Identifying Redundancies and Gaps Across Testing Levels During Verification of Automotive Software

Rohini Bisht, Selomie Kindu Ejigu, Gregory Gay
Chalmers | University of Gothenburg
Gothenburg, Sweden
rohini@student.chalmers.se, guskinse@student.gu.se, greg@greggay.com

Predrag Filipovikj
Scania CV AB
Södertälje, Sweden
predrag.filipovikj@scania.com

***Abstract*—Testing of automotive systems usually follows the V-Model, a process where sequential testing activities progress from low-level code structures to high-level integrated systems. In theory, the V-Model should reduce redundant testing and gaps in verification. To assess such benefits in practice, in a case study at Scania CV AB, we have developed a framework to identify redundancies and gaps in test cases across test levels.**

Our framework identified both redundancies and gaps in Scania's scripted testing efforts. Deviating cases were also identified where, e.g., requirements were outdated or contained incorrect details. Factors contributing to redundancy include re-verification in a new context, difficulties mapping requirements across levels, and lack of test case documentation. Both redundancies and gaps result from a lack of communication of traceability and test results across test levels. We recommend active collaboration across levels and use of coverage matrices to alleviate these issues. We offer our framework to help refine testing practices and to inspire process improvements.

***Index Terms*—Software Testing, Embedded System, Automotive Software, Testing Process, Traceability**

I. Introduction

Embedded systems are complex combinations of hardware and software, often designed for safety-critical applications with high expectations on performance and robustness. Modern vehicles contain dozens of electronic control units (ECUs)—embedded systems that control mechanical, electrical, and electronic systems. As vehicles grow in their capabilities, the number and complexity of ECUs have also rapidly increased [1]. Proper verification of software is critical, as it defines the functional behavior of the vehicle [2]. *Testing*—the application of input and analysis of the resulting output—is the most common form of verification, and ensures that both individual ECUs and their integration work as intended.

The vehicle itself offers functionalities built on individual subsystems—ECUs. Those subsystems, in turn, offer a more specific functionality serviced by code modules. Testing can take place at multiple points, with focus on either individual elements or on the *integration* of those elements to form higher-level functionality.

The "V-Model" [3] is the de-facto standard in the automotive domain. It structures the development process as a sequence of design, development, and testing activities. A generic version of the V-Model is shown in Figure 1. The left side of this model consists of design activities—e.g., requirements specification—and the right consists of implementation and testing activities. The V-Model is also being promoted as the reference model in ISO 26262 functional safety standard [4]. Companies often adapt the V-Model to fit their needs (one implementation is shown in Figure 2).

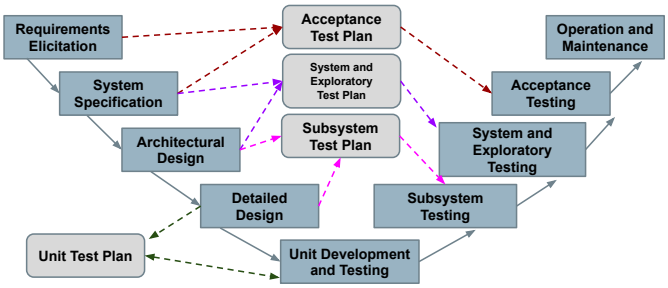


Fig. 1: A generic implementation of the V-Model.

The structuring of testing in the V-Model, in theory, enables clear scoping and progress visibility [5]. This structure allows testers to understand and communicate the objectives of each testing level, with the aim to prevent gaps—e.g., untested requirements [5]. Additionally, if the correctness of a particular functionality is established at the certain level in the V-Model, it should not need to be retested again. As tests at higher levels are often more difficult to design and more expensive to execute, the V-Model offers potential cost savings by avoiding redundant testing [5].

It is not clear that such benefits are guaranteed in practice. Consequently, it is important that organizations can assess whether their testing process is both efficient and effective—i.e., that they can assess the extent of *redundant testing* and *gaps in verification* across levels. This is especially true as past empirical studies of test process efficiency or effectiveness for embedded systems are rare, and may not be applicable to an organization's context.

Our aim is to develop a framework that can identify redundancies and gaps across testing levels. By redundancies, we refer to multiple test cases overlapping in a substantial manner—in this research, invoking the same functionality while applying input from the same equivalence partitions. By gaps, we refer to untested requirement or prescribed equivalence partitions not being applied.

We have performed this research with Scania CV AB, a major manufacturer of commercial heavy-load vehicles. Scania development follows an adaptation of the V-Model, and we focus in particular in this study on three levels—the system, user function, and complete vehicle levels—of their testing process, based on individual ECUs and forms of ECU integration. At Scania, the testing process is guided by a set of test strategy documents. The documents clearly define scope and objectives at each level. However, ensuring efficient verification is difficult, as the overall system is very complex, and the teams often work in isolation and use different documentation, tools, and environments to test. This makes it difficult to assess redundancy and gaps across levels.

