Emphasis on Evaluative Prerequisites for Decisive Software-in-the-Loop (SiL) Environments

Kushal Koppa Shivanandaswamy
RBEI/EVV4
Robert Bosch Engineering & Business Solutions Pvt Ltd
Bengaluru, India
ksk261188@ gmail.com

Chandrima Sarkar
RBEI/EVV4
Robert Bosch Engineering & Business Solutions Pvt Ltd
Bengaluru, India
chandrima.sarkar@in.bosch.com

Sivakumar Rajagopal
RBEI/EVV AE-BE/EVV
Robert Bosch Engineering & Business Solutions Pvt Ltd
Bengaluru, India
sivakumar.r2@in.bosch.com

Lakkappa Pisutre Ramachandra
RBEI/EVV1
Robert Bosch Engineering & Business Solutions Pvt Ltd
Bengaluru, India
ramachandra.lakkappapisutre@in.bosch.com

Chiranjeevi Manchasandra Chayakumar
RBEI/EVV1
Robert Bosch Engineering & Business Solutions Pvt Ltd
Bengaluru, India
chayakumar.chiranjeevimanchasandra@in.bosch.com

***Abstract*—Recent trends in virtualization have shown significant desideratum, for the simulations to be performed using credible SiL environments. The SiL environment is a plausible approach, which is decisive in verifying the complex control algorithms, embedded into various automotive ECUs. Major challenges would be associated with the approaches, processes, and manipulation of risks, in the course of the development of credible SiL environments. The need for a competent mitigation plan is required, which might postulate a paradigm shift in the software development processes. This is quintessentially required for the development of an effective, reliable, and credible SiL environment. With different approaches and processes, currently, for different SiL environments, created using different tools, from different vendors, emphasis on the evaluation of the standard set of prerequisites is very much necessary. The evaluation provides technical clarity and aids in the process of vECU generation, with a significant impact on the implementation of a credible SiL environment. This decisive SiL environment thus enables verifying & validating the vECU behavior and performance. In this paper, we provide insights into the evaluative prerequisites, considered before the creation of the SiL environment. The evaluation of these prerequisites is imperative in identifying and selecting the software components and evaluating its available information, integrated as part of vECU. With the help of use-cases, we try to highlight and emphasize the imperative prerequisites, evaluated, and the need for its evaluation, in creating a credible SiL environment.**

Keywords—virtual ECU (vECU), Software-in-the-Loop (SiL), Verification & Validation, Prerequisites

I. Introduction

With the critical assessment of the process, it has been revealed that validating a software component/module is very much necessary and imperative, at an early stage in the development process. Growing trends show that there is a substantial increase in the complexity of automotive software. Factors like productivity, cost, and time-to-market are critical and will be emphasized during the development process of the highly-complex automotive ECU software. The need for functional tests of ECU software, its underlying software components, is very much essential. The challenges with, the evolution of software component, validation costs, and availability of the prototypes in later stages of the development cycles, create overhead in testing and validating the functional ECU software[1]. Interactive simulation of the ECU software at an early stage is imperative, with different domains involved in the development processes[2]. Software-in-the-Loop (SiL) is a credible alternative, which provides an environment to either test the individual software components or the entire ECU software, as compared to the traditional Hardware-in-the-Loop (HiL) systems, for the co-development processes. SiL emerges as a fore-runner with the evolution of interactions in the simulation technologies, with the virtual ECU (vECUs), virtual Networks (vNET), parallel computations, and cloud-based solutions, to validate the complex, software-intensive ECUs, both functional and non-functional concerns[3]. Such a credible SiL environment leverages benefits, termed as 3C Benefits, as in;

- *Configure* - the ability to *configure/include* the required layers of software in a vECU, integrate various models to have more realistic responses.
- *Control* - ability to *control* the execution of vECU during its simulation, by stopping the simulation whenever required, update the configuration and repeat the simulation deterministically.
- *Capture* - ability to *capture* the responses from various models, vECUs, or from different layers of software configured as part of vECU.

