A Rapid Review on Advanced Software-in-the-Loop Methods for Automotive Software Verification

Larissa Pestana
Federal University of Pernambuco
Recife, Brazil
lcp2@cin.ufpe.br

Breno Miranda
Federal University of Pernambuco
Recife, Brazil
bafm@cin.ufpe.br

ABSTRACT

The V-cycle has been the primary software development methodology in the automotive sector. However, with the increasing complexity of software, both from an architectural and technological perspective, software verification has become inefficient compared to system verification. This inefficiency results in rework and increased costs due to late error detection. Software deemed functional during verification may prove ineffective when tested in the simulated vehicle environment, as development requirements are ever-expanding, and test simulation environments have not kept pace with this evolution. The simulation environments used in automotive development are based on a guided simulation approach, encompassing Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), Processor-in-the-Loop (PiL), and Hardware-in-the-Loop (HiL) testing. These tests are conducted between system and software verification and must be used in conjunction to achieve comprehensive validation. To address inefficiencies in software verification, techniques such as MiL, SiL, and PiL are excellent techniques. SiL, in particular, plays an essencial role in simulating code before its integration into the system. This rapid review will discuss techniques that make simulations more representative of the real vehicle system, considering the electromechanical structure of sensors and actuators outlined in the software requirements, as well as the network of ECUs, which follows a distributed communication model.

KEYWORDS

Software verification, Software-in-the-Loop (SiL), Simulation environments, Automotive Software

1 INTRODUCTION

The increasing complexity of embedded systems in the automotive industry demands the revision and adaptation of verification and validation processes. Ensuring that automotive software, now with greater component integration, undergoes rigorous validation stages is essential to guarantee its quality, safety, and functionality in critical systems.

Historically, the V-cycle has been the predominant methodology in automotive software engineering for managing the development and verification of systems. This linear model, with sequential phases of development and verification, presents several limitations: its rigidity makes it difficult to adapt to requirement changes during development, a common occurrence in complex projects; long feedback cycles result in inefficiency; and the complexity of automotive systems requires more dynamic and interactive approaches. These limitations culminate in high costs and increased risk of failures in the final stages of verification and validation.

To mitigate these limitations, Liu *et al.* [12] proposed from W cycles, the V-INC (Incremental Verification Cycle), as shown in the figure of his research . The goal is to improve the flexibility and efficiency of the verification process. The W cycle includes testing activities in the early stages of development, allowing early detection of defects and reducing costs and correction time. In the INC-V cycle, models and test cases are continually developed and integrated, with verification carried out throughout the entire life cycle, ensuring continuous and efficient verification.

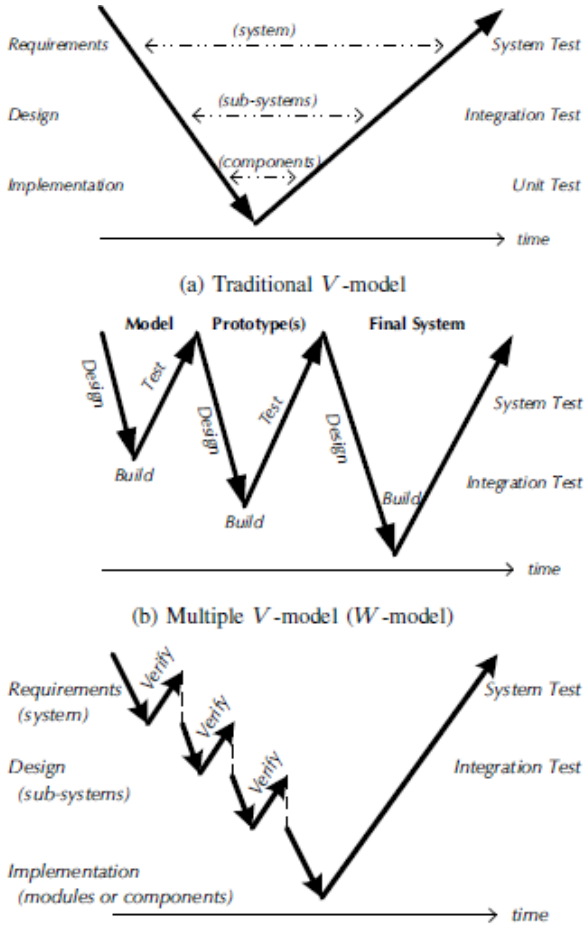

Figure 1: Comparison of Models V, W and INC-V by Liu *et al.* [12]

Falcini and Lami [5] conducted an empirical study using the Automotive SPICE methodology (Software Process Improvement and Capability dEtermination), as described by the VDA QMC Working Group 13 / Automotive SIG [20]. The results indicated that software unit verification (SWE.4) and software integration verification (SWE.5) practices often perform poorly due to the informal execution of verification activities and the lack of well-defined test strategies. The study also revealed a significant difference in maturity between system testing and software testing, with the former showing higher maturity.

These studies on development cycle methodologies and software testing practices highlight inefficiencies in software verification during development and integration. Mature identification of algorithmic errors predominantly occurs in the final system verification, resulting in significant rework from the early stages of development. This underscores the need to improve verification and validation practices from the early stages.

Considering the closed-loop simulation-based approach, the focus of these improvements could be on Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), Processor-in-the-Loop (PiL), and Hardware-in-the-Loop (HiL). However, in HiL, we typically have System Verification tests, which are not the focus of this work. For Software Verification, we have MiL, PiL, and SiL.

In MiL, there is no simulation of the real system, i.e., the vehicle, making it inefficient for the purpose of this research. In PiL, the system is simulated, but the code is running on the actual ECU hardware, which can be considered system or software verification. Since the focus of this research is the purely simulated environment of software verification, SiL is the ideal environment for implementing improvements that indeed affect SWE.4 and SWE.5.

This work is organized as follows: Section 2 presents the main approaches for automotive software verification. Section 3 details the rapid review protocol. Section 4 discusses the action research results. Section 5 presents the discussion of the results, and Section 6 concludes the work, highlighting the main contributions and suggesting future research directions.