Testers can follow our framework to map test-to-requirements traceability across testing levels. Traceability is enabled by the existence of a common logical element—functionality offered by "allocation elements", logical units of computation deployed on each ECU. We have applied this framework with independent assessment by testers:

- 4.29% of tests were redundant between levels. Contributing factors include the need to re-verify functionality in a new context, difficulty mapping testing efforts across levels, and lack of clarity on which test interactions contribute to goals of a test and which are part of setup.
- 13.06% of requirements were not fully verified by the available tests. Deviations in the form of outdated requirements, tests that could not be mapped to requirements, requirements lacking descriptions, and requirements indicating incorrect signals were also detected. In 58.63% of cases, there was no documentation on why the requirements were not fully verified by test automation.
- Both redundancies and gaps result from a lack of communication of traceability and results across test levels. We recommend active collaboration across levels and use of coverage matrices to alleviate these issues.

We offer our framework and the results of its implementation at Scania to help other organizations to refine their testing practices, and to inspire testing researchers to offer further improvements to the V-Model and similar processes.

II. Background and Related Work

Automotive Embedded Systems Testing: ECUs contain embedded micro-controllers equipped with software that monitor and control vehicles functions. ECUs interact through sensors (i.e., input) and actuators (i.e., output), and share information through different protocols and technologies—predominantly CAN and Ethernet[1]. Modern vehicles may contain 70+ ECUs, with 2500+ signals sent through sensors and actuators [1]. Increasing ECU complexity makes testing a significant task.

Testing and verification are performed in different environments where varying portions of the vehicle are real or simulated, based on the need to safely test system elements or due to lack of hardware availability [6]. Model-in-the-loop (MiL) testing is done in a environment where functions and the vehicle itself are simulated using models. During software-in-the-loop (SiL) testing, the software is tested in a simulated environment. In hardware-in-the-loop (HiL) testing, hardware and software components—e.g., ECUs—are tested in a simulated environment. HiL testing has traditionally been the main approach, but MiL and SiL have become increasingly common [6]. These techniques allow for fast development cycles with verification at different levels of abstraction.

V-Model Testing in Automotive: Scania follows a modular development approach (Figure 2). Modules of the vehicle are developed separately, and then integrated to deliver a more complex functionalities. A vehicle consists of ECUs (grey boxes in Figure 2), and each ECU controls certain aspect of a particular vehicle-level function, often referred to as User Function (UF). ECUs communicate through signals (red lines in Figure 2). Each ECU consists of allocation elements (AEs)—logical components (blue boxes in Figure 2).

Figure 3 illustrates the advanced emergency braking (AEB) UF, which is responsible for safely stopping the vehicle when an obstacle up front is detected based on camera and radar data. When there is a risk of collision, the function is activated, first with a light and sound. If the driver does not respond, braking is activated. To realize this UF, a sequence of functionalities are executed, including a speed check, monitoring, and lighting. ECUs (and their internal AEs) are responsible for these functionalities, and the code for the UF coordinates the AEs.

Companies adapt the traditional V-Model—Scania's implementation is shown in Figure 2. Broadly, the left side consists of design activities—e.g., requirements specification—and the right consists of sequential implementation and testing activities. Low-level logical elements of a system interact in service of performing higher-level functionality. In a typical object-oriented system, the system can be broken into interacting subsystems, which can be broken into interacting classes. In the V-Model, classes would be tested independently, then their interaction within subsystems would be tested, then the interactions between subsystems would be tested. Finally, the system as a whole would be tested.

Scania's adaptation of V-Model contains four main levels of testing, where some levels consist of multiple activities[2]:

- **Code Level (Module, Module Integration Test):** Unit and integration testing is performed on software modules within each individual ECU.
- **System Level (ECU System, Part Integration, Part System Test):** In ECU system testing, each ECU is tested. In Figure 3, ECU system testing verifies the light, camera, and speed checking AEs while testing their respective ECUs. Part-system integration testing verifies interface and communications of an ECU. Part system testing focuses on verification of highly-interdependent ECUs.

[1]For brevity, and without sacrificing the generality of our results, we assume that all communication is via the CAN network.

[2]We focus on the system (ECU system test), UF, and integration (complete vehicle integration test) levels.

