AUTOMATION FOR SYSTEM FUNCTIONALITY TESTING OF VEHICLE

Abstract

ECUs play a crucial role in modern automotive electronics by delivering the system's functions and the numerous features that are built right into them. The paradigm shift toward ECU technology has increased the complexity of the software because it now needs to provide functional solutions as well as safety and diagnostic services. Given this complexity, it has become very important to improve testing procedures to evaluate software behavior. To verify that CAN-based systems meet their functional and diagnostic criteria in a hardware-in-loop environment, the approach for automated software testing of CAN-based devices is presented in this work using an automation tool called ECU- TEST. The objective of this paper is to make a standard database in INCA so that all the possible ways of communication from different projects can be stored, automate the testing process with the test case packages and simplify the tester's work and save time and resources, one of our primary goals in automating the testing process is continuous testing, which is running tests repeatedly for various software and monitoring which test cases are succeeding or failing for a certain software so that we can optimize our test cases. The system's entire data flow is under control via the ECU-TEST tool. ECU-TEST is in charge of providing inputs to the device being tested, capturing the behavior associated with it, and then processing this data to provide the test results in this hardware-in-loop testing environment. This automation configuration allowed software testing of a power train unit to be completed in one hour as opposed to the four hours that were previously needed. This strategy has significantly cut down on testing time while simultaneously guaranteeing the delivery of software of higher caliber.

Keywords: ECU-TEST, Software Test Automation, Labcar, Hardware in Loop (HIL), Testbench informationsystem(TIS)

Authors

Manasa K V
PG Scholar
Department of Electronic and Communication Engineering
Nitte Meenakshi Institute of Technology
Bangalore, India
manasakvishwanath@gmail.com

Shashidhara K S
Associate Professor
Department of Electronic and Communication Engineering
Nitte Meenakshi Institute of Technology
Bangalore, India
sks.nmit@gmail.com

Kulkarni Vinayaka Ananth Rao
Technical Team Lead
Bosch Global Software Technologies
Bangalore, India
vinayaka59@gmail.com

I. INTRODUCTION

After all of the software's parts have been integrated, functional testing is done on the automotive software to ensure that it is performing as expected at the system level and in real-time. Functional testing includes evaluating the performance of in-vehicle networking protocols, diagnostic services, and functional safety elements of the product [1][2]. In a subsystem test bed, such as a hardware-in-loop system, these features are put to the test and verified. The communication between the tester device and ECU happens according to UDS (Unified Diagnostic Services) standards. UDS enables the communication between the diagnostic systems and the ECUs, enabling the systems to identify problems and, when needed, reprogram the ECUs. When a request comes in, ECU-TEST first verifies that it is valid before processing it. If the request is valid and processed appropriately, then it delivers a positive response with the desired data.

A test engineer must be present at all times during the execution of the current system for functional testing of ECU software. The test engineer sends the ECU a combination of actual and virtual signals. To analyze the success and failure criteria of the test scenario, the product's response is recorded. The functionality of the program is verified by test engineers by analyzing the recorded output and watching the physical behavior of the device under certain test conditions [3].

Due to the abundance of use cases, functional testing has grown rigorous. This emphasized how difficult it was to limit the time and effort spent performing both primary and regression testing of the software application. This involved investigating the tactics for testing automation. The following are crucial characteristics of the automated testing environment covered in this paper:

1. The system's physical signals are transmitted automatically.
2. System-wide automatic transmission of simulated signals
3. Automatic evaluation of the response provided by the product being tested
4. The execution of numerous test scenarios automatically
5. Automatic generation of test reports

The automation tool should be effective at putting these characteristics into practice. The choice of automation tools is thus one of the most important factors when developing an automation setup. Flex-Ray, Local Interconnected Network (LIN), and Controller Area Network (CAN) are a few different network protocols for in-vehicle communication [4]. The communication protocol used in the ECU being tested should be supported by the automation tool. The amount and types of automated test cases have a considerable impact on an automated system's efficiency; test cases for diagnostic communication are our core area of interest [5].

