Automotive Systems

Team 5 Project documentation

Group members:

Truong Hoang Xuan	7207032
Tejas Hosur Upendra	7206911
Swathi Sudeendra Rao	7207100
Ernest Hunko	7207277
Torben Müller	7206493

Table of Contents

Introduction

Introduction to Model-Based Systems Engineering (MBSE)

The complexity of the automotive system is increasing significantly. It is a complex system by the variety of connected components. Using document-based information exchange to engineer the modern automotive system in a proficient way is facing many challenges including both technological and social contexts [1]. Therefore, MBSE which is a system engineering methodology that focuses on creating and exploiting domain models [2] is an alternative solution for automotive system development nowadays.

MBSE uses the formalized application of modeling such as SysML, Modelica to support system requirements, design, analysis, verification and validation activities [3]. MBSE begins with the conceptual design phase, and continues throughout the development phase and later life cycle phases. MBSE provides model-centric approaches adopted by other engineering disciplines, including mechanical, electrical and software [4] which are main factors of an automotive system.

This report uses the MBSE method to implement A Parking Maneuver.

Why Parking Maneuver

We select Auto Parking Maneuver as a case study for the automotive system because parking is a common action that drivers must perform with any car. In this parking scenario, we will simulate two situations that a driver is trying to search for a parking lot in a city. The first one is that the driver finds a parking lot, but obstacles suddenly appear, and the driver must find another parking lot. The second one is that the driver also sees a space for parking, however he is not allowed to park, and the driver must find another too. Both situations are popular cases in a big city with the high density of population. In a short period of time, we can understand these situations better than the other cases which require more time and domain knowledge. Once we have a background on these situations, we can analyze them deeper and develop an auto parking solution.

Moreover, a parking scenario requires many connected components inside the car to communicate and work properly to perform safe and automatic parking. By considering this, we have the chance to understand the interaction between modules. To realize a solution, we must consider data flow from getting data from the environment, process it, transfer processed data to other modules and perform actions at the final. We also have the opportunity to validate the solution. We have a lot of useful information in the automotive domain with Auto Parking Maneuver at the end.

Automotive Software Engineering

The automotive industry has been changing by moving from a mechanical to software-intensive industry in which most innovation and competition rely on software competence [5]. Software becomes a main factor not only for customers who want to own a car but also for car manufactures. Hence, software engineering plays an important role in the automotive industry today, many companies and organizations get involved in this field.

Forty years ago, software in cars was only a small function to control the timing of ignition. At that time, a function was running isolated from the other software functions [6]. Today, software is presented in almost every part of a car, it is the combination of many different functions with a million lines of codes and continues increasing. These functions are connected and interacted real-time to ensure the higher safety level as well as the automatic operation. They are distributed throughout various electronic control units (ECUs) on several in-vehicle networks [7]. Automotive software becomes more and more complex and brings many challenges for developers.

Automotive software engineering becomes an essential discipline in the automotive industry to manage the complexity of automotive software as well as to provide the methods for further function development. Automotive industry has developed standards, solutions and platforms both in process and in production such as Automotive SPICE, ISO-26262, Motor Industry Software Reliability Association (MISRA), AUTomotive Open System Architecture (AUTOSAR), Open Systems and their Interfaces for the Electronics in Motor Vehicles (OSEK) [8]. This is an adaptation of the results and solutions offered by software engineering in other domains to the automotive domain. However, the unique characteristics, constraints, and requirements of the automotive industry [9] require different solutions. Automotive software engineering aims for such solutions by using techniques in software engineering to find the universal solution through concepts, models, and technologies to develop and validate software for automotive systems.

Knowledge

System

Automatic parking is an autonomous car-maneuvering system that moves a vehicle from a traffic lane into a parking spot to perform parallel, perpendicular, or angle parking. The automatic parking system aims to enhance the comfort and safety of driving in constrained environments where much attention and experience is required to steer the car. The parking maneuver is achieved by means of coordinated control of the steering angle and speed which takes into account the actual situation in the environment, which it receives with the help of sensors, to ensure collision-free motion within the available space. [20]

It works like this: On arriving at the designated spot for handover, the driver will trust the vehicle to navigate the route and park safely. When the driver's ready to leave, he simply launches the reverse action and the car moves back to the road from the parking spot.

Functional Safety

Safety denotes that the cause of an activity by someone or something does not involve hurting people as an effect. However, there is no possibility to make a system completely safe, so efforts are made in systems to attempt to diminish the potential for harm to an adequate level. The automotive industry is to deliver innovative and enhanced vehicle safety systems, ranging from infotainment systems to extremely complex Advanced Driver Assistance Systems (ADAS) with accident estimation and prevention capabilities. Out of such everyday scenarios, we have picked autonomous parking as a user story and had to have defined a few safety functions. These safety functions are progressively made by refined mechatronic systems and ISO 26262 has been developed to enable the design and assess such systems that can prevent dangerous failures or control them when they occur.

ISO 26262

ISO 26262 is an international Standard for Functional Safety for Road Vehicles. That is, ISO 26262 is designed to ensure that the embedded systems for Road vehicles are designed with an appropriate level of rigour for their intended application. Revision of ISO 26262:2018 was released in December 2018 with 12 parts in it. [21]

Part 1: Vocabulary
Part 2: Management of functional Safety
Part 3: Concept phase
For a given Product "Item", identify relevant safety lifecycle steps, perform a Hazard Analysis, determine ASIL, identify Safety Goals and identify Functional Safety Concept[22], [23]

Hazard Analysis and Risk Assessment[21], [22], [23]

Hazard Classification:

The identified potential hazards are classified based on the estimation of severity, probability of exposure, and controllability (S, E, C). Severity or S has 4 classes ranging from S0 or no injuries to S3 or fatal injuries. Exposure or E ranges from an E0 or extremely unusual situation to E4 or a highly likely situation. Finally, controllability C ranges from C0 or simply controllable to uncontrollable or C3. These are determined with the help of Table 1.

