Open access • Book Chapter • DOI:10.1007/978-3-319-49094-6_15

An effective verification strategy for testing distributed automotive embedded software functions : A case study — Source link

Annapurna Chunduri, Robert Feldt, Mikael Adenmark

Institutions: Blekinge Institute of Technology, Scania AB

Published on: 22 Nov 2016 - Product Focused Software Process Improvement

Topics: Automotive software, Software reliability testing, Integration testing, Software construction and Software verification and validation

Thesis no: MSSE-2016-08

An Effective Verification Strategy for Testing Distributed Automotive Embedded Software Functions

A Case Study

Annapurna Chunduri

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the degree of Master of Science in Software Engineering. The thesis is equivalent to 20 weeks of full-time studies.

Contact Information:
Author(s):
Annapurna Chunduri
E-mail: anch15@student.bth.se

University Advisor:
Prof. Robert Feldt
Dept. Software Engineering

Industry Advisor:
Mikael Adenmark
Senior Test Engineer
Scania CV AB, Södertälje, Sweden

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context. The share and importance of software within automotive vehicles is growing steadily. Most functionalities in modern vehicles, especially safety related functions like advanced emergency braking, are controlled by software. A complex and common phenomenon in today's automotive vehicles is the distribution of such software functions across several Electronic Control Units (ECUs) and consequently across several ECU system software modules. As a result, integration testing of these distributed software functions has been found to be a challenge. The automotive industry neither has infinite resources, nor has the time to carry out exhaustive testing of these functions. On the other hand, the traditional approach of implementing an ad-hoc selection of test scenarios based on the tester's experience, can lead to test gaps and test redundancies. Hence, there is a pressing need within the automotive industry for a feasible and effective verification strategy for testing distributed software functions.
Objectives. Firstly, to identify the current approach used to test the distributed automotive embedded software functions in literature and in a case company. Secondly, propose and validate a feasible and effective verification strategy for testing the distributed software functions that would help improve test coverage while reducing test redundancies and test gaps.
Methods. To accomplish the objectives, a case study was conducted at Scania CV AB, Södertälje, Sweden. One of the data collection methods was through conducting interviews of different employees involved in the software testing activities. Based on the research objectives, an interview questionnaire with open-ended and close-ended questions has been used. Apart from interviews, data from relevant artifacts in databases and archived documents has been used to achieve data triangulation. Moreover, to further strengthen the validity of the results obtained, adequate literature support has been presented throughout. Towards the end, a verification strategy has been proposed and validated using existing historical data at Scania.
Conclusions. The proposed verification strategy to test distributed automotive embedded software functions has given promising results by providing means to identify test gaps and test redundancies. It helps establish an effective and feasible approach to capture function test coverage information that helps enhance the effectiveness of integration testing of the distributed software functions.

Keywords: Verification Strategy, Distributed Automotive Embedded Software Functions, Test Coverage.

Acknowledgments

The journey of my master thesis has been a truly rewarding experience. I am grateful to have had the unique opportunity to pursue it within the research department of the automotive industry. The perks of both, the opportunity to be pursuing research and to be able to do so within a real world industrial setting have been immensely satisfying. I have had the honor to contribute to research within this industry while being guided by the most prolific and profound minds within and outside the industry.

I would like to express my deep gratitude to my university supervisor, Professor. Robert Feldt, and my industry supervisor, Mr. Mikael Adenmark, for their valuable expert advice and strong support and encouragement throughout my thesis work. Their guidance and encouragement was one of the defining pillars of my confidence to overcome obstacles and minor setbacks and emerge out successfully.

I am grateful to Mr. Niclas Clashammar, Head of Systems and Integration Test, Scania CV AB (henceforth referred to as Scania) for providing me the opportunity to conduct my thesis at Scania. Special thanks to Mr. Tom Nyman for his valuable recommendations during critical phases of my thesis work. In addition, I would like to thank all my colleagues at Scania who have been highly cooperative and enthusiastic.

Finally, I would like to express my heartfelt gratitude to my parents, Mr. C.V.R Murthy and Ms. C. Padmavathi, my sister, C. Swetha, and my uncle, late Mr. Sunil Kandlikar, for their everlasting love, support and well wishes. Last but not the least, I would like to thank one and all, who might have missed my mention inadvertently, for their help and support, without which my thesis work could not have been successful.

Thank you all.

Annapurna Chunduri

Contents

List of Tables

List of Figures

Chapter 1

Introduction

"The rapid increase in the software complexity of today's Electronic Control Units(ECUs) makes testing a central and significant task within automotive control software development" [5].

1.1 Problem Statement

Across several industries, there is an increasing use of embedded system software to provide functionality with high reliability demands within safety-relevant applications. One such industry is the automotive industry which has been significantly affected by the industrial software revolution over the past decade [6][7]. The share and importance of software within a vehicle is growing steadily [7]. Most functionalities in modern vehicles, especially safety related functions like automatic braking system, advanced driver assistant systems and anti-slip control are controlled by software [8]. It is anticipated that 90% of all future automotive innovations will be driven by software [7]. Hence, it can be inferred that the automotive industry is slowly but steadily transitioning towards being a software-centric industry.

An automotive vehicle consists of ECUs, also referred to as ECU systems, which are essentially embedded microcontrollers with corresponding software components. The ECU systems interact in order to execute the desired functionality in the vehicle like controlling the engine, displaying fuel level and operating air bags [9]. In the past, each single ECU system in the vehicle had a single dedicated function. Hence, execution of a function required the software within only one of the ECU systems to execute independently. In the early 90s, ECU systems were coupled using a single area network called the Control Area Network (CAN) through which they could communicate by sharing information. This lead to functions that were designed to be realized through the cooperation of different sub-functions and ECU systems. Moving towards the late 90s, functions highly depended on communication between multiple CAN networks. Nowadays, there is a multi-fold increase in automotive software function complexity with the inclusion of interaction between multiple CANs and outer-vehicle environment via

radio links [7]. Hence, cross-functionality i.e., a function distributed across multiple ECU systems and consequently across several system software modules is a common and complex phenomenon in today's automobile vehicles.

Such an increased significance of software based distributed functionality has resulted in various challenges for the automotive industry [8]. The growing number of functionalities in the next generation vehicles not only results in complex embedded software but also a bottleneck to design and execute effective and efficient processes of development, testing and production of the software [10]. One such critical and crucial challenge is the growing complexity of automotive software testing due to the rapid rise in, and highly distributed nature of, software within the vehicles [11]. Undetected software defects can cause damage to humans and, in a few unfortunate cases, loss of lives. Hence, a complete and thorough testing of automotive embedded software is essential. A single fault could potentially cause devastating consequences [12]. As a result, about 50% of the total time spent on management and technical activities of automotive development is dedicated to software testing [13].

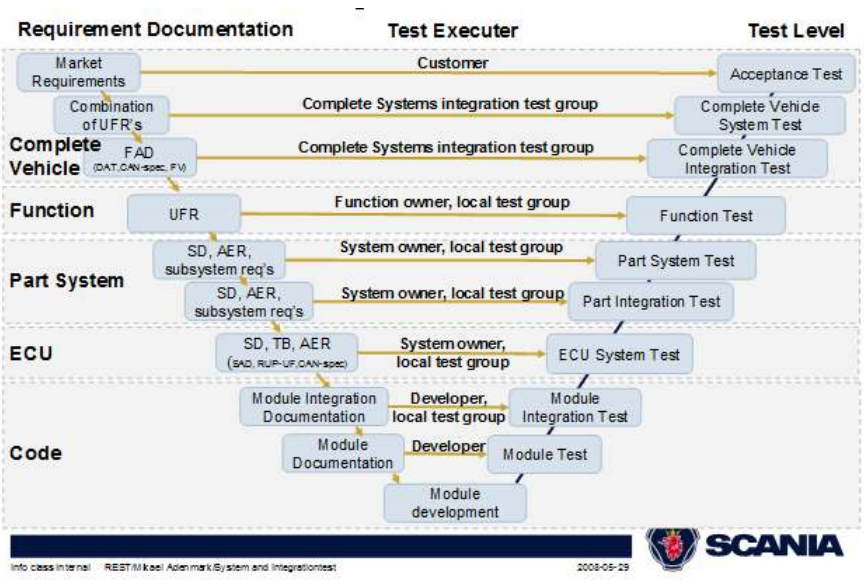


Figure 1.1: The V-model implemented at Scania to develop and test their automotive embedded software [4]

In the automotive industry, the standard V-model is most widely used for the engineering processes of embedded software development [13]. A typical V-model implemented by Scania [4], a major manufacturer of commercial heavy vehicles in the European automotive industry, can be seen in Figure 1.1. Similarly, across the automotive industry, testing of the embedded software occurs at different test levels from the lowest code test level to the highest vehicle integration test level.

Here, the term 'test level' is used to indicate the test focus. *"Each test level describes an area of test responsibility"* [4]. For instance, at the software code test level, the individual software module testing and software module integration testing is performed independent of the underlying ECU system hardware. Moving to the system test level, for each individual ECU system of the vehicle, the software modules are mounted or embedded on the corresponding ECU system hardware like the engine or the gearbox and tested independently. There after, at the vehicle integration test level, the software modules corresponding to all the ECU systems which make up the vehicle are tested together in their actual operational environment (real vehicle or simulated vehicle). The exact number of test levels and terminology used to describe each test level differs from one automotive company to another. But what remains the fundamental similarity in the concept of testing is that, at each test level, the test strategy adopted aims to address and test software behavior at a different level of abstraction and provides a different degree of coverage of the object under test [6].

In general, testing of embedded software modules is more complex than non-embedded software modules due to lower controllability and observability [14]. This, coupled with the distributed nature of the functions across several embedded system software modules further increases the complexity of the situation [15]. Exhaustive integration testing of the distributed functions at the vehicle integration test level using the numerous variants across the modules would be an excellent means of ensuring defect-free software. However, it is not feasible since projects within the automotive industry neither have infinite resources nor have the time to carry out such exhaustive testing [12]. On the other hand, going by the traditional approach of implementing an ad-hoc selection of test scenarios based on the tester's experience and expertise can lead to test gaps and test redundancies across the different test levels [16]. Hence, there is a pressing need within the automotive industry for a feasible and effective verification strategy for testing distributed software functions at the vehicle integration test level that would help achieve adequate test coverage while reducing test redundancies and test gaps across the test levels. There is limited scientific literature in the context of automotive industry, like [12], found to be focusing on solving this challenge.

Verification as defined by the IEEE standard for System and Software Verification and Validation [17] and Software Engineering Institute [18] is a process that provides evidence for whether the products under consideration fulfill the specified requirements for correctness, completeness, consistency and accuracy. There are several established techniques for performing system and software verification like testing, technical reviews and prototyping [19]. Here, testing is a verification technique which involves activities that exercise a product at various levels to verify if it satisfies the specified requirements [18]. In the context of the automotive industry, similar definitions of verification and testing have been provided by the

ISO 26262 safety standard which defines verification as "determination of completeness and correct specification or implementation of requirements" [20] and testing as "process of planning, preparing, and operating or exercising an item to verify it satisfies specified requirements, to detect anomalies, and to create confidence in its behaviour" [20]. In addition, while the term 'strategy' has yet to be defined concretely and accepted universally, it has been found to be a ubiquitous term that can be attached to any means of achieving the desired result. It is a term that is used to define an attempt to establish actions and activities in the light of the goals and capabilities. It hence goes beyond being just a plan. It is about identifying a high end challenge, establishing suitable objectives to solve these challenge and providing ways and means of fulfilling these objectives [21]. Therefore, within the current context, the term 'verification strategy for testing' can be defined as a set of actions and activities that have been laid out based on careful consideration of the current goals and capabilities, to solve the identified high end challenge(s) in the verification process of testing.

While the need from within the automotive industry to identify an effective verification strategy that can handle the steadily increasing software complexity is one of the major driving forces, there is another driving force that plays a significant role. That is the ISO 26262 Road Vehicle Functional Safety Standard [11]. According to this standard [20], functional safety of an automobile vehicle is achieved by undertaking safety measures that pertain to not only the technologies implemented within the vehicle, but also to several other influencing factors like the processes implemented for development, production, services and management among others. Hence, it can be inferred that a recommendation by the standard to build safe vehicles is to have effective supporting processes. In addition, the standard also specifies the need to provide evidence that the vehicle functions are reasonably safe. This in the current context, would imply a need to provide evidence for the effectiveness of the processes being implemented to build safe vehicle functions.

