks.

SystemClock_Config(): Configures the system clock. This function is usually automatically generated by STM32CubeMX or provided by the manufacturer. It sets up the clock sources and frequency of the microcontroller.

MX_GPIO_Init(), MX_DMA_Init(), MX_ADC1_Init(), MX_TIM3_Init(), MX_USART1_UART_Init(), MX_USART2_UART_Init(), MX_I2C1_Init(): These functions initialize various peripherals like GPIO pins, DMA (Direct Memory Access), ADC (Analog-to-Digital Converter), timers, and UART (Universal Asynchronous Receiver-Transmitter) interfaces. These functions are often generated by STM32CubeMX based on your project configuration.

```c
while (1)
{
    if (AdcConvCmplt)
    {
        /* Temperature calculation and UART transmission */
        AdcConvCmplt = 0;
        sprintf(msg, sizeof(msg), "%If\r\n", Temperature);
        HAL_UART_Transmit(&huart2, (uint8_t*)msg, strlen(msg), HAL_MAX_DELAY);
        HAL_Delay(1000);
    }
}
}
```

Inside the loop, the code checks whether the ***AdcConvCmplt*** flag is set. If it's set, it means that an ADC conversion has been completed, and the code proceeds with temperature calculation and UART transmission.

AdcConvCmplt flag is checked, and if it's true (non-zero), it means that the ADC conversion has completed.

The flag is then reset (***AdcConvCmplt*** = 0;) to prepare for the next ADC conversion.

The ***sprintf*** function is used to format the temperature value into the msg character array.

HAL_UART_Transmit is called to transmit the msg (temperature value) over UART (huart2). It sends the message as a sequence of bytes.

strlen(msg) provides the length of the message in bytes, and ***HAL_MAX_DELAY*** indicates that the function will wait indefinitely until the transmission is complete.

Finally, there's a delay of 1000 milliseconds (1 second) introduced using ***HAL_Delay(1000).***

The loop continues to run indefinitely, periodically checking for completed ADC conversions and transmitting temperature data over UART.

```c
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc)
{
    if (hadc->Instance == ADC1)
    {
        AdcConvCmplt = 255;
    }
}
```

This is a callback function that is automatically executed by the HAL library when an ADC conversion is complete. It checks if the completed conversion is from ADC1 and sets the ***AdcConvCmplt*** flag to 255.

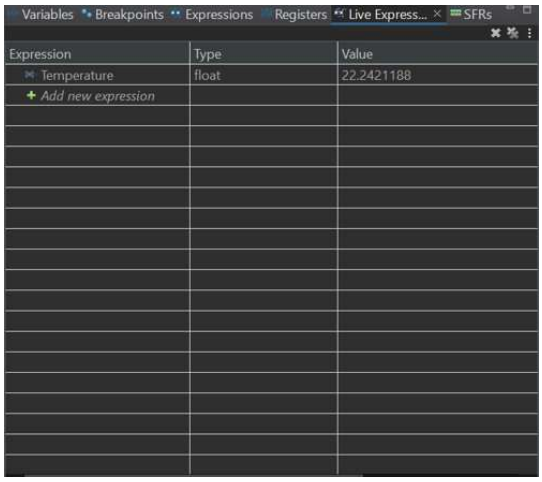
```c
void Error_Handler(void)
{
    __disable_irq();
    while (1) { }
}
```

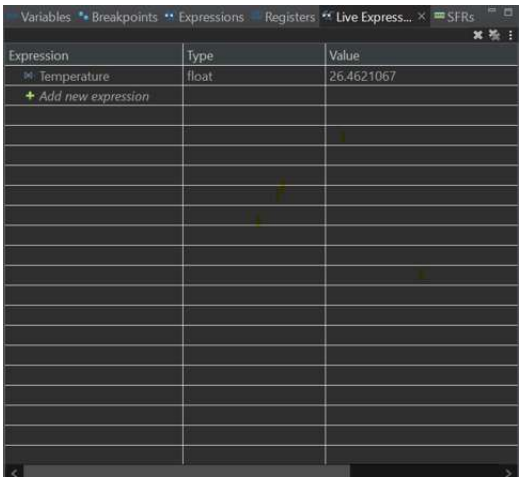
This function is called in case of an error. It disables interrupts and enters an infinite loop, effectively halting the program.

```c
#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t *file, uint32_t line)
{
    /* User-defined error reporting, if needed */
}
#endif /* USE_FULL_ASSERT */
```

This is an assertion error handler. If assertion checking is enabled (***USE_FULL_ASSERT*** is defined), this function is called when an assertion fails. Developers can add their own error reporting or debugging code here.