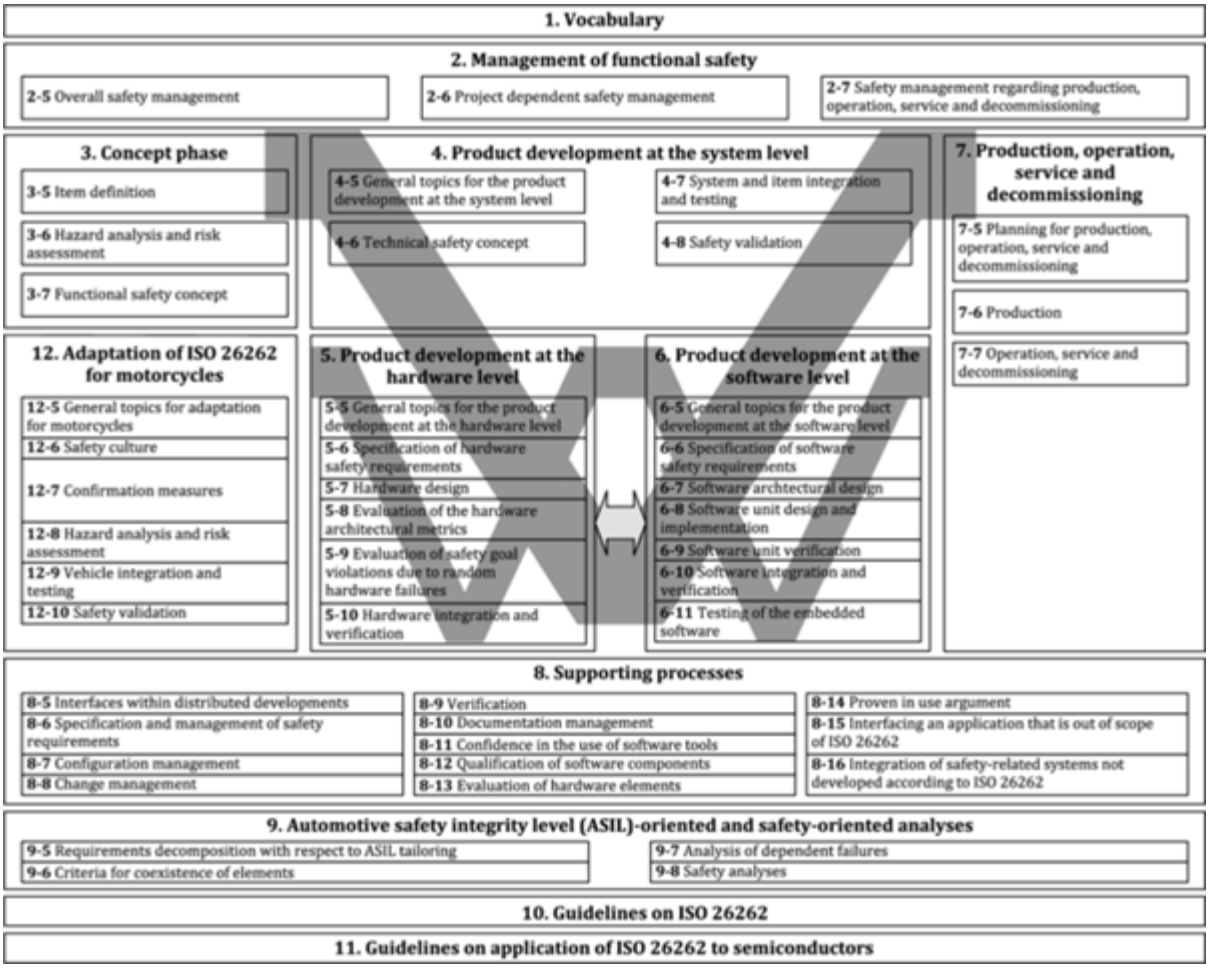

Figure1: Structure of ISO 26262:2018

ASIL Determination:

An ASIL is to be determined for each hazardous event using the parameters S, E and C as shown in Table below. Four ASILs are defined, where ASIL A is the lowest safety integrity level and ASIL D the highest one. In addition to these four ASILs, the class QM (Quality Management) denotes no requirement in accordance with ISO 26262, however any other requirements such as Quality, Reliability, and Durability need to be accounted for.

The ASIL determined for the hazardous event is assigned to the corresponding safety goal. A potential hazard may have more than one safety goal, and if similar safety goals are

determined, they can be combined into one safety goal that will be assigned the highest ASIL of the similar goals.

Part 4: Product development at System level
Part 5: Product Development at Hardware level
Part 6: Product Development at Software level
Part 7: Production, operation, service and decommissioning
Part 8:Supporting Processes
Part 9: ASIL oriented and safety oriented analysis
Part 10: Guidelines on ISO 26262
Part 11: Guidelines on application of ISO 26262 to semiconductors
Part 12: Adaptation of ISO 26262 for motorcycles

		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Table 1: ASIL Determination (Source ISO 26262) [26]

SAHARA

We saw in the previous section that an industry-wide standard for assessing the functional safety of road vehicles, covering the whole product life cycle exists in ISO26262. However, the security flaws are addressed by a method called SAHARA (Security-Aware Hazard Analysis and Risk Assessment) [24] that encompasses threats with the use of the Threat Model. This approach enables the quantification of the probability of the occurrence and impacts of security issues on safety concepts (safety goals).

Threats are quantified with reference to the ASIL analysis, according to the resources (R) and know-how (K) that are required to pose the threat and the threats criticality (T). These three factors determine the resulting security level (SecL). The SecL determination matrix is based on the ASIL determination approach and is illustrated in the following figure. The quantification of the impact of the threats, on the one hand, determines whether the threat is safety-related (threat level 3) or not (all other levels). This information is the trigger to hand over the threat for further analysis.

Level	Required Resource (R)
0	no additional tool or everyday commodity
1	Standard tool
2	Simple tool
3	Advanced tool

Table 2: Determination of the ’R’ value for required resources to exert a threat

Level	Required Know-how (K)
0	no prior knowledge (black-box approach)
1	technical knowledge (gray-box approach)
2	domain knowledge (white-box approach)

Table 3: Determination Of The ’K’ Value For Required Know-How To Pose A Threat

Level	Threat Criticality (T)
0	no security impact
1	moderate security relevance
2	high security relevance
3	high security and possible safety relevance

Table 4: Determination Of The ’T’ Value Of Threat Criticality

		Threat Level 'T'			
Required Resources 'R'	Required Know-How 'K'	0	1	2	3
0	0	0	3	4	4
	1	0	2	3	4
	2	0	1	2	3
1	0	0	2	3	4
	1	0	1	2	3
	2	0	0	1	2
2	0	0	1	2	3
	1	0	0	1	2
	2	0	0	0	1
3	0	0	0	1	2
	1	0	0	0	1
	2	0	0	0	1

Table 5: Determination Matrix - ascertains the security level from R, K and T values [24]

Development Process

Now with the functional safety defined with the ISO 26262, we need to ensure that the development follows these guidelines to create a safe system. For that we can use the development process Automotive SPICE.

SPICE is an acronym and stands for “System Process Improvement and Capability Determination”. It evolved from an ISO project and was published first in 1998. The different parts were all published from 2003 onwards with part 5 being published in 2006. The breakthrough of SPICE came in 2003 when the OEM software initiative decided to use SPICE as an evaluation for suppliers in the software and electronic field. In 2005, the Automotive Interest Group (AUTOSIG) published the Automotive SPICE-Model, which was now specifically tailored for car manufactures to perform assessments of their software and electronics suppliers. [17]

Automotive SPICE consists of two dimensions: The process dimension and the capability dimension. The process dimension is defined by the process reference model (PRM) which is a main part of Automotive SPICE. All the required processes are defined in the process reference model and grouped into three categories: Primary Life Cycle Processes, Organizational Life Cycle Processes and Supporting Life Cycle Processes. These categories are again split up to different groups which then have different processes assigned to them (Figure 2). [19]

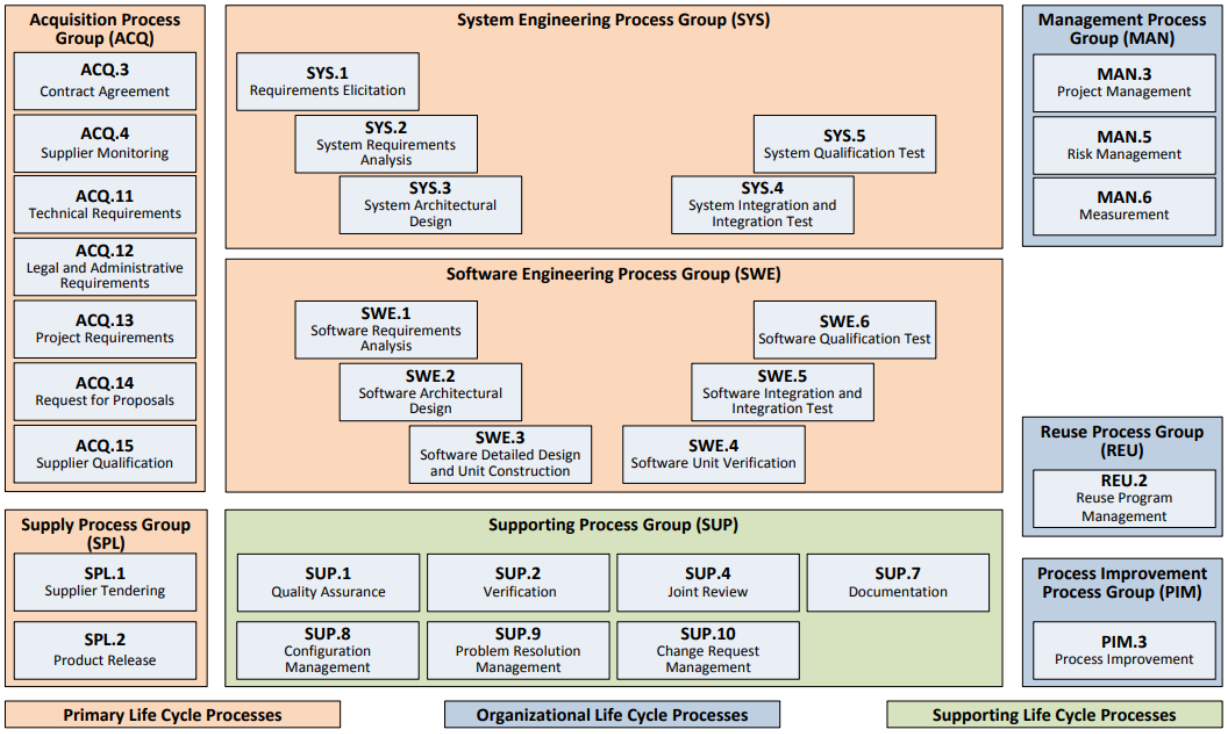

Figure 2: Automotive SPICE process reference model - Overview [19]

