Automated Continuous Verification & Validation for Automobile Software

Vinodhini Vijayaraghavan, Jagadeeswara Vijayaraghavan Angamuthu, Saravanakumar C Shanmugam

Robert Bosch Engineering and Business Solutions Private Limited

ABSTRACT

As many automobile industries are moving towards Agile, Scrum, DevOps, and other efficient process, Continuous Delivery is a key for all the above process. Continuous Delivery is possible by Continuous Integration and Continuous Testing (Verification & Validation). Automation of Verification and Validation is an integral part of Continuous Integration and is a backbone of Agile, Scrum and DevOps. This paper presents the approach for automated Verification & Validation (V&V) for different test strategies (MiL, SiL, PiL, HiL testing, static testing) of Automobile Software. This approach helps developers to increase regression test set with each release and be ASPICE compliant.

INTRODUCTION

According to Moore's Law, the number of transistors per square inch on integrated circuits had doubled every year. This resulted in increasing amount of complexity of the embedded software in Automobile Industry. Manually testing the embedded code will no longer be efficient and consistent. Also, manual testing consumes more time and effort of the developers. This would also result in delay in product delivery. Manual testing is also error prone and could easily introduce bugs. In safety-critical systems like automobile systems, bugs could result in loss of life, significant property damage, or damage to the environment.

The predominant and continually increasing number of innovative vehicle functions relating to the protection of the environment, safety, security, economic efficiency and user-friendliness can only be achieved by the introduction of complex and highly-networked software systems. To avoid life or property damage, **ASPICE** (Automotive SPICE) has published "Process Assessment Model" and "Process Reference Model", which was developed and tailored considering specific needs of automotive industry [1]. The Automotive SPICE process assessment model and process reference model is conformant with the **ISO/IEC 33004**, and can be used as the basis for conducting an assessment of process capability [1].

ASPICE STANDARD

According to ASPICE, Section SWE.4 Software Unit Verification (SUV- Figure 1) and Section SWE.5 Software Integration and Integration Test (SIT- Figure 2), gives details on embedded software unit regression testing process and software integration testing process. As per above process methods, one of the Base Practice is to "Develop a **strategy for verification of the software units including regression strategy for re-verification** if a software unit is changed. The verification strategy shall define how to provide evidence for compliance of the software units with the software detailed design and with the non-functional requirements" [1]. To ease verification, reverification of software continuously, automation plays a vital role. Hence, there is a strong demand in the market to provide automated solutions for the verification and validation process.

Our solution is to create a suitable environment with required tools to automate the SUV, SIT process using a Continuous Integration (CI) Server. A CI server is server which is capable to integrate, test and deliver working and stable software continuously.

4.4.4. SWE.4 Software Unit Verification

Process ID	**SWE.4**
Process name	**Software Unit Verification**
Process purpose	The purpose of the Software Unit Verification Process is to verify software units to provide evidence for compliance of the software units with the software detailed design and with the non-functional software requirements.
Process outcomes	As a result of successful implementation of this process: 1) a software unit verification strategy including regression strategy is developed to verify the software units; 2) criteria for software unit verification are developed according to the software unit verification strategy that are suitable to provide evidence for compliance of the software units with the software detailed design and with the non-functional software requirements; 3) software units are verified according to the software unit verification strategy and the defined criteria for software unit verification and the results are recorded; 4) consistency and bidirectional traceability are established between software units, criteria for verification and verification results; and 5) results of the unit verification are summarized and communicated to all affected parties.
Base practices	**SWE.4.BP1: Develop software unit verification strategy including regression strategy.** Develop a strategy for verification of the software units including regression strategy for re-verification if a software unit is

Figure 1 ASPICE SWE.4 Software Unit Verification [1]

4.4.5. SWE.5 Software Integration and Integration Test

Process ID	**SWE.5**
Process name	**Software Integration and Integration Test**
Process purpose	The purpose of the Software Integration and Integration Test Process is to integrate the software units into larger software items up to a complete integrated software consistent with the software architectural design and to ensure that the software items are tested to provide evidence for compliance of the integrated software items with the software architectural design, including the interfaces between the software units and between the software items.

Figure 2 ASPICE SWE.5 Software Integration and Integration Test [1]

PREVIOUS PRACTICES

The past practices of different testing techniques are briefly described below:

- Model-In-Loop Testing- (MiL) – Functional developers create models that adhere to the requirements using Model based development tools like MATLAB (MATrix LABoratory) or ASCET in a PC. Then the required test cases are written and tested for the model in Dynamic Testing tools like TPT (Time Partition Testing). This does not require any Hardware for testing.
- Software-In-Loop Testing- (SiL) – Functional developers would auto generate the embedded code from models or manually write the embedded code. This code is compiled for a targeted Electronic Control Unit (ECU) architecture. Then the required test cases are written and tested for the code in Dynamic Testing tools like TPT (Time Partition Testing). Also, this does not require any Hardware for testing.
- Processor-In-Loop Testing- (PiL) – Functional developers would use the "Executable Linkable File" which is generated for a target ECU, is flashed in ECU. The test simulation are done using in Dynamic Testing tools like TPT (Time Partition Testing). Here, the object code is tested in a real target hardware or an instruction set simulator.
- Hardware-In-Loop Testing- (HiL) - this is a real time simulation of the hardware where the final object code is tested in a Lab Car using test automation tools like ECU-Test, Lab Car PT and INCA .
- Static Testing- The code is tested for code quality, complexity during development of embedded code.

AUTOMATION OF VERIFICATION AND VALIDATION

Automating Verification and Validation steps of an ECU Software is one of the best practice, which are critical in Agile, Scrum, and DevOps development model. The demands of quality code in automotive market drive a greater need for tools and practices that enable automation. SUV, SIT are two major test process that we automate using CI server.