Output:



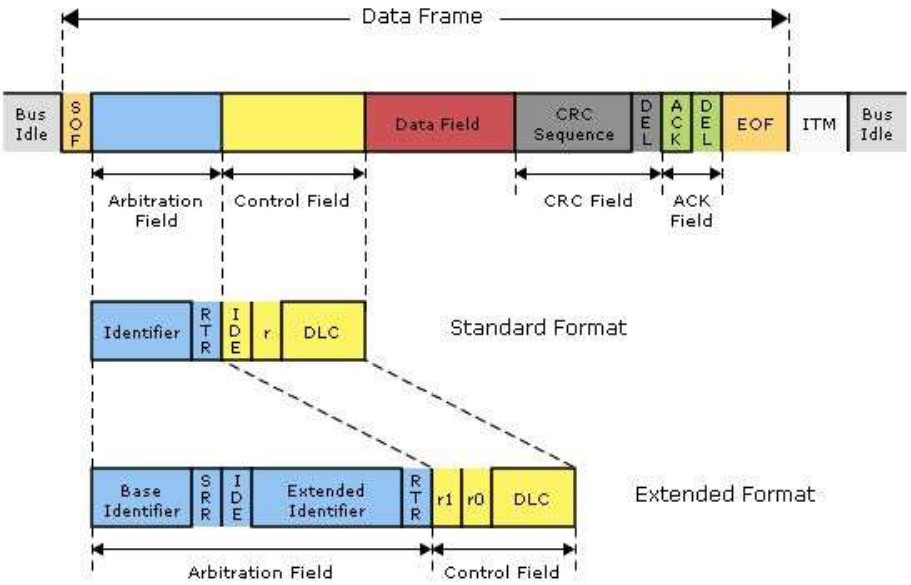


8.1 Introduction to CAN Protocol

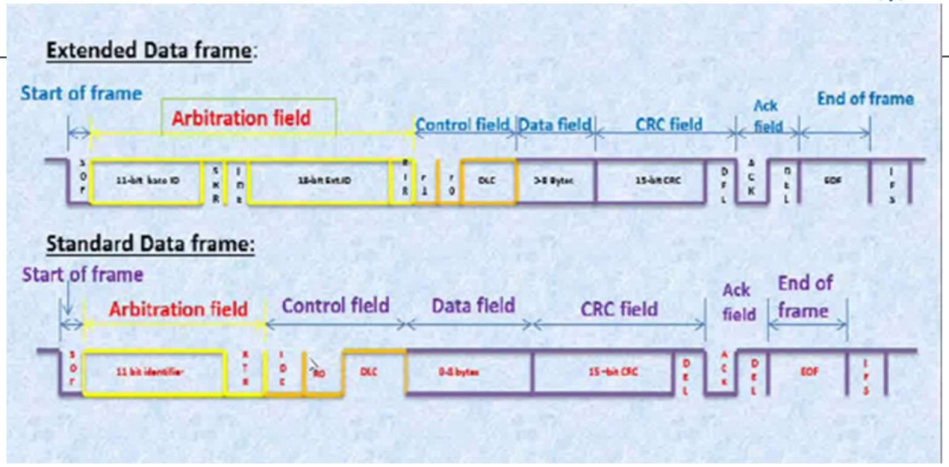
CAN stands for **Controller Area Network** protocol. CAN protocol designed to target the communication issue. The CAN bus is a broadcast type of bus. This means that all nodes can 'hear' all transmissions. There is no way to send a message to just a specific node; all nodes will invariably pick up all traffic. The CAN hardware, however, provides local filtering so that each node may react only to interesting messages.

8.2 Message Type

- The Data Frame
- The Remote Frame
- The Error Frame
- The Overload Frame

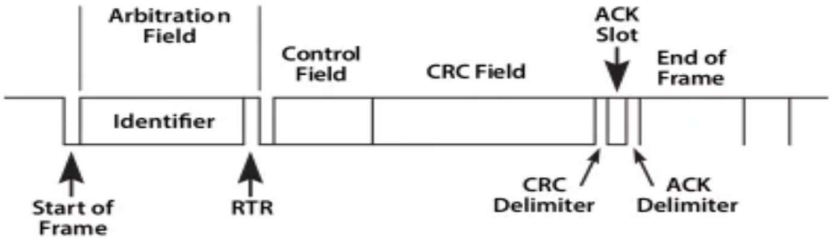


- **Data Frame**



CAN: Standard-Extended Frame Format

- **Remote Frame**



A Remote Frame (2.0A type)

- **Error Frame:** Error Frame is a special message that violates the framing rules of a CAN message, special message transmits when a node detects a fault and will cause all other nodes to detect a fault.