The second dimension is the capability dimension. The measurement framework of Automotive SPICE defined the necessary rules to determine the capability level of a given process. To determine the capability there are 6 different levels which can be assigned to a process under assessment (Figure 3). [19]

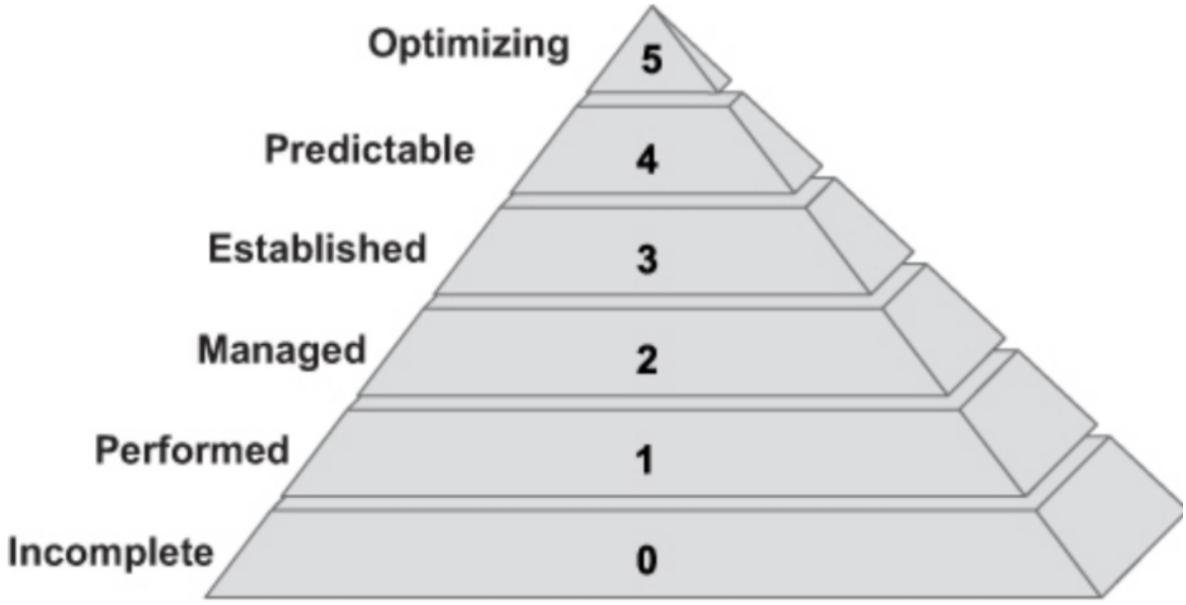

Figure 3: Capability levels of a process [17]

Level 0 describes a process that is not implemented or that does not fulfill its purpose. Project successes are possible but only based on the individual staff members efforts. Level 1 describes a process that fulfills his purpose. This could be called basic operation. Level 2 now includes planning and tracking of the process performance. Level 3 defines a standard process that could be reused in the organization for multiple projects. This process will achieve the defined results. Level 4 increases the predictability of the process by measuring and analyzing the process. This will make the quality of work quantitatively known. The last level 5 defines processes that are continuously improved by tracking quantitative process objectives based on the organization's business goals. This is the highest level a process can achieve. [19]

The last part of the Automotive SPICE is the process assessment model (PAM). It offers indicators in order to identify if process outcomes are present or not in a given set of projects. The indicators provide guidance for the assessors while collecting the relevant objective evidence to judge project capability. They are not meant as a checklist to be followed. [19]

An individual process now can be drawn as a table containing the process reference model in form of base data of the process and different process performance indicators which can be used to define the process assessment model. Figure 4 shows a template of a level 1 process. A set of these processes define the whole project and enable it’s performance and capability to be measured.

Process reference model	**Process ID**	The individual processes are described in terms of process name, process purpose, and process outcomes to define the Automotive SPICE process reference model. Additionally a process identifier is provided.
	Process name	
	Process purpose	
	Process outcomes	
Process performance indicators	**Base practices**	A set of base practices for the process providing a definition of the tasks and activities needed to accomplish the process purpose and fulfill the process outcomes
	Output work products	A number of output work products associated with each process *NOTE: Refer to Annex B for the characteristics associated with each work product.*

Figure 4: Template for the process description [19]

AUTOSAR Standard

Figure 5:: AUTOSAR Logo

Embedded software development in the automotive industry is very different from typical software development in other domains such as web development, mobile application, windows-based application [10]. Software in automotive controls a vast number of functions that communicate via linked networks. It is necessary to have a standard for handling the complex interactions of these functions and controlling the software environment. The acceleration of development, the increment of function integration as well as the increment of number of Electronic Control Units (ECUs) in a car are challenging vehicle manufacturers and their suppliers [11]. An industry standard for developing automotive software offers an effective way to manage the system-level complexity [12], keeps affordable costs and ensures safety standards.

Automotive Open System Architecture (AUTOSAR) is a standardized and open software architecture for ECUs [13]. It focuses on managing the growth of complexity in automotive software architecture, with the aim to standardize the software architecture of Electronic Control Units, enable new technologies and improve the efficiency without making compromises on quality [14]. AUTOSAR describes architecture, application interfaces and methodology [15] for automotive software. It allows different vehicle manufactures, different electronic component suppliers to work on the same software generation by increasing reuse and exchangeability of SW modules.

Benefit of AUTOSAR

The software architecture of AUTOSAR makes hardware and software widely independent. It is not necessary for each Original Equipment Manufacturer (OEM) to create software with their

own hardware. AUTOSAR makes software development time and cost are reduced significantly for OEMs by enabling software reusability.

Figure 6: Exchangeability of AUTOSAR [16]

AUTOSAR also supports suppliers in many ways. It allows suppliers to handle software variants efficiently by customizing and reusing software modules for each OEMs. By having a standard architecture, suppliers can focus application development only which helps to increase the efficiency.

Additionally, AUTOSAR gives the opportunities for tool providers by enabling them to interface with the development process and provide tools to use in the AUTOSAR development environment.

Finally, AUTOSAR opens new business models by means of standardized interfaces. Many companies and organizations are involved in developing AUTOSAR which makes automotive software understandable.

Modeling Language and Tools

3D Engineer:

This tool Is used to provide the 3-D visualizing capability of the scenario from which the sequence diagrams can be derived. The Scenarios can be captured using the models of elements needed such as vehicles ,traffic signs, pedestrians and many more in which some are already present in the tool or the models could be imported into it from websites such as sketchfab.The image below shows the basic window of the 3-D engineer tool which could be later populated with the models according to the required scenario.

Modelling Language:

Scenario-based modeling allows engineers to capture specifications in a way that is very close to how they would naturally conceive and communicate the requirements that is by describing, separate stories, how the system may, must, or must not react to certain events. Scenario Tools

supports scenario-based modeling with the Scenario Modeling Language (SML), which is a textual variant of Live Sequence Charts (LSCs).

Figure 7: 3D Engineer

The Scenario Modeling Language is a formal textual domain specific language created with the Xtext framework for designing system specifications. It provides a user-friendly editor with syntax highlighting and syntactic and semantic helper functionality.