In essence, within the context of verification, which is one of the supporting development processes identified by the standard, it can be inferred that, the standard recommends automotive industries to have an effective verification process to help produce high quality and safe vehicle functions. It also recommends to have evidence of effectiveness of this process. Introduction of the safety standard proved to be a major advancement in the automobile industry and compliance to the standard is being increasingly discussed within the industry and research [11][22]. But the study of how the effectiveness of testing process can be identified and enhanced and evidence for the same can be provided for integration testing of the distributed software functions has been found to be limited. Therefore, there is a need to study how to align or improve the current approach to test distributed software functions in the automotive industry to facilitate future need

for compliance to the recommendations of this defacto standard.

Hence, there exist challenges within integration testing of complex and distributed software functions in the automotive industry. These challenges have a potential to lead to undesirable consequences. Yet, research focusing on addressing this problem domain, like [12], has been found to be limited. Therefore, there is a need from within and outside the automotive industry for a verification strategy that can help focus on improving the effectiveness of integration testing of the distributed software functions. Here, it is essential for the strategy to help identify and improve test coverage of the distributed software functions at the vehicle integration test level, while reducing test redundancies and test gaps across different test levels.

1.2 Research Aim and Objectives

The aim of the research was to develop an effective verification strategy, termed cross-functional verification strategy, for integration testing of the distributed functions of automotive embedded software. The proposed strategy aims to help improve test coverage of the distributed software functions at the vehicle integration test level while reducing test redundancies and test gaps across different test levels. Hence, here, the primary focus of the strategy is on the high-level function requirements. The idea was to combine test results from different test levels to establish that the different verification results together can provide evidence of functional safety for such distributed functions. Since the focus was on test coverage based on requirements, only those test levels where the focus is on requirements-based test coverage and not code-based test coverage were considered to be within the scope of this research.

Given the overall aim, the primary objectives of the research were to:

- Perform a case study at Scania to identify the current test approach at different test levels used for testing the distributed automotive embedded software functions along with the major challenges in this approach.
- Capture the relevant information pertaining to the distributed software functions under consideration, like the test results and test reports, from each test level at the case company.
- Develop and propose an effective and feasible concept for combining the identified test results with a cross-functional verification strategy to verify distributed functionality.
- Demonstrate the validity of the proposed cross-functional verification strategy by studying its feasibility and evaluating the improvements in test ef-

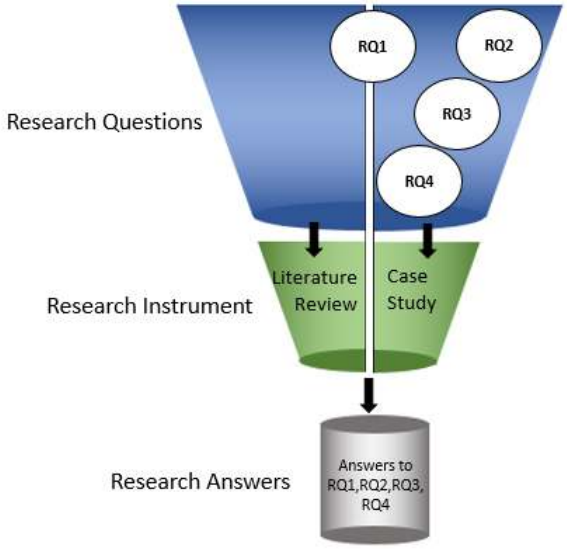


Figure 1.2: Research instrument for answering the research questions

fectiveness for distributed functionality using existing historical data at the case company.

1.3 Research Questions and Instrument

Based on the research aims and objectives, the research questions formulated for this study are reported within this section. Research question RQ1 part a) was answered by conducting a literature review. On the other hand, RQ1 part b) along with RQ2, RQ3 and RQ4 were answered by conducting a case study at Scania as illustrated in Figure 1.2.

RQ1. What is the approach at the different test levels to test the distributed automotive embedded software functions a) as described in recent literature and b) at the case company?

Motivation: The motivation behind inclusion of this research question is in two-folds. Firstly, it helps capture scientific data related to the problem domain from literature like [1]. This in turn helps identify the state-of-the-art software testing approach at different test levels used in the automotive industry. Secondly, it helps the researcher study and report the test approach at different test levels currently implemented at the case company to gain an understanding of if and how it differs from the state-of-the-art. While evidence in literature exists pertaining to how software testing is implemented in the automotive industry, it has been found to be limited [1], especially pertaining to higher level integration testing [2] which is the focus

of the current research. Therefore, on one hand, the answer to this question helps establish a firm foundation for the research, while on the other, it contributes to the sparse body of knowledge in the area of integration testing of the distributed automotive embedded software functions. In addition, it stands as the first step towards identifying and reporting where the current test approach is most likely problematic and needs to be addressed for improving its effectiveness.

RQ2. What is the current test coverage of the distributed software functions across the test levels at the case company?

Motivation: One accepted and widespread approach to report the efforts of verifying a software function for various input combinations is using test coverage information based on a suitable coverage metric like requirements-based test coverage or code-based test coverage [12]. Identifying the current test coverage information for the distributed software functions across different test levels helps establish how much has been tested. It can then be used as one of the indicators to assess the effectiveness of the current test approach. Hence, answering this question helps take a step towards achieving the aims and objectives of the research.

RQ3. What are the major issues with the current approach to test distributed software functions that are a hindrance to its effectiveness at the case company?

Motivation: While literature pertaining to challenges in the automotive industry in the context of overall software technology [7] and other key areas like requirements engineering [23], agile development [8] and testing [1][2] have been found, it has been identified to be limited especially pertaining to challenges of high-level integration testing. Moreover, owing to the increasing complexity, dependency and share of software in automotive vehicles, challenges related to all aspects of software, including testing, are expected to grow over the years in the automotive industry [7][8][10]. This makes it a dynamic, ever-changing and growing area of study. Added to this is the new dimension of challenges and complexity introduced by the need for compliance to the ISO 26262 safety standard in the near future [11]. Hence, considering all these factors, this research question has been deemed necessary for the following two reasons. Firstly, to add to the research body of knowledge pertaining to the current challenges in automotive software testing of distributed software functions. Secondly, answering this research question helps identify test issues that need to be addressed by the verification strategy to enhance the current approach to test the distributed software functions.

RQ4. What is a feasible and effective cross-functional verification strategy to improve test coverage while reducing test redundancies and test gaps across test levels for such distributed software functions?

Motivation: As presented in Section 1.1, the automotive industry today is facing a challenge of inability, in terms of inadequate resources and time, to perform exhaustive software testing [12] on one hand and test gaps and test redundancies due to the current ad-hoc test approach on the other [16]. Hence, the need in the industry is for a verification strategy that not just helps in solving the challenges at hand, but also one that would help to do so in an effective and feasible manner. Therefore, partly, the answer to the research question is a verification strategy that aims to resolve key test issues to enhance the overall test effectiveness. Another part of the answer to this question, helps address the feasibility and effectiveness aspects of the verification strategy. This provides a means of studying and reporting the significant consequences of implementing the strategy in a real world setting.

1.4 Expected Research Outcomes

The thesis report is expected to reflect the knowledge gained by fulfilling the research aims and objectives through answering the research questions. This includes:

- A cross-functional verification strategy for testing distributed functionality of automotive embedded software which helps improve integration test effectiveness by providing means to improve test coverage by reducing test redundancies and test gaps across different test levels.
- Validation of the feasibility and effectiveness of the proposed cross-functional verification strategy using existing historic data collected at Scania.

1.5 Structure of the Thesis

The thesis report fundamentally consists of four major parts, namely, Introduction, Research Methodology Results, Analysis and Conclusion, as illustrated in Figure 1.3. The Introduction has two chapters - Introduction (Chapter 1) and Background and Related Work (Chapter 2). The problem statement, research aim and objectives, and research questions are presented in the Introduction Chapter. On the other hand, to set the foundation for the research, the relevant background and related work is presented in the Background and Related Work Chapter. Following this is the Research Methodology Results part which consists of two chapters, namely, Research Method (Chapter 3) and Case Study

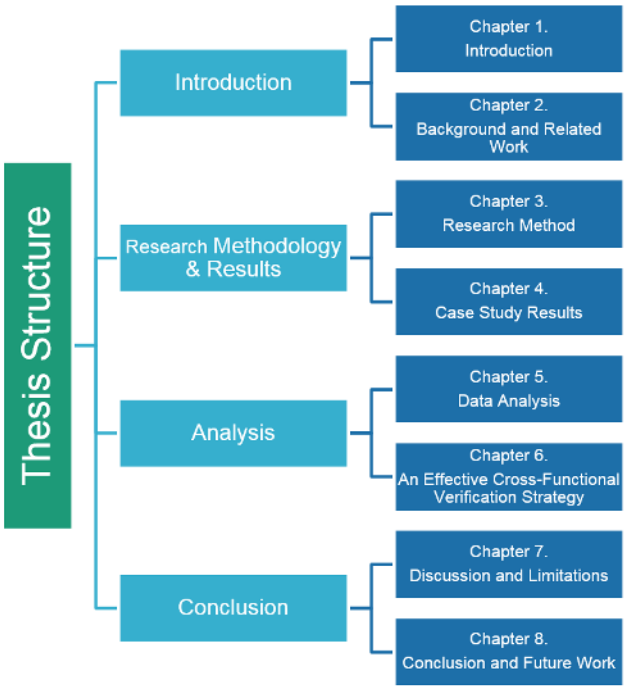


Figure 1.3: Structure of the thesis report

Results (Chapter 4). The Research Method Chapter presents the motivation for the choice of the research method and information pertaining to how the case study design protocol was planned and implemented. The Case Study Results Chapter deals with the results that were obtained on implementing the case study design steps. The next part is the Analysis which contains Data Analysis (Chapter 5) and An Effective Cross-Functional Verification Strategy (Chapter 6) chapters. The Data Analysis Chapter presents the findings on analyzing the obtained data results to answer a subset of the research questions dealing with the issues with testing the automotive embedded software functions and analyzing the alternative approaches that can be employed as verification strategies. There after, as the name suggests, An Effective Cross-Functional Verification Strategy Chapter presents the proposed verification strategy for testing distributed automotive embedded software functions and its validation based on implementation on a legacy software function- Fuel Level Display. Towards the end of the report is the Conclusion which contains two chapters. The Discussion and Limitation (Chapter 7), discusses the overall results obtained on conducting the research and the threats to its validity. The Conclusion and Future Work (Chapter 8) presents the summary of the contributions of the research and the areas that have great potential to be studied in the future to further enhance and expand the current research.

Chapter 2

Background and Related Work

To better comprehend the context of the current research, the first essential step is to understand and analyze the role of software in the automotive industry and the state-of-the-art automotive software testing approach. In addition, another key objective of the literature review conducted was to identify and compile a set of major challenges with the automotive software testing approach as presented in scientific literature. This provides the researcher with adequate research context knowledge that assists in identifying the current state of the software testing approach at the case company, along with the major challenges in this testing approach that can be addressed as part of the proposed cross-functional verification strategy.

2.1 Software in the Automotive Industry

The automotive industry has traditionally been, and to a certain extent continues to be, dominated by electrical and mechanical engineering concepts [24][25]. But this predominant nature of the industry is seeing a slow yet steady shift for nearly a decade now. Like several other industries that have traditional non-software centric applications like aeronautics, and space and defense systems, the automotive industry too has been affected by the software revolution [7]. This is what many would like to term as "Digital Automotive Revolution" [25]. Increasingly many automotive innovations, pertaining to both safety critical and non-safety critical aspects of the vehicle, are being controlled by software. Hence, studying how automotive industries handle their software is essential and is being given more prominence in today's automotive context than it was a few decades ago. Broy in [25] characterized software in this industry as follows:

Size: There is an overwhelming amount of software in today's automotive vehicles. Broy identified in his other work [26] that in 30 years, the software in a vehicle moved from zero, before it was first introduced in 1976 [27], to over ten million lines of code.