SiL also neutralizes the risk of unforeseeable hazards that are typically associated with prototype testing and actual vehicle tests [4]. Their availability and maturity processes often induce additional delays. Thus, the need for an efficient and effective process, to be established and standardized for the virtualization of ECU, as an integral part of its development life-cycle is very much essential. This would enable various domains to work towards the integration aspects of their software components, which would be integrated onto a single validation platform [5]. The different domains mean a variety of software variants and configurations, that need to be integrated as part of the vECU in the SiL environment. This yields to a co-simulation environmental setup for the vECU testing as part of the SiL environment [6]. Ideally, various kinds of models (plant models, environment models, any other FMUs, etc.)can also be integrated as part of the SiL environment, which would be compiled (source code of these models), for successful integration with the vECUs. The integration capabilities and performance of the SiL environment depends upon the factors, such as:

- compiled vECUs, with required software components configured and integrated. The interfaces of vECUs are as per the standards (like *FMI*), enabling their integration with other models
- plant models with the required computational logic modeled, included & compiled, and enabled with standard interfaces, as standard units (like *FMUs*)
- environment models with the required parameters, compiled and enabled with standard interfaces

A credible, efficient, and effective SiL environment, as such, can be envisioned and created by considering various essential factors and evaluating them before the creation of the SiL environment or during its course.

This paper focuses on highlighting the need for evaluation of the prerequisites and identifying the missing details, that would harm the creation of the SiL environment. The evaluation brings out maturity levels of the software or software components of vECU or various models considered for modeling and integration with vECU, as part of the SiL environment. Also, the evaluation enables checking the adherence levels to the standard development processes and bring out the shortfalls at an early stage. The analysis also aids in the assessment of creating adapters or wrappers (if required), for seamless integration of software components as part of vECU, and integrating with various models. The emphasis on evaluation of SiL prerequisites is highlighted in this paper with the help of use-cases, that were part of the Proof-of-Concept (POC) realization.

II. Understanding Software-in-the-Loop (SiL) Environment

The need to verify and validate the compiled production compliant software utilizing a virtual, interactive simulation environment is indispensable, with the evolution of ECU software and its intensiveness. This environment, not only can be used for verification and validation but also the integration and development of the ECU software and its constituent modules. Software-in-the-Loop (SiL), majorly involves the control software of the ECU being integrated into a virtual simulation environment, compiled on an x86 machine to run on a local PC. SiL enables to iteratively test and modify the code. Ideally, a credible SiL solution complements the traditional Hardware-in-the-Loop (HiL) systems, and do not replace them. SiL environment not only executes the control software of the ECU, but also the software of various models (plant models, environment models, other FMUs) integrated along with the vECUs, to have more realistic responses, correlate and qualify the environment by comparing with the responses from HiL systems. The basic constituent elements of a classic SiL environment and its architecture is as shown in “Fig. 1”.

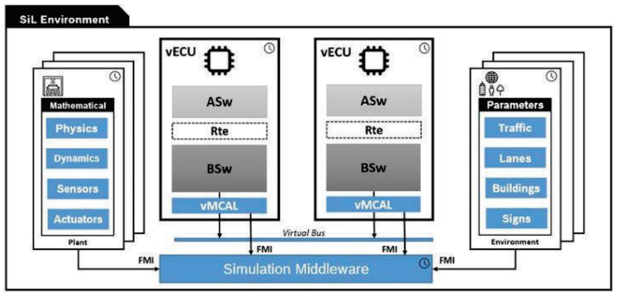

Fig. 1. Software-in-the-Loop (SiL) environment

The elements of the SiL environment being;

- ***vECU*** - virtual ECUs are units that consist of the target ECU code or parts of it, that are compiled to run on the standard x86 systems or the PC.
- ***Plant models*** - ideally a mathematical model representing the physical properties or dynamics of a system, integrated into the control unit. This model controls the control unit and is usually represented by a transfer function.
- ***Environment models*** - the powerful and real-time models, parameterized by the models of vehicles, roads, traffic, events, maneuvers, etc., majorly influencing the scenario-based testing. This is very much necessary during the virtual vehicle simulation or for vehicle-in-the-loop tests. This model comprises an intelligent driver model, vehicle model, and models for roads and traffic [8][9].
- ***Simulation Middleware*** - a sophisticated & powerful simulation and integration platform, to test and validate the ECU software, as vECUs. It also enables the integration of vECUs with the plant and environment models, as well as with other FMUs. The *simulation middle-ware* is responsible for the simulation of executable representations of vECUs, plant, and environment models. This serves as the master coordinator in an interactive SiL environment.