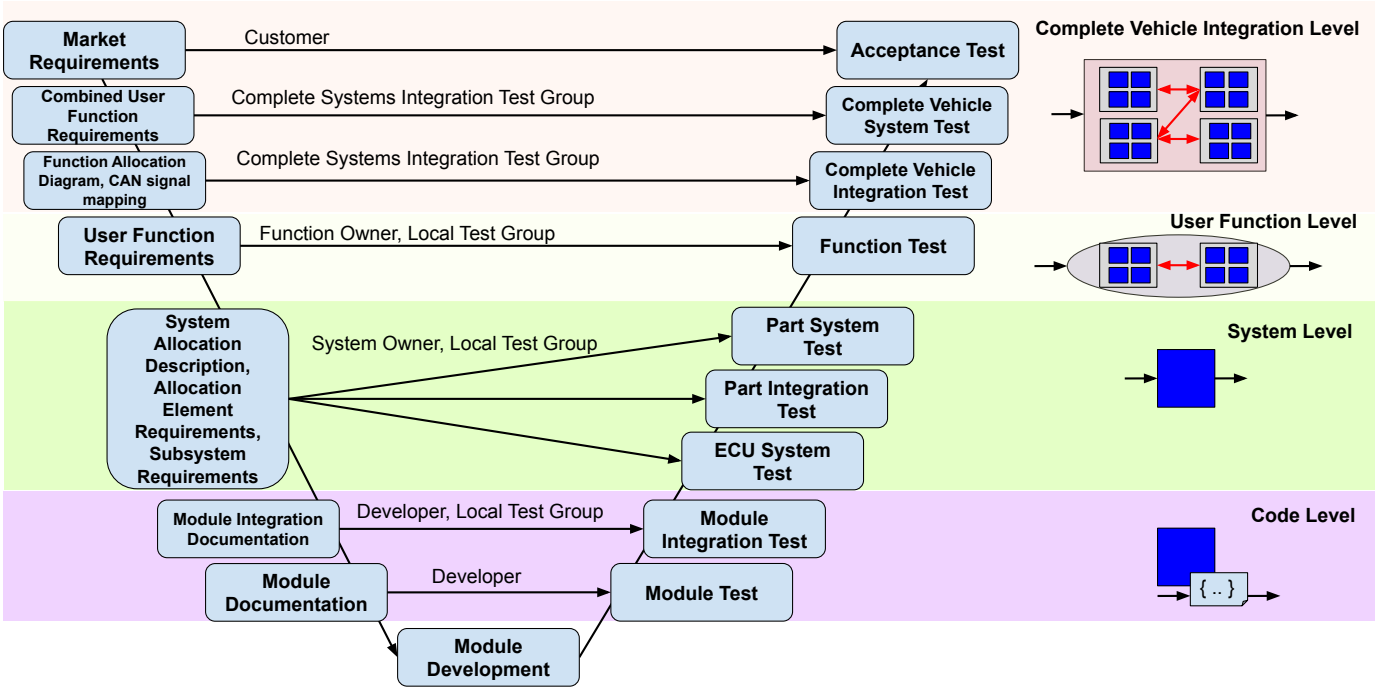


Fig. 2: V-Model development process, as implemented at Scania. Activities on the left relate to requirements specification. Activities on the right relate to testing. Labels on the arrows indicate the party responsible for conducting the testing activity.

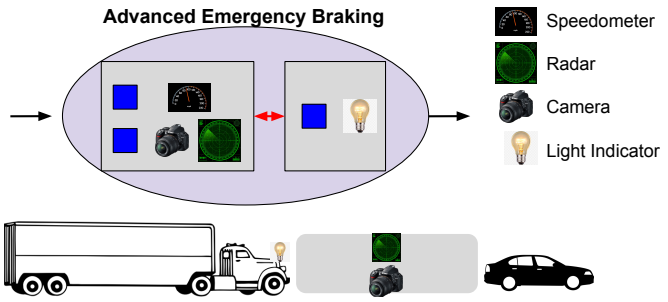


Fig. 3: Advanced emergency braking slows the vehicle to avoid collision. Makes use of functionality from three AEs (in two ECUs): speedometer, radar, camera, light indicator.

- **User Function Level (Function Test):** UFs are tested. The full advanced emergency braking system in Figure 3 would be verified at this level.
- **Integration Level (Complete Vehicle Integration, Complete Vehicle System, Acceptance Test):** End-to-end functionality testing, including tests where the AEB UF from Figure 3 is activated[3]. Finally, acceptance testing ensures that the customer requirements are met.

There are challenges in performing development under the V-Model [7]. Development and verification is strictly divided between the levels. However, automotive systems have complex dependencies between functionalities and ECUs. Strictly dividing by level introduces challenges in cross-functional cooperation between teams, requirements traceability, overlapping of test cases, and lack of documentation [8], [9].

Related Work: Several studies have been conducted on redundancy and gaps in testing efforts. The large volume of test cases needed for complex systems carries risks of redundancies [9]–[11]. However, redundancy can improve quality, and should not always be avoided [10]. Redundancy has been assessed using code coverage criteria [12], string similarity [11], and clone detection tools [13], among other measurements. Reusing test cases at different levels can avoid wasted effort on redundant testing [10].

Our study differs from related work in its method—based on traceability between tests and requirements across levels of the V-Model. Our method is highly flexible, makes use of domain and project knowledge absent from methods based on, e.g., code coverage, and avoids dependencies on particular languages or technologies.

Annapurna et al. proposed a strategy for avoiding redundancies and gaps by enhancing requirements with semi-formal use cases and scenario-based modelling [14]. Traceability is established across testing levels and system views at each level are mapped to the enhanced requirements. Their study was conducted at Scania, across the same levels. We focus on identifying redundancies and gaps after tests are designed. Therefore, our approaches are complementary.

III. Proposed Framework and Methodology

We have conducted a case study at Scania, following Runeson and Höst's guidelines [15], to address following questions:

[3]The distinction between UF and integration levels is that—at the UF level—only the involved ECUs are connected. At the integration level, all ECUs are connected and multiple UFs run simultaneously.

TABLE I: Data collected for each testing level.