Role: Functions within a vehicle, whether safety critical functions like advanced emergency braking system or non-safety critical functions like comfort and entertainment system functions, are being controlled by software. This increasing role of software in controlling automotive functions has been identified in several other researches like in [7] and [24].

Interaction and Distribution of Software: Traditionally, automotive functions were handled by software modules in one ECU system and hence required minimum interaction with other ECU system software modules. But in today's vehicles, software functions are spread across multiple ECU systems and it is not uncommon to have ECU system software with mutual dependencies and high communication to execute complex functions within the vehicle .

The above characteristics of software in the automotive industry summarizes what are commonly considered to be evidence in industrial research for software's every growing share and complexity within automotive vehicles. It thereby helps identify the need to focus empirical research and industrial study on software in the automotive industry.

Further Broy et al. in [27] characterized the automotive software engineering domain to have the following salient characteristics. This further enhances the understanding of automotive software complexity.

Heterogeneous Subsystems and Unique Organizational Structure: The automotive organizational structure for system software development mimics its modular vehicle development concepts. Here, different units of the organization are responsible for development and testing of software pertaining to different systems of the vehicle like the chassis, the engine and the gearbox. The modular systems so developed and tested are then assembled together into a vehicle. Hence, there exist possible differences among the approaches employed by different organization units to develop and test the software which is then to be coordinated efficiently to assemble a complete vehicle.

Original Equipment Manufacturer (OEM) Supplier Software: While the integration of heterogeneous systems is in itself a challenging task, within the automotive industry this task is far more complex. This is due to the fact that a large number of vehicle systems and system software are developed and manufactured by several OEM suppliers. These supplier systems and system software are then assembled and integrated with other vehicle systems by the automotive company.

Highly Configurable Software: Automotive software is highly configurable due to high variants of electronic parts and functions that make up the vehicle. Therefore, a large set of vehicle variants can be produced from modular parts repre-

senting concepts similar to product line engineering. The authors of [27] state an example of a car having nearly 80 electronic units, where a simple choice of whether to include a function or not in the vehicle leads to 2^{80} possible vehicle variants. The software can hence be configured in several ways, producing several vehicle variants that are characterized by the differences in their electronic units and the corresponding software functions.

Distributed Software and Unit Cost Models: Like discussed earlier, Broy further stresses on the fact that with increasing complexity of vehicle functions, the software used to realize these functions is distributed across several ECU systems and also across several organization units. Moreover, driven by cost pressure, the automotive industry relies greatly on unit vehicle cost models. Such a focus on optimizing cost per unit has been found to be problematic from the software perspective.

In essence, the increasing share of software brought about an evident advancement within the automotive industry with increased safety, reliability, environmental efficiency, and comfort features that cannot otherwise be easily provided using mechanical solutions alone [7]. However, the simultaneous increase in complexity has brought about several software-related challenges and issues that today's automotive industries face. Hence, in this context, several researchers [7][24][25][26][27] made an attempt to capture broad software engineering-related challenges within the automotive industry under several categories. Software engineering as a discipline covers several areas like requirements engineering, software development and software verification and validation [28]. Of these, areas of software engineering that have been most widely identified as being challenging include requirements engineering, architectural design and safety and quality assurance through system software verification and validation [7][24][25][26][27]. Further, several researchers have focused on studying such specific 'software engineering' areas within the automotive industry in terms of the advantages they provide and the challenges they face. For instance, challenges in requirements engineering within the automotive industry have been studied in [23], [29], [30] and [31]. But such a focus on studying challenges in verification and validation, especially testing, like in [1] and [2], is limited as recognized by the authors themselves.

2.2 Software Testing in the Automotive Industry: State-of-the-Art

The standard V-model is most widely used in the automobile industry for the engineering processes of system software development [13]. Therefore, it can be inferred that the automotive system software testing generally takes place at

several test levels across the right side of such a V-model. Here, the software is tested from the lowest code level to the highest vehicle integration level in different test environments, with different test objectives and against different set of requirements. The exact test levels of the V-model used, differs slightly from project to project within each automotive organization and among different organizations [1].

Generally, following the modular nature of system software development and testing, different automotive organizational units are responsible for development and testing of different ECU systems (modular part of the vehicle) like the chassis control system, engine management system and brake management system [27]. Hence, they are primarily responsible for developing and testing the software independent of the underlying ECU system hardware. There after, they perform the system integration testing by mounting the developed software modules on the corresponding ECU system hardware. This system integration testing is usually performed as Hardware in the Loop (HIL) testing where the system is executed under simulated environments [26]. Moving on, the software functions distributed across several such ECU systems of the vehicle, whether developed and system tested in-house or by suppliers, are tested through integration testing at the corresponding vehicle integration test level. At the vehicle integration test level, all the ECU systems and corresponding systems' software are integrated together and testing of the overall software functions which are realized by interaction between several systems is performed either in a real vehicle or in a HIL simulation of the vehicle. As identified by [1], this integration testing is seldom performed by each individual organizational unit responsible for the ECU systems. Rather a dedicated integration test group focuses on testing of the distributed software functions. The exact testing approach, including the testing tools used and the test artifacts generated, may vary across different automotive companies. However, the primary test activities at each test level in the automotive industry generally involve: Test Planning, Test Analysis and Design, Test Build, and Test Execution and Reporting. [1].

Owing to the difference in the test objective at each test level as discussed earlier, software functions are viewed with different levels of abstraction at each test level. Consequently, the software functions are tested against a different set of requirements pertaining to the corresponding level of abstraction. At the vehicle integration test level, where the focus is primarily on testing the execution of distributed software functions in a vehicle, the requirements usually pertain to how the function is expected to execute within the vehicle architecture. At the software and system test level, a function is viewed as a set of software modules distributed across several systems. Hence, the requirements for each system usually pertain to how each individual software module within that system should be realized to ensure the function executes accordingly when the system is integrated

into the vehicle.

As stated in Section 2.1, one facet of challenges faced by the automotive industry today is in the critical context of system software verification and validation. Yet, these set of challenges in the verification approach of testing are not studied adequately in literature and are not complemented by sufficient literature that identifies and establishes the current approach adopted by the automotive industry to test their embedded system software. There is little industrial evidence gathered by empirical research in order to identify the state-of-the-art system software testing approach in the automobile industry as also identified by researchers in [1]. More over, a review of research in test, verification, and validation in the automotive domain has revealed that majority of the research in this area is focused on low-level system specific model-based testing and verification. Studies pertaining to high-level integration testing of distributed software functions are sparse. Such a shortage has also been identified by other researchers like in [2] and [32]. Hence, limited existing research in the areas of automotive industry pertaining to studying the current approach to system software testing, challenges in the current approach and proposed solutions to tackle these challenges for high-level integration testing, motivate the need for the current research that aims to contribute to each of the above mentioned areas by answering the research questions formulated.

2.3 Challenges of Software Testing in the Automotive Industry

The following Table 2.1 summarizes the primary challenges of automotive software testing presented in the identified relevant literature [1][2][3].

Source Literature	**Core Challenge Area**	**Challenge Description**
[2]	Test Effort Measurement	Difficulty in identifying quality assurance value of testing activities at different test levels
	People Knowledge	Difference of opinion of test effort distribution
	Distributed Functionality	Increased test complexity due to distributed nature of software functions
	Test Coverage Metric	Lack of test measurement support
	Variant Handling	Combinatorial explosion of testing due to mass customization

[1]	Requirements and Traceability	Requirements related issues like lack of clearly testable high-level requirements and lack of traceability as a hindrance to testing
	Test Process	Lack of unified test process
	Knowledge Transfer	Lack of means of test knowledge transfer
	Test Tools	Lack of adequate testing tools and techniques
	Quality Assurance through Testing	Shortcomings in quality assurance and measurement
	Test and Defect Traceability	Costly defect fixing and untraceable defects
	Test Documentation	Lack of proper and updated documentation of testing effort
[3]	Unified Test Process and Test Reuse	Overlapping tests across test levels causing waste of time and resources

Table 2.1: Summary of software testing related challenges in the automotive industry as identified in literature [1][2][3]

2.4 Area of Study

Software testing has been and continues to be a vital and mature software engineering research area with research being conducted over several years in several areas like software test tools, test approaches, test practices and test processes. There has also been empirical research study in the area of software testing within the context of various industries dealing with embedded systems software. Yet, it has been identified that there is limited research that focuses on studying the high-level integration testing of embedded systems software in the automotive industry. As motivated in the previous sections, there is an evident and significant need to focus both academic and industrial research within this area. Hence, the research field of embedded systems software testing within the automotive industry is the core area of study for this research as illustrated in Figure 2.1.

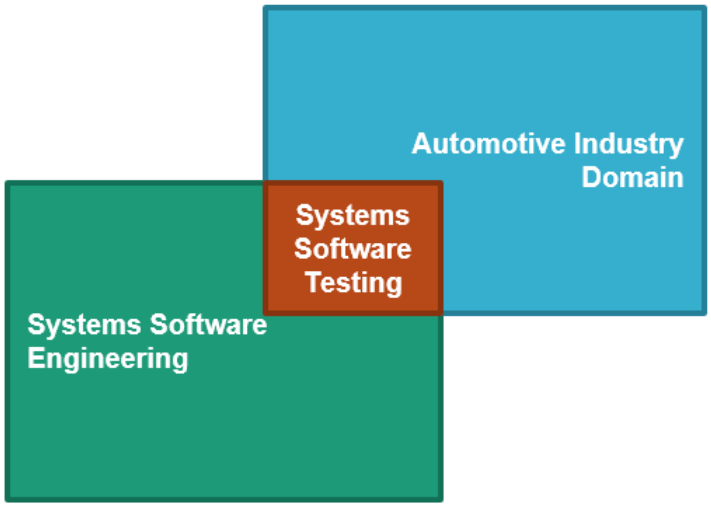


Figure 2.1: Illustration of thesis area of study

Chapter 3

Research Method

This section provides a brief description of the research design that has been laid down to achieve the research aims and objectives presented in Section 1.2. As defined in [33], a research design is fundamentally a logical step-by-step plan laid out to get from the research questions to the research answers. It includes the research method which describes the type of study used to answer the research questions, the data collection method used to collect the required data, and the data analysis method used to analyze the collected data.

An illustration of the research design used for this study is presented in Figure 3.1.

3.1 Motivation

There exist several empirical research methods that have significantly contributed to the knowledge pool of the software engineering field. Four such major research methods as identified by [34] are,

Survey involves a systematic approach of gathering and analyzing information from a representative sample of a specific population. It aims to derive conclusions about the characteristics of the population under study [35]. This research method has been found to be most suitable for descriptive studies which aim to portray the current state of a particular phenomena or situation. While, one of the primary objectives of the current research is to identify the testing approach implemented currently within the automotive industry, the overall aim of the research can be summarized as having exploratory and improving purpose. Hence, to conduct the current research through a survey has been deemed inappropriate.

Experiment can be characterized as a study in a controlled environment that aims to *"measure the effects of manipulating one variable on another variable"* [36]. The current research aims at studying the relationship between the variables (current test approach and its issues) and the effectiveness of the proposed verification strategy in an uncontrolled setting. Hence, an experiment-based controlled

research setting has been deemed inappropriate.

Action Research involves studying a phenomena or situation with an aim to influence or change certain aspects of the subject(s) under study and analyze the outcome [36]. This research method has close links to case studies. Its fundamental distinction from case studies is its inclination towards improving the phenomena being studied. While the current research aims at proposing an enhancement, this enhancement is derived from an in-depth study of the phenomena which is handled more suitably by the case study research method.

Case Study has been given several definitions over the past couple of years. All these definitions fundamentally suggest that a case study is an empirical method for *"investigating contemporary phenomena in their context"* [34]. The motivation behind the choice of selecting case study as the most suitable research method is in multiple-folds. Firstly, since the current research aims at investigating and solving a problem which is emerging across the automotive industry, a case study has been deemed apt to study the problem and aim to solve it within its real world setting. Moreover, a case study research in software engineering discipline is often most suitable for *exploratory*- find what is happening - and *improving*- try and improve the studied phenomena- purposes [34]. Since the current research objectives can be summarized as having both exploratory and improving purpose, case study has been chosen.