Within the SiL environment, each of these elements has a scheduler of its own, which can run or execute the elements in a simulation environment. In case, the execution of models is executed using a simulation middle-ware, which is the master of the entire SiL environment, then its referred to as system run-time. Otherwise, if the models or elements are executed in the tool they have been developed, and the exchange of data and control happens through tool-coupling (using tool APIs), then the SiL environment run-time is subsystem run-time.

The execution of models here means the source code of models is executed. When these models are being integrated using *simulation middle-ware*, then the models are compiled at the time of integration and upon successful compilation, the model is integrated with the vECU or other models. In the other case, models built in their specific development environments would be compiled before being executed through APIs. The vECUs are flexible, in the sense, can be configured at different levels, as in with control algorithms, multi-sensory applications, with or without BSW, or integrate with the 3rd party BSW. vECUs integrates the XCP service, to measure, calibrate, and stimulate the variables using suitable measurement and calibration tools.The variable data exchange between the vECUs and various models, including the stimuli model happens through the Data Access Points, as shown in "Fig. 2".

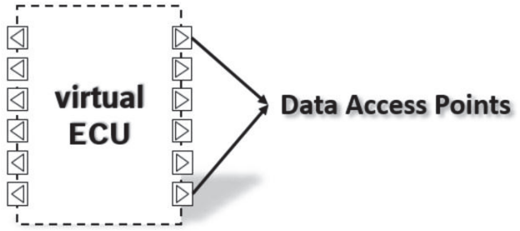

Fig. 2. Virtual ECU (vECU) with Data Access Points (DAP)

The software execution of the control unit can be either *Closed-loop* or *Open-loop*. In a *Closed-loop* scenario, the output from vECU is feedback to the plant model, which computes the conditions and alters it, based on the feedback from vECU. The input is from the plant model and *Stimuli model*. The control software decides based on the output of the plant model and inputs from the *Stimuli model*, as shown in "Fig. 3".

While in an *Open-loop* scenario, the environment is stimulated using the *Stimuli model*, as shown in "Fig. 4", wherein the output has no effect on the control decisions of the software and is completely based on the inputs. There is no plant model involved in the execution of vECU.

The plant models are modeled using various tools, and based on the level of quality of simulation responses, required to be recorded and analyzed, are integrated into the simulation environment. Also, the environment models, which are comprehensive and parameterized vehicle models, traffic and road models, integrated during vehicle-in-the-loop simulations, or virtual prototyping. In all these scenarios, it can be observed that the interfacing models is one of the key aspects. Also, the way these models are executed, their functionality, and configurations, is vital in deciding the overall performance of the credible SiL environment. All these factors impact the quality and credibility of the responses produced by the SiL environment. Some of the typical points that pose a challenge here are as typical questions seen below;

- *What?*
  - Is required from SiL environment
  - Goals to be achieved
  - Kind of models required to achieve the goals
  - Level of detailing - Configuration, is to be included
- *Why?*
  - It is required
  - Models are unavailable
  - SiL environment is not responding
  - Spurious behavior, upon the successful integration of compiled models
- *How?*
  - Immense the errors are
  - Integration of models are handled
  - Models are specified
  - Simulation is performed in-case, either of the models is not available

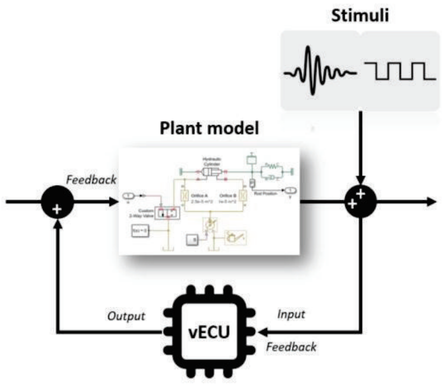

Fig. 3. Closed-loop Simulation

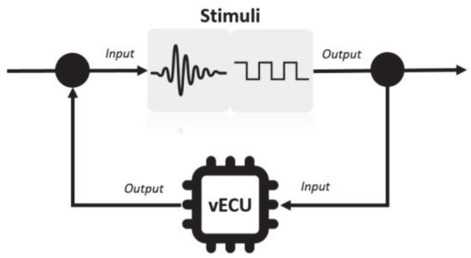

Fig. 4. Open-loop Simulation

Evaluation of these aspects is crucial in deciding various considerable facets of having a credible, efficient, effective, and reliable SiL environment.