- **Overload Frame:** Overload Frame here just for completeness. It is remarkably like the Error Frame regarding the format, and it is transmitted by a node that becomes too busy.

1. Activity 1

In Activity One, we selected a car model of our choice, a specific example: the 2023 BMW 530i, a model from the BMW 5 Series sedan lineup where Number of ECUs (Electronic Control Unit) present in selected car were 20.

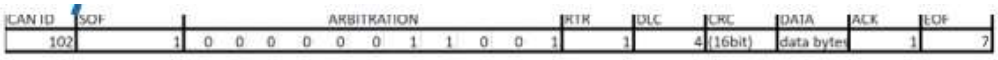
2. Activity 2

In Second activity, we select the one ECU from list which is ABS ECU uses information from **wheel speed sensor**s to detect potential wheel lock-up during braking. It then intervenes by modulating brake pressure to prevent lock-up and maintain steering control. This technology significantly improves braking performance and stability, particularly in adverse road conditions or emergency situations.

	7 bits	6 bits	5 bits	4 bits	3 bits	2 bits	1 bit	0 bit
0 byte	ws	ws	ws	ws	ws	ws	ws	ws
1 byte								
2 bytes								
3 bytes								
4 bytes								
5 bytes								
6 bytes								
7 bytes								

3. Activity

In this activity, we create the Data Frames.



Data Frames

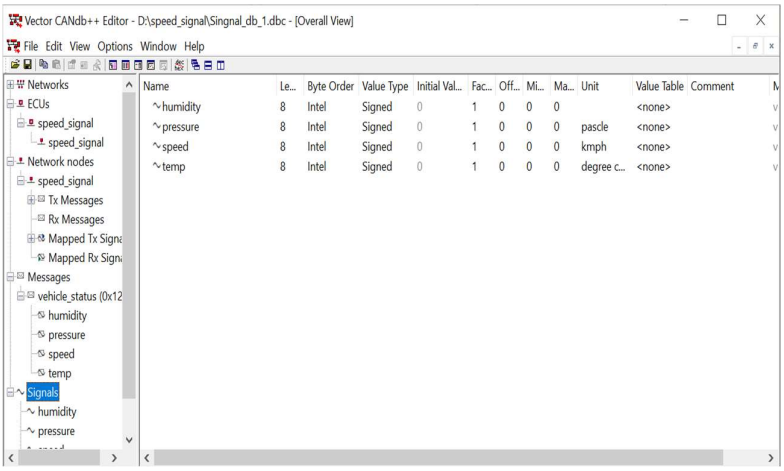
9.1 Introduction to Vector CANoe 17

Vector CANoe 17 is a widely used software tool for developing, testing, and simulating CAN (Controller Area Network) communication in automotive and embedded systems.

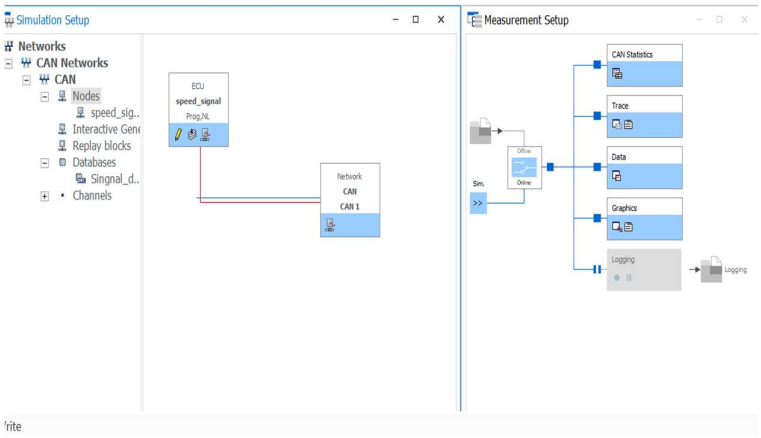
9.2 Introduction to CAPL Scripting

CAPL (Communication Access Programming Language) scripting is a feature within CANoe that enables users to create custom scripts for automating and enhancing testing and simulation processes in CAN networks.

- **Step 1:** Creating the Database.



- **Step2:** Synchronize the database with the simulation.