3.2 Literature Review Design

As illustrated in the research design (refer to Figure 3.1), the first step of the research was to conduct a literature review to identify relevant scientific literature pertaining to the approach used to test distributed automotive embedded software across the test levels in the automotive industry. This step helped capture data related to the problem domain and improve understanding of the context and background of the research study while partly answering RQ1. It helped identify the state-of-the-art software testing approach used in the automotive industry. Such a literature review has been found to often precede a case study research [34] to essentially help build knowledge which will be useful through the course of the study. There are some fundamental steps that are to be followed to conduct a literature review as reported by Rowley and Slack [37]. Based on the current simplistic nature of the literature review to be conducted, these steps have been tailored to suit the current research as follows:

Step 1: Identify the suitable information resources - As stated by Rowley and Slack [37], there exist several sources of information from where the relevant literature can be obtained. Bearing in mind the nature of the literature

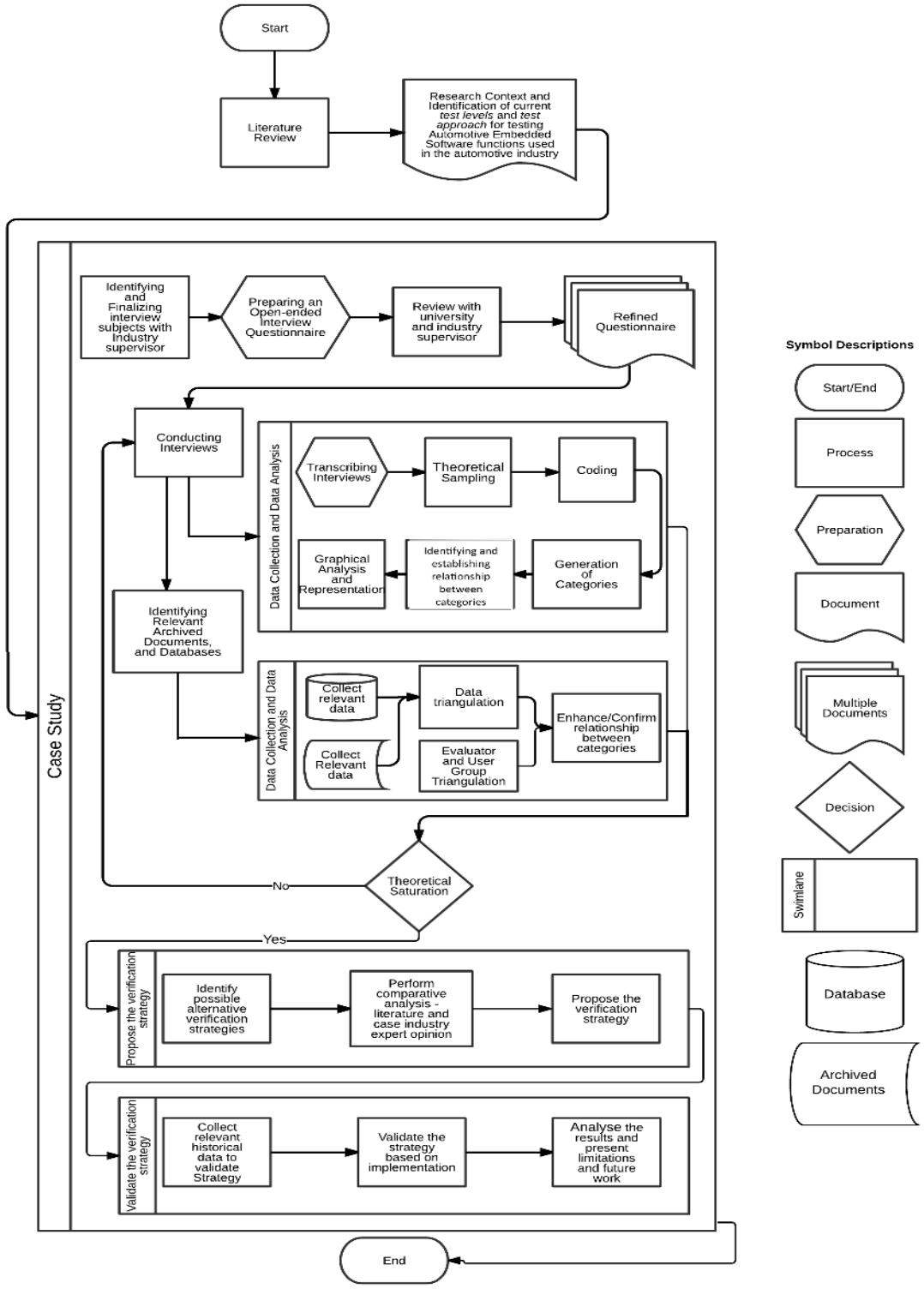


Figure 3.1: Research Design

review to be conducted, Online Databases and Books have been chosen as the information sources. It is important to note here that, due to the researcher's access to a wide range of online databases, the digital copies of the books deemed relevant to the research were used based on their online availability. The online databases that have been used for retrieving the desired scientific literature in the form of journal articles, conference papers, technical reports and books include IEEE Xplore, Engineering Village, Science Direct and SpringerLink.

Step 2: Search for relevant literature from the chosen information resources - This step involved formulating efficient search strings to search for relevant literature from the chosen information sources. Keywords including 'verification', 'software test*', 'embedded system software test*', 'automotive OR automobile' and 'test strategy OR test approach OR test process' have been used to perform the first round of filtering of the umpteen scientific literature that the chosen information sources contain. Based on the search results obtained, further enhancement of the search keywords has been done if deemed necessary. From the search results, the titles, abstracts and availability of articles were used to perform the second round of filtering. If the title and abstract were found to be relevant and the article was found to be available, the full text of the article was read to perform the final assessment of it's relevance to the current research. In this manner, the guidelines provided by [37] were followed to formulate efficient search strings and to filter search results to select scientific literature which were most relevant to the study.

Step 3: Draw up the literature together - The selected scientific literature were then studied as part of the final step of the literature review. The data relevant to the research aim and objectives was captured and documented accordingly in the Introduction (Chapter 1) and Background and Related Work (Chapter 2) Chapters.

3.3 Case Study Design

The case under study and the design laid out to carry out the study within the context of the case are presented in this section.

3.3.1 Case and its Unit of Analysis

Scania [38], is one of the major manufacturers of commercial heavy vehicles in the European automotive industry. It's Research and Development Center, located at its Headquarters at Södertälje, Sweden has been selected as the case for this research. As defined by Yin in [33], a case is fundamentally anything which is a *"contemporary phenomenon in its real-life context"*. Since, the current research

focuses on verification through testing of automotive embedded software, Scania has been chosen to be an appropriate case of a large-scale automotive company. Within the case, the unit of analysis is the test approach implemented at Scania to test their distributed software functions across the test levels.

3.3.2 Case Study Protocol

Case study protocol presents the design in the form of the procedure used to conduct the case study research. It contains the data collection and data analysis protocols, each of which are presented below.

3.3.2.1 Data Collection Protocol

3.3.2.1.1 Data Collection Methods

In order to fulfill the primary research objectives stated in Section 1.2 and thereby answer the research questions, a first degree data collection method of interviews, where the researcher is in direct contact with the subjects to collect data, has been selected as one of the data collection methods. In addition, relevant archived documents and databases, which are forms of third degree data collection methods, have been chosen as additional sources of data. Both qualitative and quantitative data was collected using the above mentioned methods. Such a combination of qualitative and quantitative data is said to often provide better understanding of the phenomenon under study [34].

3.3.2.1.2 Selection of Interview Subjects

The interview subjects were selected with the assistance of the industry and university supervisors. Initially, five distinctive and distributed software functions like fuel level display and advanced emergency braking were chosen. Then, practitioners involved in test planning, test execution and test reporting for these functions at different test levels were identified. Hence, it can be inferred that a non-probabilistic judgement based sampling technique [39] has been employed. The primary objective while selecting the interview subjects was to ensure test approaches implemented at different departments within the organization can be captured adequately. Such an approach of basing interview subject selection on differences and not by trying to replicate similarities is recommended in research [34]. Moreover, selecting participants who best represent or have most suitable knowledge of the research topic are most appropriate and recommended to ensure reliability and validity of the data collected [40]. Hence, it was decided to select participants who are involved in, and have adequate knowledge and experience in testing. The identified practitioners included a total of 13 engineers belonging to 3 categories based on test levels, which are, System Test Engineers (participant category 1), Function Owners (participant category 2), Lab and Vehicle Integration Test Engineers (participant category 3) from different departments of the

organization. The size of the interview participants is considered to be adequate, based on Rowley's [41] suggestions that as a rule of thumb for new researchers adopting a pragmatic approach he/she should aim at least 12 interviews lasting for around 30-60 minutes each.

3.3.2.1.3 Interview Design

The interviews conducted were face-to-face, semi-structured interviews with open-ended and close-ended questions. The questions fundamentally focused on capturing information related to the current test approach, issues with this approach and overall function test coverage information pertaining to the chosen distributed software functions at the case company. These interviews were conducted in three rounds. Each round included interviews of engineers belonging to each participant category and at least one from each category i.e. Round 1 with a subset of System Test Engineers, Function Owners and Integration Test Engineers and similarly for Round 2 and Round 3. Each round of interviews had a similar questionnaire framework that was slightly tailored to be suitable for each interview participant category. The interview was structured as a time-glass model where the interview starts with open-ended questions, moves towards more specific and structured questions and towards the end moves back to open-ended questions [34]. Each interview was planned to last for 30-60 minutes taking into consideration the amount of time the interviewees were willing to make available for the interviews. Each interview across all three interview rounds contained three phases. These phases, designed based on the recommendations provided by Runeson [34] and Turner [42], are as follows:

Phase 1: Introducing the research and the researcher - This phase marked the beginning of the interview where the researcher briefly introduces himself/herself (name and department). Then he/she moved forward to briefly present the aim of the research and the interview goals and objectives. This phase was essential as an initial setup of the interview environment so as to ensure the participants were more comfortable based on an increased knowledge of the interview objectives and the interviewer [42]. Interviewees are said to be less likely to fully participate if they do not know the goals of the study [43].

Phase 2: Collecting general interviewee information - In this phase, general information pertaining to the interviewee such as his/her experience, current role in testing and role responsibilities was collected.

Phase 3: Focusing on main interview questions - This phase took up the largest part of the interview and included collecting all data relevant to the current research focus from the interviewee using the interview questionnaire.

3.3.2.1.4 Formulation of Interview Questionnaire

There were four fundamental steps followed for formulating the interview questionnaire as stated below.

Step 1: Formulate questions - The first step was to formulate interview questions to be part of the questionnaire. Here, the open-ended and close-ended questions were formulated based on two major sources - the research objectives and the data collected as part of the initial literature review. Interviews are fundamentally used as a method to collect relevant data to answer the research questions. Yet, the interview questions are not expected to exactly match the research questions since there is a need for the questions to be formulated in a way that is

a) understandable, neutral and logically leads to the answers [42] and
b) encourages the interviewees to share as much information in and around the subject as possible to answer the research questions [41].

Hence, initially, the questions were formulated in a way that was deemed understandable and open-ended by the researcher based on the research objectives and efficient interview question formulation suggestions provided in [41] and [42]. There after, data collected from the literature review helped identify a major factor for developing an effective cross-functional verification strategy was to identify current test issues that act as a hindrance. Based on this understanding, the questions were enhanced to include open-ended questions that help identify the current test approach related issues in the case company that can be addressed as part of the strategy. The initial questionnaire draft hence contained questions with three broad aims. First set of questions were formulated to capture current test approach information. The next set of questions were formulated to identify function test coverage and test artifact information. The last set of questions were used to explicitly capture information pertaining to the issues in the current test approach.

Step 2: Review and revise questionnaire with industry and university supervisors - The next step was to review the questions with industry and university supervisors. This step was considered to be essential since their experience in either conducing such interviews or being a subject of such interviews in the past could be used to enhance the interview questionnaire to ensure high quality of data can be captured efficiently. Hence, initially, the formulated questions as part of the questionnaire were sent to the industry and university supervisors. Their review and suggestions captured through e-Mail and face-to-face or Skype meetings was then used to revise the questionnaire. The revised questionnaire was re-sent to the supervisors for further suggestions and review of the revised work. Hence, this step was conducted in a loop till the supervisors and the researcher agreed on the final version of the interview questionnaire. This final version was

similar to the initial draft with the major changes being focused on enhancing the language to ensure unambiguity and formulating additional questions which were to be posed to the interviewee if in case the primary question was not answered adequately. Hence, the questionnaire was designed with flexibility, which is one of the major advantages of conducting semi-structured interviews [34][41][42]. Such flexibility facilitates the participant to fully express his/her viewpoints and helps to capture a complete set of required data.

