JAIC
Journal of Applied Instrumentation and Control

On the Applicability of Behavior Driven Development for Automotive Software Testing at the Functional Model Level

José H. Roquette, Simone N. Matos, Max M. D. Santos

***Abstract*—Model-based design [MDB] is a well know and with a high level of maturity methodology for the design, build, and testing of automotive embedded systems. For the tools and technologies available, it enables to build functionalities with high quality in which the tests need to be done to verify whether requirements are attended in all stages from functional model to code stage. In the early stage of the development, the controller design might be done in a simulation environment integrated with the physical plant that allows us to test and check out whether requirements are attended. In addition, the model complexity can be increased and the testing method shall be automated in order to provide a testing coverage and consequently a high quality of the software developed. Therefore, we present the applicability of behavior-driven development for automotive software testing at the functional model level in MBD. Furthermore, we demonstrate the efficiency o four method over a small case study of power window system which enables to be deployed in other automotive functionalities to be controlled.**

***Index Terms*—Model-based design, automotive software testing, embedded systems, model-in-the loop, behavior driven development and functional model.**

I. Introduction

Model-based design [MDB] is a well know and with high level of maturity methodology for develop and testing of automotive embedded systems. For the tools and technologies available, it enables to build functionalities with high quality in which the tests need to be done to verify whether requirements are attended in all stages since functional model to code stage. As know as, we have a big dilemma in how to define the case tests in the first stage of MBD so that can be deploy in the following stages. In addition to, the testing method shall be automated in order to provide a testing coverage and consequently a high quality of the software developed. Therefore, we present the applicability of behavior driven development for automotive software testing at the functional model level in MBD. Furthermore, we demonstrate the efficiency o four method over a small case study of power window system which enables to be deployed in other automotive functionalities to be controlled.

This work focuses on tests performed at the functional model level proposing a new method of how MIL tests should be performed. After analyzing the project on power windows systems [1], it was verified that MIL tests are not performed in the best possible way, because they are done manually and in an empirical way, that is, based on the knowledge of the team involved in the project.

The proposed method aims to detect software failures even at the model level and to identify the largest number of possible test cases using techniques already applied within software engineering.

This paper is organized as follows: Section 2 presents the discussion of the problem of how the tests in automotive software development are realized after the analysis of an automatic window design for embedded systems. Section 3 describes the concepts used to elaborate a possible solution to the problem addressed in Section 2. Section 4 proposes a method that can be adopted for testing in embedded systems. Section 5 performs a case study applying the proposed method, in section 6 an analysis is made comparing its results with another method in the literature. Finally, Section 7 reports the

João Henrique Roquette - Departamento de Ciência da Computação, Universidade Tecnológica Federal do Paraná, Rua Doutor Washington Subtil Chueire, 330, Jardim Carvalho, 84017-220 Ponta Grossa, PR, Brazil
Simone Nasser Matos - Departamento de Ciência da Computação, Universidade Tecnológica Federal do Paraná, Rua Doutor Washington Subtil Chueire, 330, Jardim Carvalho, 84017-220 Ponta Grossa, PR, Brazil
Max Mauro Dias Santos - Departamento de Eletrônica, Universidade Tecnológica Federal do Paraná, Rua Doutor Washington Subtil Chueire, 330, Jardim Carvalho, 84017-220 Ponta Grossa, PR, Brazil

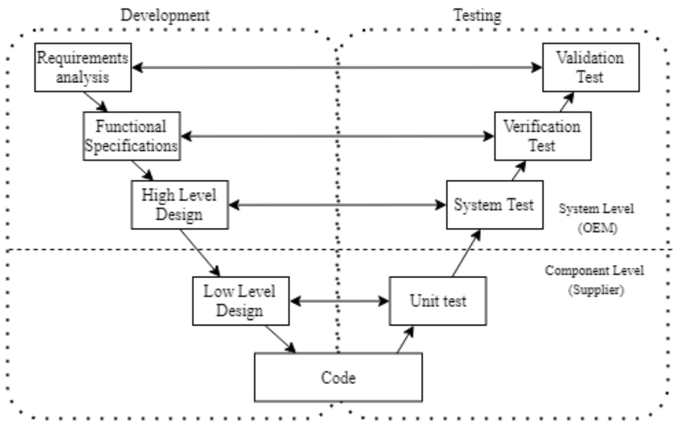

Fig. 1. The workflow of V-model for automotive embedded systems.

conclusions of this work.

II. V-Model and model-based design for automotive embedded systems

The automotive industry uses the V-model for automotive software development [2]. This model consists of two phases: development and testing. The stages of each of the phases are illustrated in Figure 1, where the stages of the development phase are on the left and the right of testing.