9.3 Activity

To create a dashboard featuring a speed gauge where the car's speedometer initially transitions from 0 to 100 and then back to 0 within the first 2 seconds. Afterward, the speed will increase based on user input, with increments of 10 every second with sender and receiver CAPL Script.

```
/*@!Encoding:65001*/

includes
{

}

variables
{
  message 0x126 speed;
  msTimer tim1;
  //counter Timer
  msTimer tim2;
  int secondsCounter=0;
  int a=0;
  float b=0;

}
on start{
  setTimer(tim1,5);
  //intitializing the Counter
  setTimerCyclic(tim2,1000);
}
```

```
on timer tim2{
  secondsCounter++; // Increment the seconds counter
  write("Elapsed time in seconds: %d", secondsCounter);
}

on timer tim1{
  speed.dlc=8;

  if(secondsCounter<=2){
    setTimer(tim1,1);
    speed.byte(0)=b;
    if(secondsCounter>=1){b-=0.1;}
    else{b+=0.1;}
  }

  else{
  setTimer(tim1,500);
  speed.byte(0)=a;
    if(speed.byte(0)>=70){
      if(speed.byte(0)==120){
        a=0;
      }
      else{
        a+=10;
        speed.byte(1)=1;
      }
    }
    else{
      speed.byte(1)=0;
      a+=10;
    }
  }
  output(speed);
}
```

Figure 1: CAPL script for sending data frames: Speed.

```
/*@!Encoding:65001*/
includes
{

}

variables
{
  message 0x456 msg;
  int a;
  int b;
}

on message 0x126{
  msg.dlc=8;
  a=this.byte(0);
  b=this.byte(1);
  msg.byte(0)=a;
  msg.byte(1)=b;
  @speed::speed=this.byte(0);
  @speed::Warning=this.byte(1);
  output(msg);
}
```

Figure 2: CAPL script for Receiving data frames and pushing it to the dashboard.

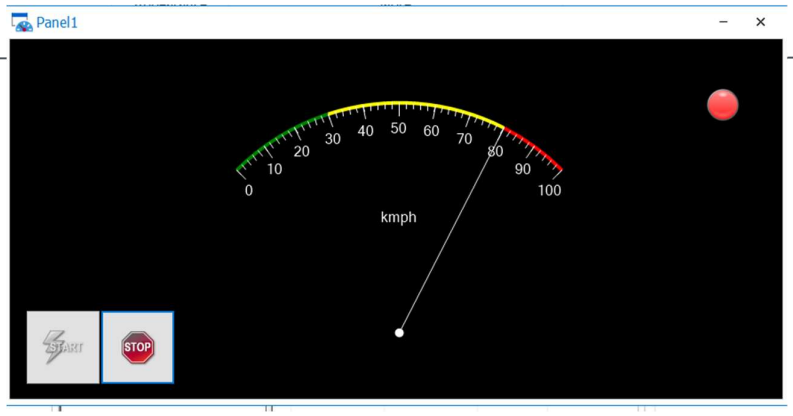


Figure 3: Dashboard for Displaying Speed

10.1 Overview

The Unified Diagnostic Services (UDS) protocol is an automotive standard used for diagnostic communication between a vehicle's Electronic Control Units (ECUs) and external diagnostic tools.

10.2 UDS services & their Sub Function

UDS provides a range of services, each with specific subfunctions, including reading and writing diagnostic data, controlling ECU functions, and managing communication.

10.3 Communication Layers

In automotive diagnostics, communication occurs through various layers, including physical, data link, and application layers, ensuring standardized and reliable communication between diagnostic tools and vehicle ECUs.

10.4 Diagnostic Trouble Codes (DTC)

DTCs are standardized codes generated by vehicle ECUs to indicate specific faults or malfunctions within the vehicle's systems, aiding in the diagnostic process.

10.5 Data Formats

Diagnostic data is typically structured using standardized formats like Diagnostic Data Records (DDRs) and Diagnostic Data Objects (DDOs) to ensure consistency and compatibility across different diagnostic tools and vehicle models.

10.6 Activity

Example:
Positive Response:
Client: 10.02
Server: 50.02 (SID+40 For Positive Response)
Negative response:
Client: 10.09 (09 sub-function does not exist)
Server:7F.28.12 (12 is NRC)
Default and Non-default:
0x28 is Not applicable for Default Session. So, we need to change its session which comes under the time limit.