Step 3: Tailor questions for each interview participant category - On approval of the final interview questionnaire by the supervisors, the questionnaire was duplicated for the different interview participant categories. This duplication process involved tailoring the questions to be more participant-category specific. For instance, the question for interviews with system test engineers across the different rounds *'What are the broad steps of testing that are undertaken at the system test level?'* was tailored to be suitable for interviews with integration test engineers by adapting the question to being *'What are the broad steps of testing that are undertaken at the vehicle integration test level?'*. A similar procedure for tailoring the rest of the questionnaire was implemented and a final set of three interview questionnaires containing open-ended and close-ended questions for the three interview participant categories was generated. Each of these interview questionnaires are presented in Appendix A.

Step 4: Update interview questionnaire - This step facilitated flexibility of the case study design, which is one of its key characteristics [34]. Since data collection and data analysis was conducted simultaneously, on identification of any new insights for which data was required to be collected, the interview questionnaire was updated for the subsequent interview rounds for the relevant interview participant categories. This enhanced the quality and completeness of the data collected.

3.3.2.1.5 Interview Planning and Setup

On selecting the interview participants and designing the interview, the next step undertaken was to plan and setup the interview. The interview participants were sent an invitation (Appendix B) over the company's dedicated Microsoft Outlook account. This interview invitation included the date, time and place of the interview along with a brief introduction of the researcher and the aim of the interview. This interview meeting included the interview participant, the researcher and the industry supervisor. The presence of the industry supervisor, who had adequate knowledge of the topics being discussed in the interview, ensured that high quality content was discussed throughout the interview as one of the means to keep the participants engaged and enthusiastic. Based on the participant's availability, the invitation was either accepted or an alternative date and time was suggested

based on which the interview was rescheduled, an updated invitation was sent and a confirmation was received. The interview was set up at one of the available meeting rooms so as to ensure minimum distraction which was essential to ensure a smooth interview was conducted [42]. The entire interview was audio recorder on taking the consent of the interview participant. The interview implementation suggestions stated in [42] have been studied and adopted while conducting the interviews to ensure they were efficient and effective.

3.3.2.1.6 Transcription

Since all the interviews conducted were face-to-face and audio recorded, the same transcription software and transcription procedure was employed for transcribing all interviews conducted. Here, an audio-to-text transcription software *ExpressScribe* was used. On concluding the interview, the audio file recorded on the researcher's phone was renamed from default to a file name that had the format *'Participant Name.Participant Category.Interview Round'* like for instance 'Jacob.SystemTester.Round2'. The audio file was then duplicated and stored on the researcher's Dropbox folder and Company Desktop folder dedicated to the interview data files. In such a way, the interview data was stored in multiple locations to ensure there was no loss of data in case of undesirable or unanticipated technology issues. The interview was then transcribed using the software either on the same day or latest by the end of the week. The generated interview text file was saved with the same file name format. At the end of the interview period, the audio recordings and corresponding text files for the 13 interviews conducted were present within a dedicated folder on the researcher's Desktop and Dropbox.

3.3.2.1.7 Identification of Relevant Databases and Archived Documents

As mentioned earlier, archived documents and databases have been chosen as the additional sources of data for this research. Data collection from these sources involved the following steps:

a) The first step was to identify the relevant archived documents and databases like the requirements documents, test strategy documents, test reports and test databases. This was achieved by ensuring that one of the objectives of the interviews was to identify relevant artifacts and request access to these artifacts from the interview participants. After the interview, the participant was sent a follow-up e-Mail thanking him/her for their participation and subtly re-requesting them to provide links and required permissions to access the relevant artifacts.
b) On obtaining the required permissions and links to the relevant archived documents and databases within the company internal folders, the documents were collected and stored within a dedicated folder for further data analysis.

3.3.2.2 Data Analysis Protocol

3.3.2.2.1 Data Analysis Methods

Fundamentally, a data analysis technique that deals with predominantly qualitative data, like in this study, has two parts- *hypothesis generation* and *hypothesis confirmation* [34]. The data collection protocol of the case study facilitated collection of both qualitative (predominant) and quantitative data through interviews, archived documents and databases. Such a combination of qualitative and quantitative data and consequently combined methods of qualitative and quantitative data analysis has been found to be a better approach to investigate most software engineering issues than either one in isolation [43].

One commonly used strategy that facilitates the analysis of qualitative and quantitative data is the Grounded Theory (GT) approach [43]. As the name suggests, GT approach provides a means of generating theory that is "grounded" in the data in an exploratory fashion [44]. It has been studied extensively in literature in different contexts [44][45][46][47][48]. GT approach allows streamlining and integrating data collection and analysis. There is no initial hypothesis formulated, rather the theory is derived form the data collected. Thus, the validity of the formulated theory strengthens with each collected input [49].

For this study, the popular GT approach, Constant Comparison Method, first introduced by Glaser and Strauss [50] has been chosen as an appropriate data analysis technique for hypothesis generation. Though Glaser and Strauss designed this method together, they soon split and presented their own versions of this GT method. Strauss and Corbin's GT version focuses on the need to have verification as an essential component of data analysis and theory building [51], the need to have research questions pre-set as a means of establishing boundaries and the need to ensure the researcher has certain pre-acquired knowledge of the phenomena under study through literature [46]. On the contrary, Glaser's GT version fundamentally opposes all the above factors and maintains that verification is not a primary concern, that research questions are developed during coding process and that pre-acquired literature knowledge should be avoided to ensure there are no prior assumptions formed by the researcher. In the current study, since research questions are pre-formulated and pre-acquired literature knowledge of the phenomena under study is relied upon by the researcher, Strauss and Corbin's GT version [49][52] has been adopted.

The theory so generated has been confirmed through an important and popular hypothesis confirmation method of triangulation [43]. Hence, the final output of the data analysis presents results that are both grounded and supported by a body of evidence. The steps followed during data analysis are elaborated in the following sections.

3.3.2.2.2 Theoretical Sensitivity, Theoretical Sampling and Theoretical Saturation

The Strauss and Corbin's GT version that has been adopted for this study supports the concept of theoretical sensitivity where the researcher uses not only the data being collected but also his pre-exisiting knowledge and literature to identify what is important [53]. To do so, the first step in data analysis using GT was to perform theoretical sampling. Theoretical sampling is defined as *"sampling on the basis of emerging concepts"* [52]. Data analysis was hence implemented simultaneously with data collection. On conducting an interview and transcribing the corresponding audio file, the generated content was exported to a word processor. This was done to correct any grammatical mistakes made during the transcription process. There after, with the final interview transcript in hand, a critical data analysis activity termed as 'micro analysis' [52] was performed. Here, the interview data was thoroughly read and scrutinized line by line. This was done in order to identify and note the concepts in the form of new questions that arise or theoretical ideas and thoughts of the researcher. These concepts can be termed as building blocks of the 'emerging theory' [52]. Each such concept identified was transformed into a 'code' that represents this concept and which was later to become a part of the theory [46], more about which is discussed in the next section where the coding process is further elaborated. The emerging theory in the form of 'codes' of the concepts identified was noted using the technique of memoing [46] within the word processor file. This information was then used to refine the interview questionnaire and align the study with the emerging results which was the primary aim of undertaking theoretical sampling. This procedure was implemented throughout the data collection period. Hence, data collection was driven by concepts emerging from the evolving theory.

There comes a point in such a theory building process where there is no new knowledge in the form of code categories or relationships among these categories that can be identified on continued data collection and data analysis. This phenomena is termed as theoretical saturation [52]. Theoretical saturation remains one of the ultimate ways of determining that the data collection and data analysis process can be concluded when there is no new data emerging regarding a category or when all categories and their relationships are well identified and established. It was hence used as an exit criteria for the data collection and data analysis process, and an entry criteria for the study to move to proposing a verification strategy based on the data analysis results.

3.3.2.2.3 Coding

According to Strauss and Corbin, coding can be defined as *"the analytic processes through which data are fractured, conceptualized, and integrated to form theory"* [52]. During micro analysis of the data, the concepts identified were converted

into 'codes' which were used to represent the concepts. These codes formed the initial emerging theory and were refined through several steps to identify all relevant *code properties* - which are its characteristics that give it a meaning, *code categories* - which represent a group of concepts that stand for a specific phenomena and *relationships* among them to establish the final theory from the data.

Coding based qualitative data analysis can be performed by adopting one of the two approaches - with the use of Computer-Assisted Qualitative Data Analysis Softwares (CAQDAS) like ATLAS/ti [54] or without the use of CAQDAS. Studies conducted to understand the merits and demerits of both these approaches like [54] and [55] have shown that each approach has its own pros and cons that are arguable. But the fundamental idea remains that CAQDAS is just a tool that is used to assist, to a certain extent, in managing the complexity that is often faced during qualitative data analysis. The adoption of CAQDAS does not replace the need for creative and meticulous data analysis that is fully dependent on the researcher. Hence, based on careful consideration of the pros and cons, data analysis was performed without the use of CAQDAS.

According to Strauss and Corbin's approach to data analysis, the coding process involves three types of coding for the data - *open coding, axial coding* and *selective coding* [46]. How each of these coding was performed is elaborated below.

Open Coding: Open coding is an iterative coding technique where the data being analyzed is broken down into meaningful units and labelled with codes that represent the concepts that are indicated by these data units [46]. This part of data analysis pertained to naming and categorizing the phenomena (represented as concepts [52]) by examining the data. On breaking down the interview data into units, these units were constantly compared for similarities and differences to identify the underlying concept they represent and then an open code was attached to this text [47]. This open code was either directly taken from the text or generated by the researcher, whichever was deemed as most suitable to aptly represent the meaning of the data unit [46]. While open codes were being generated in such a manner, the researcher also focused on identifying open codes that were conceptually similar or related in meaning, so as to group them under higher abstraction concepts called 'open code categories' [47].

For instance, consider the following snippet of an interview transcript with a function owner.

"Q. As a function owner, what would you say is your role in testing the software function?
A. We have been discussing this aspect off late that as a function owner what

should I be testing... But to be honest there is not good coverage from my side."

On reading and analyzing this data unit from the transcript, the phenomena that there might be a possibility that there is inadequate knowledge or lack of clear understanding of function owner's test role in the current test approach has been derived. This data unit (text in the transcript word processor file) was then tagged with the open code *'Ambiguity of test role at function level'* . Over time, the concept was further explored in other interview transcripts and was identified to be occurring in several instances (interview transcripts) of the function owner interviews. Hence, this open code was tagged progressively to all relevant data units in the corresponding interview transcripts. Overtime, all such relevant open codes were established which included *'Ambiguity of test role at function level'* and *'Lack of bigger picture'* . A more in-depth analysis of the similarities and difference among the mentioned open codes and other open codes showed that both of these open codes can be best categorized and represented by the open code category - *'People Knowledge Issues'* . Hence, this was established as one of the final open code categories resulting from this step of data analysis. A similar procedure was followed to obtain all the open codes and open code categories.

Axial Coding: Axial coding as defined by Strauss and Corbin is *"the process of relating categories to their subcategories, termed axial because coding occurs around the axis of a category, linking categories at the level of properties and dimensions"* [52]. It was an intermediate step that involved re-fracturing the data that was broken down in open coding [47]. So, this part of analysis involved establishing relationships between the open code categories that were identified through open coding. These axial codes were a means of elevating the data to a higher level of abstraction so as to represent the categories that encompass the open code categories [47].