The V-model consists of nine stages, Requirements Analysis obtains the understanding of the problem, verifies what should be done in the project and the business values for each functionality, as well as its acceptance criteria. The artifact generated in this stage is used in Validation Test that verifies if the system is behaving as the client requested [3]. In the Functional Specifications stage, a document with the functions that a system or component is to perform is created [4], assists in the verification test stage that evaluates the results against the pre-specified requirements. The High Level Design stage defines the high level concepts that guide the implementation of Low Level Design and the design [3], this stage works with the System Test that tests the system as a whole [4].

The stages described above are system level. Regarding the level of components, there is the Low Level Design stage which is a document that contains step-by-step detailed how the system should act and used to perform Unit Test, which performs tests of concentrated form in each component of the system [4].

To optimize the development process of the V model, agile test methodologies can be applied, one of them being Model Based Design. It makes use of simultaneous engineering with the implementation of an agile model together with incremental testing [5]. This allows the development team to incrementally perform tests and requirements are discovered throughout development, iterative is greatest within development and assists in the validation of model-level requirements [5].

Communicating the stages of development with the test steps allows the project to be run while the software is under construction. In automotive systems, five types of tests are performed:

Model-In-The-Loop (MIL): used to find bugs within the logic of the controller, makes use of the model and the controller to perform the simulation [6];

Software-in-The-Loop (SIL) The model-in-the-loop test, which uses the previous-stage model to generate the code, performs the algorithm and logic test in a real-time virtual environment [7].

Processor-In-The-Loop (PIL): the generated code is tested and loaded into the processor on the system it will run. Different machines are used to perform the tests and have the objective of finding more complex errors such as memory usage, division by zero, etc. [6].

Hardware-In-The-Loop (HIL): one of the last tests to be performed on the software, it is checked if the system continues to meet the requirements of the client. It also aims to test whether the built-in software interacts properly with the target hardware by measuring its performance [8].

Vehicle-In-The-Loop (VIL): in this test the vehicle is placed in an open or blocked traffic environment, in an environment where the system functions tests can be performed in real situations, its greatest advantages are in safety tests, such as emergency brake [9].

Rapid Control Prototyping (RCP): In RCP applications, a plant controller is implemented using a real-time simulator and is connected to a physical plant. RCP offers many advantages over implementing in actual controller prototype. A controller prototype developed using a real-time simulator is more flexible, faster to implement and easier to debug. The controller prototype can be tuned on the fly or completely modified. In addition, since every internal controller state is available, an RCP can be debugged faster without having to take its cover off. See in Fig.1 d).

III. The traditional method of functional testing

The MIL can be defined as a basic simulation where tests are carried out in the initial phase of the project in order to analyze the controller model together with the model of the system plant through a simulation [10].

Generally the controller is implemented in tools such as Matlab / Simulink [11] and the model plant is built in Simulink where there is a connection between the controller and the plant [12].

By means of these tools it is possible to assign signal inputs, which correspond to the test cases that must be tested by the test engineers. These test cases serve as inputs for the execution of the simulation that generates the outputs, called test report, as shown in Figure 3. The test reports must be analyzed to verify if the simulation is being executed as expected, if the output does not does not provide an expected value, a problem with the environment or even the model syntax has probably occurred [10].

Manual testing can bring some problems to the project such as: the test team needs experience and information on the software and hardware that will be tested [13]. There is also the possibility of not identifying all the test cases of the project, taking more time to find all the bugs in the system. This causes financial damage to the project. In addition,