2 BACKGROUND ON AUTOMOTIVE SOFTWARE VERIFICATION

Automotive software verification is an essential process to ensure the safety, reliability, and compliance of integrated systems that control crucial functions in modern vehicles. The approaches establish rigorous verification standards with the goal of ensuring software consistency and quality. Three main verification approaches are widely used: Formal Methods, Static Code Analyzer, and Simulation-Guided.

2.1 Static Code Analyzer

Static code analyzers are widely used in automotive software verification, being crucial for detecting specific coding errors. These tools are designed to assess the quality of automotive control code, identifying issues such as variable type mismatches and other syntactic errors. However, the capability of static code analyzers is limited to the static context of the code, as they cannot evaluate properties that depend on the software's dynamic behavior in its execution environment. Although they are effective in identifying syntactic errors and some logical issues, these analyzers cannot guarantee the complete functionality of the system under real operating conditions.

2.2 Formal Methods

Formal Methods are rigorous mathematical techniques used to ensure that computational systems behave correctly and safely. Torodov *et al.* [19] investigate the application of three main Formal Methods techniques in the context of automotive software: abstract interpretation, model checking, and deductive proofs.

Abstract interpretation detects runtime errors, such as division by zero and out-of-bounds array accesses, by calculating abstract representations of the code without the need for actual execution. Model checking verifies whether a formal model of the system meets specific properties expressed in temporal logic. This method is effective in identifying failures by showing trajectories that violate the verified properties through counterexamples. Deductive proofs establish mathematical properties of formal models, offering a more expressive approach but requiring a high level of expertise and not providing direct counterexamples when a property is violated.

The article proposes the integration of these techniques in automotive software development to improve the reliability and safety of systems. The study shows that the application of Formal Methods can detect errors more efficiently compared to traditional testing methods and suggests a progressive approach to incorporating these techniques into the development process.

Although Formal Methods tools provide a high degree of confidence in system correctness, they face significant challenges in terms of scalability for large and complex industrial systems, such as automotive control codes. Additionally, the time required for automotive software engineers to learn these techniques, as well as the implementation costs, are barriers that hinder the widespread adoption of Formal Methods in the automotive industry.

2.3 Simulation-Guided

Simulation-guided verification is an advanced technique used to ensure the correctness and reliability of automotive control systems, especially in powertrain control systems. This approach combines simulation and optimization to identify undesirable behaviors and ensure that systems meet specified requirements.

This method involves creating accurate models of the system to be verified and executing simulations to estimate its behavior under various operational conditions. The models may include detailed representations of the embedded controller and the operational environment of the system. Kapinski *et al.* [10] use simulations to validate functional behavior, adjust initial control parameters, and estimate system performance. Verification is performed iteratively, where a simulation engine generates system behaviors and an optimizer searches for inputs and parameters that might cause failures.

There are two main approaches: Open-Loop and Closed-Loop:

2.3.1 Open-Loop Verification. In this approach, the system model is tested in isolation, without feedback from the environment or the controlled plant (vehicle). The system inputs are provided without considering the system's responses over time. The goal is to validate

whether the controller, in this case, the electronic control unit (ECU), meets the basic functional requirements under predefined conditions.

2.3.2 Closed-Loop Verification. In this approach, the simulation includes both the controller (ECU) and the plant (vehicle), allowing dynamic feedback between them. The controller sends commands to the vehicle system, and the resulting feedback from the vehicle's behavior is used to continuously adjust the controller's output. This replicates real operating conditions, providing a more accurate representation of the system's behavior. Closed-loop test configurations include: Model-In-The-Loop (MiL), Software-In-The-Loop (SiL), Processor-In-The-Loop (PiL), and Hardware-In-The-Loop (HiL).

- MiL tests control algorithms in a virtual simulation environment, using mathematical models of the plant to provide feedback to the controller.
- SiL runs the actual control code in a simulated environment, allowing software validation with feedback from the simulated plant.
- PiL tests the control software on the final target processor (ECU), with continuous feedback from the simulated plant to evaluate the ECU hardware's performance.
- HiL integrates the control software with real hardware (ECU), using real-time simulation of the plant to provide dynamic feedback, combining computational simulation (hardware and software) of the vehicle.

The ideal scenario for software verification involves performing tests sequentially, going through MiL, SiL, PiL, and finally HiL, which is already in system verification. This sequential approach allows for progressive validation of control algorithms, software, and hardware, ensuring that issues are identified and corrected in the early stages. Integrating different levels of testing requires precise temporal synchronization, essential to ensure consistency and reliability of the results.

2.4 Defining test cases

Once the approach is defined, test cases should be designed to ensure comprehensive coverage of system requirements and maximize defect detection early in the development cycle. Barhate [2] presented a comprehensive testing strategy for automotive software that encompasses several relevant topics:

2.4.1 Test Case Reduction Using the Taguchi Method. The Taguchi method, based on Orthogonal Arrays, is employed to systematically reduce the number of test cases while ensuring comprehensive coverage. This approach significantly reduces the time and resources required for testing.

2.4.2 Test Case Prioritization. Test cases are prioritized into four levels, ensuring that the most critical and impactful tests are executed first. This prioritization helps uncover the majority of defects in the early phases of the testing process, improving the efficiency of the testing strategy.

2.4.3 Automation of Test Case Development and Execution. Automation tools are developed to generate test vectors and convert them into executable test cases. These tools facilitate the execution of tests in HiL setups, ensuring uniform implementation and reducing reliance on manual processes.

2.4.4 Avoiding the Pesticide Paradox. Regular review and update of test cases are emphasized to prevent test cases from becoming ineffective over time. This approach ensures that new defects are continuously identified, maintaining the robustness of the testing process.

3 RAPID REVIEW PROTOCOL

The research protocol model for Rapid Review was based on the model proposed by Cartaxo *et al.* [3].