It becomes challenging to determine where a test case failed when it fails. Test cases may fail as a result of abnormal responses from the components being tested on the test bench or the ECU being tested. Therefore, it is advantageous if the test reports produced by the tool include both the test result and the component where the problem occurred.

The tool used in this project, ECU-TEST, was chosen because of its many useful features, including support for a variety of automotive communication protocols, integration

with a wide range of hardware and software tools, test management tools, the ability to prepare test cases using a graphical user interface, and the ability to present test results [6]. The execution of diagnostic tasks is supported by the ECU-TEST tool as well, which helped to increase the automation coverage. Many items employ the communication protocol known as the Controller Area Network (CAN). For distributed real-time systems, CAN is indeed a multi-master, serial bus, high-speed, high reliability, and low-cost communication protocol [7]. As a result, the automated test- bench uses the CAN protocol to interact with different setup components. This indicates that the ECU-TEST tool utilizes the CAN protocol to connect with both the ECU being tested and all other nodes. ECU-TEST needs a CAN physical layer to be connected to the CAN bus. This is accomplished by configuring ECU-TEST with CAN-appropriate hardware, such as INCA, Canalyser, and Vector Hardware. The application also creates test reports in several formats, including HTML and Excel, which permit additional customization.

II. TOOLS INTEGRATED WITH ECU-TEST FOR AUTOMATION

ECU-TEST is integrated with tools like INCA, TIS web app, TIS ASM, CI dashboard, SIM, etc. forautomating the testing process. A brief about each tool is described in this section.

1. **ECU-TEST Tool:** The major tool we employ for automation is ECU-TEST. It is a tool for automation to support

   - Development and administration of test cases and their execution
   - the generation of test documentation (reports)

For assigning demands to test cases handling defect management, a connection to test management tools is also possible. It is a utility with a Python script. It is a simple and user-friendly application that allows testers to create various test cases and evaluate them.

The following automation-related tasks are carried out by ECU-TEST:

- Test scripts must be created from the available test cases to test the functional or diagnostic requirements.
- ECU-TEST offers a framework for creating test scripts visually.
- The ECU-TEST transmits CAN messages on the bus in accordance with the procedures outlined in the test scripts to communicate with the test board and ECU.
- ECU-TEST renders a verdict for the test case by comparing the actual state of the output of the ECU with the expected state of the ECU as described in the test script.

Two sorts of test results are produced after all test cases have been fully executed:

- A test report that has been compiled and shows the results of each test case (in Excel or HTML format)
- A thorough test report for each test script detailing the results of each test phase (HTML format)

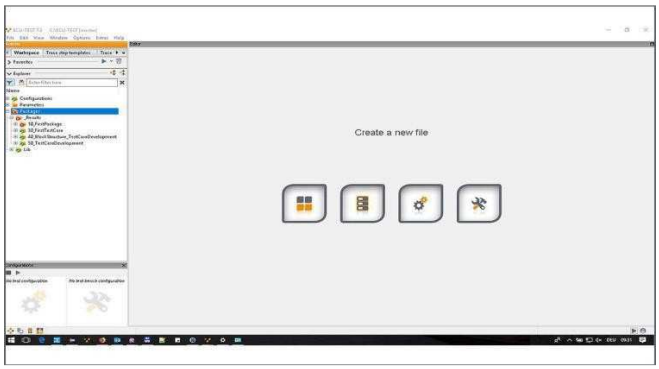

Figure 1: ECU-TEST tool

2. **Test Bench:** It is the integrated system consisting of a test ECU, battery and ignition, CAN channels, and other hardware connections that represent an actual car. It is a hardware and software integrated system where the software developed is tested against ECU (Power train).