System Level	**Requirements:**	AE Requirements (AER)	Functional requirements for AEs; also contains diagnostic trouble code (DTC) requirements and hazard analysis. DTC requirements concern situations where AE has failed. Hazard analysis relates to failure impact.
		System Interaction Requirements (SIR)	Functional requirements for ECU interactions. E.g., SIR for ECU A contains requirements on its interactions with ECUs B and C.
	Test Strategy:	Test Strategy Document	Defines testing objectives and activities to be carried out at the system level.
	Test Cases:	Executable	Executable test cases written for various automation frameworks in either Python or C#.
UF Level	**Requirements:**	UF Requirements (UFR)	Describes background, description, functional requirements, variant information, and hazard analysis for each UF.
	Test Strategy:	Test Strategy Document	Test strategy for the UF level.
	Test Cases:	Manual	Natural language tests, with test steps, acceptance criteria, and prerequisites. Additional exploratory testing takes place, but is not documented.
Integration Level	**Requirements:**	Scenarios	Scenarios derived from UFs. Consist of tasks based on user interaction with vehicle. E.g., UF advanced emergency brake system (Figure 3) may include scenarios based on activation and deactivation in different automobile variants (truck, bus).
		Message Sequence Charts (MSC)	Visualization of signal flow between ECUs via a CAN connection. Explains how ECUs should communicate with each other.
	Test Strategy:	Test Strategy Document	Test strategy for the integration level.
	Test Cases:	Manual	Natural language tests (subset), with test steps, acceptance criteria, and prerequisites.
		Executable	Executable test cases written in Python. Automation differs from system level.

- **RQ1:** How can test redundancy and gaps be identified across different levels of test abstraction?
- **RQ2:** In our case example, to what extent does test redundancy exist between the integration and preceding levels in the V-Model?
  - **RQ2.1:** How many redundancies are identified?
  - **RQ2.2:** What factors led to redundancies?
- **RQ3:** In our case example, to what extent do gaps in testing exist between the integration and preceding levels?
  - **RQ3.1:** How many gaps are identified?
  - **RQ3.2:** What factors led to gaps?

To address these questions, we performed the following:

1) We analyzed requirement and test artifacts from the case company to gain insights about their testing process. This process also included interviews with testers and study of existing literature. This process yielded insights towards answering **RQ1** (Section III-A).
2) We proposed a framework for identifying redundancies and gaps across levels in the V-Model testing process. This allows us to answer **RQ1** (Section III-B).
3) We applied the framework to selected systems to identify redundancies and gaps in their testing efforts. The identified redundancies and gaps were independently verified by testers from the case company. We also identified common explanatory factors. This allowed us to answer **RQ2** and **RQ3** (Section III-C).

A. Data Collection and Analysis

Interviews: To understand the systems and testing process, we interviewed four system owners, two UF owners, and two system test engineers. We omit interview questions due to space constraints. However, questions were open-ended and related to test objectives, how testing is conducted, system importance, testing responsibility, documentation, documents and test locations, and understanding signals used by ECUs and AEs. The developers gave us an overview of the development and testing architecture and the testing process.

Documentation Analysis: We collected and analyzed data (Table I) relevant for establishing traceability between tests (natural language scripts or code) and requirements across levels. Testing at each level is performed according to a test strategy document that defines the scope and the objectives. To analyze documents in a consistent manner, we implemented an open coding method based on four themes: documentation of correct behavior, testing goals, test objects, and testing activities. We had limited documentation for the user function level, as much of the UF testing is exploratory.

B. Framework to Identify Redundancies and Gaps

We present our framework for identifying redundancies and gaps across levels of granularity. Although we base this framework on the Scania's adaptation of the V-Model, it can be adapted to other organizations with a similar context (e.g., the automotive domain) or testing process.

Based on initial exploration, we base our framework on traceability of requirements with tests. Tests are based on requirements at each testing level, using the data in Table I. However, it is not possible to automatically establish traceability *across* levels as requirement formats, test templates, and automation frameworks differ. Instead, we employ a manual process where traceability is established from bottom-up. At each level, tests are mapped to specifications at that level. Traceability across levels can be established based on logical system elements that persist.

At Scania, a common logical element is functionality offered by the AEs. ECUs are tested individually at the system level, and requirements are based on AE functionality. We trace AE functionality to requirements, then to tests. ECUs are then integrated at the UF and complete vehicle levels. UF requirements and integration scenarios can also be mapped to AE functionality, thus establishing traceability across levels.

We present the framework for identifying redundancies and gaps in Figure 4. In short, the framework follows these steps:

1) We trace system tests with AE requirements, UF tests to UF requirements, and integration tests with scenarios.

TABLE II: Example of establishing traceability between system, UF, and integration level, based on AE functions F1-F4.

System Level			UF Level			Integration Level		
AE Req. ID	**System TC ID**	**Function ID**	**UF Req. ID**	**UF TC ID**	**Function ID**	**SCN ID**	**Integration TC ID**	**Function ID**
REQ1	STC1	F1	UFR1	UTC1	F2	SCN2	ITC5	F1
REQ2	STC2	F1	UFR2	UTC2	F1	SCN1	ITC2	F1
REQ3	STC4	F2				SCN2	ITC3	F2
REQ4		F3				SCN3	ITC4	F1
REQ5	STC6	F1				SCN4	ITC5	F4