3.1 The Practitioners' Problem

The increasing complexity of automotive software has made software validation inefficient compared to system verification, resulting in rework and high costs due to late identification of failures. Considering that the most adopted and recommended software verification methodology for this type of verification is simulation-guided, as demonstrated in the background section, SiL is essential to start the interaction of the software with the simulated system. However, deficiencies in the proper application of SiL aggravate the software verification situation, increasing the likelihood of errors not being detected until later stages of development.

3.2 Research Questions

RQ1: What are the reasons for the maturity difference between software verification and system verification in automotive development?

RQ2: Which verification approach is most effective for automotive software verification?

RQ3: What is the relevance of the XiL approach loops (MiL, SiL, PiL, and HiL) for software verification?

RQ4: Among the most relevant loops in verification, what technical gaps prevent the simulation of a more realistic system?

3.3 Search Strategy

To expedite the search for primary studies and conduct the rapid review within the deadline, we used the IEEE Xplore digital library. This database was chosen because it is one of the most relevant for the automotive sector and offers an interface that facilitates search refinement. Additionally, the efficiency of IEEE Xplore in providing high-quality results makes it ideal given the available time. We tested several versions of the search string until we found a suitable set. We presented the string to professionals and, through a feedback cycle, refined and defined the following search strings by search cycles:

- Cycle 1: "Automotive" and "Software" and "Verification"
- Cycle 2: "Automotive" and "Loop" and "Software"

3.4 Selection Procedure

To better organize and understand the search results, we applied a series of selection filters distributed across different analysis cycles, represented in Figure 2. Each cycle was designed to further refine

the search, ensuring that the selected studies meet the specific criteria defined at the beginning of the process.

- Filter 1: The study must be within the context of automotive software verification
- Filter 2: The study is a primary study
- Filter 3: The study must provide answers to at least one of the rapid review research questions

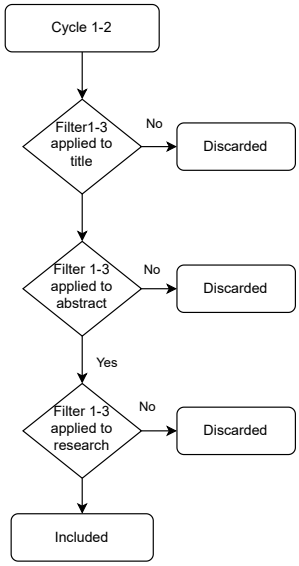

Figure 2: Model Selection Process

Table 1 presents the results of the selected studies, detailing the compliance with each selection filter in each cycle.

	Cycle 1	Cycle 2
No Filter	573	523
Filter 1-4 to title	50	80
Filter 1-4 to abstract	34	48
Filter 1-4 to research	**6**	**11**

Table 1: Research found in the selection process model

Thus, we ended up with 17 selected primary studies, as shown in Table 2.

The pair selection was not considered due to time constraints and taking into account the research participant's experience in the automotive software industry, specifically in the area of software verification. This familiarity with the processes and criteria needed for selecting relevant articles allowed for an efficient and focused analysis without compromising the quality of the selection process.

3.5 Extraction Procedure

The extraction procedure followed the same selection procedure, involving an individual researcher who is a participant in the research and the industry.

3.6 Synthesis Procedure

The data synthesis will be conducted in three main stages. First, data extraction will be performed using a standardized form to ensure consistency and integrity of the information collected from each included study. Next, a qualitative thematic analysis will be employed to identify and code the emerging themes from the data. This approach will allow for an in-depth understanding of patterns and qualitative insights. Finally, a narrative synthesis will be used to integrate the thematic data, providing a comprehensive and coherent view of the available evidence, facilitating the interpretation of results, and drawing meaningful conclusions.

3.7 Rapid Review Report

To facilitate access for professionals to the results of the Rapid Review (RR), we present the findings through an Evidence Briefing. This is a concise one-page document that exclusively highlights the main research results, providing a clear and objective summary of the most relevant evidence. Figure 3 shows the Evidence Briefing we created to report the RR. The Evidence Briefing document is also available online (see Section 7 for more details) .

Figure 3: Evidence Briefing

3.8 Limitations and Threats to Validity

This research has some limitations. The selection of articles was carried out by a single researcher, which may introduce biases. To

ID	Title	Cycle	Citation
1	Effective Test Strategy for Testing Automotive Software	1	[2]
2	Software-in-the-loop Simulation of a Test System for Automotive Electric Drives	2	[4]
3	System and Software Testing in Automotive: an Empirical Study on Process Improvement Areas	1	[5]
4	Formal analysis of timing effects on closed-loop properties of control software	1	[6]
5	An X-in-the-Loop (XIL) Testing Framework for Validation of Connected and Autonomous Vehicles	2	[7]
6	Software-in-the-Loop Simulation for Early-Stage Testing of AUTOSAR Software Component	2	[8]
7	An Automated Testing Method for AUTOSAR Software Components Based on SiL Simulation	2	[9]
8	Simulation-Guided Approaches for Verification of Automotive Powertrain Control Systems	1	[10]
9	Relative Time Synchronization of Distributed Applications for Software-in-the-loop Simulation	2	[11]
10	An Incremental V-Model Process for Automotive Development	1	[12]
11	Controlled time series generation for automotive software-in-the-loop testing using GANs	2	[13]
12	SilGAN: Generating driving maneuvers for scenario-based software-in-the-loop testing	2	[14]
13	Empirical Testing of Automotive Cyber-Physical Systems with Credible Software-in-the-Loop Environments	2	[15]
14	Emphasis on Evaluative Prerequisites for Decisive Software-in-the-Loop (SiL) Environments	2	[16]
15	Software-in-the-loop Modeling and Simulation Framework for Autonomous Vehicles	2	[17]
16	Testing Automotive Embedded Systems under X-in-the-Loop Setups	2	[18]
17	Formal verification of automotive embedded software	1	[19]

Table 2: Primary Research Included

mitigate this, we ensured that the selection criteria were clearly defined and followed a systematic approach. Additionally, while a formal quality assessment of the studies was not conducted, we incorporated multiple rounds of review and discussion with practitioners to enhance the robustness of the conclusions. Our mitigation strategy is similar to the one followed by CARTAXO in [3]. These measures were taken to reduce potential biases and increase the reliability of our findings, although these factors should still be considered when interpreting the results.