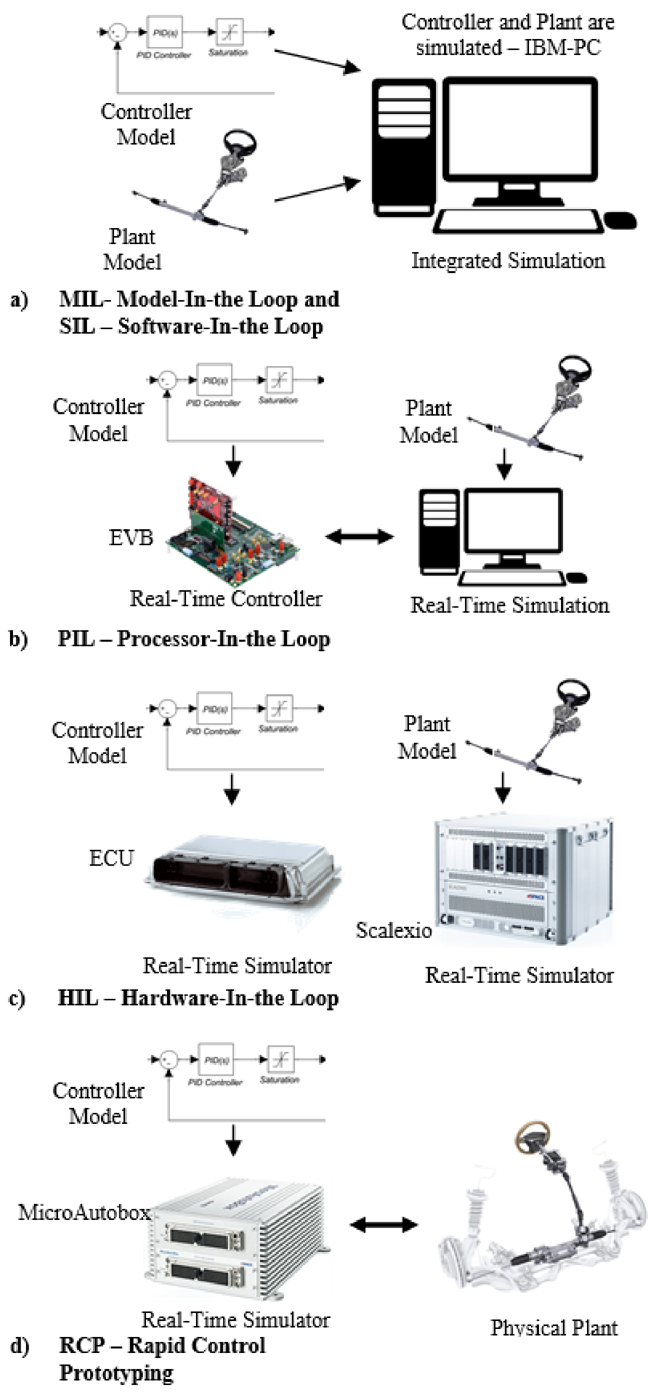

Fig. 2. Application categories of MIL, SIL, PIL, HIL and RCP.

presenting complex documentation can increase the difficulty of understanding the problem and cause misunderstandings within the development team which increases the chance of development errors. If you consider complex functions, it is often not possible to perform software testing manually.

IV. Related approach for functional testing

In this section we elaborated the theoretical concepts that were used to elaborate the method proposed in this article.

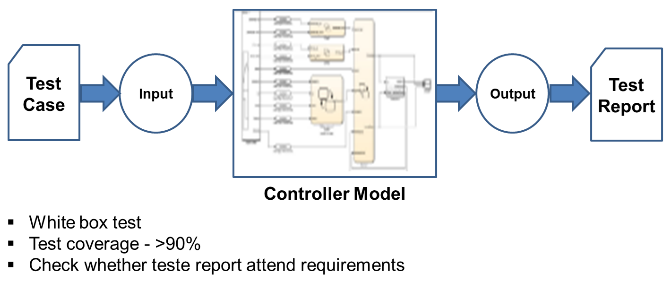

Fig. 3. The process of model in the loop testing.

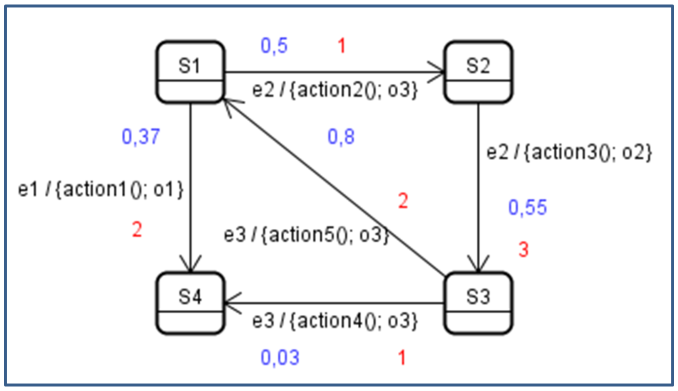

Fig. 4. Finite State Machine.

A. Finite State Machine (FSM)

A finite state machine can be defined mathematically in a tuple with six elements (S, S1, I, O, T, G), where S represents the possible machine states S = {S1, S2, ... Sn}; S1 is the initial state, always S1 $\in$ S; I and O represent the input and output values of the alphabet respectively I = {I0, I2, ..., In} and O = {O0, O2, ..., On}, each value of Ij (Oj) are functions which have a certain time to run within the state machine, which may vary according to their function. T within the tuple represents the triggering of events within the state machine that is generated by a signal that calls a transition rule represented by the letter G. Each transition rule has its structure that can be defined by a tuple, defined as follows θj = {Ssj, Sdj, ej, gj, aj, pj} where Ssj is the current state in which the state machine is located, Sdj is the destination state, and belongs to the T value of the tuple, which represents the activation of a gj is the guarding condition for this transition to take place, already aj is the action itself that occurred at that moment and finally the priority of that transaction described in the tuple by pj, in the lower its value its higher its priority over the others transactions [14]. Figure 4 illustrates a state machine and its behavior.