3. **CI Dashboard:** It is a Jenkins server where the different software developed by different developers are integrated to ensure efficiency and are ready to deliver at any time. This is the platform that interconnects the TIS web app and labcar to perform the automated testing process. The main aim of CI is to prevent integration problems. An early integration reduces interface issues/errors at a later time closer to the delivery. CI dashboard also speeds up the development and it is user-friendly; the developer can fetch the software that is integrated by the previous developer and integrate his changes. So there won't be a dependency on another person. It provides a feature of chain execution that helps to perform multiple processes along with testing which is pre-required for testing. One click from the CI dashboard connects to the TIS web app, selects the artifacts that are registered, and then connects to the labcar and carries out the testing process.

4. **Integrated Calibration and Application Tool (INCA):** For the measurement, calibration, and diagnostics of the vehicle's electronic controls, the INCA software application offers an effective environment tailored specifically for the automotive industry. Numerous ECU, bus, and measurement device interfaces are integrated by INCA. With INCA, measurements may be taken and settings can be modified while the simulation is running. INCA is used as a tool to flash the software to the ECU and to choose the appropriate ports for flashing.

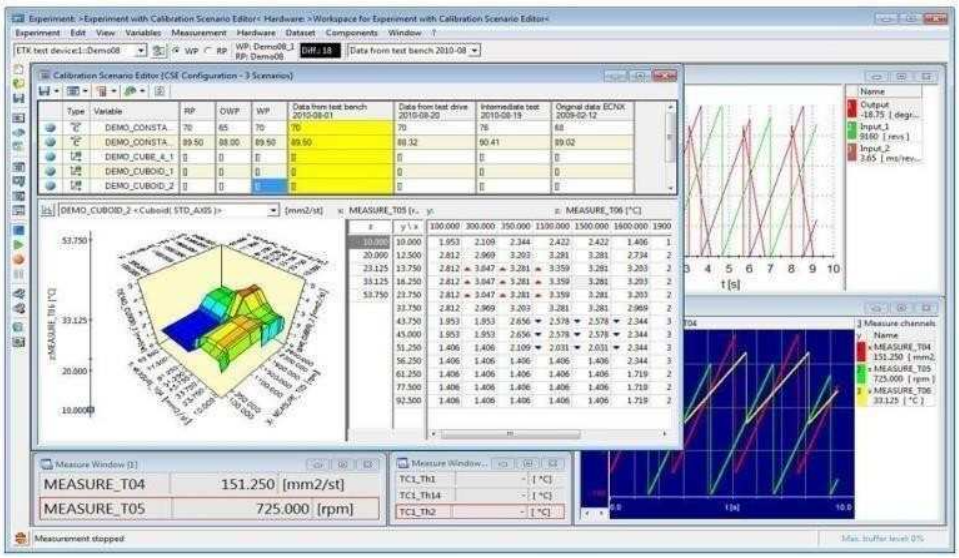

Figure 2: Experiments in INCA Tool.

5. **Test bench Information System (TIS):** TIS web app is an application where the information about the labcar or the test bench that is used for testing is configured. It is a Jenkins server that connects to the CI dashboard which is again a Jenkins server that helps in continuous testing. Different artifacts for software, labcar, and ECU-test are configured and are registered in TIS so that the CI dashboard selects the artifacts from TIS and carries out testing in a particular labcar that is selected for testing. The hardware that is connected to the particular ECU of the labcar is also chosen. The test is triggered from TIS as well to ensure the automation process is carried out correctly and then connects the artifacts to the CI dashboard.

6. **System Identification Manager (SIM):** SIM is used to lock the test bench (labcar) so that once the testing process is triggered in the CI dashboard, other users should not log in to the test bench and interrupt the testing process. The button is only enabled if the TIS- Web-App is opened on the test bench itself to prevent the 'remote' test bench lock. TIS web is designed in such a way that the test execution is carried out only if the test bench is locked.

