Passive Testing of Vehicular Embedded Systems: An Industrial Case Study with T-EARS and Napkin Studio

Aleksandra Nicaj[1](✉), Daniel Flemström[2], Eduard Paul Enoiu[1], and Wasif Afzal[1]

[1] Mälardalen University, Västerås, Sweden
anj22001@student.mdu.se, {eduard.paul.enoiu,wasif.afzal}@mdu.se
[2] RISE Research Institutes of Sweden, Västerås, Sweden
daniel.flemstrom@ri.se

Abstract. Passive testing is an approach to verify system behavior by observing logs from normal operation, without actively injecting test stimuli. This paper presents an industrial case study of applying passive testing in the domain of vehicular embedded systems, utilizing two specialized tools: Timed Easy Approach to Requirements Syntax (T-EARS) for specifying temporal requirements, and Napkin Studio for evaluating these requirements against real system execution logs. We collaborated with Volvo Construction Equipment (VCE) to translate a set of natural language requirements into structured T-EARS specifications. Then we used Napkin Studio to test these requirements against recorded machine log data passively. We evaluate the feasibility of this approach, the extent to which it can detect requirement violations or injected faults, and the perceptions of industry stakeholders regarding the adoption of such passive tests in their verification process. The results show that a majority of functional requirements can be expressed as Guarded Assertions (GAs) and validated on logs, uncovering specific issues. Stakeholders found the method promising for improving test coverage and efficiency, although integration challenges (e.g., log signal inconsistencies and tool usability issues) were noted. Overall, this work provides empirical evidence that passive testing with T-EARS and Napkin Studio can complement traditional hardware-in-the-loop testing, offering a scalable and non-intrusive verification approach in developing vehicular systems.

Keywords: Passive testing · embedded systems · T-EARS · Napkin Studio · requirements engineering · vehicular systems

1 Introduction

Embedded systems are dedicated computing units found in vehicles, aircraft, and medical devices, where reliability is critical. As they become more complex and interconnected, ensuring correct behavior in all conditions becomes increasingly challenging [19]. Ensuring the reliability of safety-critical embedded systems is of crucial importance in domains such as aerospace and automotive. Failures, such as the Boeing 737 MAX crashes, which were attributed in part to untested scenarios, underscore the catastrophic consequences of inadequate quality assurance [20]. Similarly, incidents in the automotive domain (e.g., unintended acceleration issues or autonomous driving system failures) highlight the need for rigorous verification and validation (V&V) of embedded software [15,16].

1.1 Passive Testing

Traditionally, active testing methods, which involve providing controlled inputs to a system and observing the outputs, have been widely used to verify the behavior of embedded systems [1]. While effective, active testing can be resource-intensive and may not cover all possible operational scenarios, particularly since vehicle systems often contain complex, distributed software with numerous interacting components. As a result, there is growing interest in passive testing [10], which observes and checks system behaviors during normal operation (using logs or monitors) without external intervention. Passive testing can potentially reveal issues that only manifest under real-world usage patterns and can improve the scalability of testing by leveraging in-field data.

1.2 Industrial Context and Tools Used

In this work, we explore a passive testing approach focusing on the embedded systems in the H-Series wheel loaders developed by Volvo Construction Equipment (VCE). These machines have software responsible for critical functions like braking, steering, and safety alarms. VCE's current V&V process relies mainly on manual and HIL testing. HIL testing remains essential for validating software-level safety requirements under ISO 26262 [18], but it is time-consuming and does not scale well as system complexity increases.Hardware-in-the-loop (HIL) testing is a verification technique where embedded software is tested in real time against simulated hardware environments, enabling validation of system behavior without requiring a fully assembled physical system. We propose to complement these with a structured passive testing framework using T-EARS (Timed Easy Approach to Requirements Syntax) and the Napkin Studio tool[1] [5].

T-EARS is an extension of the EARS (Easy Approach Requirements Syntax) requirements notation that incorporates timing constraints, enabling translation of natural language requirements into formal, executable checks. Napkin Studio provides an environment for specifying these T-EARS-based requirements and evaluating them against system log data in a non-intrusive manner [6]. For example, instead of actively injecting a test scenario (such as pressing a brake pedal in different conditions), we can use passive testing to verify from operational logs that the brake light activation requirement holds in all scenarios

[1] Source code available at: https://github.com/danielFlemstrom/napkin.

where the brake pedal was pressed. This allows scenario-independent validation on a large set of recorded data, potentially improving test coverage and efficiency [5,6]. Runtime verification methods aligned with ISO 26262 have also been proposed to monitor system properties at runtime [13], highlighting the feasibility of passive embedded monitors in safety-critical contexts.

1.3 Research Goals and Contributions

The goal of our study is to evaluate the feasibility and benefits of using T-EARS and Napkin Studio for passive testing in an industrial setting. The main contributions are as follows:

- We present a structured method for converting natural language requirements into T-EARS test logic and categorizing translation difficulty for 74 real requirements from a vehicular system.
- We implement and evaluate passive testing on 12 system integration test cases using actual vehicle log files, demonstrating the ability to detect safety requirement violations and discussing limitations.
- We report insights from industrial stakeholders on the practicality of applying passive testing and lessons learned (e.g., tooling challenges and data quality issues).
- We include a researcher's reflection, where notes on the entire process have been taken from the beginning of the research, resulting in detailed feedback given to the industrial stakeholder for further adoption of passive testing.