In a scenario you specify the desired system behavior. There are three types of scenarios:

1. Requirement scenario: This is a high-level requirement.
2. Specification scenario: In this scenario you specify the required system behavior.
3. Assumption scenario: This scenario is used for modeling the behavior of the environment. These assumptions are used to simulate the behavior of the real environment for the simulation and synthesis algorithms.

The scenario consists of a list of messages. While active the messages must occur in the order the scenario specifies them. If the order is violated, the system shuts off or the scenario becomes invalid, depending on the modalities of the violated message.

A Message can be requested or not requested. If it is requested the system must make sure the message occurs eventually. If it does not occur, the liveness property of the system is violated. Messages can also be parameterized, and you can specify different behavior for various parameters.

You can not only specify messages in scenarios, but also loops, alternatives and parallel messages for further modelling possibilities. You can also define variables in the scenarios and use them as counters for loops or as parameters for messages.

Test-Driven Scenario Specification (TDSS)

It is the combination of formal scenario-based specification with the analysis of test-driven development (TDD). Test-Driven Development starts with designing and developing tests for every small functionality of an application.The simple concept of TDD is to write and correct the failed tests before writing new code (before development). This helps to avoid duplication of code as we write a small amount of code at a time in order to pass tests. Following steps define how to perform TDD tests.

1. Add a test.
2. Run all tests and see if any new test fails.
3. Write some code.
4. Run tests and Refactor code.
5. Repeat.

In TDSS, we assume that we are developing a reactive software component and have a source of requirements (e.g., a document or human stakeholder) that specifies what events or data are the inputs and outputs of that components as well as what the desired relationship of these inputs and outputs is over time. Then, TDSS consists in a repeated process where we take a requirement, and first create a test scenario for it, which is expected to fail. We then add one or several scenarios to the specification to formalize the requirement and satisfy the test case. If the test fails, this might reveal a simple modeling problem, or an inconsistency in the requirements modeled thus far. If the test passes, the process is repeated, first with further tests for the same requirement, e.g., covering corner cases, and then the process is repeated for the next requirement. Finally, we obtain an executable requirements and test specification of the component to be developed.Simple diagram explaining the process is below:

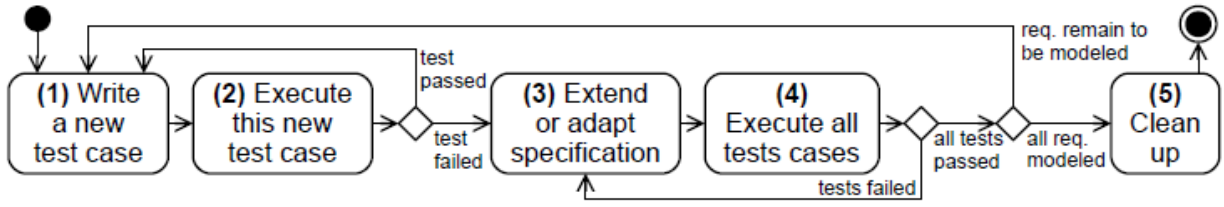

Figure 8: Test-Driven Sequence

SCiL Test environment

This is a new test environment used to test the scenarios using feature files which is a plain text file which can be understood easily as it is a simple text file containing the requirements. which would be later converted to a kotlin language file and the scenario modelling language approach would be used to evaluate the test cases for the requirements. The example for the usage of

this environment would be explained with our solution for the parking maneuver in the following section.

Our Solution

Application Environment Model (Information flow):

In the Environment model, we show what information flows between the vehicle and its environment is for the autonomous parking scenario. The vehicle needs to scan the area of the environment for a parking spot. The vehicle needs to get the positions of vehicles in the parking slot or moving into or out of the parking slot with the help of radar and sonar sensors. It also needs to look out for the obstacles and take suitable action based on the obstacle. It will also scan for the parking sign and if no parking sign is detected the vehicle should move ahead and not stop.

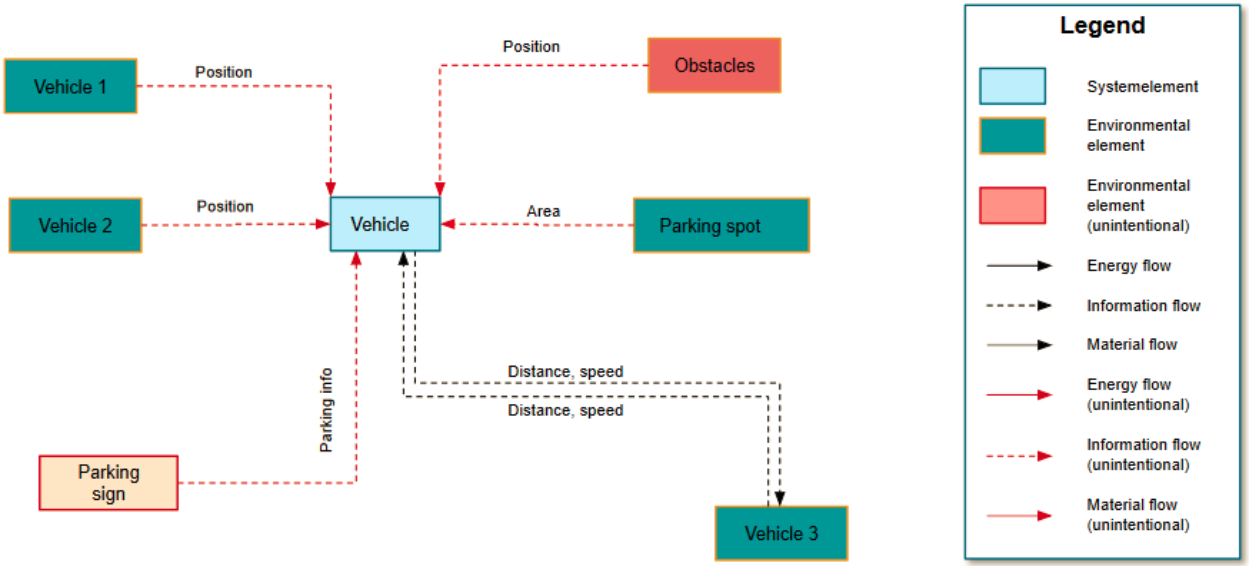

Figure 9: Application Environment Model

Sequence Diagram (Use case):

The following sequence diagram depicts the behavior of the vehicle when it detects the NO parking sign. The sequence flow on the right could be visualized with the diagrams on the left with the help of the tool 3-D Engineer. The sequence of steps followed are shown below:

- The vehicle detects the parking sign.
- The vehicle recognizes the No parking symbol.
- The vehicle adjusts the speed.
- The vehicle checks for new parking slot
- The vehicle continues to drive.

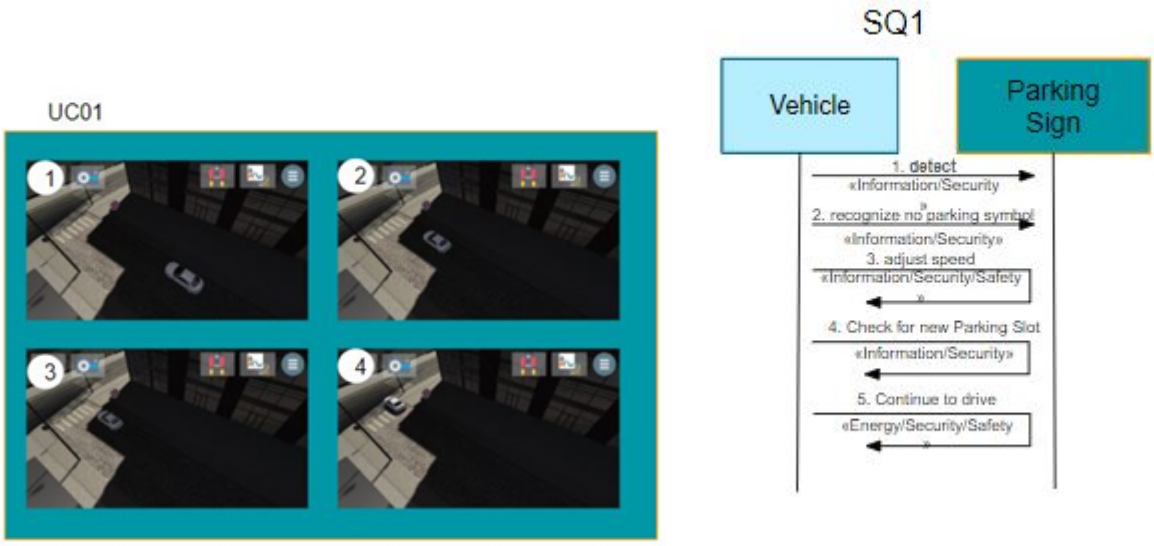

Figure 10: Use case and sequence

Sequence Diagram (Threat case):