4 ACTION RESEARCH RESULTS

The rigidity and limitations of the V-cycle for automotive software verification, as discussed by Liu *et al.* [12], reveal even greater weaknesses in tests that do not consider the complex system in which the software will operate. In the V-cycle, system verification activities occur in the final stage of development, while software verification takes place in the early and intermediate stages. However, software verification is often conducted through static analyses and non-standardized tests, which are unable to measure the software's behavior after its integration with the vehicle's software and its deployment on the hardware (ECU - Engine Control Unit). These tests, ineffective for complex systems, result in late-stage rework, as issues are only detected during system verification.

Falcini and Lami [5], in their empirical study using the SPICE methodology, evaluated the entire development cycle and found inferior performance in software unit verification (SWE.4) and software integration verification (SWE.5) practices, further reinforced by the superiority of system verification compared to software verification. They also identified that the lack of standardized testing practices significantly contributed to this maturity gap, leading to less effective software verification processes and late detection of failures that could have been identified earlier.

> Answer to RQ1: The lack of standardization in testing practices during software verification. The effectiveness of tests is limited, as they fail to replicate environments that reflect the real-world context in which the software will operate.

In the development cycle, there are two main approaches to testing: Formal Verification and Simulation-Guided. Torodov *et al.* [19] detail the application of Formal Verification, which, despite offering a high degree of reliability, comes with significant challenges in terms of complexity and scalability for large and complex industrial systems, such as automotive control codes. For this reason, we opted to follow the Simulation-Guided approach.

Kapinski *et al.* [10] used the Simulation-Guided approach to conduct validation simulations. Accurate models of the systems are created in open or closed loops, simulating the operating environment at different stages of maturity. For the purpose of this rapid review, the focus was on Closed-Loop to ensure the highest level of verification rigor.

Simulation environments can verify: the software model alone, MiL (Model-in-the-Loop); the simulated operating environment combined with the software, SiL (Software-in-the-Loop); the software on the ECU hardware, PiL (Processor-in-the-Loop); or the software on the ECU hardware in communication with the vehicle hardware simulator, HiL (Hardware-in-the-Loop).

To improve software verification in the SWE.4 and SWE.5 phases, making it as robust as system verification, it is essential to bring the testing environment closer to real-world conditions. Therefore, the use of Simulation-Guided closed loops in Software-in-the-Loop (SiL) becomes crucial, as it simulates the vehicle environment without the need for actual hardware.

Answer to RQ2: Simulation-Guided with Closed-Loops in Software-in-the-Loop (SiL)

Barhate [2] emphasizes the importance of defining test cases focused on reduction, prioritization, and automation—techniques that should be integrated into the SiL environment to ensure a more efficient and targeted verification.

In the following sections, we will address development criteria and system credibility, explore improvements for more realistic simulations, temporal synchronization, and integration with other in-Loop systems. We will also discuss how SiL can be adapted to the new challenges of automotive systems, particularly in electric and autonomous vehicles. All theoretical foundations and methodologies mentioned here are detailed in **??** and **??** as the basis for the discussions and recommendations presented.

4.1 Development Pipeline

4.1.1 Prerequisite Assessment. Koppa *et al.* [16] highlight the importance of assessing prerequisites for creating reliable and efficient SiL environments. This assessment is necessary to identify and select software components, evaluate available information, and ensure that all necessary elements are integrated into the vECU (virtual electronic control unit). The main prerequisites assessed include:

- The software architecture must be well-defined, comprising software elements or components, their relationships, and properties. The architecture assessment ensures that all definitions, configurations, and elements are available and correct.
- Evaluation of software functionalities, including control algorithms, main functions and sub-functions, and raster rates. These details are essential for the configuration of the vECU.
- Software interfaces, or Port Interfaces, must be evaluated to ensure the correct exchange of data between components. In AUTOSAR environments, these interfaces are defined and implemented according to standards.
- The AUTOSAR architecture follows a three-layer structure, and the configuration of the software layers must be assessed to ensure proper integration of components.
- The configuration of software components, including the management of stubs and the integration of protocols such as CAN, LIN, Ethernet, and FlexRay, must be carefully evaluated.
- Identification and definition of test concerns, including function calls, sequential execution, variable states, and operation modes, to ensure that all critical aspects of the software are tested.
- The qualification of the SiL environment and its constituent elements is necessary to ensure that the vECU and the simulation environment meet the expected performance and accuracy criteria.

4.1.2 Implementation and Automated Testing. Jeong *et al.* [8] propose in their research a simulation-based testing method for AUTOSAR software components using the Software-in-the-Loop (SiL) concept. The aim of the research is to validate software components in the early stages of development, before the availability of real hardware, allowing for rapid prototyping and early error detection.

Software development following the AUTOSAR methodology employs a structured and standardized approach, facilitating the modular and scalable creation of embedded systems for automobiles. The initial process involves system configuration, where software component models are created and their interactions defined through the Virtual Functional Bus (VFB). This step generates a system description document in ARXML format, serving as a reference for subsequent phases.

Next, developers implement the internal behavior of the software components as specified in the ARXML document, ensuring each component performs its designated functions. Following this stage, components are mapped to the ECUs and subsystems, including the integration of basic software (BSW), which comprises the operating system, device drivers, and communication modules. All elements are integrated to form the executable software intended for specific ECUs, allowing its installation in the vehicle.

The AUTOSAR architecture, which includes the VFB, the Runtime Environment, and the Basic Software, is designed to be hardware-independent. VFB simulation allows software components to interact through a common AUTOSAR interface, eliminating the need for physical hardware.