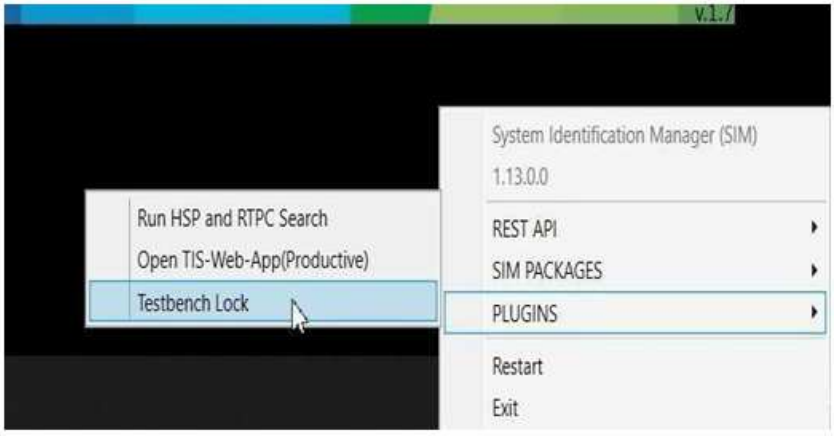

Figure 3: SIM

7. **TIS Artifact Container Manager (TIS ASM):** A container of any kind designed to store and manage code or packages is referred to as an artifacts container. There are several other types of containers in TIS, including XCUSW, vVeh LCO, test ECU-TEST, and others. This graphical user interface is used to create the artifact containers for the TIS web app. To automate the testing process three containers are created one for software, one for ECU-test, and one for labcar model.

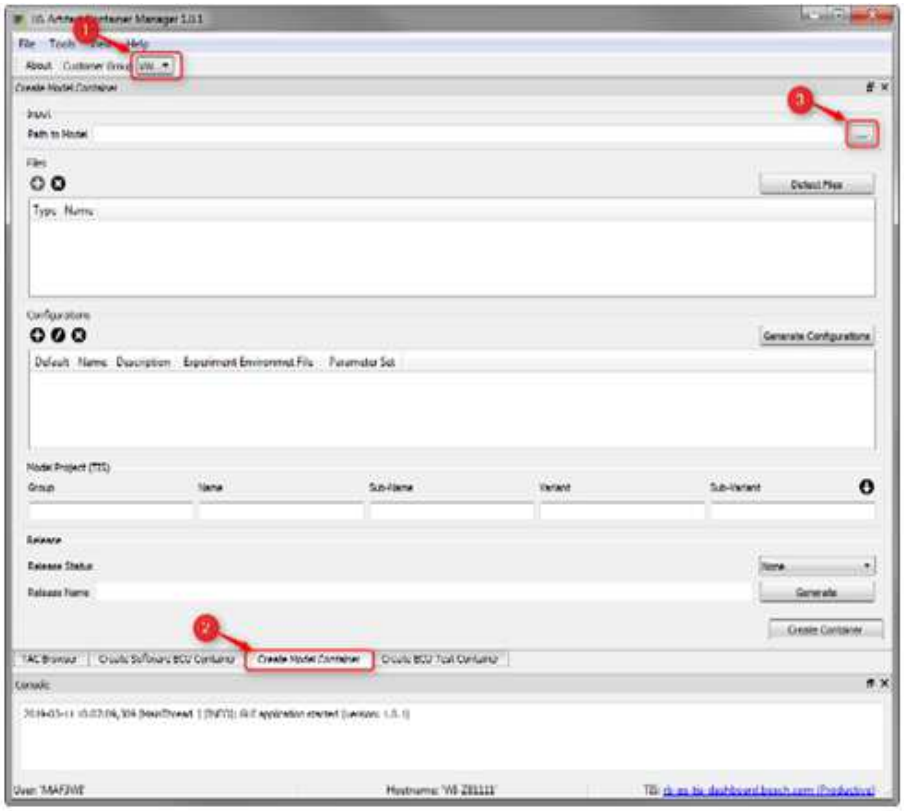

Figure 4: The Working Window of TIS ASM.