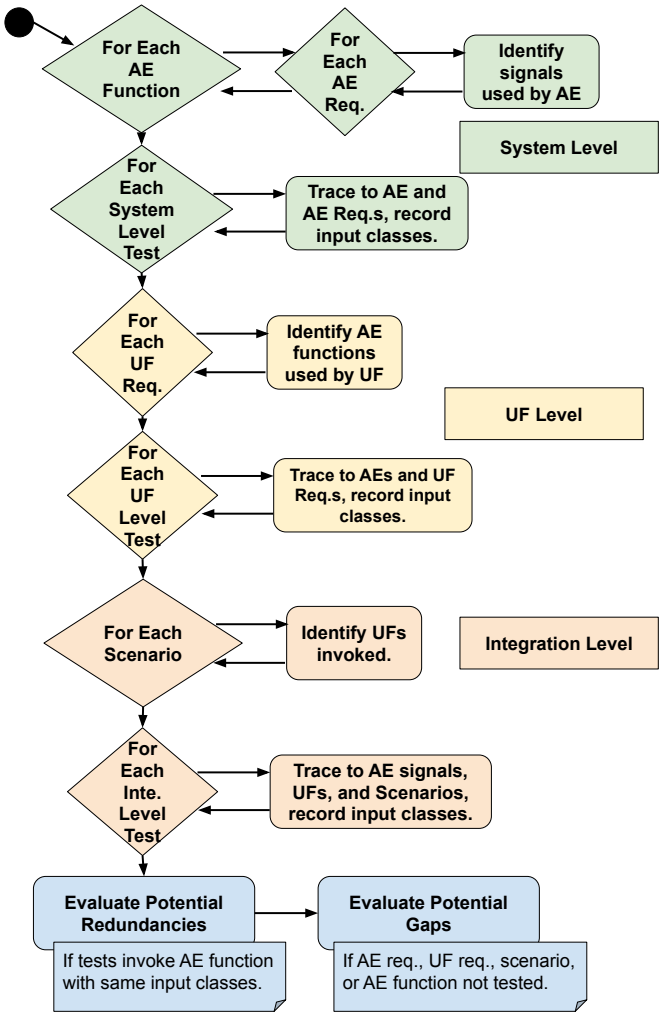


Fig. 4: Framework flowchart for overlap mapping

2) We trace AE requirements, UF requirements, and scenarios to AE functionality.
3) We compare tests from the system, UF, and integration levels that invoke common AE functionality.

If multiple tests invoke the same functionality, and with inputs belonging to the same equivalence partition, we consider those tests **redundant**. If a requirement is not tested or an equivalence partition for an input prescribed in the requirements has not been covered by a test at any level, or if—across all levels—a AE functionality is not tested, we identify a **gap**.

Tests that invoke the same functionality should not automatically be considered redundant. At Scania, test inputs are selected based on partitioning of the input domain into value classes. If the same value classes are chosen, we consider tests redundant, subject to verification by a tester.

TABLE III: Example requirements for each level for the advanced emergency braking system (Figure 3)

Type	Requirements
AE Requirement	The camera must identify the preceding vehicle and share information about the vehicle through CAN signal X.1.
UF Requirement	A warning message must be displayed to the user when advanced emergency braking conditions are valid
Scenario	Activate advanced emergency braking.

Framework Elements: The core elements needed to establish traceability include requirements, functionality, and signals. Each level has different requirements. AE requirements focus on functionality of a single AE, while UF requirements and scenarios focus on user-observable functionality. Sample requirements relevant to the advanced emergency braking UF are described in Table III.

At the system level, tests verify AE requirements. Each AE offers a specific functionality. In Figure 3, one ECU manages sensors, and its AEs offer functionality related to, e.g., detecting vehicles with a camera. Tracing requirements to AE functionality links system level tests to higher levels, where this functionality is invoked by the user functions.

ECUs interact through signals—input through sensors and output through actuators [16]. AEs are invoked and issue output through specific signals. Establishing traceability to higher testing levels can be done concretely through signals. For example, the AE referenced in Table III issues output through signal `X.1`. Requirements at the UF and integration levels may not mention this functionality by name, but the test cases may monitor `X.1`. If a reference exists to `X.1` in, e.g., an integration level test case, we know that the test case interacts with that functionality.

Mapping Example: We offer an example based on Figure 3. Abbreviated traceability results are shown in Table II.

At the **system level**, the tester iterates over each AE. They use AE requirements to identify functionality, and assign each an identifier. They identify signals corresponding to each functionality. They then analyze the system level tests, and map each to corresponding requirements. They note the value classes applied as input in each test. The system level portion of Table II indicates that four functions were identified—e.g., test case `STC1` invokes function `F1` and verifies that requirement `PREQ1` is met.

At the **UF level**, the tester maps each UF requirement to functionalities invoked. The AEB UF invokes a functionality

TABLE IV: Data collected for evaluation.

System Level	**Requirements:**	AER Document (Systems 1, 2) SIR Document (System 3)
	Test Cases:	Executable (Python; Systems 1, 2) Executable (C#; System 3)
Integration Level	**Requirements:**	Scenarios, Message Sequence Charts
	Test Cases:	Executable (Python)

related to, e.g., radar sensing. Tests at this level should be mapped to the UF requirements. These tests are performed manually and signals are not typically documented. However, AE functionalities can be identified using UF requirements. This establishes traceability to the system level. In the UF level in Table II, e.g., requirement `UFR1` is traced to test case `UTC1`, and functions `F2` and `F3` are invoked.

At the **integration level**, the tester identifies the UFs invoked in each scenario, establishing traceability to the UF level and indirectly to the system level. The tester then maps each test to its scenarios. Manual tests at this level can be linked to UFs invoked. Executable test cases are based on signals, allowing direct traceability to the system level.

The tester then checks for potential redundancies or gaps. In Table II, we see that `STC1`, `STC2`, and `STC6` invoke function `F1` at the system level, `UTC2` invokes `F1` at the UF level, and `ITC2`, `ITC4`, and `ITC5` invoke `F1` at the integration level. They compare these seven tests, and check whether they apply the same input value classes. Functionality `F3` is not tested at any level (and requirement `REQ4` at the system level is not tested). These represent gaps.