B. Basic Path Test(White-Box Test)

In the Basic Path Test the tester uses a logical complexity of a project and this measure defines a set of execution paths. Test cases created to practice basic set will surely execute all instructions in a program at least once [15].

This type of test works as a flow graph, which will represent traversing the flow of logical control, thus having all the

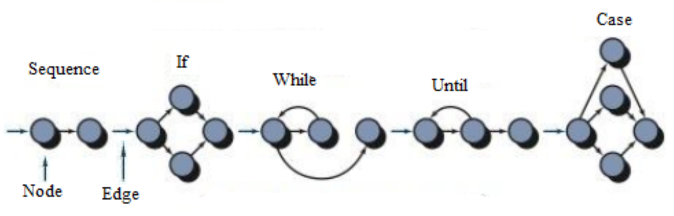

Fig. 5. . Structure of a flow graph [15]

conditions of a code. Figure 5 [15] shows an example of the structure of a flow graph.

In a flow graph, each circle represents a node that means a procedural command, the arrows represent the edges that show the flow that the graph must follow, having a flow graph of an algorithm, we can analyze all the independent paths [15].

An independent path is any path through the code that introduces a set of processing commands or a new condition, an independent path must include at least one edge other than other paths already defined.

Having the paths, it is possible to construct the test cases that execute such commands covering all possible directions of the system.

C. Behavior Driven Development (BDD)

One practice of software engineering is to help teams build and deliver a program with higher quality and with less time. This practice uses Test-Driven Development (TDD) and Domain-Driven Design (DDD), but its main feature is a common language for scenario creation, facilitating communication among project members [16].

In BDD it is common to begin by identifying what the program objectives are and to think what features the system can have to achieve those goals. In collaboration with the user scenarios are developed for these functionalities and are automated in the form of executable specifications which are used for development and documentation. Code-level BDD can help developers write high-quality, best-tested, documented, and easy-to-maintain code.

Scenarios are simple frameworks that clearly show what the purpose of functionality is that can become automated testing. Each has a title that describes what it represents and has common vocabulary expressions such as: Given, When, Then. Each expression has a meaning: Given describes the precondition and prepares the test; When describes the test action; and Then describes what is expected from the test. Table 1 presents the structure of a BDD scenario.

TABLE I
STRUCTURE OF A SCENARIO IN BDD

Title: <title of the scenario>
Given <initial context>
When <an action or event>
Then <expected result>

Scenarios in the BDD format can be extended in any of the steps using ”and” and its synonym ”but”, so in certain cases it becomes easier to understand about such a case. These scenarios are used to automate the writing of tests and serve as documentation for the project specifying how a system should behave.

This test development methodology can bring some benefits to the project, because with its use it can be perceived a reduction of costs and time, due to the increase of focus in functionalities that must be programmed. With reduced time spent maintaining code, this practice makes it possible to make changes to the code without causing damage to the project more easily [16].

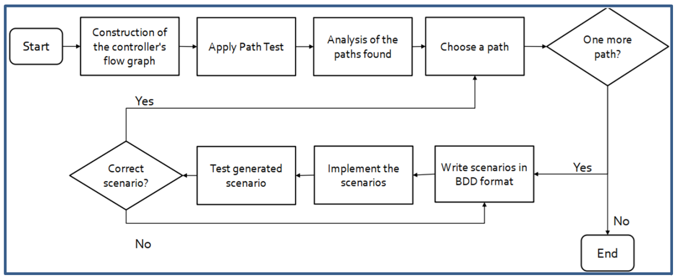

Fig. 6. Flowchart method.

V. The Method proposed to define the functional testing

The proposed method for carrying out automotive software testing using the concepts of Road Testing and the scenarios in BDD format is shown in Figure 6.