Process to convert to non-default: -

Client: 10.02(function to convert SID to Non default)
Server:50.02(Positive response)
Client:28.02 (Now it is converted to Non default session)

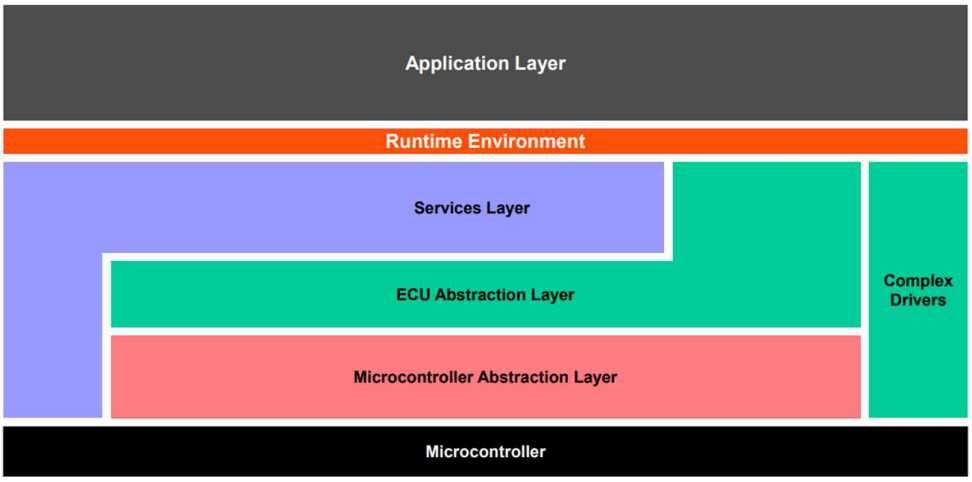
11.1 Autosar

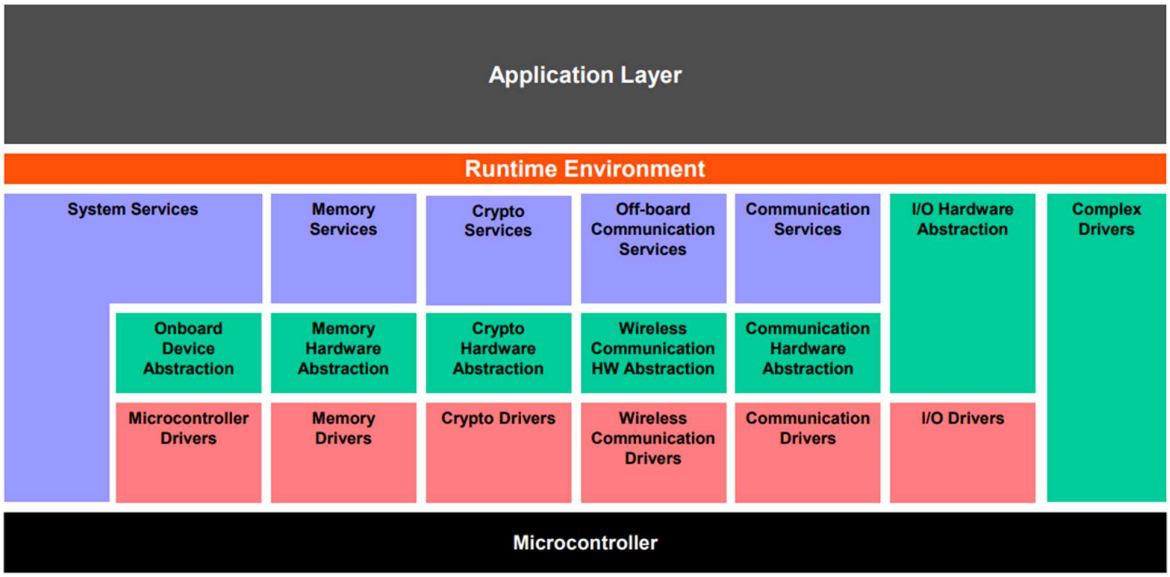
Classic AUTOSAR, or Automotive Open System Architecture, is a standardized software architecture and development methodology in the automotive industry. It aims to simplify the development, integration, and maintenance of embedded software components for electronic control units (ECUs) in vehicles.

11.2 Architecture

Classic AUTOSAR follows a layered software architecture that includes an application layer, runtime environment, basic software layer, and hardware abstraction layer (HAL). This architecture enhances modularity, reusability, and configurability in automotive software development. It also defines standardized interfaces for communication and allows for component-based development.

- **Application Layer:** This top layer contains the software components responsible for specific vehicle functions.
- **Runtime Environment (RTE):** It provides a communication layer between the application layer and the basic software layer, facilitating data exchange.
- **Basic Software Layer:** This layer offers services like communication, diagnostics, and operating system functions to the application layer.
- **Hardware Abstraction Layer (HAL):** The bottom layer interfaces with the hardware, providing abstraction and standardization for various ECUs and microcontrollers.