In this Sequence diagram, we show the threat case scenario when the vehicle finds an obstacle in the parking spot and rejects the parking spot and goes ahead to find a new parking spot. The sequence flow can be visualized by the diagrams flow below by using the 3-D Engineer tool. The sequence of steps followed are shown below:

- The silver vehicle searches for a parking spot.
- The silver vehicle starts the measurement of the area of the parking spot.
- After confirming the area required for parking the vehicle starts the parking process.
- During the Parking process, the vehicle detects an obstacle in its course.
- The vehicle aborts the parking process.
- The silver vehicle detects another black vehicle/car approaching on the road and decides to stop.
- Wait for the black vehicle to go away from the range of the vehicle.
- Leave the parking spot.
- Continue driving to find a new parking spot.

Figure 11: Threat case and sequence

White box sequence diagram

White box view of a sequence diagram can be used to show the components that work together to provide the functionality of the sequence of activities among these components. The end-to-end flow through system components can be documented in this way.

White Box Sequence Diagram For Sequence Diagram 1: Use Case 01

The structural White Box Sequence Diagram (fig., 12), with necessary interfaces, for the first defined use case, is represented by the following diagram. The above defined actions in the Black box diagram are executed by various interfaces and are defined by these interactions. Various components such as Multi-purpose camera, near range camera, DASY, CGU, VCU perform the following task to execute the automatic no-parking sign detection and maneuver the car thereby with the help of Electronic engine control and electric power pack.

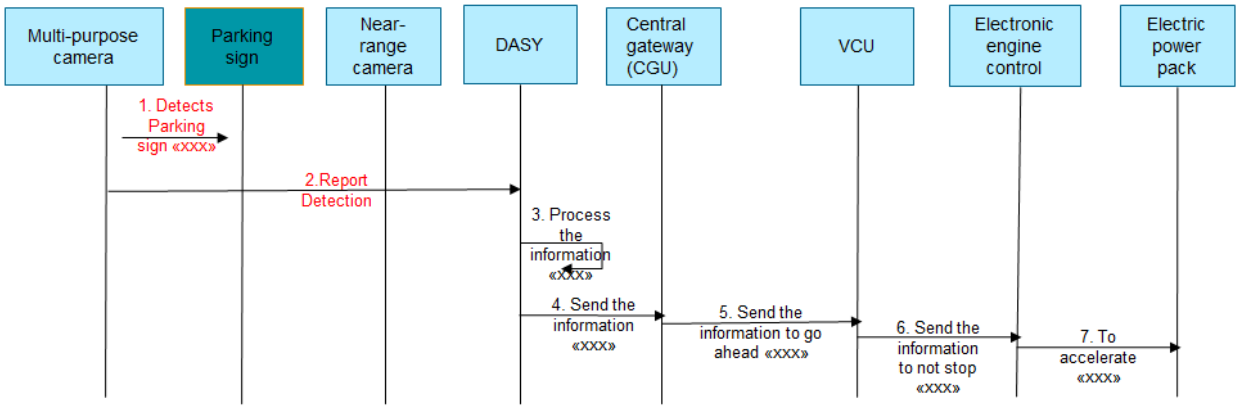

Figure 12: White Box Sequence Diagram For Use Case 1

After a careful observation made as per the SAHARA method, the following functions of “Detecting a parking sign” and “Reporting detection” were checked for their levels of SecL. The levels of ASIL are mentioned in the below diagrams for the same functions as well.

No	Function	Stereotype	K	R	T/S	SecL	S	E	C	ASIL
1	Detect parking sign	Information	1	3	3	3	2	2	3	B
2	Report detection	Information	2	2	2	2	1	2	2	A

Figure 13: Function set for White box Sequence Diagram 1 with SecL>0

White Box Sequence Diagram For Sequence Diagram 02

The structural White Box Sequence Diagram with necessary interfaces for the second, defined use case, are represented by the following diagrams. The actions defined in the Black box diagram are executed by various interfaces and are defined by these interactions as well. Components such as Ultrasonic sensor, Mid-range radar sensor, near range camera, DASY, CGU, VCU perform the following task to execute the automatic obstacle detection, stop/ decelerate the car and then proceed eventually from the parking position safely thereby, with the help of Electronic engine control and electric power pack.