Unlike earlier studies [5,7,9] in the railway domain, where T-EARS was introduced by its creators and evaluated on requirements captured during an experimental testing session, the present work applies T-EARS in a new domain using a HIL regression test suite and requirements that were transformed for passive testing by a researcher (the first author) who had no prior involvement in developing T-EARS or Napkin Studio. Furthermore, this study is the first to apply the passive testing approach and toolset specifically to requirements used within HIL testing environments for a construction vehicle embedded system.

1.4 Paper Structure

This paper is organized as follows. Section 2 reviews related work. Section 3 explains the case study design. Section 4 details the implementation of passive testing in the vehicular context. Section 5 presents the results of applying passive testing in practice. Section 6 presents a discussion, and Sect. 7 concludes the paper, outlining directions for future research.

2 Related Work

Passive testing has been studied as a non-intrusive complement to active testing. Cavalli et al. [4] survey formal methods for both, emphasizing how passive testing checks the conformity of system traces to specifications without altering behavior. For timed and distributed systems, frameworks based on extended finite-state machines and timed automata have been proposed to detect violations in logs [2,3]. Beyond automotive, passive testing has also been applied to networks and cloud systems, with more than 100 techniques cataloged in recent surveys [14].

Our work builds on these ideas by extending them into a vehicular embedded systems setting using structured requirement-based logic. The concept of Guarded Assertions (GAs), runtime checks derived from requirements, was first introduced as Independent Guarded Assertions in earlier work on model-based testing using Uppaal [12]. This concept was further developed by Flemström et al. through the SAGA Toolbox [8], which supported the interactive creation and evaluation of assertions on system logs. The toolbox, later referred to as NAPKIN Studio in subsequent publications [6], also introduced the T-EARS language, a structured requirements notation designed to enable systematic derivation of GAs from natural language specifications.

Later studies refined and extended T-EARS to better support timing constraints and tolerance expressions [5,7,9]. In a 2021 industrial case study, T-EARS was used to specify passive tests for a vehicular safety-critical system based on 116 requirements, demonstrating the feasibility of applying structured passive testing at scale in an industrial context [9]. Writing testable requirements is challenging. The Easy Approach to Requirements Syntax (EARS) offers a lightweight template designed to reduce ambiguity [17]. T-EARS extends EARS by adding timing constructs like *WHEN* and *WITHIN* for temporal conditions. Prior studies have shown that structured styles, such as EARS, can aid in test automation [17]. We translated legacy free-text requirements into T-EARS and measured the effort and complexity, including the number of signals per requirement. We also categorized translation difficulty and found patterns unsuitable for passive testing, such as cross-checks and event logging.

While these studies demonstrate the potential of structured requirement-driven passive testing, evaluations based on operational field data remain limited. Our work addresses this gap by applying T-EARS and GAs to a real-world construction vehicle embedded system and by contributing an empirical analysis of the feasibility and effectiveness of passive testing in a deployment context.

3 Case Study Design and Methodology

For this study, we used a case study methodology [22] to investigate the use of passive testing for embedded vehicular systems at VCE. The goal is to explore how passive testing can be applied in practice and to identify the benefits and challenges that arise when integrating it into existing processes.

The research is guided by answering three research questions:

- **RQ1:** How can requirements from the vehicular domain be translated into structured formats suitable for passive testing in HIL environments?

- **RQ2:** How feasible is passive testing of vehicular systems using T-EARS-based requirements and corresponding system logs?
- **RQ3:** What are stakeholders' perceptions of the passive testing workflow in the context of vehicular systems development, and what insights were gained from the researcher's experience during its industrial implementation?

Figure 1 provides an overview of the workflow of implementing the passive testing workflow at VCE. The case study was conducted in six stages. Data was collected from integration and system testing, focusing on requirements (e.g., Parking Brake, Seat Belt Reminders), and real execution logs were captured using Vector CANalyzer [21]. Natural language requirements were manually translated into T-EARS specifications, which supported temporal conditions and guarded assertions, with VCE experts consulted to resolve any ambiguities. These formalized specifications were then automatically evaluated using Napkin Studio, and each requirement was categorized by translation difficulty and testability.

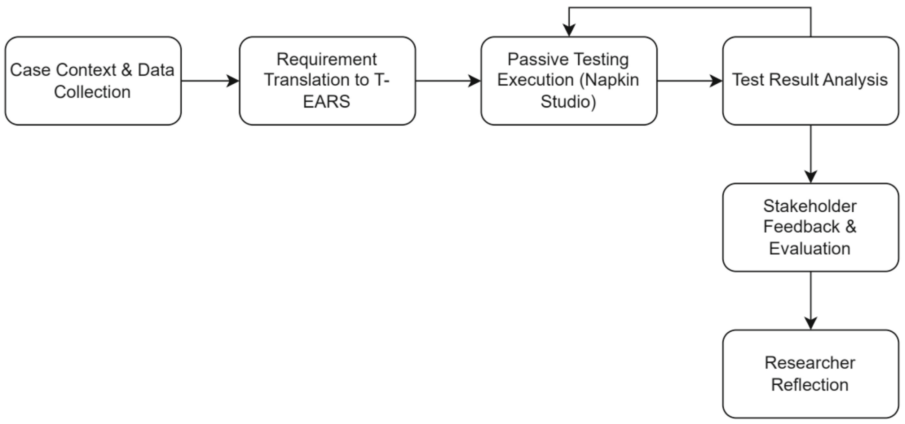

Case Context & Data Collection
Requirement Translation to T-EARS
Passive Testing Execution (Napkin Studio)
Test Result Analysis
Stakeholder Feedback & Evaluation
Researcher Reflection