To implement VFB simulation, software component descriptions are converted into C code, representing the port connections and communication interface between components. Communication is facilitated by the use of shared memory for data exchange. The simulated VFB has an event manager to handle temporal events and other triggers, as well as a scheduler that coordinates the execution of components based on time. The result is a code simulator that can run on a PC, faithfully simulating the behavior of software components as if they were in a real system.

Continuing their research in SiL, Jeong *et al.* [9] conducted a case study on an automated method for testing AUTOSAR software components based on SiL simulation. Understanding the limitations of manual testing methods, they introduced an automated testing module, which in its execution, automatically collects test case results during simulation, eliminating the need for manual intervention and increasing the efficiency of the testing process.

A method was developed to convert test cases into temporal format, facilitating the automation of tests in closed-loop control systems. The analysis and comparison of simulation results are performed automatically by the testing module, which records successes and failures in detailed reports, allowing for more rigorous and efficient validation of software components.

4.2 Credibility of the Simulation

The methodology for validating the credibility of SiL published by Raghupatruni *et al.* [15] promotes the reliability, efficiency, and safety of automotive cyber-physical systems. The separation of concerns and the definition of specific environments ensure rigorous testing of both functional and non-functional aspects, increasing system reliability. The approach facilitates adaptation to open-context environments and supports continuous updates.

4.2.1 Separation of Concerns. Each functional aspect (what the software should do) and non-functional aspect (how the software

should behave under different conditions) must be addressed independently.

4.2.2 Define Domain-Specific Simulation Environments. Each simulation environment is configured to test specific aspects of the system:

- Feature SiL (F): Focuses on testing specific software functionalities, ensuring individual functions operate as expected.
- Component SiL (C): Focuses on testing individual components, such as electronic control units (ECUs), assessing both their functional and non-functional capabilities.
- System Integration SiL (S): Focuses on integrating complete systems, including multiple components and their interactions, ensuring all elements work together harmoniously and efficiently.

4.2.3 Establish a Test Selection Method. The test selection ensures that all relevant aspects, including functional and non-functional concerns, are addressed. Tests should cover the verification of designs and the falsification of models to ensure they meet specified requirements.

4.2.4 Establish and Maintain a Socio-Technical System. Collaboration between people, processes, and technologies to ensure all aspects of the software are verified and validated throughout the product lifecycle. This includes coordination between different teams and maintaining contracts that specify performance and functionality requirements.

4.3 Realism and Control

Parthasarathy *et al.* [13], collaborators from the Volvo Group and Chalmers University of Technology, used controlled generation of time series for SiL testing using Generative Adversarial Networks (GANs). The goal of using this technology is to expose the system to a variety of input scenarios that could occur under real operating conditions, allowing for more comprehensive and efficient validation and verification of automotive software.

To simulate input values in SiL, vehicle operation sequences under various conditions are reproduced, and the signals generated during operation are collected and used as a learning base for the GAN model. The GAN uses these recorded signals to learn to generate new synthetic signal sequences that mimic real data. Thus, the model can create various realistic input patterns for testing, without the need for repeated field tests.

The SiL method allows continuous integration and verification of systems by running the System Under Test in a virtual environment. However, generating realistic and varied input stimuli is a challenge, as the input ranges are broad and continuous, requiring systematic coverage.

GANs are unsupervised learning models composed of a generator and a discriminator. The generator creates samples that the discriminator tries to distinguish from real ones. The goal is for the generator to create fake data that resembles real data, while the discriminator tries to differentiate between real and fake data. The generator continuously improves its samples to fool the discriminator, and the discriminator enhances its differentiation capability. This iterative process results in the generation of extremely realistic data.

To generate input stimuli, the VAE/GAN model uses two techniques: VAE (Variational Autoencoder) and GAN. The VAE compresses the data and attempts to reconstruct it, while the GAN improves the quality of these reconstructions, making them more realistic.

Advancing their research, Parthasarathy *et al.* [14] adopted a new approach using GANs called SilGAN. Unlike the hybrid VAE/GAN model used previously, SilGAN focuses on generating specific driving maneuvers for SiL testing. While the VAE/GAN model focused on creating realistic time series inputs, SilGAN significantly advances by automating the generation of stimuli for tests, using templates to specify driving scenarios.

This new method not only facilitates the creation of realistic multi-dimensional signal transitions but also expands these maneuvers to include transitions before and after the main event, providing a more complete and realistic view of vehicle behavior.

4.4 Relative Time Synchronization

The automotive system is composed of multiple distributed subsystems, each responsible for specific functions such as engine control, braking, navigation, and communication. These subsystems need to operate in a coordinated and synchronized manner to ensure the efficient and safe functioning of the vehicle.

In automotive systems, communication between Electronic Control Units (ECUs) is carried out through internal networks, such as CAN. Each ECU operates independently and sends messages on-demand, that is, only when another ECU requests information or when it detects a specific condition. This efficient communication model reduces data traffic on the network and improves response to requests, ensuring that the vehicle's multiple distributed subsystems function in a coordinated and effective manner.

Lee *et al.* [11] proposed a relative time synchronization method to improve SiL performance, providing a more realistic environment with the actual vehicle environment, with distributed systems. Instead of ensuring absolute synchronization with a global clock, the relative approach synchronizes the operations of distributed systems based on their operation sequences, thus reducing overhead and improving simulation performance.

Traditional global clock synchronization methods, although precise, introduce efficiency problems. Each tick of the global clock requires synchronization of all nodes, resulting in significant overhead. The proposed relative synchronization approach eliminates the need for continuous and detailed synchronization at each global clock tick. Instead, it focuses on synchronizing the operations of distributed systems based on their own operation sequences, which is more efficient and suitable for automotive SiL simulations.

The authors presented a simplified synchronization protocol that uses only tick and ticked messages, eliminating the need for complex delay calculations and other complications found in traditional methods like IEEE 1588 [1]. This protocol is efficient and suitable for the SiL simulation environment, where relative precision is more important than absolute precision.