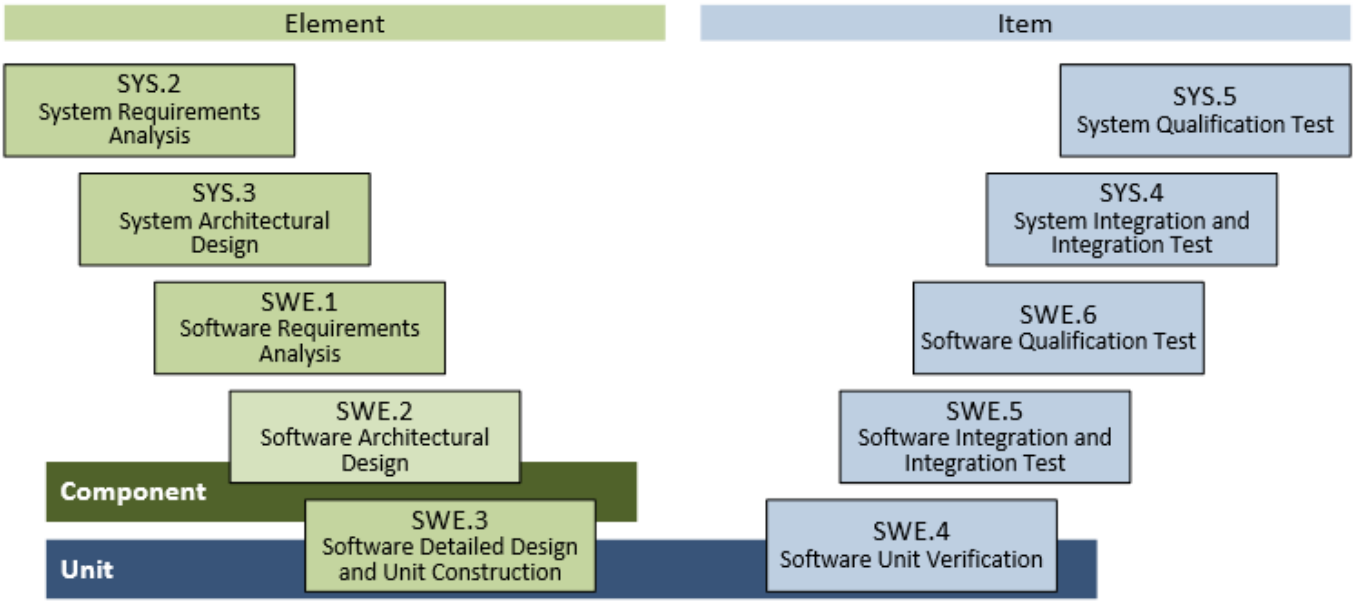

Figure 3 Element, Item, Unit and Component in a V model [1]

AUTOMATION OF SOFTWARE UNIT VERIFICATION (SUV)

The unit is a smallest piece of testable embedded code as shown in Figure 3. This smallest unit could be verified and re-verified with evidence of compilation in a CI Server. Regression testing of the units are also automated in CI server. The SUV must be executed whenever there is a change in either source code or the test cases. For automation, the test cases and the functional component together is considered to be single unit [7]. We automate the SUV for MiL, SiL, PiL, HiL environments.

SUV = Black Box Testing (requirement -based, dynamic testing) + measuring Model/Code Coverage

Figure 4 Definition of SUV [8]

Features of automated SUV [7]:

- It is mainly PC-based testing
- It is related to ASPICE - Software Unit Verification SWE.4 [1]
- It is a regression test – not only delta test
- It is a test which includes a code coverage measurement
    - statement coverage
    - branch/decision coverage

Prerequisites:

- Related artifacts must be present in Source Code Management (SCM)
    - MiL – MATLAB Model or ASCET Model with supporting files
    - SiL – Project Settings file to generate DLL using ECU Code Generator
    - PiL – PLS UDE Debugger workspace file
    - HiL – ECU Test file and other supported files
- TPT test cases (MiL, SiL, PiL) and ECU-Test file (HiL) must be present in Source Code Management (SCM)
- Jenkins machine with required Jenkins plugin
    - MiL – MATLAB Plugin or ASCET Plugin, TPT Plugin
    - SiL – ECU Code Generator Plugin, TPT Plugin
    - PiL – UDE Plugin, TPT Plugin
    - HiL – ECUTest Plugin, Lab Setup

AUTOMATION OF SUV IN MIL TESTING ENVIRONMENT USING MATLAB/SIMULINK AND TPT- The block diagram is shown in Figure 5