III. Evaluative Prerequisites of SiL

The credibility of the SiL environment is arguable and specific for an intended application domain, and a constant improvement is required, as proposed by Raghupatruni Indrasen et.al [3]. The basis for the SiL environment is certain specific factors, which are to be evaluated before the creation of the SiL environment. The evaluation would also recommend subsequent enhancements of the SiL environment, based on the evaluation timeline. These factors are thus considered as evidently crucial and pivotal prerequisites for an efficient vECU to be generated. vECU is the driver in the SiL environment, and the control unit wherein, the control strategies are executed and the outcomes are produced. Various tools from different vendors assist in the configuration and generation of vECU. The prerequisites are independent of the software tools and are considered majorly for;

- creation & generation of vECUs
- integration of various models onto a single platform, for simulation
- modeling plant and environment models
- The evaluation of these prerequisites is essential in the effort, cost, implementation, or realization time estimations, and analyzing the maturity-levels of software, for certain crucial decisions.

A. Software Architecture

*Software Archit*ecture attributes the integral structure of the software component or system. It is kind of an exoskeleton description and serves as a blueprint for the software component or system. Ideally, the *software architecture* comprises software elements or components, their relations, and their properties. The software architecture defines a structured solution for various technical and operational attributes of the software. These attributes yield to a certain significant set of operational decisions like;

- structure of the software component or system
- interfaces composed by the software component or system
- the behavior of the software component or system
- composition and behavioral classification
- architecture standards
- modularity

The *software architecture* is usually exemplified as “the design decision that needs to be made early in the development stages”. The decisions have a considerable impact on the quality, maintainability, and performance of the system.

In automotive software, with huge complex systems, the need for software to be scaled as per the variants, associated with transfer capabilities, and with considerable availability of the software requirements, led to the development of standardized software architecture, called “AUTOSAR”. *AUTOSAR* is a three-layered architecture, with Basic Software (*BSW*), Runtime Environment (*RTE*), and Application Software (*ASW*). The *AUTOSAR* architecture provides the system and ECU configuration descriptions are made available through **.arxml* file. The evaluation that needs to be done here is to check whether the definitions, configuration, elements, and attributes are available as part of the **.arxml* file. Also if the details are divided into multiple **.arxml* files, then all the files need to be integrated and evaluated for their integration and data validity. It can also be that the application software layer is *Non-AUTOSAR*, meaning, there is no configuration **.arxml* file available for it, as in for legacy code. In this case, the technical details and information of the *software architecture* to be extracted from the available artifacts (*usually source code (*.c,*.h) files* or *block diagrams*), manually. For instance, the extracted details regarding interfaces of the component can be as a **.xml* file. The extracted information is then scrutinized and validated by developers or architects. The approaches followed, in either case, are different and the latter requires some kind of adapter or wrapper code to be generated, to enable the integration of software components as part of the vECU, as it doesn’t include standard architectural policies.

B. Software Functionality

Software Functionality correlates to the software aspects, corresponding to the control logic computation, that software can perform and produce a response, upon execution. The control algorithm decides the outcome, that is expected from the software component as part of the vECU or models. The *software functionality* represents the system outcome based on the set of tasks coupled with its behavior, for a software component. Software functionality gauges the value of the software component behavior. As part of the SiL environment, for vECUs, the software component functionality metrics like tasks, functions (*also called as runnable in AUTOSAR*), their raster rates (*scheduled periods*) are derived from the *AUTOSAR* configuration **.arxml* file. In the case of *Non-AUTOSAR*, the legacy code files (**.c,*.h*) are to be explicitly analyzed and the corresponding technical details are extracted manually, like;

- main function - responsible to update the state of the component or which produces the outcome, cyclically for every raster time.
- sub-functions - control algorithm that is responsible for the computation and updates the values across when called in the main function
- raster rate - the time under which the functions are classified and executed to produce an outcome

The above-mentioned details are pertinent for the configuration of vECUs, as the details of main functions, sub-functions, and raster rates are necessary for the configuration of the Os module, of vECU. Also, the availability of artifacts like **.c, *.h* files that correspond to the software functionality, required for the generation of vECU, are evaluated. The artifacts represent the behavior of software components integrated as part of vECU. It is also crucial in evaluating that all the required artifacts are available and analyzed.