In contrast to the simplicity and efficiency of relative synchronization, Frehse *et al.* [6] adopted an approach that focuses on formal analysis of time effects on the properties of closed-loop control systems. Unlike the relative synchronization used in SiL

simulations, which prioritizes efficiency, this technique integrates timing models with functional models to verify control system properties in detail. Using hybrid automate, which combine continuous dynamics and discrete system events, it is possible to consider effects such as jitter and latency.

Synchronization is crucial here to ensure that software controllers respond within expected time limits, avoiding issues like overshoot and variable response times. This approach allows rigorous verification of the system's functional properties in the presence of temporal effects, ensuring that control systems meet performance specifications in a realistic industrial environment.

4.5 Synchronization: SiL-XiL

The integration of Software-in-the-Loop (SiL) with other in-Loop methodologies, such as Model-in-the-Loop (MiL), Processor-in-the-Loop (PiL), and Hardware-in-the-Loop (HiL), collectively known as XiL, enables verification throughout the entire development cycle. The focus of this rapid review is on SiL, as it is a critical point that often leads to rework. However, it should be noted that the greatest benefit will be achieved through XiL integration, and therefore, its integration techniques should be carefully studied.

In research conducted by Tibba *et al.* [18], collaborators from ETAS GmbH and the Institute of Real-Time Computer Systems at the Technical University of Munich, the X-in-the-Loop (XiL) approach for testing automotive embedded systems was explored.

The research reviews various works related to the development and implementation of the Functional Mockup Interface (FMI) standard. Since the first release of the standard, efforts have been made to implement and test functional mockup units (FMUs). Several studies explored the creation of FMI-based simulation environments for continuous systems, deterministic execution of FMUs for co-simulation, and the limitations and extensions of using FMI in electronic control unit (ECU) software.

The Functional Mockup Interface (FMI) standard allows the integration of simulation models from different tools, promoting interoperability and co-simulation in complex system development environments, such as automotive embedded systems. Functional mockup units (FMUs) are software packages that encapsulate simulation models, facilitating the exchange between different simulation tools. Each FMU contains a model description file, model equations, and optional resource files. The description file includes definitions of variables, units, and general information about the model.

There are two variants of the FMI standard: FMI for Model Exchange (FMI-ME) and FMI for Co-Simulation (FMI-CS). The FMI-ME variant allows the exchange of models described by differential, algebraic, and discrete equations, which are solved by the host simulation tool's solver. This variant offers flexibility in handling the synchronization of continuous and discrete models. The FMI-CS variant facilitates co-simulation where the simulation models include their own solvers. Communication between models occurs at discrete communication points, allowing each model to be solved independently between these points.

FMI functions include initialization and instantiation, progression, getters and setters for reading and setting values, and termination for unloading components and freeing memory. The integration and co-simulation of models from different tools are facilitated by the FMI standard. This is important for the development of mechatronic systems that require the combination of continuous, discrete, and event models. Co-simulation with FMI allows models to operate as black boxes, reacting to inputs and producing outputs at discrete time steps. Synchronization between models is managed by a master algorithm that coordinates the execution of FMUs.

The orchestration of FMUs for hybrid simulation is central to the XiL approach, allowing the integration of continuous and discrete models on a single simulation platform. The appropriate choice of FMI variants (FMI-ME for model exchange and FMI-CS for co-simulation) is necessary to meet synchronization and accuracy in simulations.

Experimental results demonstrated the capability of the XiL approach to provide a seamless transition between MiL and SiL, maintaining simulation accuracy and performance. The study highlighted the efficiency in reusing test cases between different XiL levels and product generations, showing the practical applicability of the approach in the development of automotive embedded systems.

> Answer to RQ3: Individually, verification loops ensure software quality at different stages of development. However, the integration between these loops significantly enhances this quality. In MiL, control algorithms and designed functions are tested in a simulated environment, ensuring they work as expected. This allows the focus in SiL to be exclusively on the integration of the software with the simulated environment, ensuring that the code behaves correctly under conditions that closely resemble real-world operations.

4.6 Adaptation for Electric and Autonomous Vehicles

The growing demand for electric vehicles (EVs) and autonomous vehicles is rapidly shaping the automotive industry, highlighting the need for advanced testing and simulation methodologies. The use of XiL techniques, particularly SiL, is crucial to ensure the safety, performance, and reliability of these complex systems.

Casolino *et al.* [4] developed an innovative testing system for automotive electric drives using a real-time platform where the electric vehicle model is implemented. This system simulates motor and load behavior, allowing for an accurate assessment of electric drives under different control methods.

The testing system is based on an inverter controlled by a real-time platform that emulates motor and load behavior at the drive terminals. Three control methods are compared through simulation, showing real-time results for different load hypotheses. The architecture of the testing system is described in detail, highlighting the need to precisely select the parameters of the virtual load

to adequately replace the synchronous permanent magnet motor (PMSM) model that represents the electric vehicle.

Ahamed *et al.*[17] addressed autonomous vehicles by discussing a SiL modeling and simulation framework developed using the Robot Operating System (ROS) and Gazebo. This framework allows the creation of multiple vehicle models and customized environments, facilitating the validation and comparison of different algorithms in various scenarios.

The MIR (Mobile Intelligent Robotics) framework developed by Ahamed *et al.* [17] offers a complete set of tools to add vehicle models and environments. The integration of sensors such as cameras, LiDAR, and RADAR into the vehicle model allows for comprehensive validation of perception and control systems, essential for the development of autonomous vehicles.

Gupta *et al.* [7] present a modular X-in-the-Loop (XiL) framework for the validation of connected and autonomous vehicles (CAVs). This innovative framework allows the integration of vehicles with traffic simulations, autonomy software, computational resources, and human drivers through the use of a mixed reality headset. This setup enables the validation of CAVs in a controlled environment, reducing the need for public road testing and increasing the safety and efficiency of the validation process.