Module 12: Deep Dive in Classic Autosar

12.1 Application Software Development

- **VFB Virtual Functional Bus**

a) Create Software Component

b) Create Interface

c) Create Port

d) Mapping interface

e) Repeat above steps.

- **RTE Run Time Environment**

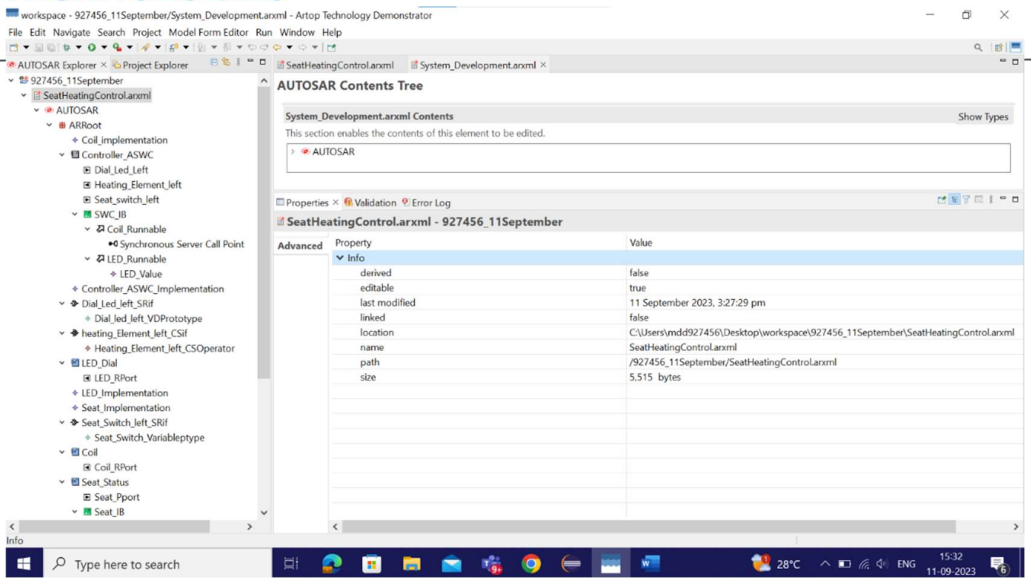
1) Create Runnable

2) Map internal behavior

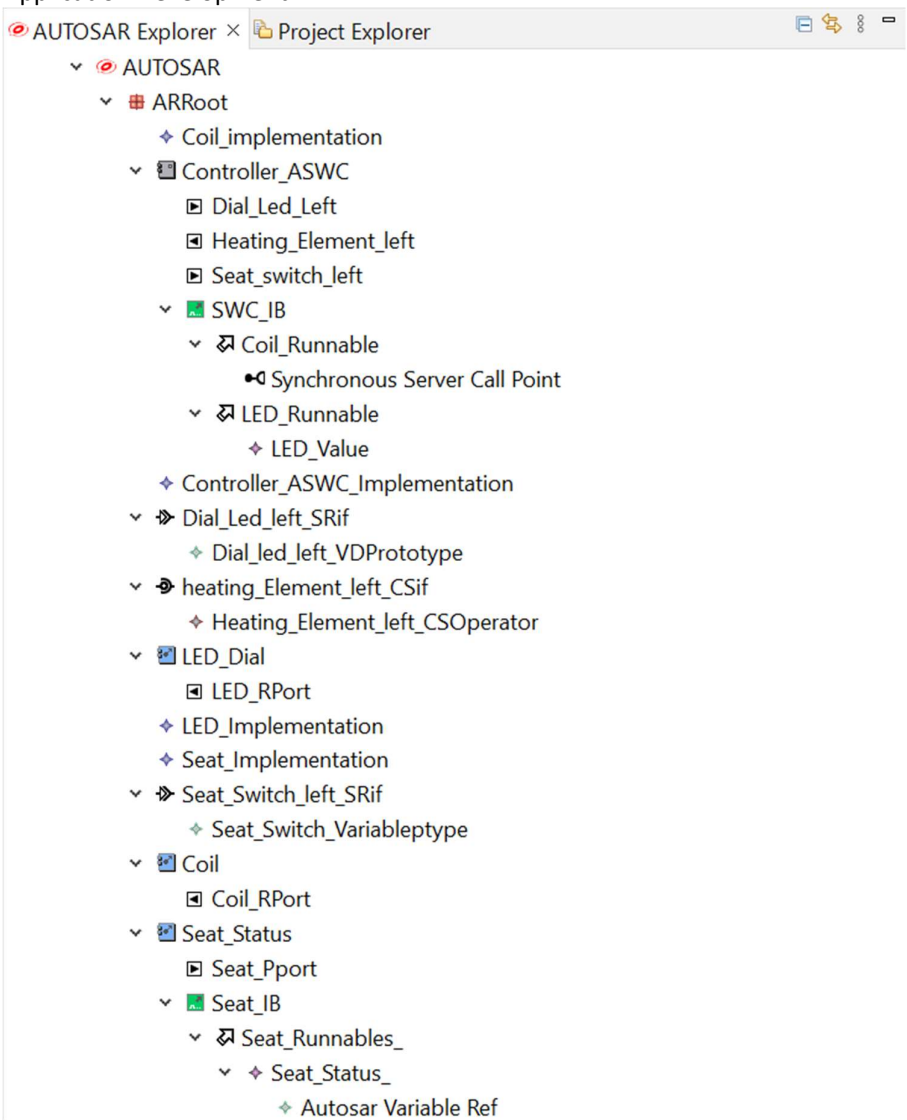
- **Implementation**

12.2 Activity

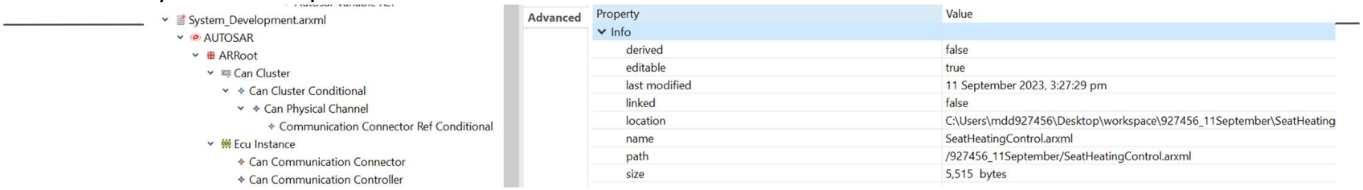
- Application software component development and system development with CAN Controller for seat heating system.
- The heating system is shown below so the configurations are done in artop tool.
- So it is done for the left hand side which includes the Led, Coil, Heating controller ASWC and the status switch.



Application Development



System Development:



14.1 Agile

Agile is an iterative and flexible approach to software development and project management that prioritizes collaboration, customer feedback, and rapid, incremental progress. It contrasts with traditional "waterfall" methodologies that rely on extensive upfront planning. Here's some information about Agile:

- **Values and Principles**: Agile is guided by the Agile Manifesto, which outlines four key values and twelve principles. The values prioritize individuals and interactions, working software, customer collaboration, and responding to change over processes and tools.

- **Iterative and Incremental:** Agile projects are divided into small, manageable increments known as iterations or sprints, each typically lasting two to four weeks. At the end of each iteration, a potentially shippable product increment is delivered.