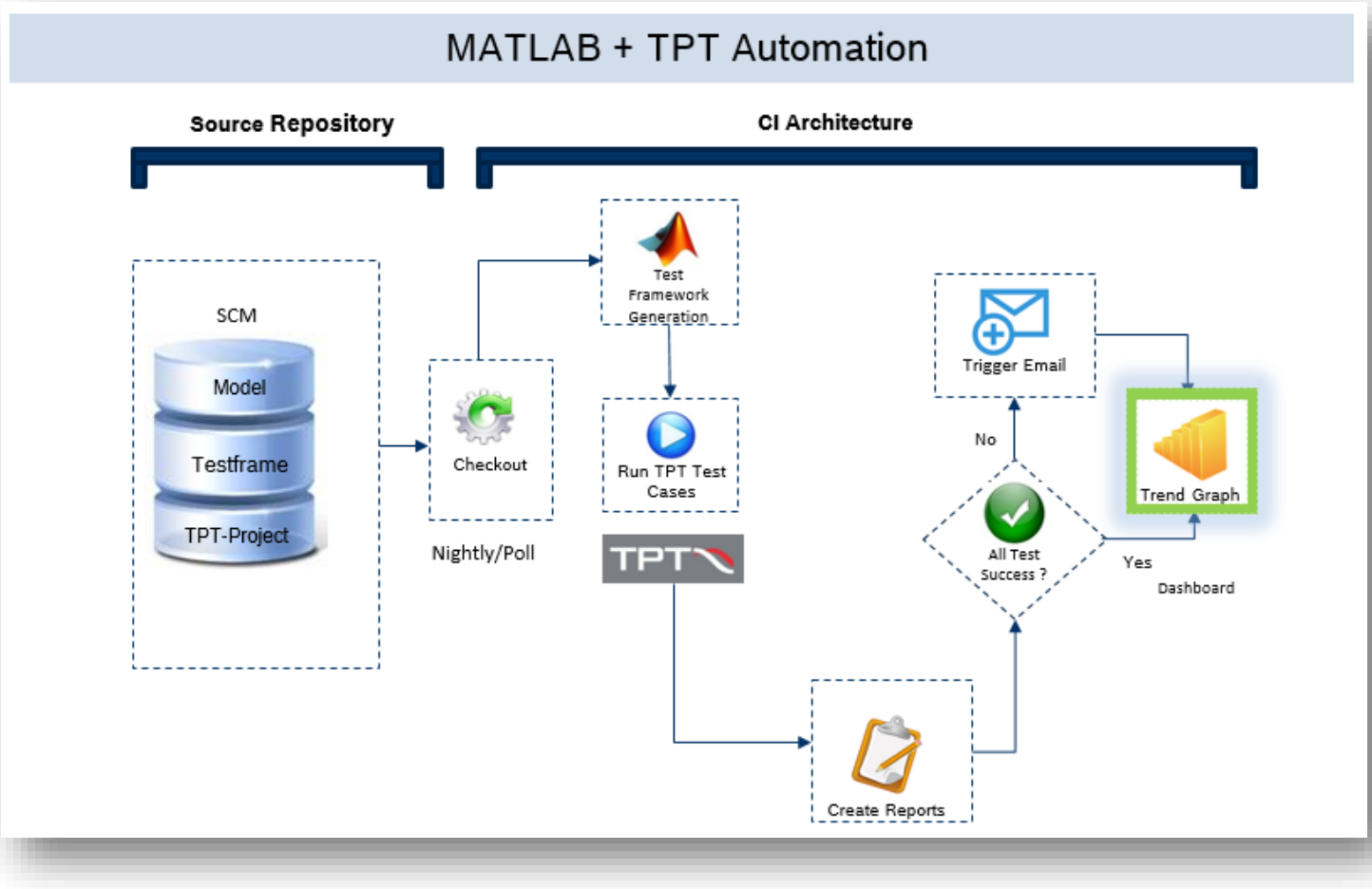

Figure 5 Automation of MiL (MATLAB+TPT) [2], [4]

Prerequisites:

- MATLAB/Simulink models must be present in Source Code Management (SCM)
- TPT test cases must be present in SCM
- CI server with MATLAB plugin and TPT plugin

Required Model and Test cases are checked out from the SCM system whenever there is a change in the SCM for the given component. Using the checked out model, test framework is automatically generated in the CI server with Matlab tool. On successful creation of model framework, test cases are executed which will provide the test artifacts like test reports, coverage reports etc. It also captures the complete trend of the test execution and maintains the history of the test progress. Mailing can also be added to share the feedback and reports on the complete test.

AUTOMATION OF SUV IN MIL TESTING ENVIRONMENT USING ASCET AND TPT- The block diagram of MIL is shown in Figure 6