The virtual environment is realized through traffic microsimulation software, populating virtual vehicles, traffic lights, road signs, etc. The virtual environment can interact with the real experimental vehicle, transmitting its current orientation, position, speed, and acceleration, being replicated as a digital twin in the simulation. The use of the Robot Operating System (ROS) software to implement autonomy on the vehicle platform accelerates the integration and deployment of self-driving algorithms, configuring a modular and hierarchical hierarchy that allows the execution of steering and acceleration commands based on the planner's reference trajectory.

The framework allows human-driven vehicles to be included in the traffic flow of CAVs to investigate how their interaction affects the control and energy consumption of autonomous vehicles. A mixed reality device is added as an onboard simulation platform, allowing drivers to visualize the test site augmented with virtual traffic objects. This approach facilitates the study of human reactions in mixed traffic environments (autonomous and human).

5 DISCUSSION

As automotive systems become increasingly complex, ensuring that software verification processes accurately simulate real-world conditions is essential. The effectiveness of verification loops, such as Model-in-the-Loop (MiL), Software-in-the-Loop (SiL), Processor-in-the-Loop (PiL), and Hardware-in-the-Loop (HiL), plays a vital role in the development of safe and reliable automotive software. However, despite advancements in these methodologies, there are still significant technical gaps that prevent these loops from fully replicating the complexities of a real system.

To better understand these limitations, it is important to identify the key challenges that hinder the achievement of a truly realistic simulation. These challenges not only impact the accuracy of the verification process but also limit the ability to detect and resolve potential issues before the software is implemented in the vehicle. Due to their greater integration with system verification and their physical hardware structure, HiL and PiL do not present the greatest challenges within the XiL approach. Instead, MiL and SiL are the most challenging verification loops for development, particularly SiL, which must computationally simulate the complexity of the vehicle for integration testing.

A weakness of SiL is the limitation in simulating inputs to the system. With the aid of GAN, the ability to create realistic and varied input stimuli was explored, expanding test coverage and allowing for more comprehensive and efficient software validation. This technology evolved into SilGAN, focused on generating specific driving maneuvers, demonstrating the effectiveness of GANs in automating and improving the quality of SiL tests.

The automotive system is composed of multiple distributed subsystems, each ECU operating independently and sending messages on demand. Operating with absolute synchronization and a global clock is not the best solution to simulate a realistic environment within the vehicle. Relative time synchronization and formal analysis of the effects of time on control system properties are complementary approaches aimed at improving the accuracy and efficiency of SiL simulations. Precise temporal synchronization is crucial to ensure the consistency and reliability of results, especially in complex distributed systems.

Another factor previously mentioned in the research is the need for XiL integration. SiL is particularly dependent on this integration, especially with MiL, due to the inherent complexity of the computational simulation of the vehicle. Any functional error in the code that disrupts the tests can result in a complete rework of the entire integration within the test suite, making this integration even more essential to ensure the efficiency and accuracy of the verification process.

In concluding the answers and perspectives presented in this rapid review on the gaps to be addressed in the development and research focused on SiL, it is essential that all technology and software development in the automotive sector keep pace with the sector's evolution, regardless of its individual rate of progress. Currently, the automotive sector is moving towards electric and renewable energy-powered vehicles, as well as autonomous vehicles. All developed technology must not only meet these new demands but also evolve in parallel with the sector's continuous development, thereby ensuring the relevance and effectiveness of the implemented solutions.

Answer to RQ4: Integration in the operation of XiL loops;
Limitations in purely computational simulations, SiL;
Behavior of inputs and outputs with vehicle dynamics;
Temporal synchronization between ECUs in SiL;
Modeling of electrical systems in simulators and test cases
for autonomous vehicles.

6 CONCLUSIONS

The increasing complexity of embedded systems in the automotive industry demands more efficient methods for software verification and validation. Improvements in development cycles, such as the W and V-INC cycles, demonstrate the importance of early inclusion of verification activities and continuous integration of models and test

cases. However, Simulation-Guided techniques, especially Software-in-the-Loop (SiL), stand out as the most effective approach for software verification in simulated environments.

SiL enables continuous validation of software components before system integration, eliminating the dependence on real hardware and significantly reducing rework and costs. The use of Generative Adversarial Networks (GANs) to create realistic input stimuli further enhances the effectiveness of SiL tests, while precise temporal synchronization ensures the consistency and reliability of results.

The adaptation of SiL techniques for electric and autonomous vehicles is essential, requiring specific monitoring of electric drives and restructuring to simulate new test conditions. The modular X-in-the-Loop (XiL) approach for the validation of connected and autonomous vehicles offers a robust solution to these challenges.

In summary, the adoption of advanced simulation and automation techniques in automotive software verification is fundamental to addressing the challenges posed by the increasing complexity of embedded systems, ensuring continuous and efficient validation, and improving the quality and safety of automotive systems.

7 ARTIFACT AVAILABILITY

A replication package has been made available, to allow for independent verification and replication. Our data, including files and primary research table, and evidence briefing, are available at https://github.com/LarissaPestana/RapidReviewSAST.git.

ACKNOWLEDGMENTS

This work was partially supported by a grant from the National Council for Scientific and Technological Development (Grant CNPq-Universal 408651/2023-7).