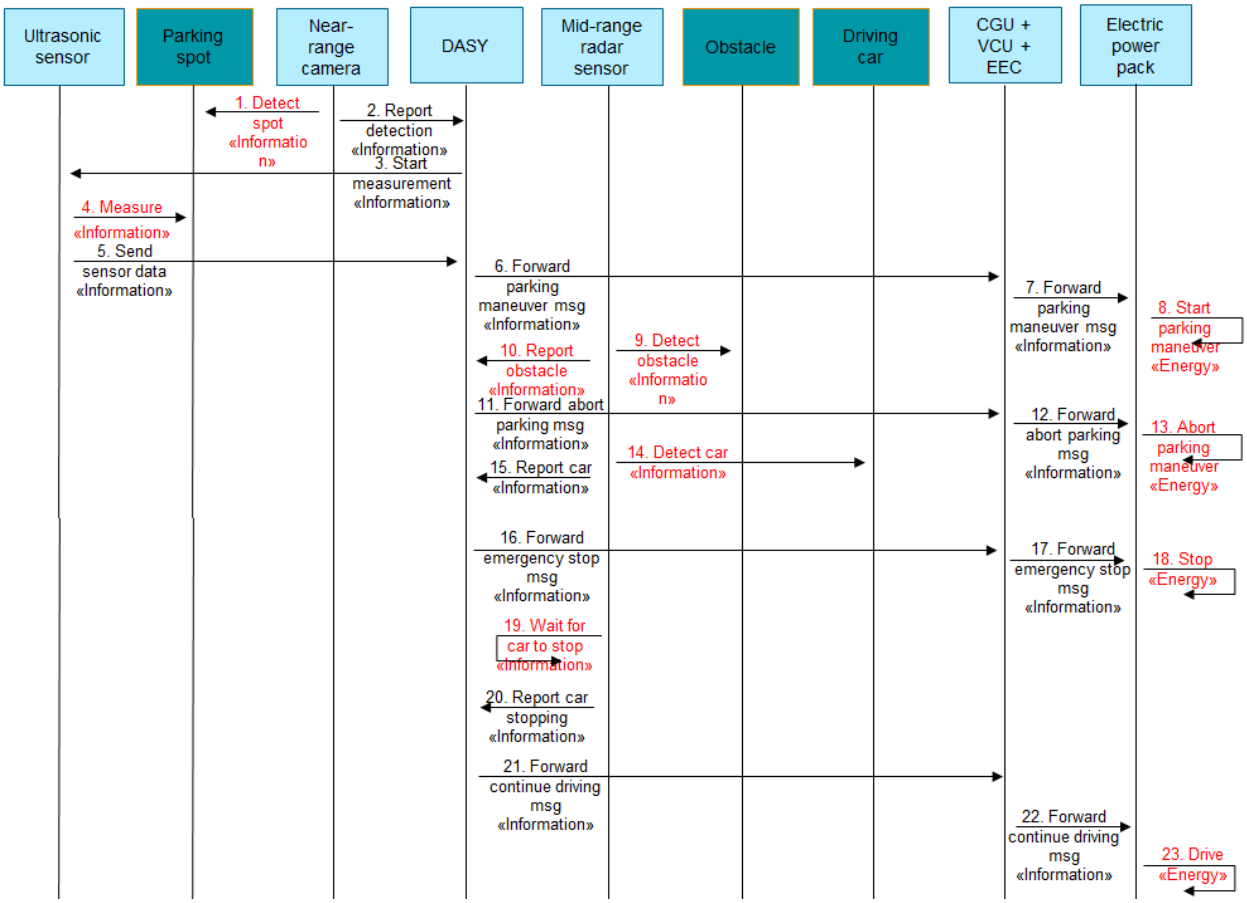

Figure 14: White Box Sequence Diagram For Use Case 2

No	Function	Stereotype	K	R	T/S	SecL	S	E	C	ASIL
1	Detect spot	Information	1	2	3	2	3	1	3	A
4	Measure	Energy	1	2	3	2	1	3	2	A
8	Start parking maneuver	Energy	2	2	3	1	2	4	1	A
9	Detect obstacle	Information	1	1	3	3	2	3	3	B
10	Report obstacle	Information	1	3	3	1	2	3	2	A
13	Abort parking maneuver	Energy	2	3	3	1	3	1	3	A
14	Detect car	Information	1	2	3	2	1	3	2	A
18	Stop	Energy	2	3	3	1	3	1	3	A
19	Wait for car to stop	Information	1	2	3	2	1	3	2	A
23	Drive	Energy	2	3	3	1	3	1	3	A

Figure 15: Function set for White box Sequence Diagram 2 with SecL>0

After a careful observation made as per the SAHARA method, the functions of “Detecting spot”, “Measure”, “Start parking maneuver”, “Detect obstacle”, “Report Obstacle”, “Abort parking maneuver”, “Detect Car”, “Stop”, “Wait for car to stop” and “Drive” were checked for their levels of SecL. The levels of ASIL are mentioned in the below diagrams for the same functions as well.

System Architecture

The proposed system uses the following modules onboard:

- VCU
- Central gateway
- DASY
- Multipurpose camera
- Mid-range sensor
- Near-range camera
- Ultrasonic sensor
- Electronic engine control
- Electric power pack

This entire set of modules is needed to successfully interact with the external objects and resolve different situations occurring in the everyday life on the road. The list of the external objects or situations, that system is ready to interact with or recognize:

- Obstacle - that can interrupt the process of the parking
- Moving car - that can interrupt the process of the parking or distract the sensors from the action
- Parking sign - that should be recognized in the case of the user’s safety
- Parking spot - the main target of all sensors to successfully recognize and finish maneuver
- Street

Recognition of the situation occurs through various sensors or cameras, and then this information is being processed.

This is clear, that the main module onboard is module DASY that provides connection between near-range, mid-range, multipurpose or ultrasonic sensors or cameras and central gateway for further data processing.

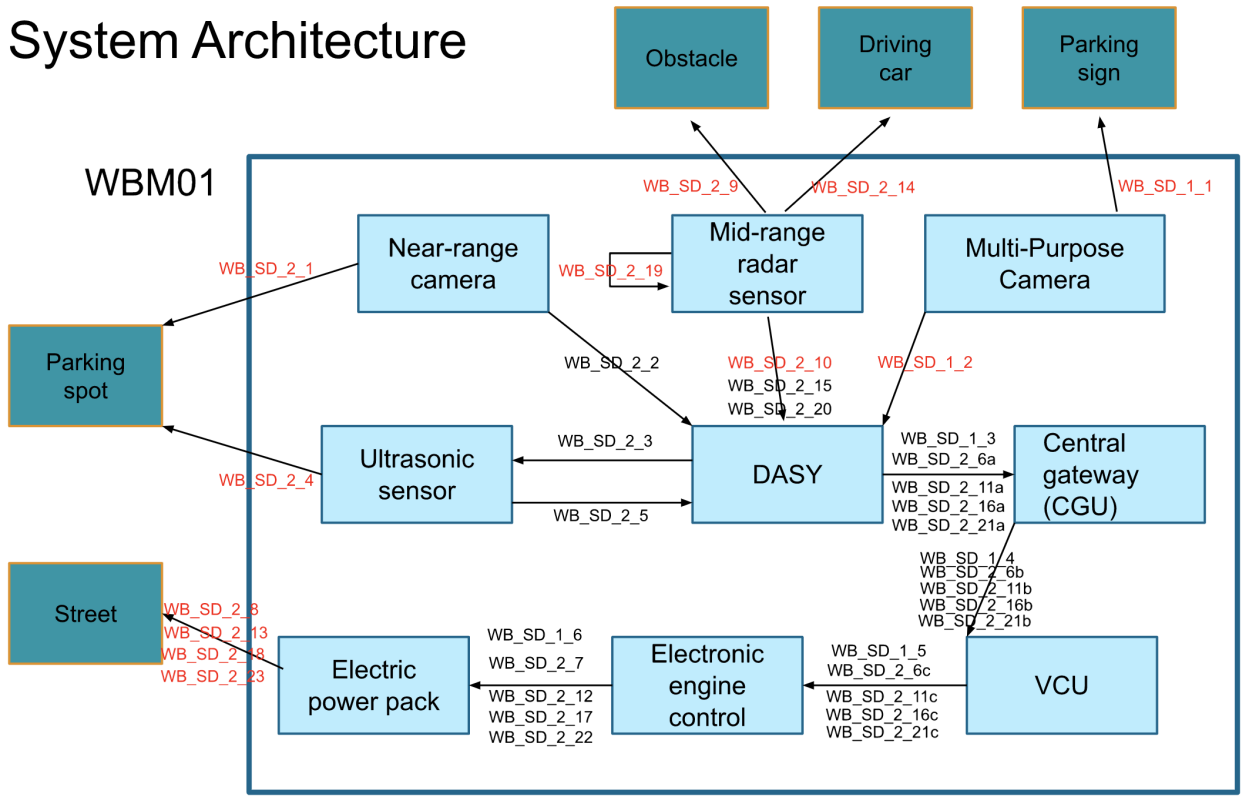

Figure 16: System architecture - modules and interactions between them

Requirement

After defining the parking scenario, the sequence for the interactions between components, and the system architecture, we start writing the requirement for Parking Maneuver in the table below. These requirements describe the main behavior of the system only.

Each requirement includes the attributes which are used for traceability as well as for evaluation.

- ID: make sure each requirement is unique
- Description: provides the content of requirement
- SHARA, ISO: evaluate the security and safety level for requirement accordingly base on that we know the importance of requirement

- Derived from: is used for traceability. Each requirement links to other contents. By that, we will know where the requirement come from or which feature will be satisfied

ID	Description	SAHARA	ISO 26262	Derived from			
		Security Level (0-4)	Safety Level (0-4)	Requirements	System Architecture	Use Cases + Sequence Diagram	Threat Cases + Sequence Diagram
R01	The vehicle system shall be able to detect parking spots automatically, using a camera sensor system.	2	2		WBM01		TC01, SQ2
R02	The vehicle shall be able to detect obstacle both in front and rear with a predefined range by using sensors	3	4		WBM01		TC01, SQ2
R03	The vehicle shall be able to measure the distance to obstacles both in front and rear	3	4		WBM01		TC01, SQ2
R04	The vehicle shall automatically stop within a predefined time in case the distance to obstacles less than a predefined distance	4	4		WBM01		TC01, SQ2
R05	The vehicle shall be able to detect and understand the parking signs on the road	2	3		WBM01	UC01, SQ1	
R06	In case the user is searching for a parking spot, the vehicle shall NOT perform the parking action when "No parking sign" is nearby.	2	2		WBM01	UC01, SQ1	
R07	In case the user is searching for a parking spot, the vehicle shall perform the parking action automatically when a space is greater than a predefined number and the "No parking sign" is not nearby.	2	1		WBM01	UC01, SQ1	
R08	The vehicle shall abort parking if obstacle is in parking spot	3	4		WBM01	UC01, SQ1	
R09	The vehicle shall be able to make a decision to stop or move forward in case another vehicle is nearby	3	4		WBM01		TC01, SQ2
R10	The decisions regarding detection must be based on information from the camera sensor system and the ultrasonic sensor system.	3	2		WBM01	UC01, SQ1	TC01, SQ2
R11	The decision message must be forwarded to the Electric Power Pack.	1	2		WBM01	UC01, SQ1	TC01, SQ2
R12	The vehicle shall be able to measure a detected parking spot using the ultrasonic sensor	2	2		WBM01		TC01, SQ2

Table 6: List of Requirements

Test Cases

With the parking system now fully specified, we can start working on test cases to test the defined interactions between components.