C. *Software Component Interfaces*

An interface is a method by which the exchange of data or control happens between two or more different software components, in a system. The interfaces of a software component implement the ports. In AUTOSAR, these interfaces are termed as *Port Interfaces*, which characterizes the information provided (output) or required (input) by a port of a software component. It is with the help of these interfaces the exchange of data is managed. In AUTOSAR, all the interfaces are defined and implemented as per the standard, with their datatype mappings (*application or implementation datatype or base datatype*). The implementation of interfaces is managed by their scope, i.e. they are defined as global or local or static. In the case of Non-AUTOSAR, the interfaces implementing the variables of software components, within the legacy code, need to communicate via the global scope. The need for functions to be void, as well the parameters, are certain prerequisites that are considered for evaluation. The above explanation is for the vECU generation. Here, the need to manually analyze the scope of parameters and include them in the **.xml/*.xlsx* file, for vECU creation is adopted. While for the former, the approach is straightforward, as the definitions are available with the configuration and its coherent properties in a **.arxml* file. During the compilation for the generation of vECU, these interfaces are then converted into the FMI compliant *Data Access Points*. It is via these Data Access Points, vECU communicates with all the other models, via the *simulation middle-ware*.

The interfaces play a major role in the SiL environment, as the vECUs are integrated with the plant and environment models or to the stimuli model using *simulation middle-ware*. The plant and environment models integrated into the SiL environment also needs to generate the FMI standard-compliant interfaces for successful integration with the vECUs. And thus, this is one of the important prerequisites to be evaluated for a successful vECU generation and also for a credible SiL environment.

D. *Layers of Software*

As elucidated earlier, the *AUTOSAR* architecture follows a three-layer architecture [10]. The top-most layer being the *Application Software* layer, wherein the application software components that interact with the *RTE*, with complex control strategies are integrated. The standardized software modules, which offer the services to run the application software components, are called the *Basic Software* layer. As the *AUTOSAR* architecture focuses on modularity, the vECUs created can be modular with only specific layers configured and integrated with the required software components or modules. This strategy is mainly to facilitate;

- exchange of software
- update the software
- cooperate on standards
- incremental build approach, minimizing the complexities, with modularity and enhancing the performance

With this, many variants of vECUs can be created, with different layers of software components or modules included as part of the vECUs. This is the task of integrating only the portions of ECU code, for the faster test of application software components or the basic software modules. To do this, it is necessary to have the information of the layers of software, included as part of vECU, and to evaluate their dependencies on other software components or modules. As the layers are cut, the dependencies would be exposed and it is required to take care of these dependencies employing efficient stub management. For the software with *AUTOSAR* architecture, it is quite easy in evaluating and analyzing the dependencies, while for *Non-AUTOSAR*, it would require quite some excessive manual effort. The application software is Non-*AUTOSAR* meaning, the configuration files are not available, and the configurations are to be extracted from available software architectures, if any, or else the legacy code has to be analyzed deeply to trace and extract their dependencies information. Stub management, here means that either the dependencies are commented-out and explicitly defined or declared, either the ways there shouldn't be any adverse impact on the performance of ECU software, as part of the vECU. This is one of the influential prerequisites that need to be evaluated, exquisitely.

E. *Configuration*

Configuration deals with the integration of required software components or modules, from specific layers. The configuration also handles the stub management process, to ease the execution of the software employing a temporary replacement of the lower-level modules, thus ensuring similar responses from the vECU. The key aspects considered under configuration are;

- integration of software components - architectural, behavioral and configuration elements (like **.arxml/*.c/*.h/*.a2l/*.x*ml)
- configuration of vECU for the generation of specific layers like *RTE* & other required basic *BSW* modules, for stimulating the vECU
- configuration of protocols like CAN/LIN/Eth/FlexRay
- creation of EcuExtract and SystemExtract
- creation & configuration of EcuInstances
- generation of vECUs - successful compilation and build
- debug errors if any
- enhance the debugging process with the integration of debugger