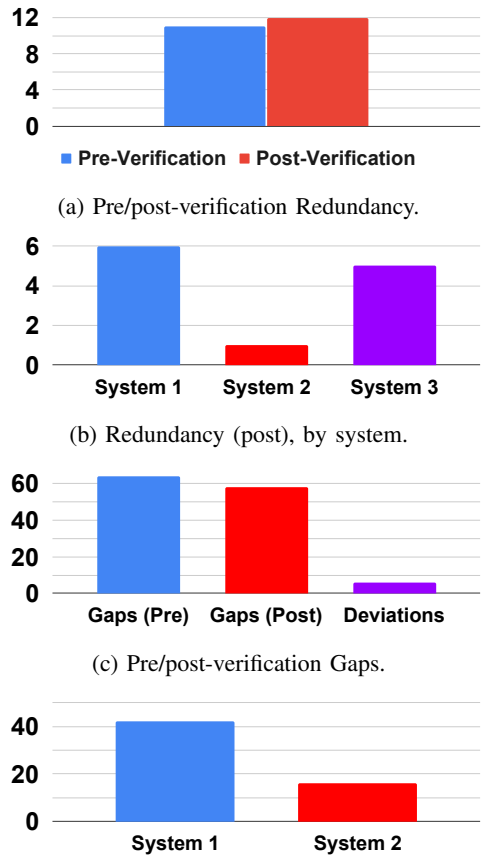
C. Framework Evaluation

Case System Selection: The examined vehicle consists of 24 ECUs. Out of these 24 ECUs, three were selected, based on importance, time constraints, and the availability of test artifacts. These ECUs relate to front-end displays and back-end processing. We selected ten AEs from the first ECU and five AEs from each of the other two ECUs. AEs with the highest hazard severity were selected.

Application of the framework on selected systems: Automated test cases and requirements were gathered for the three ECUs across each level, as shown in Table IV.

At the system level, tests are based on either an AER document (Systems 1 and 2) or a System Interaction (SIR) document (System 3). At the integration level, automated test cases are derived from scenarios. A message sequence chart was used to understand the signal flow between the ECUs. Exploratory testing was conducted to test UFs, and the test artifacts were not maintained. Therefore, we lacked sufficient information to consider the UF level.

For Systems 1 and 2, the framework was applied as described. Minor adaptations had to be made for System 3, as system tests were based on a SIR document. SI requirements are similar to AE requirements, but are not linked to AEs. To identify integration test cases for the corresponding SIR, we need to know the respective AE. Therefore, we identified the AE using the signals. At the system level, functionalities were extracted from the SIR document. The system level automated test cases and SIR requirements were mapped with the functionalities.



(a) Pre/post-verification Redundancy.

(b) Redundancy (post), by system.

(c) Pre/post-verification Gaps.

(d) Gaps (post), by system.

Fig. 5: Redundancies and gaps identified using the proposed framework. The Y-axis for redundancies represents the number of redundant test cases. The Y-axis for gaps represents the number of requirements not covered by test cases.

Independent Verification: Two Scania test engineers working at the integration level reviewed identified redundancies and gaps. We presented our framework, along with its components and motivation. The identified redundancies and gaps were presented and discussed with both testers. Their responses were incorporated into the final results. This verification helps to confirm redundancy and gap validity, and lessons from false-positive identifications improve the framework.

IV. Results and Discussion

Figure 5(a) presents the results of our redundancy analysis between the system and integration testing levels, before and after independent verification by Scania testers. The post-verification redundancies are shown, by system, in Figure 5(b). Figure 5(c), likewise, presents the results of our gap analysis before and after independent verification. A small number of additional "deviations" were also identified during verification. Post-verification gaps are illustrated, by system, in Figure 5(d).

A. Redundancy Across Test Levels

We initially identified 11 redundant test cases between the system and integration levels. Scania testers deemed two of the redundancies as invalid. However, they identified three additional redundant tests as a result of discussing the presented redundancies. Thus, a final total of twelve redundant test cases were identified—six for System 1, one for System 2, and five for System 3. Of a total of 213 system level and 67 integration level tests, **4.29% were redundant**.

Testers noted reasons that redundancies were *invalid*. Although it seemed that two test cases overlapped, they actually verified part of a requirement differently. It was noticed that this requirement was written ambiguously. In a second case, our interpretation of how inputs should be partitioned differed from the interpretation of the testers.

They also identified additional redundancies. At the system level, input is provided to System 3 directly. At the integration level, System 3 reacts to ECUs that it is integrated with. Multiple environmental conditions, based on the state of the other ECUs, can trigger different output behaviors from System 3. As part of the initial evaluation, we considered these different trigger conditions to be unique. However, the testers determined that these differences were localized to the test environment and were treated by System 3 as equivalent input. Therefore, three additional tests at the integration level were considered redundant to system level tests for System 3, as they did not verify additional functionality.