Fig. 1. Overview of the passive testing implementation and evaluation process

Passive tests were executed by applying T-EARS rules to the collected logs, assessing whether requirements could be verified based on available signal data. Requirements were classified as fully testable, partially testable, or untestable, emphasizing gaps in logging coverage and requirement clarity. Stakeholder feedback was collected through a structured session with six engineers and managers, which included a technical presentation, a live demonstration, and a detailed evaluation form. Insights covered tool usability, workflow fit, potential use cases, and improvement areas. In parallel, the researcher's reflections, guided by Gibbs' Reflective Cycle [11], captured lessons learned about practical challenges, tool limitations, and opportunities to refine the passive testing process.

3.1 Stakeholder Evaluation

To understand the practical impact, we conducted a stakeholder evaluation involving five test engineers and system developers at VCE. After we obtained preliminary results from the passive test execution, we organized a demo and feedback session. We presented an overview of the passive testing workflow, demonstrated the Napkin Studio evaluation on a sample log (including how a requirement is checked and how a failure is reported), and then discussed results such as which requirements passed or failed in the logs. Stakeholders were asked to fill out a short survey (mix of Likert-scale ratings and open comments) addressing: the clarity of the T-EARS requirement syntax, the usability of the Napkin Studio interface, their trust in the passive test results, and the perceived usefulness of integrating this approach into their existing process. We also prompted them with open-ended questions about what they saw as the main benefits or drawbacks, and what would be required to adopt passive testing in their workflow. The feedback was analyzed qualitatively (looking for common themes) and quantitatively (simple counts of responses for questions like "*Would you recommend using this technique in future projects?*").

3.2 Researcher Reflection

We structured the reflection using Gibbs' six-stage model: description, feelings, evaluation, analysis, conclusion, and action plan. These reflections, focused on tool usage, technical challenges, and process insights, guided workflow improvements throughout the collaboration with VCE.

At the end of the study, we delivered four supporting documents: a T-EARS translation guide, a Napkin Studio manual, a quick-start guide for engineers, and a summary for testers and managers to support the ongoing adoption of passive testing.

4 Passive Testing Workflow Implementation

In this section, we explain how the passive testing approach was applied in practice at VCE from requirement translation to evaluation and reflection.

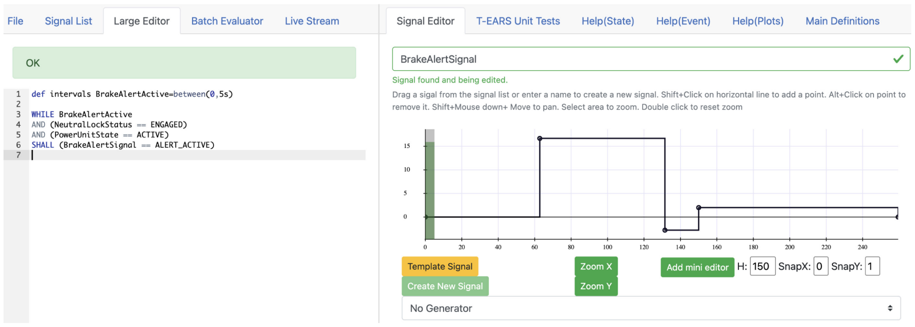
4.1 Requirements Translation with T-EARS

We gathered 74 functional requirements from VCE's internal documentation for the Parking Brake (56 requirements) and Seat Belt Reminder (18 requirements) functions. These requirements, originally in natural language, describe various expected behaviors (e.g., R1 "*If the parking brake is engaged while the machine is moving, an alarm shall sound within 2 s*"). We first pre-processed the requirements to clarify any ambiguous language and to split certain compound statements into separate, atomic requirements. Each requirement was then manually translated into the T-EARS structured format. T-EARS extends the EARS language with executable semantics and temporal evaluation capabilities for passive testing. It leverages existing EARS constructs such as WHEN (denoting triggering events) and WHILE (denoting persistent conditions), and introduces WITHIN to express timing constraints. Each T-EARS requirement is parsed and converted into an executable rule resembling a small state machine. These rules are evaluated over signal logs by detecting state transitions and enforcing timing conditions. While T-EARS is not formally grounded in LTL or timed automata, its semantics are defined operationally through these evaluators, implemented in JavaScript for rapid prototyping in Napkin Studio. This pragmatic approach enables scalable verification over real system logs, though with known trade-offs in formality and traceability. This format uses keywords such as *WHEN, AND, WHILE, THEN, and WITHIN* to define the context, trigger, and expected outcome of the requirement in a formalized way. For example, the parking brake alarm requirement (R1) was expressed as:

Listing 1.1. T-EARS Translation example

```
WHILE BrakeAlertActive
AND (NeutralLockStatus == ENGAGED)
AND (PowerUnitState == ACTIVE)
SHALL (BrakeAlertSignal == ALERT_ACTIVE) WITHIN 2s
```

This T-EARS representation (called a Guarded Assertion) captures the condition and timing explicitly. It expresses the requirement that when the neutral lock becomes engaged and the power unit is active, the brake alert signal must become active within 2 s. The WITHIN 2 s clause encodes the timing constraint explicitly, ensuring temporal correctness is evaluated during passive testing. In total, we produced 148 T-EARS test case instances (each requirement was considered in both a "positive" scenario where the event should occur and a "negative" scenario where it should not), though many followed similar templates. Each requirement is evaluated for correction in Napkin Studio, as shown in Fig. 2.



OK

```
def intervals BrakeAlertActive=between(0,5s)

WHILE BrakeAlertActive
AND (NeutralLockStatus == ENGAGED)
AND (PowerUnitState == ACTIVE)
SHALL (BrakeAlertSignal == ALERT_ACTIVE)
```

BrakeAlertSignal ✓

Signal found and being edited.

Drag a sigal from the signal list or enter a name to create a new signal. Shift+Click on horizontal line to add a point. Alt+Click on point to remove it. Shift+Mouse down+ Move to pan. Select area to zoom. Double click to reset zoom

H: 150 SnapX: 0 SnapY: 1

No Generator

Fig. 2. Requirement Evaluation in Napkin Studio

We then categorized the translation difficulty level for each requirement based on the number of distinct signals it involved, repetitiveness, and compatibility with the approach.

4.2 Log-Based Passive Test Execution

After translating the requirements, we implemented and executed passive tests using Napkin Studio. Napkin Studio is a passive testing tool that supports the import of T-EARS specifications and their evaluation against recorded system logs. It provides a graphical interface where requirements, represented as Guarded Assertions (GAs), are listed and assessed over loaded log files. Logs can be imported in formats such as CSV or JsonDiff, and test cases can be organized and executed interactively. The tool visualizes pass/fail results on a timeline, with failed GAs highlighted along with contextual details, including the triggering condition and the violated expectation, enabling engineers to trace requirement failures back to specific signal changes. Napkin Studio also supports step-by-step evaluation and result export for reporting and debugging purposes. VCE provided us with a set of system logs from wheel loader machines running the relevant software. Each log contained timestamped messages (signals and state changes) recorded during various machine operations. We first filtered and pre-processed the logs to ensure they contained the signals needed for our requirements. Out of an initial set of 70+ log files, only 12 logs had sufficient coverage of the scenarios and signals of interest (some logs were too short or missing certain sensor channels). Each of the 12 selected logs contained between 200,000 and 270,000 timestamped events, resulting in over 3 million events analyzed in total. We then loaded those 12 logs into Napkin Studio, along with the corresponding T-EARS test cases (organized into 12 test scenarios corresponding to existing HIL test cases for Parking Brake and Seat Belt functions). During evaluation, Napkin Studio automatically checks each Guarded Assertion (GA) against the log data. If a requirement's condition or triggers are met in the log, the tool verifies whether the expected outcome occurred within the specified time window. Napkin Studio provides practical vacuity detection by tracking which Guarded Assertions (GAs) are activated during log evaluation. If a requirement's triggering condition (e.g., a WHEN clause) is never satisfied in the log, the GA remains inactive and is explicitly reported as such. This allows users to distinguish between satisfied requirements and those that were not testable on a given log. This feature supports coverage analysis and helps identify scenarios or signals that are missing from test data. We treated any violation (e.g., expected event not observed, or observed outside the allowed time) as a test failure. To simulate potential faults and assess the detection capability, we also performed fault injection on some log data. For instance, we altered a copy of a log to omit a braking alarm that should have occurred, to see if the GA would catch the discrepancy. Overall, we executed 128 GAs derived from 51 distinct requirements across the 12 log scenarios[2]. Each test run produced a pass/fail result for

[2] The remaining requirements from the original 74 were either duplicates or deemed not applicable to passive testing, as discussed later.

each GA, along with timing information on when conditions were triggered. We measured the number of GAs that flagged failures in the unmodified logs (which could indicate either a genuine issue or a limitation, such as missing data) and the number that flagged failures in the intentionally fault-injected logs (which is an indicator of the fault detection capability).

5 Evaluation Results

All 74 chosen requirements were expressed in the T-EARS language, suggesting that, from a requirements engineering perspective, even these system-level informal specifications can be mapped to the GA templates. However, the complexity of translation varied. Figure 3 summarizes the difficulty categorization of the requirements. About one-third of the requirements (25 out of 74) were "easy" to translate; they involved only one primary signal or condition and mapped directly into a single GA.