For that we use the tool-suite ScenarioTools which is an Eclipse based tool-chain that builds on the scenario modeling language. It especially targets the modeling and analysis of systems where the component structure can change at run-time. For an autonomous parking example this could mean a suddenly approaching person or a standing car that starts to move. With ScenarioTools you can define these different components and test their interaction. [18]

We use ScenarioTools together with Kotlin. First we create the scenario specification. For that we use the “Class and object model” to define the components of our system as classes. Later we can create instances of this scenario to then perform tests on. The components used in this scenario specification are the same as in the system architecture since this is supposed to represent our system. For each component (class) we define actions (events) this component can take to interact with other components. These events are placed inside the corresponding class and are later available to be used. Figure 17 shows the implementation for the mid range sensor. Since this sensor is responsible for detecting either obstacles or moving cars, we have defined two events which represent these actions directly.

```
class MidRangeRadar{
    fun detectObstacle() = event(){}
    fun detectMovingCar() = event(){}
}
```

Figure 17: Example definition of the component mid range radar

After all the components have been defined we created the scenarios in which these components act. A scenario defines a set of events that are executed in a if / then style. If the entry event of the scenario occurred, then a defined resulting event will be emitted. In this way you can chain events together to simulate information travel through the system. Figure 18 shows the event of a moving car detection by the mid range radar sensor being propagated to the electronic power pack which then sends the event “stop”.

```
scenario(midRangeRadar.detectMovingCar()){ this: Scenario
    request(dasy.processSensorData("movingCar"))
},
scenario(dasy.processSensorData("movingCar")) { this: Scenario
    request(centralGatewayUnit.forwardMessage("movingCar"))
    request(vehicleControlUnit.forwardMessage("movingCar"))
    request(electronicEngineControl.forwardMessage("movingCar"))
},
scenario(electronicEngineControl.forwardMessage("movingCar")) { this: Scenario
    request(electricPowerPack.stop())
},
```

Figure 18: Example scenario definition for detecting a moving car

With the scenarios now in place, we can define test cases based on expected behaviour and test if the system reacts in the expetect way. Defining a test base consists of sending an initial event and then waiting if an expected event is emitted eventually or not. In Figure 19 there are the defined tests for our system which all pass.

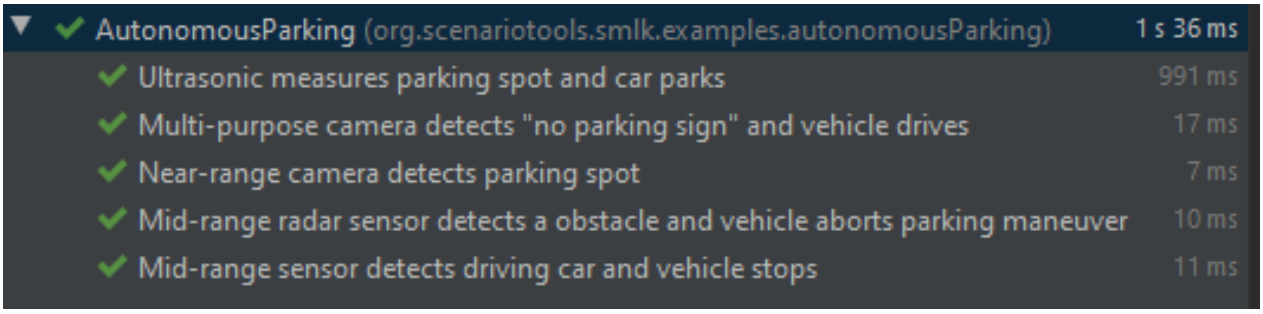

Figure 19: Test cases for the automated parking system

SCIL Scenarios

The SCIL method takes testing a step further by introducing features. A feature is a logical set of scenarios that can be grouped under a common name. The scenarios are then defined in plain text to best describe their purpose in the system. For that we use two feature files. The detection feature collects all scenarios related to detection and sensors. The movement feature collects scenarios based on the movement of the car.

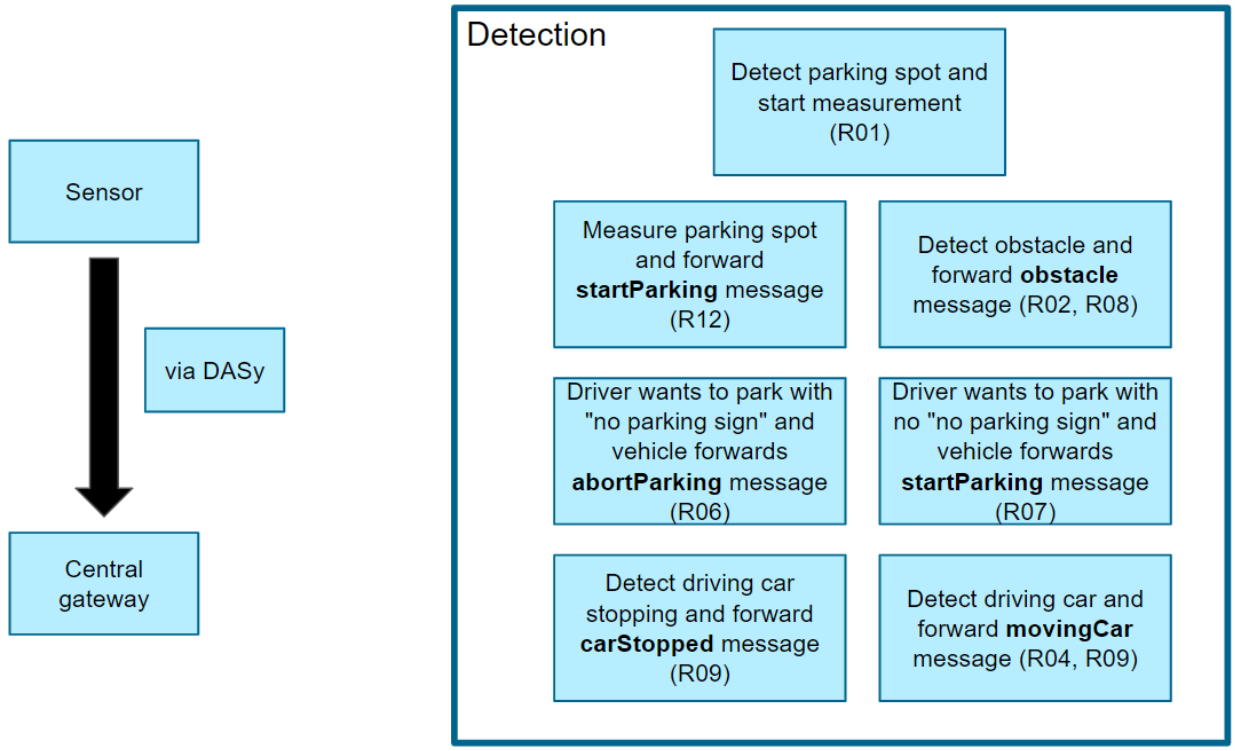

Figure 20: Contents of the detection feature

The detection feature is responsible to forward all sensor related scenarios to the central gateway. On the right side of Figure 20 you can see the different scenarios that the detection feature can face based on actions taken by the driver or other members of the traffic. The scenarios are triggered by sensor data or driver actions and result in sending message events. In bold you can see these message events which are being forwarded to the central gateway. From there they will be picked up by the movement feature and reacted upon accordingly.