Based on our observations and discussions, we have identified the following factors that may contribute to redundancies:

- **Necessary repetition in a new context:** Not all redundancy is negative. In some cases, the integration teams must verify the same requirements at a different scope or in a different test environment (e.g., different combinations of simulations and ECUs). Behavioral differences may emerge in new environments, and there can be value in redundant testing in such cases.
- **Test teams work in isolation:** Because of level-wise division of responsibilities, teams at different levels only collaborate in limited ways. This can lead to a loss of information. Even though test artifacts can be accessed, it may not be clear how to map tests, test objects and documentation between levels.
- **Test setup at integration level:** If a functionality is triggered in an integration level test, that does not mean that the objective of that test is to verify that specific function. For instance, an AE functionality is often triggered as part of test setup at the integration level to meet prerequisites for testing end-to-end flow of specific UFs. This can lead to overlap if an integration test does nothing "new" other than activate functionality in a way that was already done at the system level. Testers at the integration level should clearly indicate the interactions that are part of test setup and ensure that the complete test does not overlap with a system level test.

B. Gaps Across Test Levels

As shown in Figure 5(c), we initially identified 64 gaps in the verification of 444 requirements across the three ECUs—either complete requirements or input partitions that were not tested[4]. **Scania testers determined that 58 of the identified gaps were valid (13.06% of requirements)**. As shown in Figure 5(d), there were 42 gaps for ECU1 and 16 for ECU2.

The remaining six gaps were not invalid, but were found to not match our definition of a gap. These "deviations" correspond to anomalies in the requirements or test cases. Three deviations were identified for System 1 and three for System 3. All deviations were from the system level.

In two cases, a requirement document had not been updated, leading to a mismatch between the test cases and the requirements due to outdated requirements. In another case, a test could not be mapped to any requirement. Two deviations were matched cases where requirements existed, but lacked full descriptions. Finally, one deviation was due to incorrect mapping of a signal with an SI requirement. The signal indicated in the requirement did not match the AE.

For 24 of the 58 requirements (41.37% of gaps), the testers were unable to properly set up the ECU at the system level or adequately monitor the ECU at the integration level to verify that the requirements were met.

For the other 34 requirements (58.63%), we could not find corresponding test cases. Potentially, this could be due to the fact that these requirements have been tested using exploratory test methods, which do not require documenting the test cases and linking them to requirements. The data as to whether these requirements have been tested is stored in the test management system, however, that data was not included in our study, as we were only looking into the scripted test cases only.

C. Advice for Practitioners

Our framework can be applied in organizations with a similar testing process to help the testers better understand their own efforts. Such application is useful when tests have already been created, and can help enrich an existing test suite to cover any gaps. Redundant test cases could be adjusted to apply different inputs to potentially detect new faults. However, it is also important to examine why redundancies and gaps may exist in the first place and ensure that future projects are better tested.

The rigid segmentation of testing teams can potentially hinder testing efforts. In our analysis, limited sharing of information is a clear factor that causes overlap and gaps. If teams collaborate—and documentation is kept and rigorously updated—then gaps can be minimized and the scope for new testing efforts can be better established. We recommend:

Collaboration between teams: Detailed results about what is tested and how should be shared between teams across levels. The integration team could narrow their scope based

[4]It should be noted that we could only consider available testing artifacts. It is likely that many of these requirements were covered during exploratory testing or were documented in unavailable artifacts.

on the requirements coverage of the preceding test levels (UF or system level), or include additional scenarios that are not considered at the system level. At the integration level, the resources and time required to perform testing are high, so any reduction in unnecessary redundancy is important. Teams must share test cases, documentation, and test results across levels. It is also very important to document when requirements cannot be verified, and why this was the case. Documentation should be kept up to date and stored in a known location. It should also be documented who was responsible for testing in case follow-up information is needed.

Improving collaboration is especially important because ECUs have requirements written in different standards, different languages and frameworks are used for testing at different levels, and different test environments are employed (e.g., hardware or software-in-the-loop). Multiple types of system understanding are needed to test a complete automobile. Teams must educate each other and work together to adequately verify the complete vehicle.

Improving collaboration between test teams might sound like an obvious recommendation for improving the process, but in reality, making this improvement can be quite complex. As a vehicle can contain dozens of ECUs, establishing and maintaining traceability between requirements, test cases, and functionality can quickly become impractical. Consequently, there is an imperative need for designing a lean processes and tools to support collaboration without imposing too much added complexity. The exact form this collaboration should take, and identification of the scope and limits of this collaboration, are areas that must be explored in future research.

Requirement traceability and coverage matrices: The integration team is often unaware of what requirements have been verified, and how they were verified. A traceability matrix could be shared between teams to help higher level teams shape their testing efforts. This matrix should indicate which test cases cover which requirements at each level. It should also indicate input partitions applied, testing methods applied (e.g., manual, executable tests, exploratory testing), and which testing environments the tests took place in.

Such matrices can help each team avoid gaps and redundancies. If requirements could not be covered at a lower level, then it may be possible that a team at a higher level could add tests to cover them—e.g., they may be able to create required setup conditions when employing real ECUs that could not be created purely in simulation. They can also avoid repetition and use the matrix to help brainstorm new scenarios for test suite diversification. Again, it should be noted that documentation and matrices must efficiently provide information to testers without overwhelming them or requiring expertise in all functionality and requirements at all testing levels. Research should be conducted to establish an efficient means to inform testers of information they need in a form they can comprehend without significant burden.

V. Threats to Validity

Construct Validity: Interviews were performed to understand the testing process. Interview questions may not be interpreted in the way that we intended. To overcome this threat, questions were developed based on Scania's documentation and test strategy documents by two of the authors of this study, and were evaluated independently by the other two authors.