For instance, three of the open code categories established from open coding and encompassing the relevant open codes include *people knowledge issues*, *knowledge transfer issues*, and *explicit test issues*. Analysing these categories more extensively, has lead to an understanding that there are two fundamental types of relationships that exist among them - *causes* relationship and *hindrance* relationship. The causes relationship between issues is used to represent a set of issues that are caused or exist due to another set of issue in a cause-and-effect manner. Hence, solving the root cause issues is expected to solve the resultant effect issues. On the other hand, issues that are connected through the hindrance relationship indicate that one set of issues are causing a hindrance to solving another set of issues. Hence, solving issues under both these categories is required to get a complete solution. Going by these definitions, it was established through further analysis of the data that knowledge transfer issues of having inadequate means of sharing knowledge among test engineers across test levels are a hin-

drance to solving people knowledge issues of having a lack of unambiguous and clear bigger picture of the testing approach. Moreover, both these issue code categories along with other issue categories are a cause of the explicit test issues. On establishing all such possible relationships between all open code categories, relevant axial code categories were identified. For instance, the axial code category - *People Issues* was seen to encompass the open code categories of *people knowledge issues* and *knowledge transfer issues* since both of these were found to be predominantly people related issues. A similar procedure was followed to identify all open code category relationships and generate all axial code categories.

Selective Coding: The final step of the coding process was selective coding which involved integrating the axial categories and refining the theory [52]. At this point, a 'core category' around which the theory was built was identified. This core category was central to all the axial categories and the final theory was established.

3.3.2.2.4 Triangulation

Triangulation is the hypothesis confirmation method used in this study. It deals with gathering different types of evidence to confirm the theory generated [43]. There exist several approaches to triangulation which can be categorized into two main groups - *between-methods* triangulation and *within-methods* triangulation [56]. For this study, a between-methods triangulation approach of data triangulation [34] and two within-method triangulation approaches of evaluator triangulation and user group triangulation [56] have been used. A brief description of each of the three triangulation methods adopted is as follows:

Data Triangulation: Data triangulation is a between-methods approach that involves using more than one source of data in order confirm the theory generated [34]. Here, data from relevant archived documents and databases was used to confirm the theory generated from the analysis of the data collected from interviews.

User Group Triangulation: User group triangulation is a within-method approach where in multiple users from the same group and/or multiple user groups are used to confirm whether the theory is specific to a particular group or is more generic [56]. Here, conducting interviews with participants belonging to different categories and having multiple participants from each category ensured user group triangulation was implemented.

Evaluator Triangulation: Evaluator triangulation is a within-method approach that involves having more than one evaluator or facilitator evaluate the same phenomena to see if they all agree on the results obtained [56]. Here, having several practitioners and researchers evaluate the results of data analysis and the theory generated ensured evaluator triangulation was implemented.

3.3.2.2.5 Validation using Existing Historic Data

The cross-functional verification strategy proposed as part of the study is based on the data analysis results. This proposed strategy has been planned to be validated using one of the following two methods: validation using historic data or validation through survey. Owing to the nature of the proposed verification strategy, and considering the recommendations from the supervisors, validating the strategy using existing historic data has been deemed apt. Here all required and relevant data pertaining to a distributed software function - Fuel Level Display has been collected during data collection and was used to implement the proposed strategy and analyse the results of the strategy to assess its feasibility and effectiveness.

Chapter 4

Case Study Results

This chapter presents the results obtained on conducting the research as per the research design laid out and reported in Chapter 3.

4.1 Current Approach to Test Distributed Software Functions at Scania

This section presents the current approach implemented at the case company, Scania, in order to test their distributed software functions. This information has been captured to answer RQ1 through the interviews conducted and from relevant Scania internal documents like [4] specifying the test roles and responsibilities for each test level.

A function of the software of an automotive vehicle is viewed as a set of inputs and outputs across one or more ECU systems that results in the execution of a specific behaviour of the vehicle. Examples of such software functions include fuel level display, oil level display and more safety critical functions like advanced emergency braking. A graphical representation of how a software function, termed as a User Function (UF), is viewed at different test levels across the V-model at Scania is illustrated in Figure 4.1.

Considering the left side of the Scania V-model, owing to the increasingly distributed nature of software functions across several systems of a vehicle, a UF is viewed as a set of Allocation Elements (AEs) at the system test level in case of those ECU systems that are developed in-house. AEs are fundamentally software modules that are allocated or embedded to a particular ECU system. Hence, each UF at Scania is viewed as a set of AEs distributed across one or more ECU systems which on interaction result in the execution of the function's desired behaviour within the vehicle. A particular UF can require one or more AEs allocated to one ECU system for its execution, and, on the other hand, one ECU system can contain AEs required for the execution of more than one UF. Hence, at the system test level, the system requirements take the form of a set of

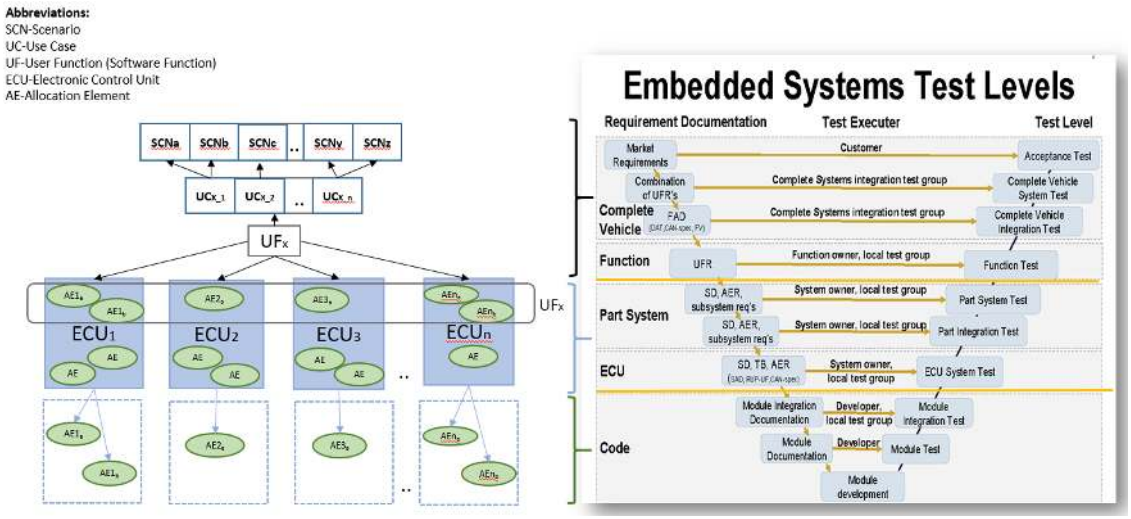


Figure 4.1: How distributed software functions are tested across different test levels of the V-model at Scania

AE requirements written in structured natural language.

At the vehicle integration test level, a UF is viewed as a set of *Use Cases* and its *Scenarios*. A use case according to the OMG-Unified Modelling Language (UML) specification is *"the specification of a sequence of actions, including variants that a system (or a subsystem) can perform, interacting with actors of the system"* [57]. This definition remains unchanged in the OMG-Systems Modelling Language specification [58]. A use case hence describes a piece of behaviour of a system [59]. Putting this in the current context, use case of an automotive embedded software function can be defined as 'the specification of sequence of actions, including variants, that a function can perform, based on interaction among one or more vehicle sub-systems and the environment'. Here, the environment can be the driver, another vehicle, etc. and vehicle sub-system is used to refer to any ECU system like the Engine Management System, Brake Management System and Instrument Cluster. Therefore, use cases of a function act as part functions describing different behaviours of the function that together make up the entire function and its comprehensive behaviour.

Each single sequence of interaction between the ECU systems and the environment is termed a scenario of the use case [59]. For instance, one scenario for a function use case may pertain to the sequence of steps it follows to execute the desired behaviour in a vehicle with a gas engine and another scenario may pertain to an alternative unique sequence of steps it follows to execute the same desired behaviour but in a vehicle with a diesel engine. Therefore, a use case can be considered to be a set of scenarios. Scenarios are popularly used for

requirements specification and testing of the overall system implementation and have been recognized by UML as being part of the behavioural diagrams called Sequence Diagrams [60]. Scenarios are usually represented using formal graphical notations like the Message Sequence Charts (MSCs), Live Sequence Charts, etc. [59]. At Scania, the scenarios of a software function's use case are represented using MSCs. Visually, an MSC consists of a number of interacting ECU systems and the environment each represented with a vertical line. The communication among the different ECU systems is represented using horizontal arrows between the vertical lines from the sending to the receiving system. The MSCs generally include notes that are present within the diagrams to provide more information regarding the interaction and the function execution.

At the vehicle integration test level, in addition to the use cases and scenarios, each UF has a corresponding requirements document in which the function requirements take the structured natural language form.

Mapping the structural breakdown of a UF to the right side of the Scania V-model pertaining to testing the UF as illustrated in Figure 4.1, each test level was identified to have the following primary test objectives and test approach. It is important to note here that, since the scope of the thesis research is limited to test coverage based on requirements, the code test level where the test focus is inclined towards code coverage was not considered. Therefore, only the system, function and vehicle integration test levels were considered.

The ***System Test Level*** at Scania is where the system testing of the individual ECU systems is performed. In the V-model, this test level corresponds to ECU and Part System levels (refer to Figure 4.1). Different test groups belonging to different departments across the organization take up the responsibility for testing one or more of the ECU systems.

At this test level, the test target is the software of the ECU system under test. This includes all software modules (AEs) allocated to that ECU system. The software is tested by mounting it on the corresponding system hardware. Here, white-box and black-box testing of the individual ECU system is done to ensure the system's software behaves as expected. Generally, the test environment used for system test level is a HIL test rig lab where parts or all of the ECU system's interfaces are simulated. As mentioned earlier, the expected system behaviour is presented in the form of the system's requirements (set of AE requirements documents) which generally take a structured natural language form. Hence, testing is done and test results are reported against these system requirements. The test coverage of the system is therefore presented in terms of the number of system requirements that have been tested. In case of supplier systems, testing at this system test level is performed at the suppliers end, and it is up to the sys-

tem test leader to decide how much system testing should be taken up there after.

The dynamic nature of the automotive industry requires continuous development of software and corresponding systems. This in turn leads to multiple software versions that need to be tested across all the test levels. Owing to this factor, testing of the software at the system test level at Scania is taken up in short-cycles called test rounds all around the year where each test round spans from one to four test weeks. Hence, test reports are generated once every week to every four test weeks depending on how each ECU system test group has planned to execute its testing activities.

Moving on, as the name suggests, ***Function Test Level*** deals with testing each of the distributed software functions across all the ECU systems that are involved in its execution. The responsibility for testing each function at this test level is given to an individual called the 'function owner' in an appropriate organization department.

At this test level, the test target is the corresponding software modules across all the ECU systems, regardless of their relative importance, that are required for the execution of the concerned function in a vehicle. Hence, the function requirements are to be tested and not a specific ECU system containing major part of the functionality. Here, black-box testing of the function is performed based on the function owner's knowledge of the function to check for the correctness of the function output in several scenarios. The test environment is either a HIL test rig lab, where part or all of the systems involved in the function are simulated, or a real vehicle. At this test level, it was identified that testing is mostly performed based on function owner's knowledge and not systematically against any function requirements. Moreover, no test reports are generated from this test level.

The ***Vehicle Integration Test Level*** is where all the distributed software functions of the vehicle are aimed to be tested. It is at the highest level in the Scania V-model corresponding to the complete vehicle level in Figure 4.1. This test responsibility is taken up by dedicated integration test groups at Scania.

At this test level, the test target is the interface communication among the ECU systems for the execution of all software functions of the vehicle. Hence, each software function is aimed to be tested by testing the communication among the involved ECU systems using both a white-box and black-box testing approach. One test group performing the integration testing in a HIL test rig lab is responsible for white-box testing of the communication among the ECU systems as a means of testing the software functions. On the other hand, the test group performing the integration testing in real vehicles is responsible for black-box testing of the communication among the ECU systems as a means of testing the software

functions. Here, the integration testing by both the test groups is performed and test results are reported against the function scenarios. Hence, currently, the test coverage of distributed software functions is in terms of the number of scenarios of the total scenarios of the function that have been tested.

As mentioned earlier, due to continuous development of software leading to multiple software versions that need to be tested at all levels, the integration test level at Scania plans and executes the testing activities in one-month short cycles called test rounds. During this one-month span, two test weeks are spent on test planning and the remaining two test weeks are spent on test execution. The test reports are hence generated once every four test weeks.

4.2 Summary of Interviews

With one of the primary research objectives being the identification of issues with the current approach to test distributed software functions at the system test level, function test level and the vehicle integration test level, interviews with 13 test engineers from the aforementioned test levels have been conducted in three rounds. A brief description of the interview participants as a means to present their suitability as interview subjects is stated in Table 4.1. The description is in terms of their test role title, test level to which their role can be accurately mapped to and experience in their current role.