The method developed is structured into the following eight steps as shown in Table II

TABLE II
STRUCTURE OF A SCENARIO IN BDD

1. Perform the requirements analysis to obtain an understanding of the problem in question.
2. Create in appropriate tools, the model of the system controller and a state machine that meets the necessary requirements.
3. Realize the analysis of the model and from it build a behavioral flow graph of that state machine.
4. Analyze all possible independent paths, which can be both manual and automated. With this we will get the knowledge of all the test cases that must be programmed.
5. Choose a path and interpret it.
6. Write the path chosen in BDD scenario format specifying the initial context of that event, the action that will occur on that path, and the expected output of that test case.
7. Implement scenarios on test tools following their conditions.
8. Test the generated scenarios and analyze if they are correct. If they are not returning to step 6 and rewrite the scenario. If the scenario is meeting expectations, return to step 5 and consider if there are more paths. If they do not have more paths, this means that all test cases have been implemented and tested.

VI. Case study

In this section the proposed method was applied in the Power Windows System project [1]. In this application we have already defined the steps of requirements analysis and the development of the controller. Therefore, the explanation of the method starts in Step 3, defining the flow graph of the

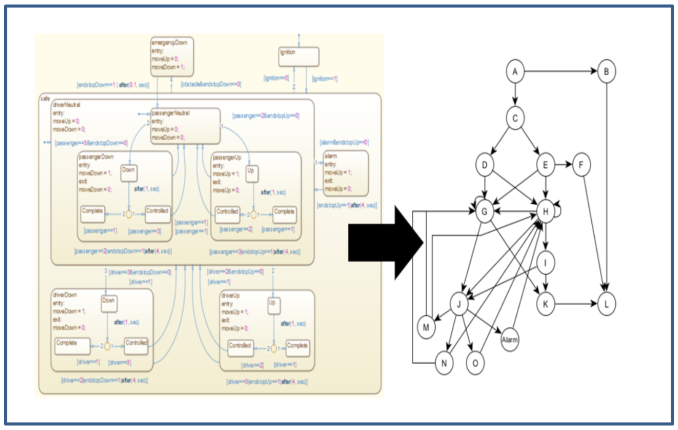

Fig. 7. The graph flow of the controller in operation.

controller after conducting a controller analysis, the following flow graph was elaborated in Figure 7.

For the construction of the flow graph, in Figure 6, we abstract the actuators of the system relating to their variables: the ”Driver” actuator responsible for the variables ”Driver Up” and ”Driver Down”, ”Passenger” can trigger ”Passenger Up” and ”Passenger Down” and lastly ”Nobody” that controls the ”Obstacle” and ”Alarm” variables.

We check which events can occur in the system. The following have been identified: ”sending a signal for more than one second to open or close the window”, ”sending a signal for less than one second to open or closing the window”, ”the automatic glass has encountered an obstacle”, ”Interruption in glass handling” and ”the car alarm has been triggered”

The graph in Figure 6 has its initial state in ”A”, the glass is stopped. The first ”if” defines whether the battery is on or not, if it is off, it will go to the ”B” state and end process in ”L”. If it is on it will go to state C, whose ”if” indicates which driver has or has not sent a signal. If yes, proceed to ”D”, otherwise to ”E”, in ”E” you have the same situation, but now we check the passenger’s signal. If he sends the signal, it goes to the ”G” or ”H” state, if not to ”F”, which means that nobody sent a signal then going to the ”L” state and finish the process.

In states ”B” and ”D”, one can move to the ”G” and ”H” states, where they respectively represent the up and down movement of the glass. Both states can have the same destination ”I” and ”K”. The second represents that the signal sent was greater than 1 seconds, being moved up to the duration of its signal and stopping (L). As for the ”I” state, this means that its signal lasted less than 1 second and should continue its movement (J), node ”M” represents that the system did not find any interruption or obstacle, ”N” means interruptions that can occur, being they up and down. The ”O” node means the encounter of an obstacle through the window and finally the node relating to ”Alarm”.

With the flow graph constructed, path analysis was performed. In this step, all possible independent paths were found using a tool [17] which assigns a graph as input and as output all possible independent paths are obtained. The tool used has limitations, it does not accept loops within the graph. The solution to this problem was to find and remove all existing loops, for this was created new nodes that supplied such a restriction.

We identified 24 different paths, that is, is possible found 32 test cases that should be tested at the model level. The paths were divided into 3 categories, ”Neutral”, in which no one sends a signal, ”Driver”, are the paths where the driver sends the first signal and ”Passenger” represent the ways in which the passenger sends the first signal, but after analyse this test cases we noticed that there were redundant cases, In Table III we can see how the test cases were classified.

TABLE III
ANALYSIS OF THE PATHS FOUND

Scenario	**Quantity**
Impossibles Situations	2
Referring to Driver	13
Referring to Passenger	13
Neutral	4
Total	32

The next step was to choose a path and write it in the BDD scenario format. To illustrate, this article shows the BDD scenario for the path A-B-E-G-H-J-OBJ-L which represents that the passenger sent a signal up and found an object, as can be seen in Table IV. However, 14 scenarios were identified that were written in the BDD format, and from them were generated 48 test cases.