Internal Validity: The application of the framework was performed by the authors of this study, leading to potentially biased results. We mitigate this potential bias through independent verification by Scania testers. Our conclusions are also potentially biased by the selection of ECUs and AEs when developing and evaluating our framework. To mitigate this threat, we selected case examples through a severity analysis, as suggested by Scania testers. We focused on the most important ECUs and AEs for analysis.

External Validity: Due to artifact availability and time constraints, only a limited set of systems were considered for both developing and evaluating the framework. In addition, our case study is performed within a single organization. This may limit the generalizability of our findings. However, we have designed our framework to be adaptable to other contexts—while it makes use of the specific documents available at Scania, the process of mapping across levels should be possible with other document types. At minimum, it should be applicable to other organizations in the automotive domain or that adapt a version of the V-Model. In future work, we will examine applying this framework at other organizations in order to generalize and solidify its design.

VI. Conclusion

We propose a framework that can be applied to identify redundancies and gaps in test cases across levels in the V-Model. We observed that factors contributing to redundancy include the need to re-verify functionality in a new context, difficulty in mapping testing efforts across levels, and lack of clarity on which test interactions contribute to goals of a test case and which are part of setup. Deviating cases were also identified where requirements were outdated, tests could not be mapped to requirements, requirements lacked descriptions, and requirements indicated incorrect signals. Both redundancies and gaps result from a lack of communication of traceability and test results across test levels. We recommend active collaboration across levels and use of coverage matrices to alleviate these issues.

We offer our framework case study to help refine testing practices and to inspire process improvements. Ultimately, the adoption and value of a framework such as ours will be judged through the prism of added complexity to an already-complex development and verification process. Therefore, in future work, we aim to further refine both this framework and the surrounding development process. We will consider requirement severity to prioritize redundancies and gaps, explore how our framework can be automated to the extent possible, and apply this framework at additional organizations. We will

also explore how teams can better collaborate across levels, including both the scope and forms of collaboration.

VII. Acknowledgments

We thank Marco Bauer, Markus Byström and Björn Adolfsson from Scania for constructive discussions and valuable feedback on our research.

References

[1] N. Navet and F. Simonot-Lion, *Automotive embedded systems handbook*. CRC press, 2017.

[2] B. Czerny, J. D'Ambrosio, P. O. Jacob *et al.*, "A software safety process for safety-critical advanced automotive systems," in *Proceedings of The International System Safety Conference*, 2003.

[3] L. Shuping and P. Ling, "The research of v model in testing embedded software," in *2008 International Conference on Computer Science and Information Technology*, 2008, pp. 463–466.

[4] ISO, "Road vehicles – Functional safety," 2011.

[5] C. Mindrum *et al.*, *Netcentric and Client/server Computing: A Practical Guide*. CRC Press, 1998.

[6] G. Tibba, C. Malz, C. Stoermer, N. Nagarajan, L. Zhang, and S. Chakraborty, "Testing automotive embedded systems under x-in-the-loop setups," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE Press, 2016, p. 1–8. [Online]. Available: https://doi.org/10.1145/2966986.2980076

[7] R. Pfeffer, G. N. Basedow, N. R. Thiesen, M. Spadinger, A. Albers, and E. Sax, "Automated driving - challenges for the automotive industry in product development with focus on process models and organizational structure," in *2019 IEEE International Systems Conference (SysCon)*, 2019, pp. 1–6.

[8] A. Kasoju, K. Petersen, and M. V. Mäntylä, "Analyzing an automotive testing process with

[11] D. Flemström, W. Afzal, and D. Sundmark, "Exploring test overlap in system integration: An industrial case study," in *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2016, pp. 303–312.

evidence-based software engineering," *Information and Software Technology*, vol. 55, no. 7, pp. 1237–1259, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584913000165

[9] D. Flemström, D. Sundmark, and W. Afzal, "Vertical test reuse for embedded systems: A systematic mapping study," in *2015 41st Euromicro Conference on Software Engineering and Advanced Applications*, 2015, pp. 317–324.

[10] E. Engström and P. Runeson, "Software product line testing – a systematic mapping study," *Information and Software Technology*, vol. 53, no. 1, pp. 2–13, 2011. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584910001709

[12] T. Long, I. Yoon, A. Porter, A. Sussman, and A. Memon, "Overlap and synergy in testing software components across loosely coupled communities," in *2012 IEEE 23rd International Symposium on Software Reliability Engineering*, 2012, pp. 171–180.

[13] S. P. R. Asaithambi and S. Jarzabek, "Towards test case reuse: A study of redundancies in android platform test libraries," in *Safe and Secure Software Reuse*, J. Favaro and M. Morisio, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 49–64.

[14] A. Chunduri, R. Feldt, and M. Adenmark, "An effective verification strategy for testing distributed automotive embedded software functions: A case study," in *Product-Focused Software Process Improvement*, P. Abrahamsson, A. Jedlitschka, A. Nguyen Duc, M. Felderer, S. Amasaki, and T. Mikkonen, Eds. Cham: Springer International Publishing, 2016, pp. 233–248.

[15] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, p. 131, 2008. [Online]. Available: https://doi.org/10.1007/s10664-008-9102-8

[16] B. Broekman and E. Notenboom, *Testing embedded software*. Pearson Education, 2003.