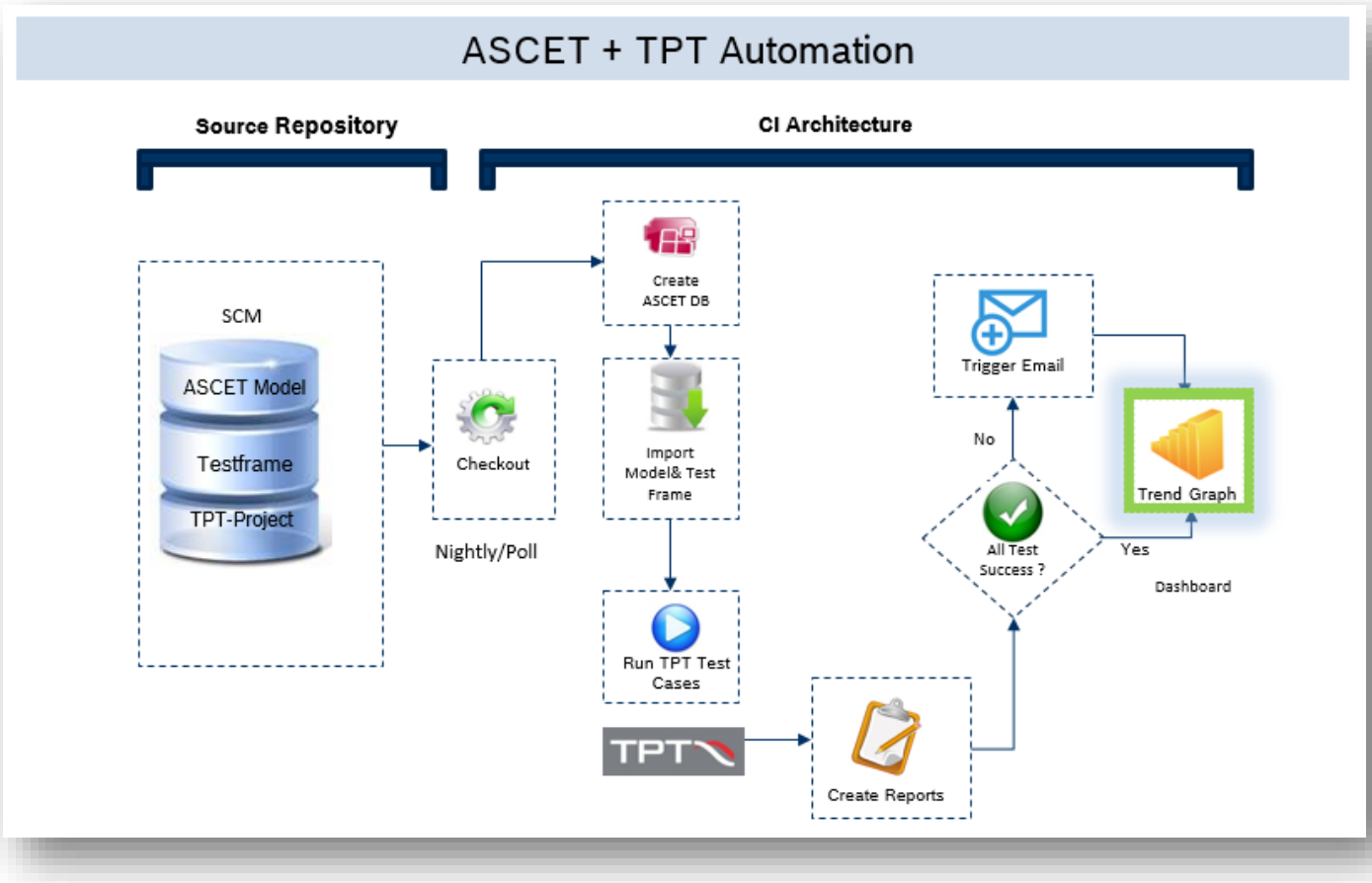

Figure 6 Automation of MiL (ASCET + TPT) [5] [2]

Prerequisites:

- ASCET models must be present in Source Code Management (SCM)
- TPT test cases must be present in SCM
- CI server with ASCET plugin and TPT plugin

Required ASCET Model, test frame and Test cases are checked out from the SCM system whenever there is a change in the SCM for the given component. ASCET tool is executed in the CI server to create DB and generate the model framework automatically. On successful creation of model framework, test cases are executed which will provide the test artifacts like test reports, coverage reports etc. It also captures the complete trend of the test execution and maintains the history of the test progress. Mailing can also be added to share the feedback and reports on the complete test.

AUTOMATION OF SUV IN SIL TESTING ENVIRONMENT USING DLL AND TPT - The block diagram of SIL is shown in Figure 7.

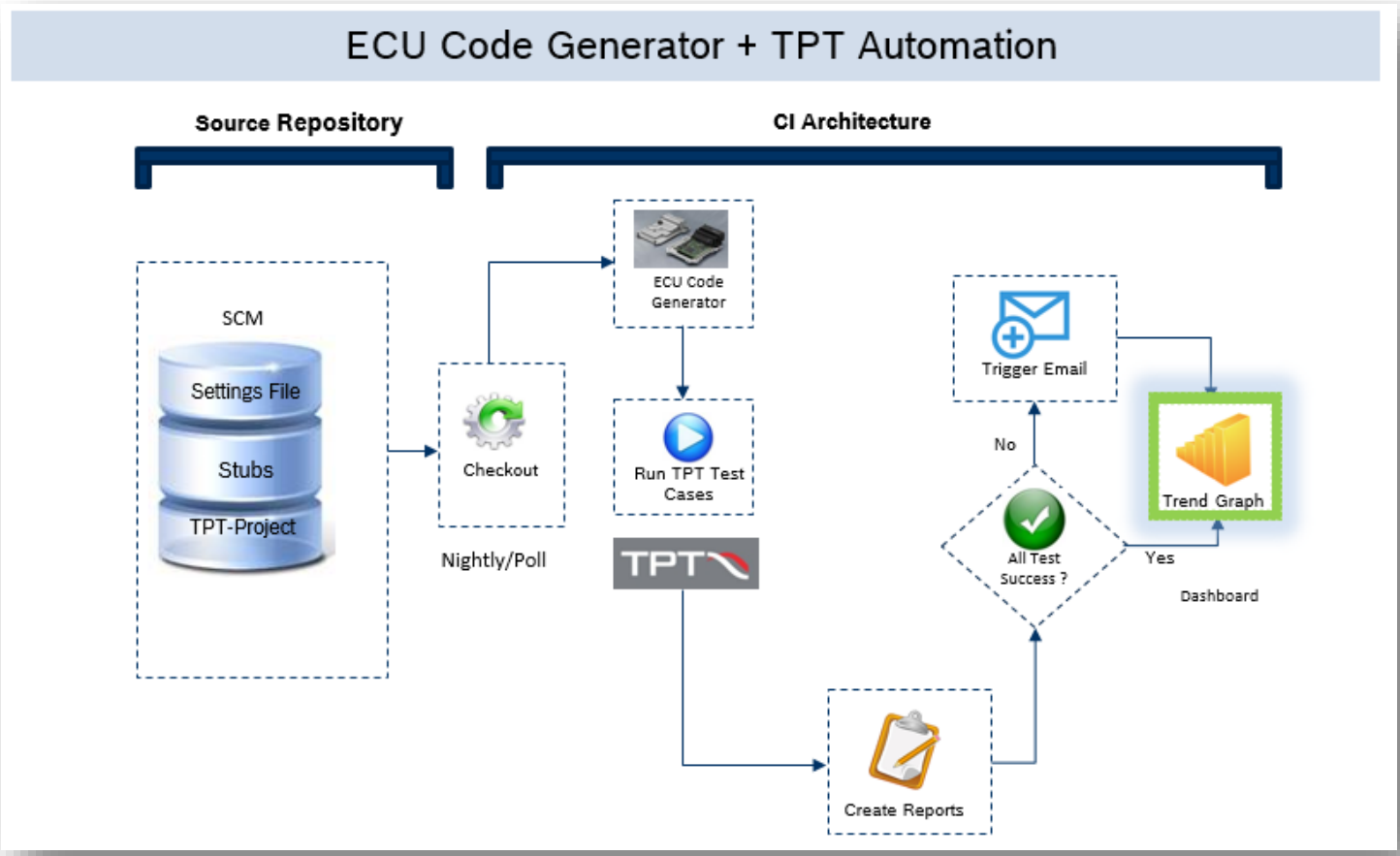

Figure 7 Automation of SiL (ECU Code Generator + TPT) [2]

Prerequisites:

- DLL settings must be present in Source Code Management (SCM)
- TPT test cases must be present in SCM
- CI server with DLL generator plugin and TPT plugin

SiL automation is achieved by generating the DLL and executing the test cases in the CI server. Required functional component and Test cases are checked out from the SCM system whenever there is a change in the SCM for the given component. DLL generator will be executed using these checked out items in the CI server automatically. On successful creation of DLL, test cases are executed which will provide the test artifacts like test reports, coverage reports etc. It also captures the complete trend of the test execution and maintains the history of the test progress. Mailing can also be added to share the feedback and reports on the complete test.