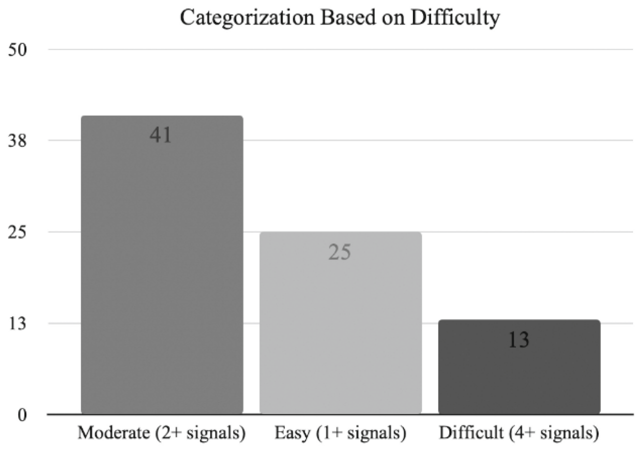


Fig. 3. Requirements Translation Difficulty Categorization

The majority (56%, 41 requirements) were of "moderate" difficulty, typically involving 2–4 signals or multiple conditions. These required careful handling but were still translatable using the T-EARS syntax. The remaining 17% (13 requirements) were judged "hard" to translate because they involved complex logic or more than four signals (often indicating multiple interacting subsystems). In these cases, multiple interpretations of the requirement were possible, or a single linear temporal rule could not capture the requirement. We often had to either break these into multiple GAs or simplify assumptions to fit the T-EARS format. A majority of requirements (moderate) involved 2–4 signals, whereas a notable subset (hard) with 5 or more signals or complex logic posed challenges for direct translation.

We further analyzed the feasibility of passive testing for each requirement. As shown in Fig. 4, after eliminating redundant requirements, 56 unique requirements remained. By reviewing each from the perspective of passive testing constraints, we found that only 26 out of these 56 (around 46%) were fully compatible with the passive approach, as shown in Fig. 5a. The compatible ones tended to be state-based or timing-based conditions that could be directly checked in a log (e.g., if event X happens, system does Y within t seconds). The other 30 requirements (about 54 %) had aspects that could not be verified solely through logs.

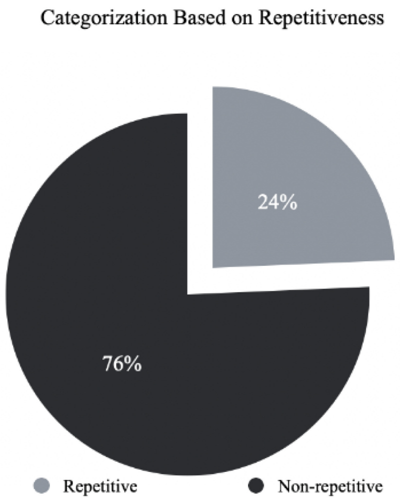
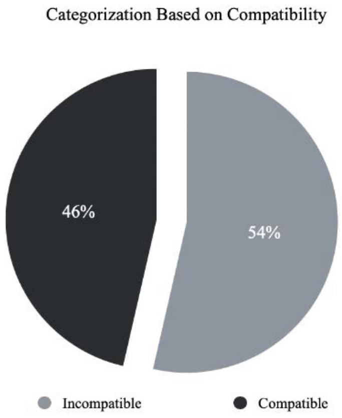
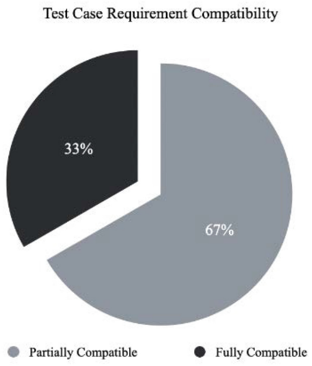


Fig. 4. Requirement Redundancy Breakdown

Reasons for this included reliance on redundancy checks, which require two separate signals to activate and can be difficult to verify if one signal is missing or the check is not explicitly recorded. Another factor was the presence of implicit system behaviors that are not captured in the available logs, such as verifying whether an internal flag was reset or an event was stored in an internal log, which external logging does not reveal. This indicates that while passive testing can validate many functional requirements, those involving redundant signal checks or internal system states still require active testing or improved logging. In our test implementation, we marked such requirements as "*untestable via passive*" and excluded them from the GA set, focusing on the 26 that were feasible.



(a) T-EARS Compatibility



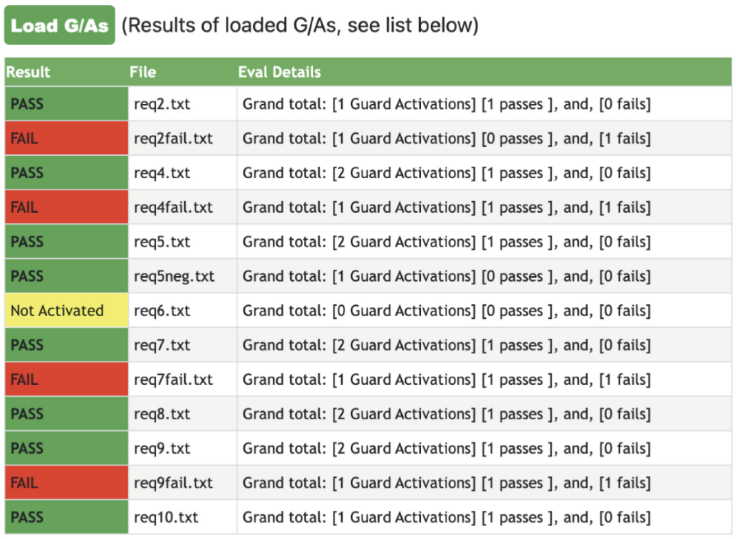
(b) T-EARS Compatibility of Requirements for a Test Case

Fig. 5. Comparison of T-EARS requirement compatibility for the entire set and an individual test case.