TABLE IV
STRUCTURE OF A SCENARIO IN BDD

time = 10;
passenger_up = 0;
passenger_down = 0;
driver_up = 0;
driver_down = 0;
alarm = 0;
accessory = 0;
run = 0;
crank = 0;
battery = 1;
obstacle = 0;

Title: Passenger sent signal up and found an object
Given the passenger window is open
When $variable1 is fired in the interval %time1 to %time2 seconds
And find a $variable2 during the interval %time3 to %time4 seconds
Then the glass should find the obstacle and descend approximately 10 cm

variable1	*variable2*	*time1*	*time2*	*time3*	*time4*
passenger_up	*obstacle*	3	5	6	7
passenger_up	*obstacle*	3	3.5	6	8

In this test case, we declare the title ”Passenger sent signal up and found an object”, in this work we show the execution of the first line of values of the table described below under the scenario, whose variable1 and variable2 represent respectively the sign of the ”passenger up” and the obstacle, and the time values in which they were triggered are assigned the variables time1, time2, time3 and time4.

With the written scenarios, we can implement the scenario using tools like Simulink and carry out the simulation of the test cases and analyze its output comparing with the Then step

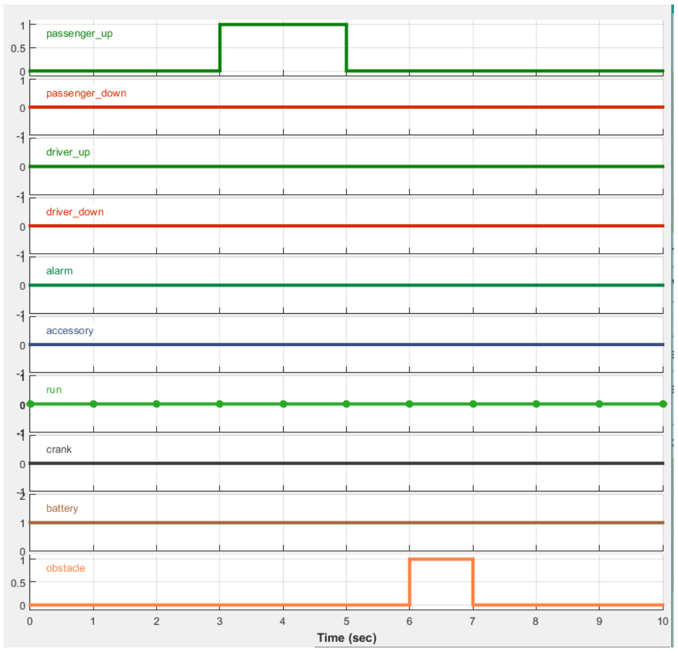

Fig. 8. The input signals of test case.

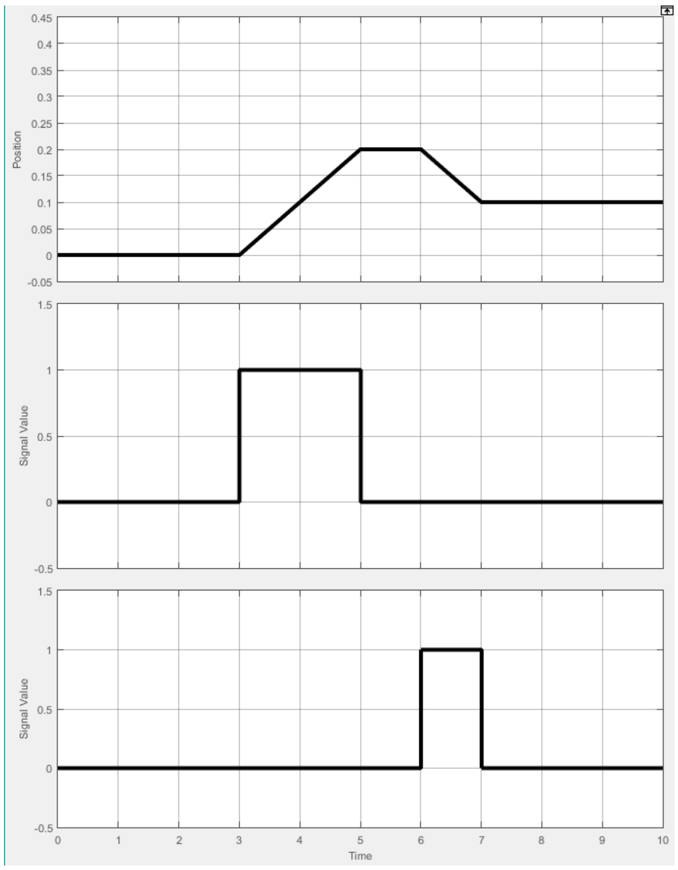

Fig. 9. . The output signals of test case.

of the scenario. We implemented the scenario within the tool, as can be seen in Figure 8.