AUTOMATION OF SUV IN PIL TESTING ENVIRONMENT USING UDE Debugger AND TPT- The block diagram of PIL is shown in Figure 8.

Prerequisites:

- PLS UDE workspace must be present in Source Code Management (SCM)
- TPT test cases must be present in SCM
- CI server with PLS UDE plugin and TPT plugin

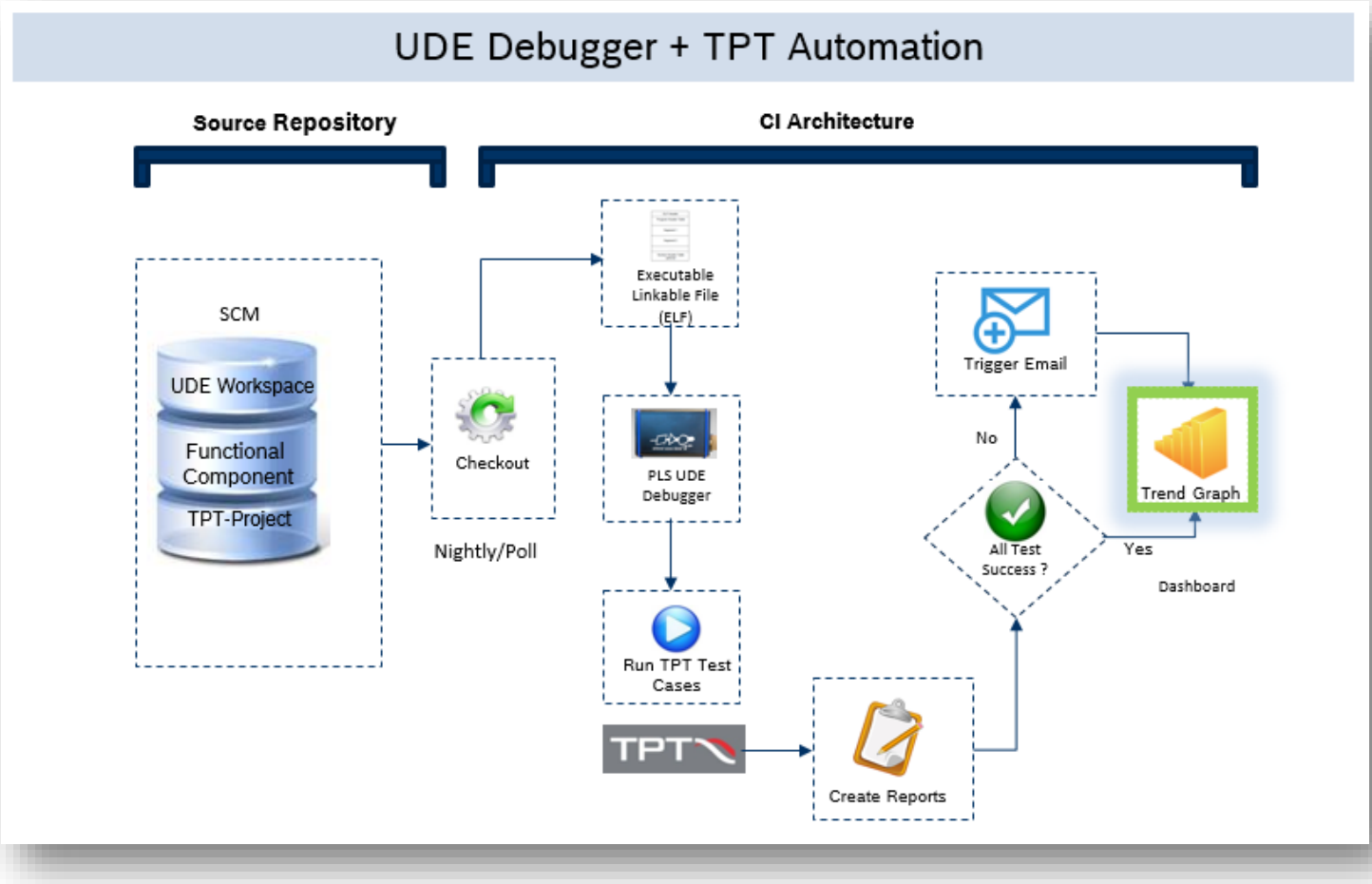

Figure 8 Automation of PiL (UDE + TPT) [6] [2]

In a PIL simulation, the generated code runs on the target hardware or on an evaluation board. Memory consumption and runtime information are transferred to the PC. SUV in PiL Testing can be achieved by automating the UDE Debugger and TPT in Jenkins.

Fresh copy of the Functional component, UDE workspace and test cases are checked out from the SCM whenever there is a change. ELF file is generated automatically and loaded to PLS UDE debugger. Test cases are then executed which will provide the test artifacts like test reports, coverage reports etc. It also captures the complete trend of the test execution and maintains the history of the test progress. Mailing can also be added to share the feedback and reports on the complete test.

AUTOMATION OF SUV IN HIL TESTING ENVIRONMENT USING LAB CAR PT and ECU-TEST- The block diagram of HIL is shown in Figure 9.

HiL testing systems contribute to quality assurance during the early phases of ECU development. They facilitate the testing of the functions or diagnostic behavior of ECUs in the laboratory [3]. SUV in HiL Environment can be achieved by automating the LAB Car PT, INCA and ECU-Test in Jenkins. LAB Car PT generates and measures signals for engine ECU testing [3].