The configuration is almost the same process, in the majority of the tools. While it is a bit different in the way steps are enabled to achieve the necessary artifacts at the required levels, for the generation of vECU. The process of configuration differs slightly between the *AUTOSAR* and *Non-AUTOSAR* architectures, as with the former, the process is to import the configuration files and make some slight adaptations in the tool to enable the configuration. Later import the behavioral artifacts like **.c,*.h* files, defining their functionality. In the latter, the process slightly deviates as there is a need for the wrapper code or an adapter to be developed, to convert the *Non-AUTOSAR* format into a standard format, that the tool accepts and understands. It has to be noted here that, none of the tools support *Non-AUTOSAR*, legacy code integration for vECU generation, as-is, without any adaptations. It is the responsibility of the integrator to take care of the information included as part of the wrapper code. The wrapper code or adapter is project-specific. However, generic adapters are provided by various tool vendors for the specific vECU generation tools, which has certain prerequisites, that are mandatory. The configuration is a pivotal parameter that is evaluated for the generation of vECUs.

F. *Test Concerns*

Test Concerns brings out the concerns within vECU that needs to be tested or that can be tested, in a SiL environment. The scope of the test that is presented, with the vECU and other models being included in the scope, for measurement and analysis. This provides an overview, in terms of;

- What is in the module?
- What can be tested in the module?
- How can it be tested?
- What needs to be tested?

This prerequisite is very much essential in knowing what all inherent components are to be included as part of the SiL test environment, in addressing the concerns over testing the control logic. The concerns related to the components addressed upon evaluation, are;

- function calls
- sequential execution
- states of variables
- different operating modes/conditions
- protocols, if any included

Test Concerns are majorly derived by the architects and developers for the software function that would be integrated as part of the vECU in the SiL environment. This also provides an overview of the models included as part of the SiL Test environment. The concerns related to models are;

- category of model
- model functionality
- states of model operations

Therefore, this is also one of the evaluative prerequisites that are pertinently evaluated and are emphasized for evaluation.

G. *Qualification*

This prerequisite is associated with the credibility of the SiL environment and its constituent elements. This prerequisite attributes to the performance claims, wherein it is claimed that the vECU would perform to the required level, with the desired outcomes within the simulation environment. Here, the simulation environment is nothing but the SiL environment, and bench-marking is required for the satisfactory performance of the vECU. For *qualification*, it is required to define the scope of the system and its interfaces, as each element of the SiL environment is to be qualified and checked for its credibility against certain standard criteria.

The *qualification* process includes the requirements, SiL environment, and its consistency in producing the same results against time, as compared with the traditional HiL system responses, with accuracy. Also, this activity is performed for the SiL environment, for a specific application domain. The outcomes of *qualification* processes are documented. These artifacts are later utilized for claiming the credibility of the SiL environment and its constituent elements. The *qualificatio*n process instills a sense of maturity and confidence, in using the SiL environment for realization and implementation.

IV. Use-Case

Vehicle Access Control is a next-generation vehicle access system, with the main feature being to use a mobile phone/key card/tags as a key with restricted and time-limited rights. *Vehicle Access Control* supports access via Radio Frequency (RF). *Vehicle Access Control* provides a vehicle ECU with information about the presence and location of authorized phones. Standard RKE functions are implemented so the existing look and feel of the vehicle are maintained. *Vehicle Access Control* enables the new use cases for private and fleet vehicle via flexible key management.

With this system being highly complex and software-intensive, the need to analyze and access the software behavior and quality of the software by detecting and analyzing the software defects at an early stage is imperative. Software-in-the-Loop (SiL) process is adopted to enable analysis and assessment of the software behavior, quality of software upon integration, and tracing the defects, at an early stage in the development process. With SiL, the ability to simulate and test software at an early stage, address and assess the ECU concerns (like diagnostic, COM & memory), front-loading with PC-based simulations creating vECUs and capabilities to configure, control, and capture the behavior of software, are enabled.

The process for implementation is, as shown in "Fig. 5". 3 main phases were followed for the generation of vECUs, integration of vECU in SiL environment with the *Stimuli model,* and performance measurement and calibration activities.