5.1 Passive Test Execution Results

Out of the 12 test scenario logs analyzed, as shown on Fig. 5b only four had complete signal coverage for all applicable GAs, meaning those logs were fully compatible with passive testing. The remaining eight logs were only partially compatible, with certain requirements not testable due to missing or incomplete signal data. As a result, while Napkin Studio flagged some GA failures during evaluation, these were mostly attributable to limitations in passive testing, rather than actual system faults. For example, in one scenario, a failure was reported for a redundant brake signal check simply because one of the required signals was absent in the log, rather than due to incorrect system behavior. After filtering out such false negatives caused by signal unavailability, we found no evidence of requirement violations in the unmodified logs. This indicates that, where logs contained sufficient data, the system generally behaved as expected.

We injected a total of 30 instances of altered behavior across various scenarios (e.g., removing an expected alarm activation, or changing a sensor value to simulate a stuck signal). The GAs correctly detected 20 out of 30 injected faults, immediately flagging the corresponding requirement as failed. The remaining ten injected faults did not trigger GA alerts, but this was traced to the nature of those faults: in many cases, the scenario conditions to trigger the requirement were not present in that particular log segment, so the GA never got evaluated (i.e., if we inject a fault for a situation that never occurred in the log, the passive test cannot detect it). The distribution of pass, fail, and inactive results under fault injection is shown in Fig. 6.


Load G/As (Results of loaded G/As, see list below)

Result	File	Eval Details
PASS	req2.txt	Grand total: [1 Guard Activations] [1 passes], and, [0 fails]
FAIL	req2fail.txt	Grand total: [1 Guard Activations] [0 passes], and, [1 fails]
PASS	req4.txt	Grand total: [2 Guard Activations] [1 passes], and, [0 fails]
FAIL	req4fail.txt	Grand total: [1 Guard Activations] [1 passes], and, [1 fails]
PASS	req5.txt	Grand total: [2 Guard Activations] [1 passes], and, [0 fails]
PASS	req5neg.txt	Grand total: [1 Guard Activations] [0 passes], and, [0 fails]
Not Activated	req6.txt	Grand total: [0 Guard Activations] [0 passes], and, [0 fails]
PASS	req7.txt	Grand total: [2 Guard Activations] [1 passes], and, [0 fails]
FAIL	req7fail.txt	Grand total: [1 Guard Activations] [1 passes], and, [1 fails]
PASS	req8.txt	Grand total: [2 Guard Activations] [1 passes], and, [0 fails]
PASS	req9.txt	Grand total: [2 Guard Activations] [1 passes], and, [0 fails]
FAIL	req9fail.txt	Grand total: [1 Guard Activations] [1 passes], and, [1 fails]
PASS	req10.txt	Grand total: [1 Guard Activations] [1 passes], and, [0 fails]

Fig. 6. GA Results with Fault Injection

In a couple of cases, insufficient log detail (e.g., coarse timestamp resolution) caused the GA to miss a subtle timing violation. These results indicate that passive testing can detect a large proportion of issues, assuming the relevant scenarios are captured in the logs and the log quality is sufficient. It also shows that passive testing is constrained by the logs themselves; if a scenario never occurs or required data is missing, certain issues may go undetected. From an efficiency perspective, executing the passive tests was very fast. Once the logs were loaded, Napkin Studio evaluated all 128 GAs in a matter of seconds per log. This is a stark contrast to running 12 separate HIL tests in real time (which could take hours of setup and execution). While preparing the logs and translations required initial effort, these resources can be reused continuously. The ability to monitor logs means that any new log (from field operation or continuous integration testing) can be checked against the entire suite of requirements automatically. Stakeholders pointed out that this could enable a form of regression testing on in-service data, for instance, after a software update is deployed, operational logs could be passively tested to ensure no safety requirements were violated over thousands of hours of operation.

5.2 Stakeholder Feedback

Feedback from five VCE stakeholders was overall positive, highlighting the value of passive testing as a complementary method. Participants appreciated the traceability between requirements and test results. T-EARS syntax was generally found understandable, while Napkin Studio's interface received mixed feedback. Stakeholders emphasized the need for automation, standardized logging, and low effort tooling. Table 1 summarizes the evaluation responses. Overall, they clearly saw benefit in adapting the method into their process and recommended pilot projects to explore broader adoption.