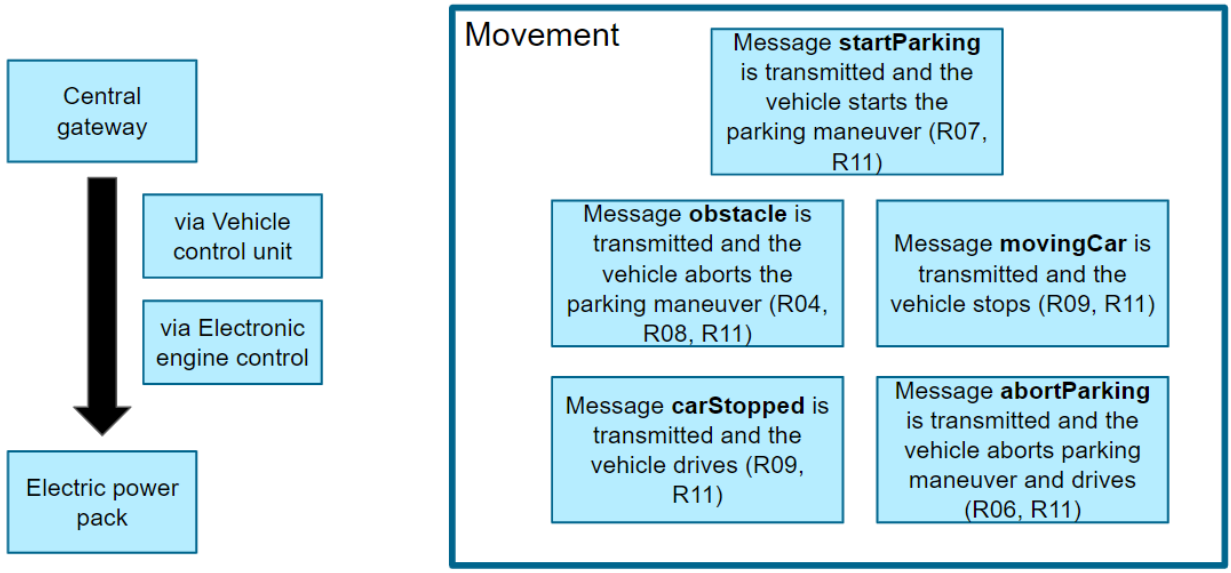

Figure 21: Contents of the movement feature

As you can see in Figure 21 the scenarios in the movement feature start at the central gateway and propagate through the vehicle control unit and the electronic engine control to the electric power pack where they will be translated to movement related actions.

```
Feature: Movement
    Background: init test setup
        Given init test setup

    Scenario: Message startParking is transmitted and the vehicle starts the parking maneuver (R07, R11)
        When the central gateway forwards startParking message
        Then the vehicle starts parking maneuver

    Scenario: Message obstacle is transmitted and the vehicle aborts the parking maneuver (R04, R08, R11)
        When the central gateway forwards obstacle message
        Then the vehicle aborts parking maneuver

    Scenario: Message movingCar is transmitted and the vehicle stops (R09, R11)
        When the central gateway forwards movingCar message
        Then the vehicle stops

    Scenario: Message carStopped is transmitted and the vehicle drives (R09, R11)
        When the central gateway forwards carStopped message
        Then the vehicle drives

    Scenario: Message abortParking is transmitted and the vehicle aborts parking maneuver and drives (R06, R11)
        When the central gateway forwards abortParking message
        Then the vehicle drives
```

Figure 22: Excerpt of the movement feature

With the features now fully defined (Figure 22), we generate steps from the feature files which include each plain text definition in the feature. For each “When” or “Then” definition there will be a step which needs to be filled with the right events to match the plain text definition. For example the step “Then the vehicle starts parking maneuver” will be filled with the action to wait for the event “startParkingManeuver” which is defined in the electric power pack. When all the steps are filled, then the system can be tested. The tool then combines the right steps each scenario and tests, if the defined behaviour in the steps is reached by the specified system. As you can see in Figure 23 all of our scenarios in both features have passed.

Figure 23: Results of the SCIL tests

Summary / Conclusion

This document describes the whole process of designing a system in an automated context. We created an Environment model which shows the basic interactions of our system in a black box style. Then we defined sequence diagrams in form of use cases and threat cases to model the behaviour our system should follow. Based on this we could derive the white box sequence diagram which includes the behaviour of internal components, whose need was now obvious by the defined behaviour. Here we also calculated the security level and ASIL of the individual behaviour steps to determine safety critical functions in our system. With the detailed behaviour in place we could define the system architecture depicting all the internal components and the messages they send between each other. Then we elicited requirements based on the created models and finished by creating tests for our designed system.

We determine that the MBSE method is very good to specify a automotive system, since it covers all important connections between components inside the systems and enables developers to design a safe system.

References

[1] C. Haskins, “Using patterns to transition systems engineering from a technological to social context,”Syst. Eng., vol. 11, pp. 147–155, Summer2008.

[2] Model-based systems engineering. [Online]. Available: https://en.wikipedia.org/wiki/Model-based_systems_engineering

[3] [4] MBSE Wiki, Object Management Group, Inc. [Online]. Available: http://www.omgwiki.org/MBSE/doku.php?id=start

[5] [6] [7] [8] [9] Alireza Haghighatkhah, Ahmad Banijamali, Olli-Pekka Pakanen, Markku Oivo, Pasi Kuvaja, Automotive software engineering: A systematic mapping study, The Journal of Systems & Software (2017)

[10] [11] [12] [13] [14] [15] AUTOSAR Basic Information Short Version 2014. [Online]. Available: https://web.archive.org/web/20151219053935/http://www.autosar.org/fileadmin/files/basic_information/AUTOSARBasicInformationShortVersion_EN.pdf

[16] AUTOSAR Introduction The vision, The partnership and current features in a nutshell [Online]. Available: https://www.autosar.org/fileadmin/ABOUT/AUTOSAR_EXP_Introduction.pdf

[17] Klaus Hoermann, Markus Müller, Lars Dittmann, Jörg Zimmer: Automotive SPICE in Practice. Surviving Interpretation and Assessment - Klare, konsistente und konstruktive Richtlinien und Hinweise fürs Assessment. Von führenden Automotive SPICE®-Experten. Rocky Nook, Santa Barbara, Ca., USA, 2008

[18] Joel Greenyera, Daniel Gritznera, Timo Gutahra, Florian Königa, Nils Gladea, Assaf Marronb, Guy Katz, “ScenarioTools A Tool Suite for the Scenario-based Modeling and Analysis of Reactive Systems”, 2017

[19] VDA QMC Working Group 13 / Automotive SIG, Automotive SPICE® Process Reference Model Process Assessment Model Version 3.0, 2015

[20] Automatic Parking - Wikipedia. [Online]. Available: https://en.wikipedia.org/wiki/Automatic_parking

[21] ISO 26262 Functional Safety Draft International Standard for Road Vehicles: Background, Status, and Overview, by Barbara J. Czerny, Joseph D’Ambrosio, Rami Debouk, General Motors Research and Development, Kelly Stashko, General Motors Powertrain

[22] A quick guide to ISO26262 by Feabhas Ltd., www.feabhas.com

[23] 2012 International Symposium on Safety Science and Technology, The Automotive Standard ISO 26262, the innovative driver for enhanced safety assessment & technology for motor cars, Peter Kafka.

[24] SAHARA: A Security-Aware Hazard and Risk, Analysis Method by Georg Macher, Harald Sporer, Reinhard Berlach, Eric Armengaud and Christian Kreiner, Institute for Technical Informatics, Graz University of Technology, AUSTRIA.

[25] https://en.wikipedia.org/wiki/ISO_26262
[26] https://www.iso.org/obp/ui/#iso:std:iso:26262:-12:ed-1:v1:en