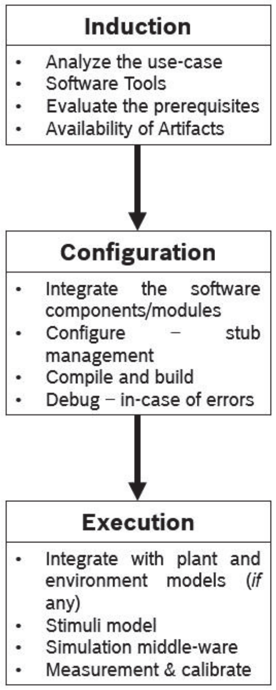
Induction
Analyze the use-case
Software Tools
Evaluate the prerequisites
Availability of Artifacts
Configuration
Integrate the software components/modules
Configure – stub management
Compile and build
Debug – in-case of errors
Execution
Integrate with plant and environment models (if any)
Stimuli model
Simulation middle-ware
Measurement & calibrate

Fig. 5. Workflow – Use-case implementation

The 3 phases are, as follows;

- *Induction* - in this phase, the use-case is analyzed. Here the use-case or software component, that was considered, from Vehicle Access Control project, *2 application software* components. Here, it can be noted that *application software* component *1* follows the *AUTOSAR* architecture, while the *application software* component *2* follows *Non-AUTOSAR* architecture. The availability of Software tools to perform integration, configuration, and generation of vECU was evaluated. Also, the above mentioned evaluative prerequisites were extensively evaluated for compliance and consideration. Then the availability of artifacts was verified and based on its availability, the configuration was done.

- *Configuration* - the *application software* component *1* was integrated with minimal adaptations, while the Tag Interface component, required quite an extensive manual analysis, and the wrapper code was generated for its integration as part of the vECU. The stubbing process was also managed during this phase for both the components. The EcuInstance created was successfully compiled and built, to generate the vECU. In either case, there were no **.a2l* files available.

- *Execution* - there were no plant models or environment models available for integration with generated vECU. Only the Stimuli model was integrated, to stimulate the vECU. The integration with the stimuli model was via the simulation middle-ware, from the leading SiL solution vendor. The measurement and calibration were performed.

During its realization, effort estimates were logged and a collective representation of the data for both the varied architectures are provided in the tables, "TABLE I. " and "TABLE II. ", respectively.

TABLE I. USE-CASE 1: VEHICLE ACCESS CONTROL – AUTOSAR ARCHITECTURE

Prerequisites	*Adapters / Wrapper code*	*Effort Estimation (in hrs)*
Software Architecture	NR[a]	8
Software Functionality	NR[a]	3
Software Component Interfaces	NR[a]	2
Layers of Software	NR[a]	5
Configuration	NR[a]	27
Test Concerns	NR[a]	10
Qualification	NR[a]	10

TABLE II. USE-CASE 2: VEHICLE ACCESS CONTROL – NON-AUTOSAR ARCHITECTURE

Prerequisites	*Adapters / Wrapper code*	*Effort Estimation (in hrs)*
Software Architecture	NR[a]	27
Software Functionality	NR[a]	34
Software Component Interfaces	Required	34
Layers of Software	Required	34
Configuration	Required	68
Test Concerns	NR[a]	42.5
Qualification	NR[a]	17

[a] Not Required

The effort estimates logged were analyzed, as shown from "Fig. 6". It can be seen that with the *AUTOSAR* architecture standard compliance, processes were performed and the solution was realized with ease and faster. While with *Non-AUTOSAR* architecture, it required more manual effort than the former. Hence, more effort is required. With this analysis, and emphasizing the evaluation of prerequisites, the effort estimates can be re-worked as it provides more clarity of the topics and the kind of adaptations or effort that would be involved with the processes.

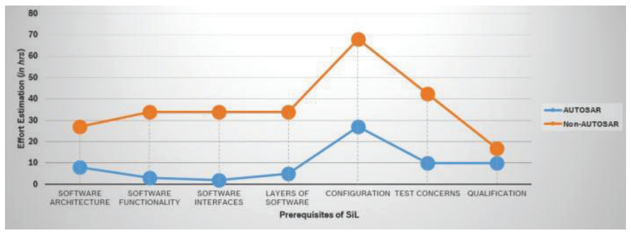

Fig. 6. Analysis – AUTOSAR Vs Non-AUTOSAR

V. CONCLUSION

The credibility of the SiL environment and its constituent elements are decisively attributed to the imperative evaluation of the prerequisites. The evaluation of these compelling factors accords the efforts in realizing or generating a vECU. In this paper, we elaborate on the need for evaluation of the software components against the prerequisites, explained in detail, of the vECU in the SiL environment. This provides more insights into the effective approaches that can be incorporated into the generation of vECU. The intensification of performance, test times, and cost factors are comprehended with this evaluation. The need for emphasis on evaluating these factors is justified with a classic use-case, in this paper. The evaluation instills confidence about the software component and ideally helps in decision making, selecting the software component or module for its integration as part of vECU. Also in doing so, a credible, reliable, effective, and efficient SiL environment is created, for testing the complex ECU software functions. This process boosts the maturity levels of the SiL environment. Even though some more factors are also very essential in improvising the prospects of the SiL environment and not only focusing on the aspects of vECU, it is planned to be included under the scope of future work.