- **Scrum and Kanban:** Two popular Agile frameworks are Scrum and Kanban. Scrum organizes work into fixed-length iterations with defined roles (Scrum Master, Product Owner, and Development Team). Kanban is more flow-based, allowing for continuous delivery and focusing on visualizing work and limiting work in progress.

- **Cross-Functional Teams:** Agile teams are typically cross-functional, meaning they include members with diverse skills (e.g., developers, testers, designers) who collaborate closely to deliver value.

- **Customer-Centric:** Agile places a strong emphasis on customer involvement and feedback throughout the development process. This ensures that the product aligns with customer needs and expectations.

- **Adaptability:** Agile embraces change and encourages teams to be flexible and responsive to evolving requirements. It acknowledges that customer needs and market conditions can change rapidly.

- **Continuous Improvement:** Agile teams regularly reflect on their processes and performance, aiming to improve efficiency, quality, and collaboration.

- **Transparency:** Agile promotes transparency by making work visible through tools like task boards and burndown charts, allowing team members and stakeholders to understand progress.

- **User Stories and Backlogs:** Requirements are often expressed as user stories, concise descriptions of a feature from an end-user's perspective. These stories are managed in a product backlog, a prioritized list of work items.

- **Testing and Quality Assurance:** Agile encourages continuous testing and quality assurance throughout development, not just at the end of a project.