After completing its implementation, when analyzing the output of the implemented scenario it can be verified that the scenario is correct, because when comparing the output generated by the tool with the stage Then of the scenario it is observed that in the first Graph of Figure 9, that the electric window identifies an object in the time of 6 seconds and moves down 10 cm, considering the scenario condition and requirement 6.

Therefore, this test case should receive the approved status, so we must go back to the Path and Analysis step, so that another path can be tested, until there are no more paths to execute.

In Table V, we describe all the results of the 48 test cases performed, following their ID, which group they belong to, what requirements they cover, and whether the test was approved or not. The ”OK” means that the test was approved and meets the conditions set forth in the scenario and ”FAIL” means that the development team should be reported, so that the error is corrected and tested again.

TABLE V
ANALYSIS OF THE PATHS FOUND

Test Case id	Group	Requirements	Test Report
#0, #4	Driver	1, 2, 3, 10	OK
#1, #5	Driver	1, 2, 3 5, 10	OK
#2, #3	Driver	2, 6, 7, 10	OK
#6, #7	Driver	2, 5, 10	OK
#8, #9	Driver	2, 5, 10	FAIL
#10 , #18	Driver	2, 5, 10	OK
#19	Driver	2, 5, 10	FAIL
#20, #21	Driver	2, 5, 10	OK
#22	Passenger	1, 2, 3, 10	FAIL
#23	Passenger	1, 2, 3, 5, 10	OK
#24, #25	Passenger	2, 6, 7, 10	OK
#26	Passenger	1, 2, 3, 10	FAIL
#27	Passenger	1, 2, 3, 5, 10	FAIL
#28, #29	Passenger	2, 5, 10	FAIL
#30, #31	Passenger	2, 5, 10	OK
#32 , # 35	Passenger	2, 5, 8, 10	OK
# 36, # 37	Passenger	2, 5, 10	OK
#38	Passenger	2, 5, 10	FAIL
#39, #43	Passenger	2, 5, 10	OK
#44, #45	Neutral	9, 10	OK
#46, #47	Neutral	10	OK

When analyzing Table V, it is observed that the majority of the implemented scenarios were approved in their test, finding 9 cases in which they were failed. The number representing the number of tests implemented and approved is 81% and 19% of tests were disapproved.

VII. Comparative analysis with related work

After the conclusion of the case study, an analysis of the method performed in the article about the automotive window system [1] and the one proposed in this work was performed.

An important fact to note is the number of test cases found in each of the studies, in the first one that has the identification of the test cases in an empirical way and manually only four test situations were found, while with the method proposed in this work where the search for test cases happened in an automated way and using software engineering concepts attending 48 test cases, in Table VI we can see the comparison of the works.

TABLE VI
ANALYSIS OF THE PATHS FOUND

Scenario	**Proposed Approach**	Santos (2015)
Number of test cases identified	48	4
Number of approved test casesr	39	4
Number of failed test cases	9	0
Number of requirements met	9	10

Comparing the results of the work, it is observable that the proposed approach identified a larger number of test cases compared to that used by Santos et al (2015) in his work. We identified 48 test cases that should be performed, while the article that was used for comparison found only 4 cases.

Another important factor is the number of failed tests, while this work identified 9 test cases with errors, the related work did not present any failed test, which may result in future problems for the developing system, since errors were found in the controller. It is important to note that even 9 cases were disapproved, 38 test cases were verified and validated. We conclude that with the number of cases found, the reliability of the designed system can be increased.

The tests performed by Santos et al (2015) covered all the requirements encountered, whereas the approach of this work covered 9 of the 10 requirements identified, requirement number 4, "The control system must operate between a voltage of 12.5V and 14, 5V ", it was not possible to be answered, this being due to a simulation we could not, in this work, verify the voltage in which the system was working. For this reason, we were unable to validate all the requirements at this stage of testing.

In addition, as another criterion for comparing the documentation, in the original work the test cases are documented in a complex way, presented greater difficulty of understanding with what should be implemented by the development team, while the use of scenarios in BDD is efficient when describing a clear and objective functionality, avoiding misunderstandings about what should be programmed in this test case, besides the scenarios written in BDD format can be used in other stages of the test procedure as validation of the features since we have specified in it must the behavior of the same.