III. STEPS IN MANUAL TESTING

The manual testing process requires a lot of time and technical know-how and effort to teach a new person about the testing process. Below are the steps that are followed in manual testing:

1. Book the labcar prior to use to avoid overlapping between two person's slots.
2. Login to the labcar
3. Select the labcar project specific to a specific project
4. Turn on the battery and ignition
5. Download the software that should be tested from the CI dashboard
6. Open INCA and load the software. Then do a data merge (data merge is a process of merging the differences between the developed software and the software which contains the threshold values of the variables).
7. Select a proper CAN channel and the flashing ports that are configured in a particular labcar

8. Flash the software to the ECU.
9. After the flashing is done, INCA should be closed because it should be connected through ECU-TEST.
10. Open ECU-test and load the software in TCF and keep the state of tools that need to be connected through ECU-test as always in TBC.
11. Open the test cases in ECU-test which are pre-configured and run the test cases.
12. Collect the reports and close the test bench.
13. The above-mentioned steps are to be done manually and it takes about one hour to set up the conditions for testing and extra time for execution of test cases. Automating the testing process saves 75% of the time that is consumed in manual testing.

IV. SYSTEM OVERVIEW

The System Overview gives a quick explanation of how the connected tools and integrated automation test-bench operate. The CAN communication protocol is used to implement the automation.

1. **Process flowchart:** Process flowchart tells about the flow in which automation takes place and gives a brief about the stepsstarting from triggering build to obtaining reports.

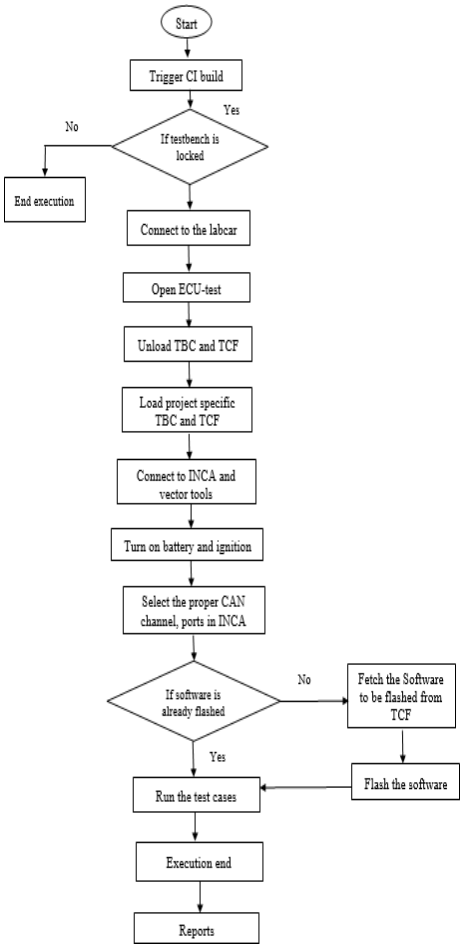

Figure 5: Process flowchart of ECU-TEST automation package

- The first step is to trigger the build in the CI dashboard. Once the build is triggered then the following steps take place automatically
- If the test bench is locked then move to the next step and if the test bench is not locked the execution is ended. Connect to labcar
- Open ECU-TEST of the configured version
- Unload TBC (Test bench configuration) and TCF (Test configuration) if any other configuration is already present.
- Load the project-specific TBC and TCF
- Connect INCA, vector hardware, canalyser from ECU-TEST
- Select the proper ports and CAN channel for flashing
- If the software is already flashed then move to the next step, if not flash the software to be tested
- After the software is flashed execute the test cases that should be tested
- The reports are generated automatically.

2. **System block diagram:** The system block diagram gives information about how the tools are connected to each other to facilitate automation and provides the details of t h e working of the test bench setup for testing.