REFERENCES

[1] 2008. IEEE Standard for a Precision Clock Synchronization Protocol for Networked Measurement and Control Systems. https://standards.ieee.org/ieee/1588/6825/

[2] Sujit Sopan Barhate. 2015. Effective test strategy for testing automotive software. In *2015 International Conference on Industrial Instrumentation and Control (ICIC)*. 645–649. https://doi.org/10.1109/IIC.2015.7150821

[3] Bruno Cartaxo, Gustavo Pinto, and Sergio Soares. 2018. The Role of Rapid Reviews in Supporting Decision-Making in Software Engineering Practice. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (Christchurch, New Zealand) *(EASE '18)*. Association for Computing Machinery, New York, NY, USA, 24–34. https://doi.org/10.1145/3210459.3210462

[4] Giovanni Mercurio Casolino, Milad AlizadehTir, Alessandro Andreoli, Mariano Albanesi, and Fabrizio Marignetti. 2016. Software-in-the-loop simulation of a test system for automotive electric drives. In *IECON 2016 - 42nd Annual Conference of the IEEE Industrial Electronics Society*. 1882–1887. https://doi.org/10.1109/IECON.2016.7794145

[5] Fabio Falcini and Giuseppe Lami. 2021. System and Software Testing in Automotive: an Empirical Study on Process Improvement Areas. In *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*. 253–262. https://doi.org/10.1109/ICST49551.2021.00035

[6] Goran Frehse, Arne Hamann, Sophie Quinton, and Matthias Woehrle. 2014. Formal Analysis of Timing Effects on Closed-Loop Properties of Control Software. In *2014 IEEE Real-Time Systems Symposium*. 53–62. https://doi.org/10.1109/RTSS.2014.28

[7] Prakhar Gupta, Rongyao Wang, Tyler Ard, Jihun Han, Dominik Karbowski, Ardalan Vahidi, and Yunyi Jia. 2023. An X-in-the-Loop (XIL) Testing Framework for Validation of Connected and Autonomous Vehicles. In *2023 IEEE International Automated Vehicle Validation Conference (IAVVC)*. 1–6. https://doi.org/10.1109/IAVVC57316.2023.10328040

[8] Sooyong Jeong, Yongsub Kwak, and Woo Jin Lee. 2016. Software-in-the-Loop simulation for early-stage testing of AUTOSAR software component. In *2016 Eighth International Conference on Ubiquitous and Future Networks (ICUFN)*. 59–63. https://doi.org/10.1109/ICUFN.2016.7536980

[9] Sooyong Jeong and Woo Jin Lee. 2017. An automated testing method for AUTOSAR software components based on SiL simulation. In *2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN)*. 278–283. https://doi.org/10.1109/ICUFN.2017.7993793

[10] James Kapinski, Jyotirmoy Deshmukh, Xiaoqing Jin, Hisahiro Ito, and Ken Butts. 2015. Simulation-guided approaches for verification of automotive powertrain control systems. In *2015 American Control Conference (ACC)*. 4086–4095. https://doi.org/10.1109/ACC.2015.7171968

[11] Sunghee Lee, Bueng Il Hwang, Kang-Bok Seo, and Woo Jin Lee. 2016. Relative Time Synchronization of Distributed Applications for Software-in-the-Loop Simulation. In *2016 IEEE Intl Conference on Computational Science and Engineering (CSE) and IEEE Intl Conference on Embedded and Ubiquitous Computing (EUC) and 15th Intl Symposium on Distributed Computing and Applications for Business Engineering (DCABES)*. 753–756. https://doi.org/10.1109/CSE-EUC-DCABES.2016.273

[12] Bohan Liu, He Zhang, and Saichun Zhu. 2016. An Incremental V-Model Process for Automotive Development. In *2016 23rd Asia-Pacific Software Engineering Conference (APSEC)*. 225–232. https://doi.org/10.1109/APSEC.2016.040

[13] Dhasarathy Parthasarathy, Karl Bäckstrom, Jens Henriksson, and Sólrún Einarsdóttir. 2020. Controlled time series generation for automotive software-in-the-loop testing using GANs. In *2020 IEEE International Conference On Artificial Intelligence Testing (AITest)*. 39–46. https://doi.org/10.1109/AITEST49225.2020.00013

[14] Dhasarathy Parthasarathy and Anton Johansson. 2021. SilGAN: Generating driving maneuvers for scenario-based software-in-the-loop testing. In *2021 IEEE International Conference on Artificial Intelligence Testing (AITest)*. 65–72. https://doi.org/10.1109/AITEST52744.2021.00022

[15] Indrasen Raghupatruni, Thomas Goeppel, Muhammed Atak, Julien Bou, and Thomas Huber. 2019. Empirical Testing of Automotive Cyber-Physical Systems with Credible Software-in-the-Loop Environments. In *2019 IEEE International Conference on Connected Vehicles and Expo (ICCVE)*. 1–6. https://doi.org/10.1109/ICCVE45908.2019.8965169

[16] Kushal Koppa Shivanandaswamy, Chandrima Sarkar, Sivakumar Rajagopal, Lakkappa Pisutre Ramachandra, and Chiranjeevi Manchasandra Chayakumar. 2021. Emphasis on Evaluative Prerequisites for Decisive Software-in-the-Loop (SiL) Environments. In *2021 8th International Conference on Computing for Sustainable Global Development (INDIACom)*. 450–457.

[17] Mohamed Fasil Syed Ahamed, Girma Tewolde, and Jaerock Kwon. 2018. Software-in-the-Loop Modeling and Simulation Framework for Autonomous Vehicles. In *2018 IEEE International Conference on Electro/Information Technology (EIT)*. 0305–0310. https://doi.org/10.1109/EIT.2018.8500101

[18] Ghizlane Tibba, Christoph Malz, Christoph Stoermer, Natarajan Nagarajan, Licong Zhang, and Samarjit Chakraborty. 2016. Testing automotive embedded systems under X-in-the-loop setups. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 1–8. https://doi.org/10.1145/2966986.2980076

[19] Vassil Todorov, Frédéric Boulanger, and Safouan Taha. 2018. Formal Verification of Automotive Embedded Software. In *2018 IEEE/ACM 6th International FME Workshop on Formal Methods in Software Engineering (FormaliSE)*. 84–87.

[20] VDA QMC Working Group 13 / Automotive SIG 2017. *Automotive SPICE Process Assessment Model* (version 3.1 ed.). VDA QMC Working Group 13 / Automotive SIG. Available at: https://www.vda-qmc.de/en/publications/software-process-assessment-automotive-spice/.