Interview Round	**Interview Participant**	**Test Level**	**Test Role Title**	**Experience**
1	Interviewee 1	Complete Vehicle	Actual Vehicle Tester	11 years
	Interviewee 2	Complete Vehicle	HIL Lab Integration Tester	1.5 years
	Interviewee 3	Function	Function Owner for fuel level display	8 months
	Interviewee 4	System	Main ECU 2 Lead Test Engineer	1 year
2	Interviewee 5	Complete Vehicle	Actual Vehicle Tester	5 years
	Interviewee 6	Function	Function Owner for obstructed camera warning	8 years
	Interviewee 7	Function	Function Owner for advanced emergency braking	1 year

2	Interviewee 8	System	Engine Management System Tester	4.8 years
3	Interviewee 9	Complete Vehicle	HIL Lab Integration Tester	4.3 years
	Interviewee 10	Function	Function Owner for oil level display	10 months
	Interviewee 11	Function	Function Owner for gearbox status presentation	5 years
	Interviewee 12	System	Main ECU 1 System Lead Test Engineer	1.5 years
	Interviewee 13	System	Gear Management System Test Developer	3 years

Table 4.1: Brief description of the interview participants' background

As presented above, each of the three interview rounds included a subset of participants such that, in each round there was at least one participant from each of the three test levels. Hence, each interview round ensured that issues from the three relevant test levels could be captured and analyzed to improve the subsequent data collection approach for the remaining interview rounds. Moreover, the experience of the interview participants from each test level ranges from 1 year or less (termed by the participants themselves as being relatively new to the role) to a couple of years (termed by the participants themselves as being very accustomed to their role). Such a heterogeneous set of interview subjects, in terms of their varied experience of test role, ensured that the data collected was not biased and was from different viewpoints - viewpoint of practitioners who are new to the role, of practitioners who are setting into the role and of practitioners who are highly accustomed to their role.

4.3 Transcription

As mentioned earlier in Section 3.3, all 13 interviews conducted were face-to-face interviews that were audio recorded. Each of the audio interview file was transcribed within one week from the interview date using 'ExpressScribe' software. Figure 4.2 presents a snapshot of the transcribing software interface. It visualizes the naming pattern used for the files and the features of the software that assisted in transcribing the audio files.

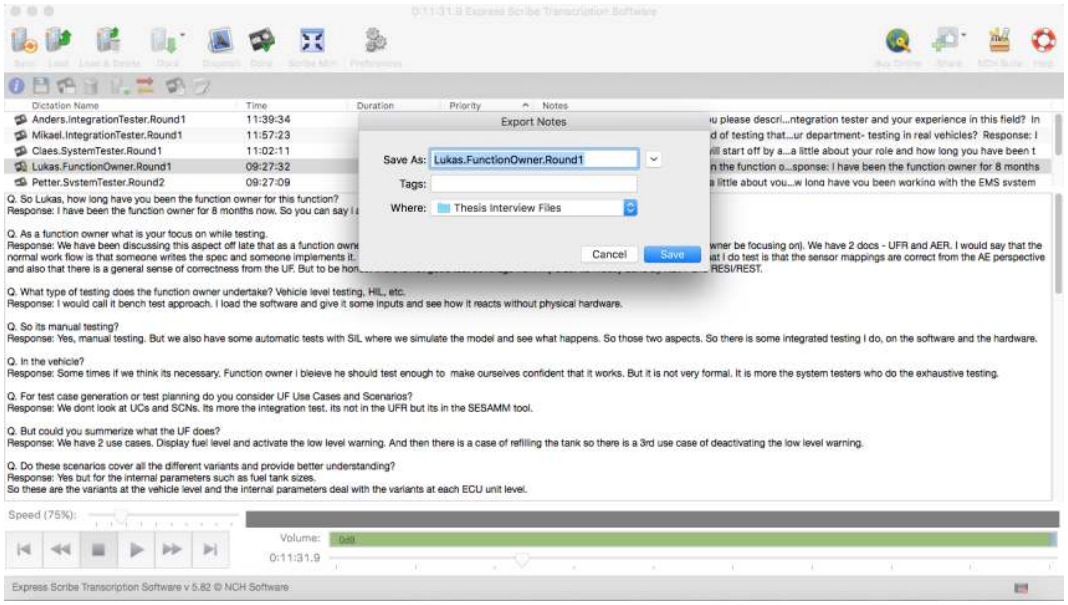


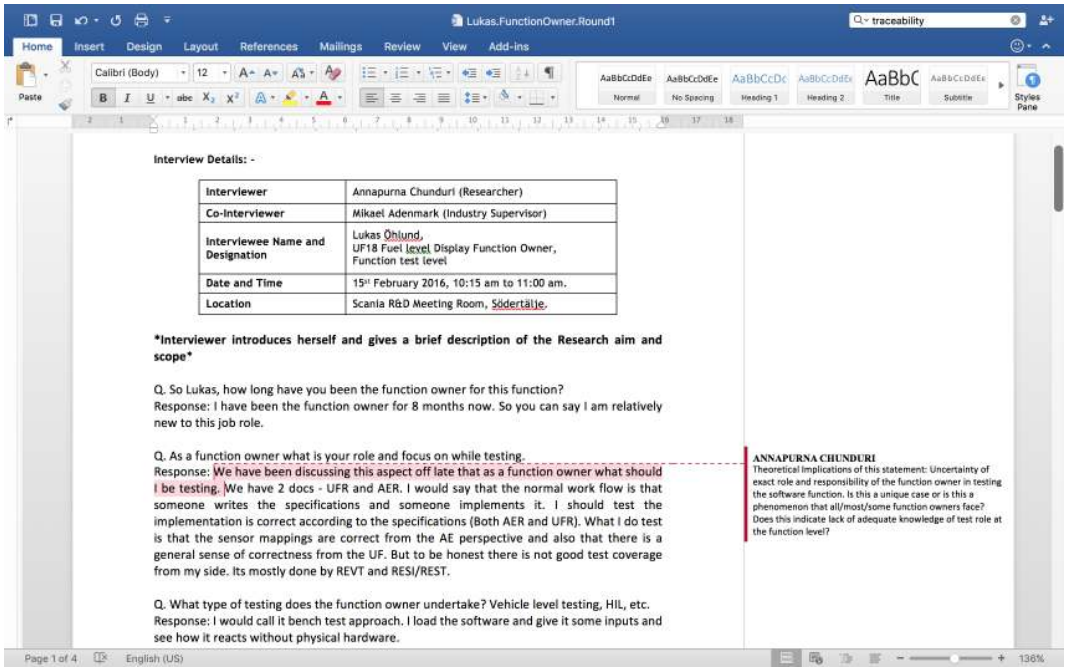
Figure 4.2: Snapshot of ExpressScribe software used to transcribe the interview audio files

On concluding the transcription process, the transcribed interview content was exported to Microsoft Word Processor with the same naming pattern as the audio file to maintain consistency. One of the primary purpose of opting to handle the final version of the transcribed interview using a word processor was to perform grammatical spell checking on the transcribed content. This was necessary to ensure effective transcription of the interviews. Finally, towards the end, the interview participant name, test role, test level, and interview date, time and location were added to the final interview transcript to ensure all details pertaining to the interview are consolidated within this document.

4.4 Post Interviews - Theoretical Sampling

On obtaining the final version of each transcribed interview, the next step was a post interview activity of thoroughly reading the transcript in order to note any new questions or theoretical implications that were derived from the content. These notes were made by highlighting the corresponding text and adding the new questions or theoretical implications identified from this text to a comment added as a memo within the word processor file. An instance of an interview transcript containing text from where an initial theoretical implication was identified by the researcher is presented in Figure 4.3.

A similar instance of an interview transcript containing text from where a new question arose is presented in Figure 4.4. Based on analyzing the criticality, and



Interview Details: -

Interviewer	Annapurna Chunduri (Researcher)
Co-Interviewer	Mikael Adenmark (Industry Supervisor)
Interviewee Name and Designation	Lukas Öhlund, UF18 Fuel level Display Function Owner, Function test level
Date and Time	15th February 2016, 10:15 am to 11:00 am.
Location	Scania R&D Meeting Room, Södertälje.

Interviewer introduces herself and gives a brief description of the Research aim and scope

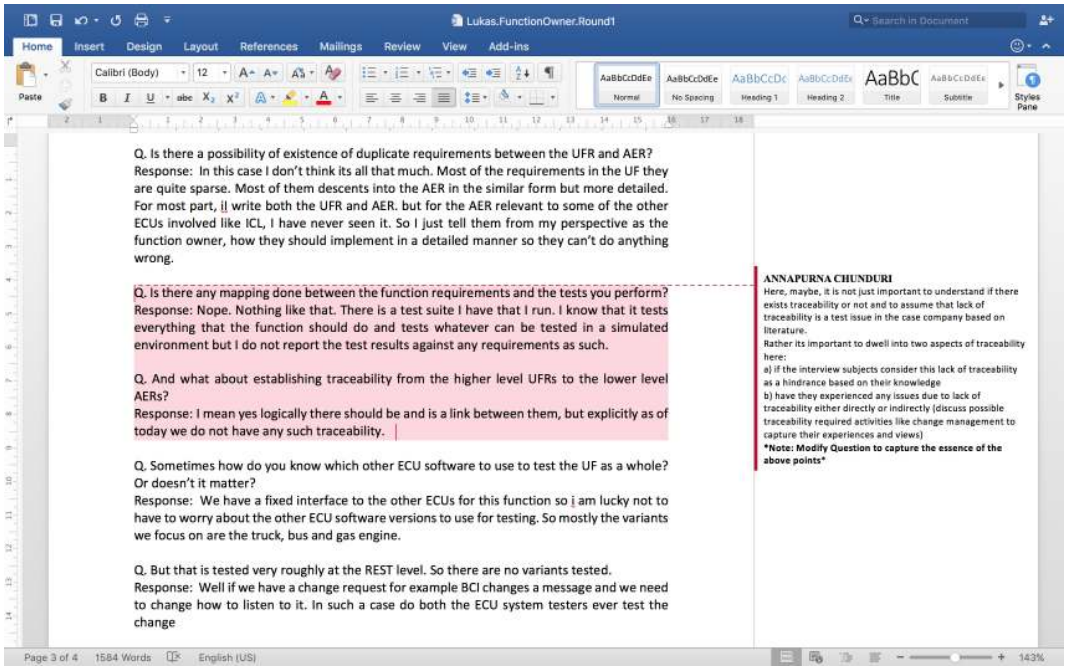
Q. So Lukas, how long have you been the function owner for this function?
Response: I have been the function owner for 8 months now. So you can say I am relatively new to this job role.

Q. As a function owner what is your role and focus on while testing.
Response: We have been discussing this aspect off late that as a function owner what should I be testing. We have 2 docs - UFR and AER. I would say that the normal work flow is that someone writes the specifications and someone implements it. I should test the implementation is correct according to the specifications (Both AER and UFR). What I do test is that the sensor mappings are correct from the AE perspective and also that there is a general sense of correctness from the UF. But to be honest there is not good test coverage from my side. Its mostly done by REVT and RESI/REST.

Q. What type of testing does the function owner undertake? Vehicle level testing, HIL, etc.
Response: I would call it bench test approach. I load the software and give it some inputs and see how it reacts without physical hardware.

ANNAPURNA CHUNDURI
Theoretical implications of this statement: Uncertainty of exact role and responsibility of the function owner in testing the software function. Is this a unique case or is this a phenomenon that all/most/some function owners face? Does this indicate lack of adequate knowledge of test role at the function level?

Figure 4.3: Snapshot of interview transcript with a note of researcher identified theoretical concept



Q. Is there a possibility of existence of duplicate requirements between the UFR and AER?
Response: In this case I don't think its all that much. Most of the requirements in the UF they are quite sparse. Most of them descents into the AER in the similar form but more detailed. For most part, il write both the UFR and AER. but for the AER relevant to some of the other ECUs involved like ICL, I have never seen it. So I just tell them from my perspective as the function owner, how they should implement in a detailed manner so they can't do anything wrong.