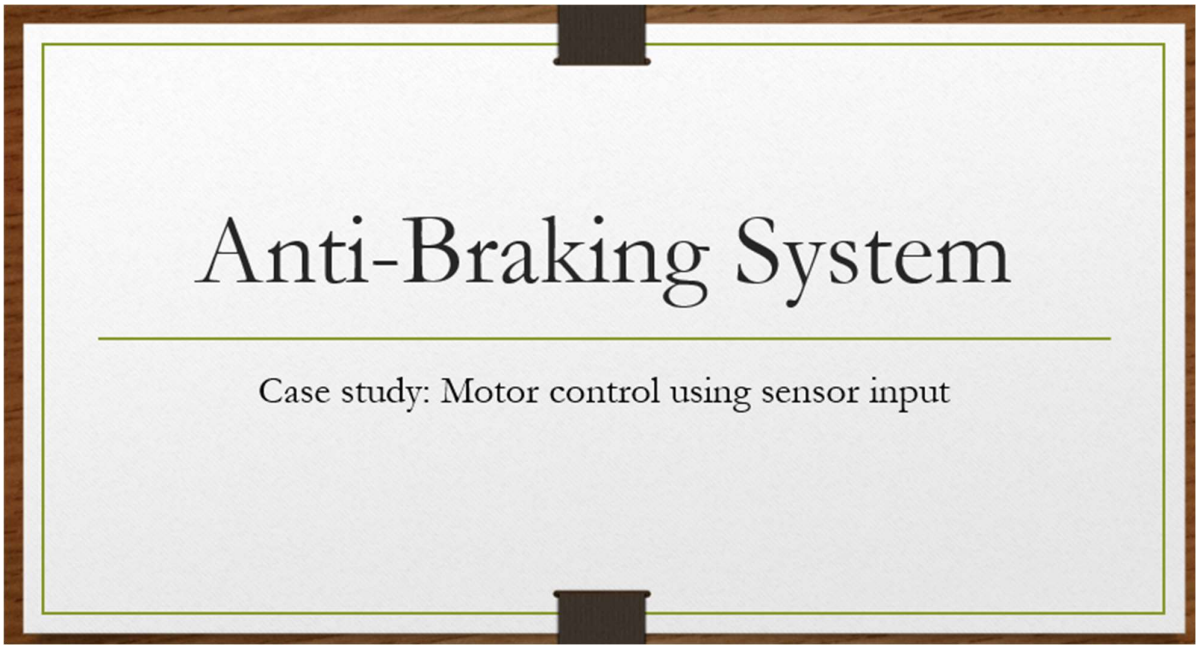
- **Frequent Deliveries:** Agile teams aim to deliver small, valuable increments of a product frequently, providing stakeholders with opportunities to provide feedback and adjust priorities.

- **Scaled Agile:** For larger organizations or complex projects, frameworks like SAFe (Scaled Agile Framework) and LeSS (Large Scale Scrum) extend Agile principles to multiple teams or across entire organizations.

- **Popular Tools:** Various tools support Agile processes, including JIRA, Trello, and Kanban Flow for project management, and Continuous Integration (CI) and Continuous Delivery (CD) tools for automated testing and deployment.

- **Cultural Shift:** Agile often requires a cultural shift within organizations, emphasizing collaboration, empowerment, and trust among team members.

- **Widespread Adoption:** Agile methodologies have been adopted not only in software development but also in other industries, including marketing, manufacturing, and healthcare.



Team Members

PO and SM

- Product Owner: Thakshak Shetty
- Scrum Master: Meet Dave

Developers:

- Arfan Sayyad
- Clifton Martis
- Sri Nidhin Tankala
- Deviprasad Nallaboyina
- Bhuvana R
- Chandana N
- Suchitra
- Mouisha N
- Disha S K

Sprint 1(Hardware) (Priority – 35%)	SP	Sprint 2(Software) (Priority – 25%)	SP	Sprint 3(Testing) (Priority – 40%)	SP
Wheel Speed Monitoring and Control	8	Using CAN communication data from the sensor is given to intended microcontroller-6	6	Normal Braking (ABS activates, wheel lock prevented) -6	6
Preventing Skidding and Maintaining Steering Control	8	Using stm32microcontroller connect the sensor to the controller and their function-7	7	Slippery Surface (ABS activates, maintains control) - 8	8
Shortening Braking Distances	4	By using LIN communication, transmitting the signals from mc to intended actuator(break)-7	7	emergency Avoidance (Steering input allowed by ABS) -6	6

/TataTechnologies

@tatatechnologies

/TataTech_News

/TataTechnologies

/TataTechnologies