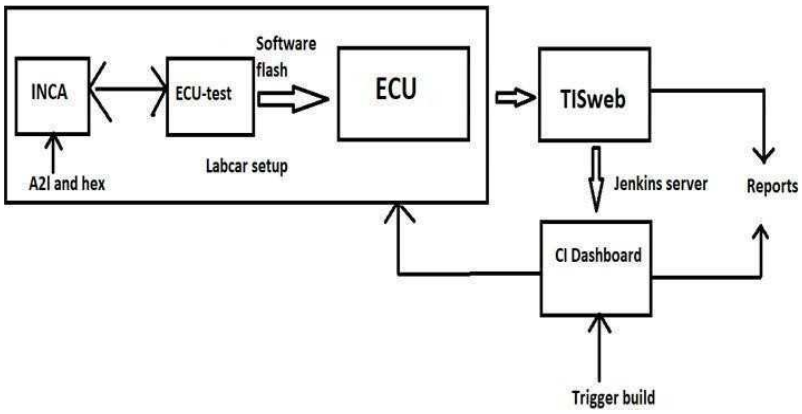

Figure 6: System Block Diagram.

The artifacts (software, testbench, and ECU-TEST) that are registered in the TIS web app are integrated into the CI dashboard. So when the build is triggered from the CI dashboard it connects to SIM that opens the TIS web app and gets the information of the software, testbench, and ECU- TEST and connects to the testbench. Once the testbench is accessed the ECU-TEST tool is opened and starts executing the test cases that are developed setting up pre-conditions for lab setup and flashing of software. The automated test cases are developed in such a way that it executes all the steps that are done in the manual process like turning on battery and ignition, setting up proper communication between INCA and ECU-test, flashing the software to ECU, executing the test cases that should be tested and finally obtaining the reports. All these steps are executed in a single click from the CI dashboard. The reports can be obtained both from the CI dashboard and the TIS web app. Following attributes of a CAN message are requested with a transmitter CAN id 0x6a8

1. Message length
2. Service identifier
3. Name of the data identifier

The response to the request is from the CAN id 0x688 with the value from the ECU. In manual testing, these signals are sent manually and require a lot of time and technical knowledge whereas in automation we use the ECU-TEST tool to send these signals to ECU that are pre-configured in the test cases. Eliminating human involvement in sending CAN signals to the ECU is one of the tactics used while automating the testing process.

3. **ECU-TEST – Creating an Environment for Automation:** Setting up a test environment in ECU-TEST is necessary for automatically running a test case. ECU-TEST needs both a Test Bench Setup (TBC) and a Test Configuration for a configuration to be successful (TCF). The tools that are directly connected to TBC are configured. In our case, the tools connected to ECU-TEST are INCA, vector hardware, battery, and ignition units. It controls the instruments that have direct access to the test variables. A specific test object, also known as a device under test or ECU, is configured using TCF.

   Using the ECU-TEST editor, the test cases that were created using the software requirement document are transformed into test scripts. The package is an editor made available by ECU- TEST where the test cases can be written. By dragging and dropping test objects into the package, the graphical user interface allows for the creation of test stages.

   One test case or several test cases may be included in a package. It is possible to compile and automatically run many packages in a serial fashion. After running the test case, a thorough test report in HTML format is produced for each test step. Results from many test cases can be downloaded in MS Excel and HTML formats.

4. **Steps in a Test Case for Setting Up Labcar Environment for Testing:** The steps involved in setting up the labcar environment are written in the ECU-TEST package

   - After opening ECU-TEST, if any other configurations are loaded they should be unloaded.
   - Load the TBC and TCF files which are configured according to test bench requirements.
   - Set the global constants which contain information about the project and labcar settings.
   - Collect the information about the INCA version, user, user file path, user patch file, etc.
   - Open current test configurations and set proper global constants value.
   - If the selected tool is labcar then continue the steps in the flashing package.
   - Flash the software.

5. **Steps Involved in a Test case for Flashing the Software to ECU through INCA:** In the ECU-TEST package, the test steps for the test case are visually programmed. No human involvement is required during the entire test case's execution. Every test case is scripted in a different package and run sequentially. Steps in flashing the software to ECU are listed below

   - Select the tool from which the flashing has to be done, in our case it is INCA.
   - If INCA tool status is 1 then initialize flashing.