VIII. Conclusion and future work

This paper presents a proposal for a new method to test automotive software at the functional model level in order to optimize this process. He also reports a work done in the area showing that the tests performed on him were not performed in the best way, since in this work in automatic electric window identified only four test cases while with the proposed method were identified 14 test cases that should be performed before progress to the next stage of development.

With the implementation of this test method, it was possible to identify nine errors during the initial phase of the project, which could still benefit users, such as reducing time and cost, reducing errors in future stages of the project.

The proposed method has better documentation on test cases, since the scenarios written in BDD format serve as a comprehensive documentation of what is happening in that process, so it can also be used to verify that test reports match what is waiting for this functionality. Because the scenarios are written in natural language scenarios and with simple language, it helps in removing misunderstandings within the team and facilitates the client's understanding of what is being implemented in that functionality.

As a future work, we propose that the automation of the process of implementing the scenarios in the BDD be performed for a tool that is capable of performing the simulation, thus being able to automate another stage of the automotive software testing process.

References

[1] F. R. F. M. M. D. Santos, J. H. Neme, "Rapid contro prototyping automotive software in power windowns systems."
[2] J. D. P. D. M. J. A. T. Hermans, P. Ramaekers, "Incorporation of autosar in an embedded systems development process: a case study."
[3] ISO, "Iso 24765: Systems and software engineering —vocabulary," 2010.
[4] R. S. Yadav, "Improvement in the v-model."
[5] V. Since, "Implementing a model-based design and test workflow," 2012.
[6] A. K. S. M.-M. P. T. Tulpule, A. Rezaeian, "Model based design (mbd) and hardware in the loop (hil) validation: Curriculum development."
[7] M. S. A. Popp, "Real-time co-simulation platform for electromechanical vehicle applications."
[8] B. B. H. S. W.Chaaban, M. Schwarz and J. B. Since, "A hil test bench for verification and validation purposes of model-based developed applications using simulink® and opc da technology," 2012.
[9] G. F. T. Bock, M. Maurer, "Validation of the vehicle in the loop (vil) – a milestone for the simulation of driver assistance systems."
[10] M. H. H. Shokry, "Model-based verification of embedded software."
[11] "Simulink, simulation and model-based design," 2018. [Online]. Available: http://www.mathworks.com/products/simulink/?BB=1
[12] S. D. D. A. Vidanapathirana, "Model in the loop testing of complex reactive systems."
[13] Y. C. S. Jang, H. Kim, "Manual specific testing and quality evaluation for embedded software."
[14] M. Natale and H. Zeng, "Task implementation of synchronous finite state machines," 2012.
[15] R. S. Pressman, Ed., *Software Engineering: A Practitioner's Approach*. New York: McGraw-Hill Science, 2014.
[16] J. F. Smart, Ed., *BDD in Action*. New York: Manning Publication, 2015.
[17] C. Felchar and J. H. Roquette, "all$_{dag}$$_{p}$$aths,''$ 2019.$[Online].Available : https : //gist.github.com/htmk/$

Received: 14 April 2021;
Accepted: 18 March 2022;
Published: 21 March 2022;

© 2022 by the authors. Submitted for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC-BY) license (http://creativecommons.org/licenses/by/4.0/).

Aplicação do Desenvolvimento Dirigido a Comportamento para Testes de Software Automotivo em Nível de Modelo Funcional

***Resumo:* —O *Model-based design* [MDB] é uma metodologia bem conhecida e com um alto nível de maturidade para o projeto, construção e teste de sistemas automotivos embarcados. Para as ferramentas e tecnologias disponíveis, ele permite construir funcionalidades com alta qualidade nas quais os testes precisam ser feitos para verificar se os requisitos são atendidos em todas as etapas desde o modelo funcional até a etapa de código. No estágio inicial do desenvolvimento, o projeto do controlador pode ser feito em um ambiente de simulação integrado com a planta física que nos permite testar e verificar se os requisitos são atendidos. Além disso, a complexidade do modelo pode ser aumentada e o método de teste deve ser automatizado a fim de fornecer uma cobertura de teste e, conseqüentemente, uma alta qualidade do software desenvolvido. Portanto, apresentamos a aplicabilidade do desenvolvimento orientado ao comportamento para testes de software automotivo no nível do modelo funcional em MBD. Além disso, demonstramos a eficiência do método de quatro sobre um pequeno estudo de caso de sistema de janelas de energia que permite que seja implantado em outras funcionalidades automotivas a serem controladas.**

***Palavras-chave:* —Model-based design, testes de software automotivo, sistemas incorporados, model-in-the loop, desenvolvimento orientado ao comportamento e modelo funcional.**