REFERENCES

[1] Phatak, S., Chen, H., Xiao, Y., Wang, C. et al., "Virtual Multi-ECU High Fidelity Automotive System Simulation," SAE Technical Paper 2016-01-0013, 2016, doi:10.4271/2016-01-0013.

[2] Schneider, S.-A.: Presentation on the "7th MODPROD Workshop on Model-Based Product Development", Linköping University, February 5–6, 2013, BMW, Berlin

[3] Raghupatruni I., Burton S., Boumans M., Huber T., Reiter A. (2020) Credibility of software-in-the-loop environments for integrated vehicle function validation. In: Bargende M., Reuss HC., Wagner A. (eds) 20. Internationales Stuttgarter Symposium. Proceedings. Springer Vieweg, Wiesbaden. https://doi.org/10.1007/978-3-658-30995-4_30

[4] Schmidt, S., Henning, J., Wambera, T. et al. Early PC-based Validation of ECU Software Using Virtual Test Driving. ATZ Elektron Worldw 10, 14–17 (2015). https://doi.org/10.1007/s38314-015-0508-y.

[5] Goyal, R., S, K., and Mistry, P., "Standard Process for Establishment of ECU Virtualization as an Integral Part of Automotive Software Development Life-Cycle," SAE Technical Paper 2020-01-5007, 2020, https://doi.org/10.4271/2020-01-5007.

[6] L. Mikelsons and R. Samlaus, "Towards Virtual Validation of ECU Software using FMI", Proceedings of the 12th International Modelica Conference, 2017.

[7] ETAS COSYM, Software Products & Systems, https://www.etas.com/en/products/cosym-co-simulation-platform.php.

[8] IPG CarMaker, Simulation Software, https://ipg-automotive.com/products-services/simulation-software/carmaker/

[9] Mosser Alexander, "Vehicle Environment Model for Autonomous Driving", MSc Thesis, Graz University of Technology, 8010 Graz, Austria, 2016.

[10] "AUTOSAR: The worldwide automotive standard for e/e system", ATZextra, Springer Fachmedien Wiesbaden, 18: 9–10, October 2013, ISSN 2195-1454

[11] dSPACE VEOS, Simulation Software, "https://www.dspace.com/en/inc/home/products/sw/simulation_software/veos.cfm"

[12] Anne Geburzi, Holger Krisp, Karsten Krügel, "The Next Generation of Validation - Experimenting Conveniently with Virtual ECUs", dSPACE GmbH, Hanser Automotive, Tech. Report, Jul 2011

[13] Jordan, Y., von Wissel, D., Dolha, A. et al. Virtual ECUs Used to Develop Renault's Engine Management Software. ATZ Elektron Worldw 13, 36–39 (2018). https://doi.org/10.1007/s38314-018-0058-1

[14] A. Junghanns, J. Mauss, M. Seibt, "Faster Development of AUTOSAR compliant ECUs through simulation", ERTS-2014, Toulouse

[15] Morishima, K., Oho, S., and Shimada, S., "Virtual ECU and Its Application to Control System Analysis - Power Window System Demonstration", SAE Technical Paper 2016-01-0022, 2016, doi:10.4271/2016-01-0022., 1963, pp. 271–350.

[16] Bilgaiyan, S., Mishra, S. & Das, M. Effort estimation in agile software development using experimental validation of neural network models. Int. j. inf. tecnol. 11, 569–573 (2019). https://doi.org/10.1007/s41870-018-0131-2

[17] Lobato, S., Poncela, J. & Aamir, M. General framework for testing using formal languages. Int. j. inf. tecnol. 9, 41–48 (2017). https://doi.org/10.1007/s41870-017-0007-x.