- Set battery and ignition on
- Check software on the device, if the selected software is already flashed then exit, if notcontinue to thenext step.
- Get the path of the software (a2l and hex file)
- Adaption of prof command which contains the information of the ports that are connected, profdirectory, and the configuration of INCA.
- Start flashing
- Restart ECU after flashing
- Turn off battery and ignition

V. STATISTICS OF REPORT ON TEST AUTOMATION

The figure shows the result of 10 test cases executed and also the number of test cases passed, failed, errors, etc. The time taken for manual testing of 757 test cases is 4 hours and through automation, the time consumed is 1 hour. Thus automation reduces the testing time by 75%.

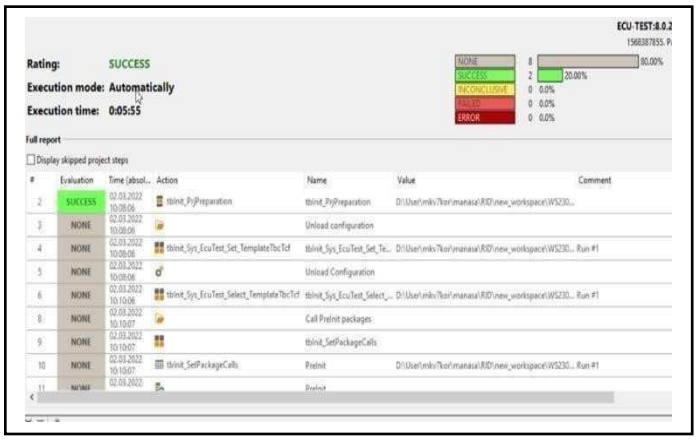

Figure 7: Statistics of the report of one of the test cases executed with automation

VI. CONCLUSION AND FUTURE SCOPE

In this paper we have developed the ECU-TEST packages for automation of the Testbench process and a proper INCA database is set up so that there is no need to set up INCA every time testing is done. Labcar, TIS, and CI dashboard tools are integrated with each other which achieves the aim of automation that is one click from the CI dashboard carries out the testing process, and reports are generated that gives the information on the number of test cases executed and the success and failure rate. The automation process reduced the time requirements of test engineers for testing. Automation enables continuous testing and regression testing which increases the efficiency of testing and helps to know the errors and strengthen the test cases.

1. **Future Scope:** With the increase in demand for new features in automobiles, there is a demand for testing of the features. Automation of the testing process eases the process of testing. As a future extension the feature of the CI dashboard which enables chain action that includes ehooks and INCA data merge is utilized and implemented for the full fledge automation. Ehooking is done to change the values of the variables in the software

and test the functionality. Data merge is required to merge the software to be tested with the software that contains the threshold value of variables.

VII. ACKNOWLEDGMENT

I wholeheartedly thank **Dr. Shashidhara K S**, Associate Professor, Department of ECE, NMIT, and **Mr. Kulakarni Vinayaka Ananth Rao**, Technical team lead (BGSW/EEP) Bosch Global Software Technologies, Bengaluru for their guidance, support, and encouragement during the development of this work.

VIII. REFERENCES

[1] V. Ili, S. Popi and M. Kovači, "Data flow in automated testing of the complex automotive electronic control units," 2016 Zooming Innovation in Consumer Electronics International Conference (ZINC), Novi Sad, Serbia, 2016, pp. 1-3, doi: 10.1109/ZINC.2016.7513639.

[2] V. Garousi and F. Elberzhager, "Test Automation: Not Just for Test Execution," in IEEE Software, vol. 34, no. 2, pp. 90-96, Mar.-Apr. 2017, doi: 10.1109/MS.2017.34.

[3] Malik, Hafiz & Avatefipour, Omid & Hafeez, Azeem & Raj, Prudhvi. (2017). Comparative Study of CAN-Bus and FlexRay Protocols for In- Vehicle Communication. 10.4271/2017-01-0017.

[4] R. Boot, J. Richert, H. Schutte and A. Rukgauer, "Automated test of ECUs in a hardware-in-the-loop simulation environment," Proceedings of the 1999 IEEE International Symposium on Computer Aided Control System Design (Cat. No.99TH8404), Kohala Coast, HI, USA, 1999, pp. 587-594, doi: 10.1109/CACSD.1999.808713.