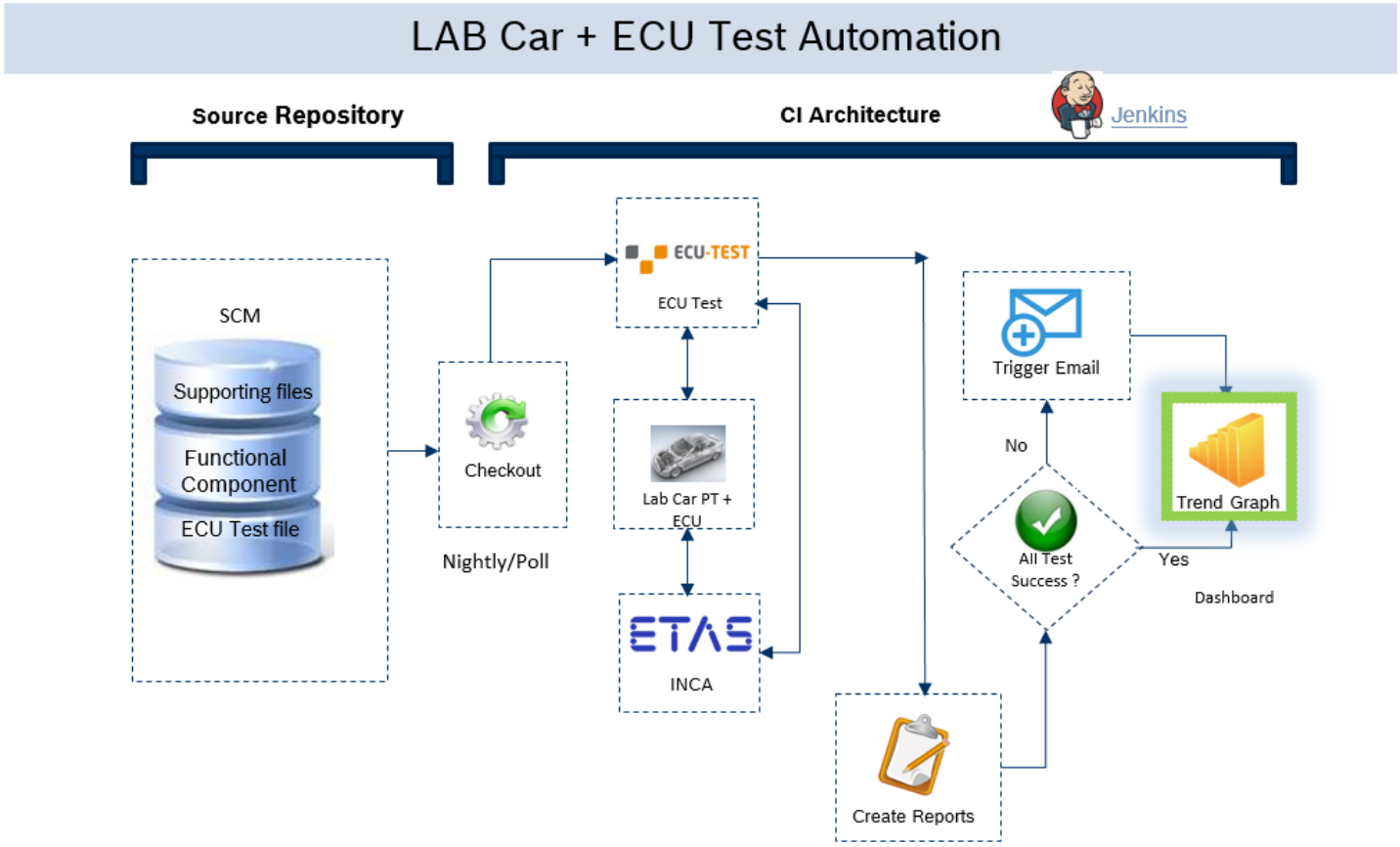

Figure 9 Automation of HiL (LCA, INCA, ECU-Test) [8] [3]

AUTOMATION OF SOFTWARE INTERGRATION TEST (SIT)

Software Integration Test validates the correct implementation of the functional requirements defined on System Function (SF) level. It may consist of one or more Functional Component and optionally by some Hardware [7]. It complies with ASPICE Standard, which recommends regression testing of the System elements driven by customer requirements [7]. The SIT must be executed whenever there is a change in either the source code or the test cases [7]. SIT for MiL, SiL, PiL, HiL environments could be automated in a similar manner as of SUV.

Features of automated SIT:

- It is related to ASPICE – Software Integration and Integration Test SWE.5 [1]
- It is also a regression test – not only delta test, so that the change in one functional component has not introduced bugs in other functional component of the system function [7].
- It validates the correctness of the new functional requirements and other unchanged functional requirements [7].

STATIC TESTING

Static testing is a type of testing wherein the quality of deliverable is evaluated without executing the code [Figure 10]. In ECU development, validating the containers using the MISRA standard has become a mandatory practice across the automobile industry. As it is well-known, that evaluating the embedded code using the MISRA standard facilitates the safety, security, portability and reliability of the software.

Features of automated static testing:

- Generates report of code quality
- Captures the trend of the code quality issues
- Provides quicker response thereby reducing the risks of schedule and cost overrun