Table 1. Stakeholder Evaluation Summary (n = 5)

Evaluation Criteria	Most Frequent Answer	Key Remarks
Clarity of T-EARS Syntax	Somewhat Easy (3/5)	Described as "*easy format and easy to understand*" by one participant
Napkin Studio UI Usability	Somewhat Intuitive (2/5)	One noted "*unclear buttons and terminology*"; others found it manageable
Perceived Usefulness	Very Useful (5/5)	Considered effective for logic validation and timing checks
Trust in Results	Potentially (5/5)	Dependent on signal availability and log integrity
Workflow Fit at VCE	Needs Improvement	Issues include non-standardized signals and varying log formats
Recommend Further Use	Yes (5/5)	All respondents suggested future pilots and broader adoption

6 Discussions and Limitations

The results of this study indicate that passive testing with T-EARS and Napkin Studio is both feasible and useful in the context of vehicular embedded systems. The approach validated a considerable set of functional requirements and was adequate in detecting specific injected faults in most test scenarios considered. Stakeholders found the T-EARS syntax clear and accessible, and appreciated the traceability it provided between requirements and test outcomes. The speed and scalability of passive test execution also suggest a potential for integration into continuous testing or regression workflows. However, the study also highlighted important limitations. Passive testing depends entirely on the availability and quality of log data. Although all requirements could be formally represented using the T-EARS syntax, certain types of requirements remained untestable through passive testing due to inherent visibility constraints. Specifically, requirements involving redundant signal logic, internal flags, or memory events could not be evaluated because the required signals or system states were not observable in the available logs. This reflects a limitation of passive testing rather than the expressiveness of T-EARS. Inconsistent signal naming and incomplete coverage across logs also introduced false failures or limited the ability to evaluate certain guarded assertions. Incomplete signal coverage in several logs limited the number of GAs that could be executed. In these cases, it was the absence of key signals needed to evaluate specific requirements. This constraint led to some tests being skipped or marked as failed due to missing preconditions. Notably, fault injection experiments confirmed that when relevant scenarios were present and signal data were complete, Napkin Studio reliably detected violations. These results suggest that incomplete log coverage primarily impacts testability, not the validity of the approach itself. Improving log signal availability and consistency would enhance coverage and strengthen the evaluation. These issues underscore the need for more structured and comprehensive logging practices to fully realize the benefits of passive testing. To fully benefit from passive testing, it is necessary to improve how logs are generated during system operation. Since this testing approach depends entirely on the data captured in the logs, any missing or incomplete signals make it impossible to properly evaluate certain requirements. Better logging, including more relevant signals and clearer event capture, is essential to make passive testing reliable and effective. Although passive testing did not uncover previously unknown system faults, it revealed significant gaps in log coverage and signal availability. These insights are valuable in themselves, as they help identify limitations in current verification practices and highlight opportunities to improve future data collection strategies.

From a practical perspective, passive testing is viewed as a complement to existing methods used in the development process of vehicular embedded systems, such as HIL testing. It can efficiently validate timing and state-based behaviors during or after system operation, especially when logs are available from simulations, test rigs, or in-field deployments. To support adoption in industry, organizations should consider categorizing requirements by testability early in the development cycle and standardizing log formats to ensure signals are always recorded. While passive testing is not a replacement for active methods, its non-intrusive nature, low execution overhead, and potential for continuous verification make it a good option for inclusion in vehicular embedded system development processes. This study does not include a formal comparison with traditional verification methods such as HIL testing, active testing, or runtime verification (RV). Future work could investigate the relative effectiveness, effort, and defect detection capabilities of passive testing versus these alternative approaches in similar scenarios.

Several factors may affect the validity of this study, including subjective requirement translation, limited log data quality, tool constraints in Napkin Studio, and a small stakeholder group used in the evaluation. To address these issues, we employed a structured approach to transform requirements into T-EARS and conducted expert reviews. The empirical evaluation involved only five participants, which limits the generalizability of usability findings and stakeholder perceptions. Future studies should aim to involve a broader set of users to strengthen these insights. We selected and preprocessed logs to ensure the availability of usable data and documented the limitations of the passive testing tool. Stakeholder feedback is supported through targeted participant selection in a real-world context.

7 Conclusion and Future Work

This study indicates the feasibility and potential benefits of applying passive testing to embedded systems in a vehicular industrial context. A set of structured requirements was translated, and passive tests were capable of validating behaviors non-intrusively using existing system logs. However, our evaluation also revealed limitations: some requirements remained too complex for current tools or were untestable due to missing or inconsistent log data.

To support industrial adoption, we recommend that practitioners and researchers in the area of passive testing: (1) introduce structured logging requirements early in the development process to ensure necessary signals are captured; (2) enhance Napkin Studio with better traceability, usability, and support for requirement coverage analysis; (3) extend the expressiveness of the T-EARS language to accommodate more complex logical and temporal constructs; and (4) conduct full-scale pilots across one or several product lifecycles to assess defect detection, integration effort, and return on investment.

From the perspective of a practitioner embedded within the industrial setting, this work also highlights the value of collaboration between tool developers and practitioners. Clear requirement specification, suitable logging, and accessible passive testing methods are all critical for bridging the gap between prototypes and industrial workflows.