[5] Tracetronic, "ECU-TEST product Datasheet" URL: https://www.tracetronic.de/produkte/ecu test/datenblatt_ecu- test_en.pdf

[6] Bosch. "CAN Specification version 2.0". Robert Bosch GmbH, 1991

[7] F. Zhou, S. Li and X. Hou, "Development method of simulation and test system for vehicle body CAN bus based on CANoe," 2008 7th World Congress on Intelligent Control and Automation, Chongqing, China, 2008, pp. 7515-7519, doi: 10.1109/WCICA.2008.4594092.

[8] Vaijayanti Gajul, Jitendra Kumar D. MishraSarika Tavhare, "Automation solution for software testing of CAN-based ECU" 2021 Fourth International Conference on Electrical, Computer and Communication Technologies (ICECCT) | 978-1-6654-1480-7/21/$31.00

[9] ©2021 IEEE | DOI: 10.1109/ICECCT52121.2021.9616902

[10] Shruthi T S , K H Naz Mufeeda, "Using VT System for Automated Testing of ECU ", IOSR Journal of Computer Engineering (IOSR-

[11] JCE) e-ISSN: 2278-0661,p-ISSN: 2278-8727, Volume 18, Issue 3,Ver. V , PP 28-31, May-Jun. 2016

[12] Robert I. Davis. "Impact case study: Guaranteeing the real-timeperformance of in-vehicle networks", Technical report, UniversityofYork, 2015. URL: https://www-users.cs.york.ac.uk/~robdavis/papers/ImpactCaseStudyVolcano.pdf

[13] Freescale Semiconductor, "S12 MagniV Microcontrollers", MC9S12ZVC-Family Reference Manual and Datasheet, 8th February2016

[14] NXP Semiconductors Application Note, "Hardware Design Guidelines for S12ZVC MagniV Mixed-Signal MCUs for CAN Applications",Document Number: AN4867, Rev. 3, July 2016

[15] Freescale Semiconductors Application Note, "Using MSCAN on the MagniV Family", Document Number: AN4975, Rev. 0, Aug. 2014

[16] P. Peti, A. Timmerberg, T. Pfeffer, S. Muller, C. Ratz and V. Informatik, "A quantitative study on automatic validation of the diagnostic services of Electronic Control Units," 2008 IEEE International Conference on Emerging Technologies and Factory Automation, Hamburg, Germany, 2008, pp. 799-808, doi: 10.1109/ETFA.2008.4638490.

[17] Warren, J., Adams, J. and Molle, H., 2011. Arduino robotics. [New York, NY]: Apress.

[18] Y. Erol, H. Balik, S. Inal, and D. Karabulat, “Safe and Secure PIC Based Remote Control Application for Intelligent Home”. International Journal of Computer Science and Network Security, Vol.7, No.5. 2007.

[19] R. Boot, J. Richert, H. Schutte and A. Rukgauer, "Automated test ofECUs in a hardware-in-the-loop simulation environment", *Proceedings of the 2020 IEEE International Symposiumon Computer Aided Control System Design*

[20] *(Cat. No.99TH8404)*, pp.587-594, 2020.

[21] F. Zhou, S. Li and X. Hou, "Development method of simulation and test system for vehicle body CAN bus based on CANoe", *2021 7th World Congress on Intelligent Control and Automation*, pp. 7515- 7519, 2021.

[22] P. Peti, A. Timmerberg, T. Pfeffer, S. Muller, C. Ratz and V. Informatik, "A quantitative study on automatic validation of the diagnostic services of Electronic Control Units", *2021 IEEE International Conference on Emerging Technologies and Factory Automation*, pp. 799-808, 2021

[23] Hafiz Malik, Omid Avatefipour, Azeem Hafeez and Prudhvi Raj, Comparative Study of CAN-Bus and FlexRay Protocols for In- Vehicle Communication, 2020.