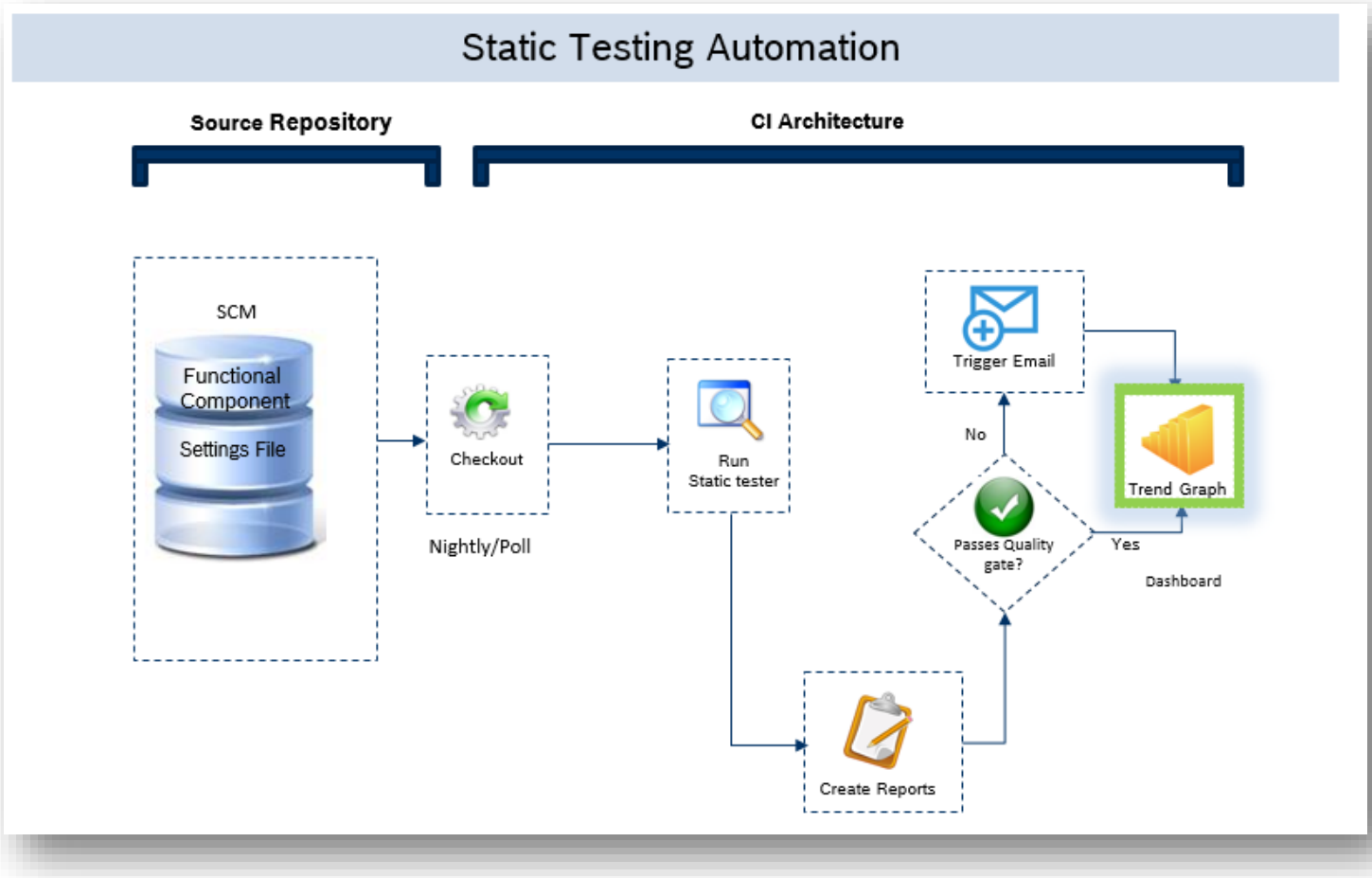

Figure 10 Automation of Static Testing

CONCLUSION

There are numerous advantages when we automate Verification and Validation process (SUV, SIT). Due to the feature of parallel execution in CI server, the SUV and SIT could run in parallel for different functional component and system components. This also gives more transparency in the build and test process. The advantages is shown in Figure 11. By automating, we get faster regression test results, quicker & continuous delivery, and improvement in productivity of developers and testers. Also, this increases efficiency & quality of delivered ECU software. This approach helps developers to increase regression test set with each release and to be ASPICE compliant. For every delivery of an ECU functional component, automated tests are triggered for all test sets for all variants. Using minimum required inputs from user, we automate the testing process and maximize the productivity of the users.

The characteristics of our automated tests are simple commands to trigger, self-checking, instant feedback, End-to-end testing, timeline comparison with previous release, trend Graph. As the tools used were commonly used for any ECU Software development, we could reuse the plugin framework for automating any domain ECU software (like Aerospace domain, Automobile domain, etc.)

Overall benefits by using our solution:

1. Reduces Cost and Schedule risk
2. Increases Code Quality and Development Efficiency
3. Drastic effort reduction in Manual Testing
4. Reusability
5. Increases Productivity of Users