While this study indicates the feasibility of using T-EARS and Napkin Studio for passive testing in a vehicular context, several avenues for future exploration remain. One area is enhancing support for requirements involving redundant signals or complex dependencies that could not be validated passively. Improving signal availability and standardizing logging practices may increase the number of testable requirements used for passive testing. Another area is the automation of requirement translation into T-EARS, which could reduce manual effort and improve consistency. Future work could also involve integrating passive testing earlier in the development cycle, allowing real-time feedback during simulation or system integration phases. Finally, evaluating this approach across multiple vehicle platforms would help generalize its applicability and reveal system-specific constraints.

Acknowledgments. The authors thank Volvo Construction Equipment, especially the Verification & Validation team, for their support and collaboration. We are grateful to Jörgen Stenmark and Mrinal Kumar for their guidance, and to the engineers who participated in our evaluation for their valuable feedback. This work is further supported by the Software Center project, Eureka's Sustainability Call project SUS2022-039: NRPCES & the MATISSE project, an EU-funded initiative under Horizon Europe GA no.101056674.

Disclosure of Interests. The authors have no conflicts of interest to declare that are relevant to the content of this paper.

References

1. Ammann, P., Offutt, J.: Introduction to Software Testing. Cambridge University Press (2008)
2. Andrés, C., Merayo, M.G., Núñez, M.: Formal passive testing of timed systems: theory and tools. Softw. Test. Verif. Reliab. **22**(6), 365–405 (2012)
3. Cavalli, A., Gervy, C., Prokopenko, S.: New approaches for passive testing using an extended finite state machine specification. Inf. Softw. Technol. **45**(12), 837–852 (2003)
4. Cavalli, A.R., Higashino, T., Nunez, M.: A survey on formal active and passive testing with applications to the cloud. Ann. Telecommun. annales des télécommunications **70**(3), 85–93 (2015)
5. Flemström, D., Afzal, W., Enoiu, E.P.: Specification of passive test cases using an improved T-EARS language. In: Mendez, D., Wimmer, M., Winkler, D., Biffl, S., Bergsmann, J. (eds.) SWQD 2022. LNBIP, vol. 439, pp. 63–83. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-04115-0_5
6. Flemström, D.: Industrial system level test automation. Ph.D. thesis, Mälardalen University, Västerås, Sweden (2021)
7. Flemström, D., Enoiu, E., Afzal, W., Sundmark, D., Gustafsson, T., Kobetski, A.: From natural language requirements to passive test cases using guarded assertions. In: 2018 IEEE International Conference on Software Quality, Reliability and Security (QRS) (2018)
8. Flemström, D., Gustafsson, T., Kobetski, A.: Saga toolbox: interactive testing of guarded assertions. In: 2017 IEEE International Conference on Software Testing, Verification and Validation (ICST) (2017)
9. Flemström, D., Jonsson, H., Enoiu, E.P., Afzal, W.: Industrial scale passive testing with t-ears. In: 2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST) (2021)
10. GeeksforGeeks: Difference between active testing and passive testing (2025). https://www.geeksforgeeks.org/difference-between-active-testing-and-passive-testing/. Accessed 23 July 2025
11. Gibbs, G.: Learning by Doing: A Guide to Teaching and Learning Methods. FEU (1988)
12. Gustafsson, T., Skoglund, M., Kobetski, A., Sundmark, D.: Automotive system testing by independent guarded assertions. In: 2015 IEEE Eighth International Conference on Software Testing, Verification and Validation Workshops (ICSTW) (2015)
13. Heffernan, D., MacNamee, C.: Runtime observation of functional safety properties in an automotive control network. J. Syst. Architect. **68**, 38–50 (2016)
14. Itkin, I., Yavorskiy, R.: Overview of applications of passive testing techniques. In: Proceedings of the Workshop on Modeling and Analysis of Complex Systems and Processes MACSPro. CEUR Workshop Proc (2019)
15. Koopman, P.: A case study of toyota unintended acceleration and software safety (2014). https://users.ece.cmu.edu/~koopman/pubs/koopman14_toyota_ua_slides.pdf. Accessed 23 July 2025
16. Koopman, P., Wagner, M.: Challenges in autonomous vehicle testing and validation. SAE Int. J. Transp. Saf. **4**(1), 15–24 (2016)
17. Mavin, A., Wilkinson, P., Harwood, A., Novak, M.: Easy approach to requirements syntax (ears). In: 2009 17th IEEE International Requirements Engineering Conference (2009)

18. ISO 26262: Road vehicles - Functional safety. International Standardization Organization, Geneva, Switzerland (2018)
19. Sonigara, B., et al.: Xandar: verification & validation approach for safety-critical systems. In: 2023 IEEE 36th International System-on-Chip Conference (SOCC) (2023)
20. Travis, G.: How the boeing 737 max disaster looks to a software developer (2019). https://spectrum.ieee.org/how-the-boeing-737-max-disaster-looks-to-a-software-developer. Accessed 23 July 2025
21. Vector Informatik GmbH: Canalyzer: Comprehensive ECU & network analysis software, version 18 (released 30 October 2024), windows 7–11 (2024). https://www.vector.com/us/en/download/canalyzer-full-installer-18-sp4/. Accessed 23 July 2025
22. Yin, R.: Case Study Research and Applications: Design and Methods. SAGE Publications (2017)