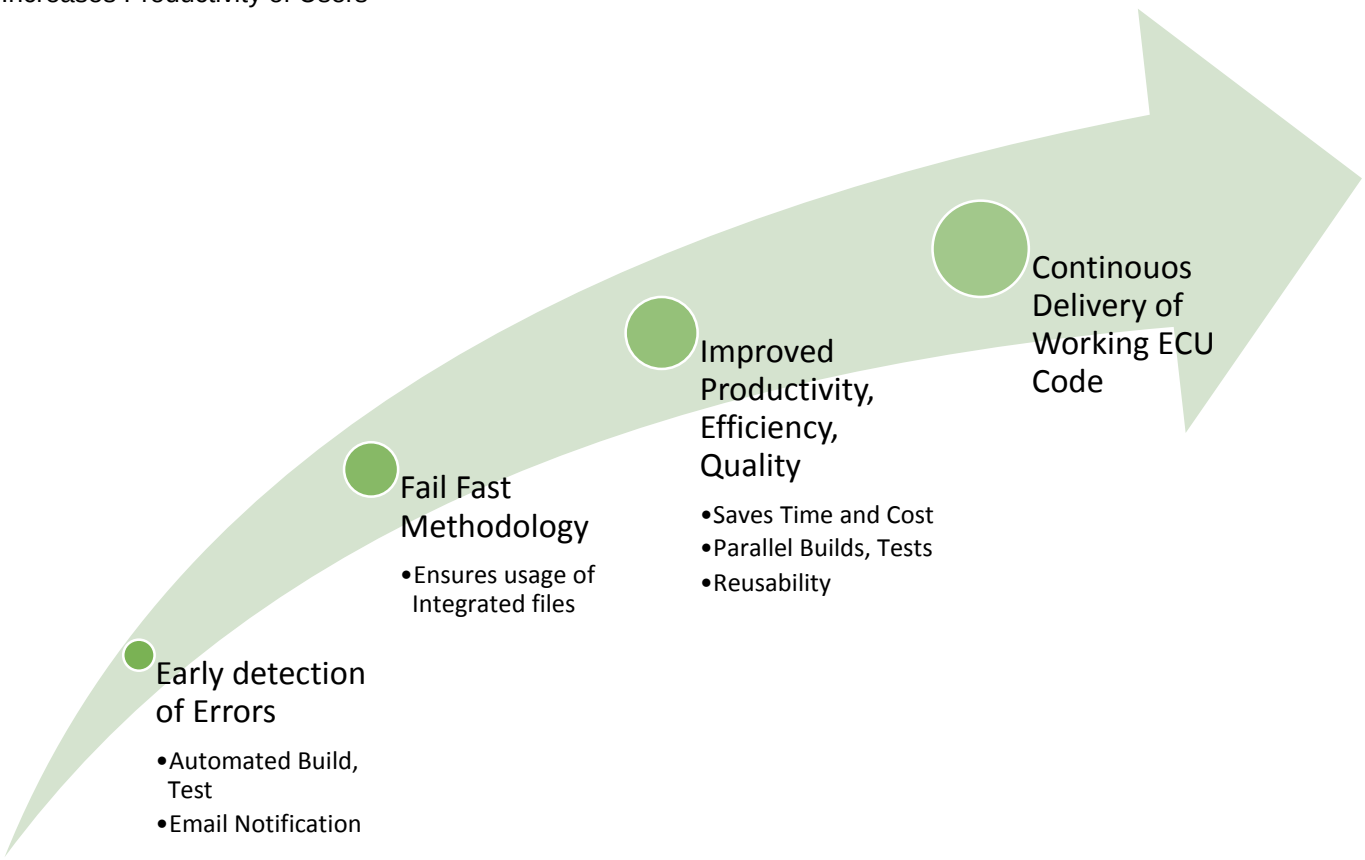

Figure 11 Advantages of Automation of V&V (SUV, SIT)

ACKNOWLEDGMENTS

We are immensely grateful to Mr. Franze Rayko [Product Owner, DGS-EC/ESB1, Robert Bosch GmbH] for giving us the seed for this project, valuable documents, motivation, valuable and timely suggestions, review of the project and support for the team.

REFERENCES

[1] VDA QMC Working Group 13 / Automotive SIG, Automotive SPICE Process Reference Model, http://vda-qmc.de/en/certification/automotive-spice/,http://www.automotivespice.com/fileadmin/software-download/Automotive_SPICE_PAM_30.pdf#page=52&zoom=auto,54,676
[2] TPT introduction and images are referred from https://www.piketec.com/de/2/tpt.html
[3] ETAS introduction and images are referred from http://www.etas.com/en/
[4] MATLAB introduction and images are referred from http://in.mathworks.com/products/matlab/
[5] ASCET images are referred from http://www.etas.com/en/etas_group.php
[6] PLS UDE Debugger images are referred from http://www.pls-mc.com/universal-access-devices-uad/universal_access_devices-a-797.html
[7] Documents referred from Internal BOSCH Training
[8] ECU-Test introduction and images are referred from https://www.tracetronic.com/products/ecu-test/
[9] Scrum Reference Card, http://scrumreferencecard.com
[10] Agile, https://www.agilealliance.org/agile101/what-is-agile/
[11] Martin Fowler, http://martinfowler.com/articles/continuousIntegration.html
[12] Jez Humble and David Farley, “Reliable Software Releases through Build, Test, and Deployment Automation” Book
[13] DevOps, From Wikipedia, the free encyclopedia

CONTACT

Vinodhini Vijayaraghavan
Technical Specialist,
Robert Bosch Engineering and Business Solutions Private Limited
CHIL SEZ Unit, Keeranatham Village, Coimbatore, Tamil Nadu 641035, India
+91(422)67-61108
Vinodhini.Vijayaraghavan@in.bosch.com

Jagadeeswara Vijayaraghavan Angamuthu
Associate Project Manager,
Robert Bosch Engineering and Business Solutions Private Limited
CHIL SEZ Unit, Keeranatham Village, Coimbatore, Tamil Nadu 641035, India
+91(422)67-62401
JagadeeswaraVijayaraghavan.Angamuthu@in.bosch.com

Saravanakumar C Shanmugam
Senior Software Engineer,
Robert Bosch Engineering and Business Solutions Private Limited
CHIL SEZ Unit, Keeranatham Village, Coimbatore, Tamil Nadu 641035, India
+91 (422) 67-62581
Saravanakumar.CShanmugam@in.bosch.com

DEFINITIONS, ACRONYMS, ABBREVIATIONS

- V&V: Verification and Validation
- ASPICE: Automotive SPICE
- MiL : Model in Loop
- SiL : Software in Loop
- PiL : Processor in Loop
- HiL: Hardware in Loop
- SUV: Software Unit Verification
- SIT: Software Integration and Integration Test / Software Integration Test
- SCM: Source Code Management
- MISRA: Motor Industry Software Reliability Association
- TPT: Time Partition Testing
- FC: Functional Component
- Scrum: Scrum is a management framework for incremental product development using one or more cross-functional, self-organizing teams [9].
- Agile: The ability to create and respond to change in order to succeed in an uncertain and turbulent environment [10].
- Continuous Integration: Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible. Many teams find that this approach leads to significantly reduced integration problems and allows a team to develop cohesive software more rapidly [11].
- Continuous Delivery: Continuous Delivery is an approach in which teams ensure that every change to the system is releasable, and that we can release any version at the push of a button. Continuous Delivery aims to make releases boring, so we can deliver frequently and get fast feedback on what users care about. It is the natural extension of Continuous Integration [12].
- DevOps: DevOps (a clipped compound of "development" and "operations") is a culture, movement or practice that emphasizes the collaboration and communication of both software developers and other information-technology (IT) professionals while automating the process of software delivery and infrastructure changes. It aims at establishing a culture and environment where building, testing, and releasing software, can happen rapidly, frequently, and more reliably [13].