Q. Is there any mapping done between the function requirements and the tests you perform?
Response: Nope. Nothing like that. There is a test suite I have that I run. I know that it tests everything that the function should do and tests whatever can be tested in a simulated environment but I do not report the test results against any requirements as such.

Q. And what about establishing traceability from the higher level UFRs to the lower level AERs?
Response: I mean yes logically there should be and is a link between them, but explicitly as of today we do not have any such traceability.

Q. Sometimes how do you know which other ECU software to use to test the UF as a whole? Or doesn't it matter?
Response: We have a fixed interface to the other ECUs for this function so i am lucky not to have to worry about the other ECU software versions to use for testing. So mostly the variants we focus on are the truck, bus and gas engine.

Q. But that is tested very roughly at the REST level. So there are no variants tested.
Response: Well if we have a change request for example BCI changes a message and we need to change how to listen to it. In such a case do both the ECU system testers ever test the change

ANNAPURNA CHUNDURI
Here, maybe, it is not just important to understand if there exists traceability or not and to assume that lack of traceability is a test issue in the case company based on literature.
Rather its important to dwell into two aspects of traceability here:
a) if the interview subjects consider this lack of traceability as a hindrance based on their knowledge
b) have they experienced any issues due to lack of traceability either directly or indirectly (discuss possible traceability required activities like change management to capture their experiences and views)
Note: Modify Question to capture the essence of the above points

Figure 4.4: Snapshot of interview transcript with a note to modify interview questionnaire based on interview data analysis

deducing the need to explicitly include the new question that arose, the corresponding questionnaire was modified for the next round of participants.

Hence, one evident advantage of performing theoretical sampling was identified to be the modification and improvement of the interview questionnaire that assisted in enhancing data collection. In addition, this step proved to be a crucial pre-activity to perform coding. Codes were initially identified from the memos and comments written during theoretical sampling and were later followed by steps that were executed to further strengthen the base for generating these codes.

4.5 Open Coding - Preliminary Results of the Interviews

The preliminary results pertaining to the issues with the approach implemented to test distributed software functions across the three test levels at the case company were identified based on performing the first step of data analysis. This step included performing open coding of the interview data till theoretical saturation was reached. It was performed iteratively over the interview period. The process followed to derive the open codes and open code categories, along with an instance of this process is as presented in Chapter 3.

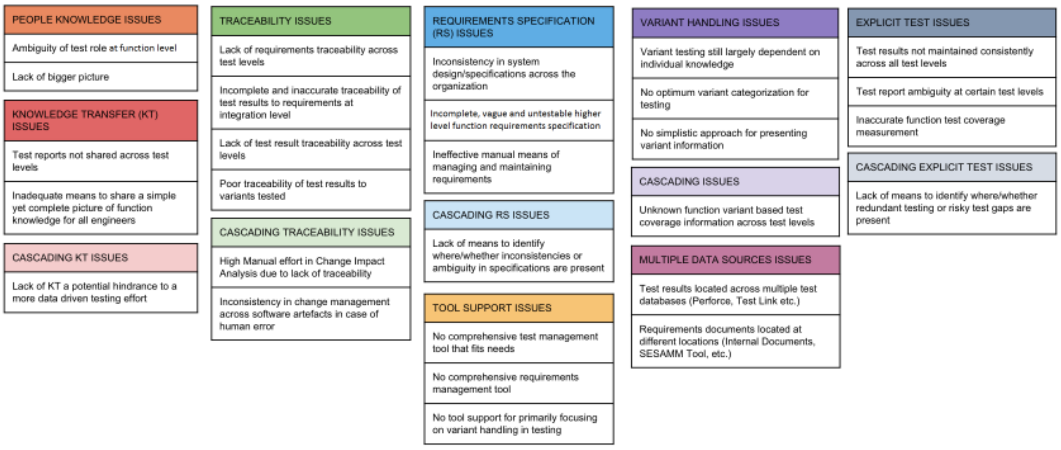


Figure 4.5: Open codes and open code categories

The process of open coding resulted in a set of 28 open codes, each pertaining to a test issue. These open codes were then placed within 8 primary open code categories based on the similarities and differences among the open codes. In addition, it was observed that there were certain open codes pertaining to test issues that were fundamentally present due to the existence of other issues that

were placed within one of the eight primary categories. Such open codes were categorized under 'cascading issues' for each of the primary categories for which this phenomena was observed. This lead to a total of thirteen open code categories that emerged from this data analysis step. A final set of the open codes, the primary open code categories and the secondary open code categories in the form of cascading issues is illustrated in Figure 4.5. Wherever applicable, the primary and secondary open code categories are color coordinated in order to enhance understandability.

Studying the preliminary qualitative data analysis results using a quantitative approach provides a means of motivating the inclusion of each of the identified open code categories in the final set of categories emerging from this step of data analysis. Hence, the frequency of occurrence of each one of the eight primary open code categories among the thirteen interviews conducted was identified.

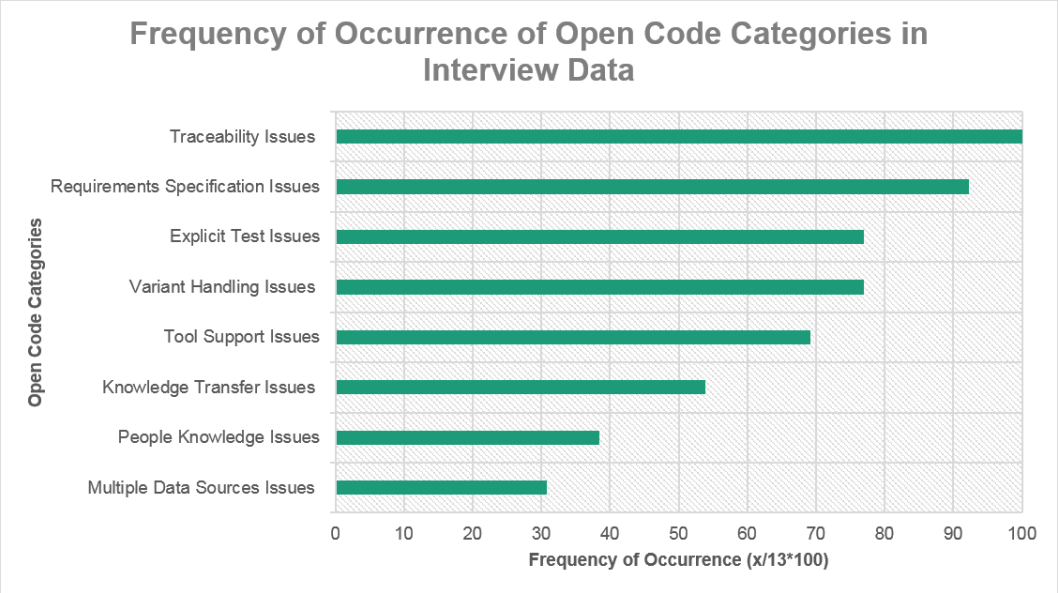


Figure 4.6: Frequency of occurrence of open code categories

The presence of any one of the open codes in the interview transcript was considered as one occurrence of the open code category it belonged to. Hence, the frequency of occurrence of an open code category was calculated as the number of interviews(x) where any of the codes under the category was identified, divided by the total number of interviews(y). It is important to note here that, in this context, the frequency of occurrence is not used to analyze the relative criticality or the relative importance of the corresponding issues. Rather, this aspect is studied to analyze the level of awareness of the test issues among the interview participants and the magnitude of presence of the identified issues in the case

company. A graphical representation of the frequency of occurrence of each one of the eight primary open code categories is presented in Figure 4.6. The data pertaining to the graph is presented in Table C.1 in Appendix C.

4.6 Axial Coding - Establishing Relationships between Categories

On identifying and establishing a final set of open codes and open code categories that represent the test issues and test issue categories respectively, the next data analysis step was undertaken using axial coding. The objectives of this data analysis step were to identify

a) the relationships that exist among the identified test issues and
b) possible higher abstraction issue categories that can assist in representing the underlying theory at a more generic level.

The process followed to perform axial coding to fulfill the aforementioned objectives and an instance of how the process was implemented is as discussed in Chapter 3.

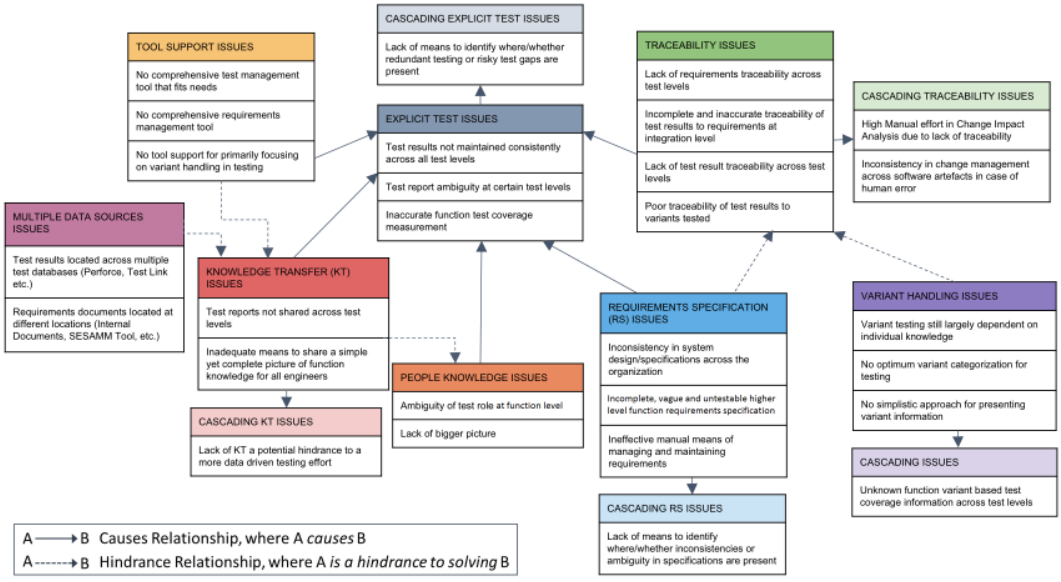


Figure 4.7: Relationships among open code categories

A visual illustration of the relationships that were identified among the open code categories to fulfill Objective a) is presented in Figure 4.7. The two primary relationships that were identified among the open code categories, as defined and discussed in Chapter 3, are the *'causes'* and the *'hindrance'* relationship. Each

of the two kinds of relationships among the open code categories are represented uniquely to explicitly illustrate the difference. For the *causes* relationship, the direction of the arrow is used to point from the causing issue to the effected issue. Similarly, for the *hindrance* relationship, the direction of the arrow is used to point from the issues that cause a hindrance to the issues that are effected by it. The relationships between open code categories which take the form *A causes/is a hindrance to B and B causes/is a hindrance to C so A causes/is a hindrance to C* are not explicitly visualized but are implied.

It is important to note here that, besides the relationships that exist among the primary open code categories, the relationships between the primary and secondary open code categories have also been explored, established and suitably represented in Figure 4.7. As the name suggests, cascading issues which are represented by the secondary open code categories are a consequence of the corresponding primary issues within the primary code categories. Hence, there is a *'causes'* relationship that has been identified to exist between them. For instance, the Knowledge Transfer (KT) issues (refer to Figure 4.7) *'cause'* the cascading KT issue of possible hindrance to a more data-driven testing approach. Similar is the case with the other primary open code categories that cause corresponding cascading issues.

The next step of axial coding was to identify higher abstraction axial code categories to fulfill Objective b). On iteratively analyzing the relationships between the open code categories, a final set of four axial code categories have been identified. The four axial code categories were identified to accurately represent all the open code categories at a higher level of abstraction. These include: *People Issues*, *Process Issues*, *Technology Issues* and *Explicit Test Issues*. Figure 4.8 provides a visual depiction of the axial code categories and the corresponding open code categories that they encompass. Each axial code category can be defined as follows:

People Issues: People issues category pertains to all those issues that are fundamentally related to the human resources involved directly or indirectly with the testing activities and the test approach at different test levels considered. For instance, issues like 'Lack of bigger picture' and 'Test reports not shared across test levels ' are predominantly pertaining to people aspect of the test approach and hence included within this category.

Process Issues: Process issues category is used to refer to any issue pertaining to processes surrounding and undermining the effectiveness of the current test approach. These include issues with requirements specification, variant handling and traceability-based alignment of requirements and test approach.

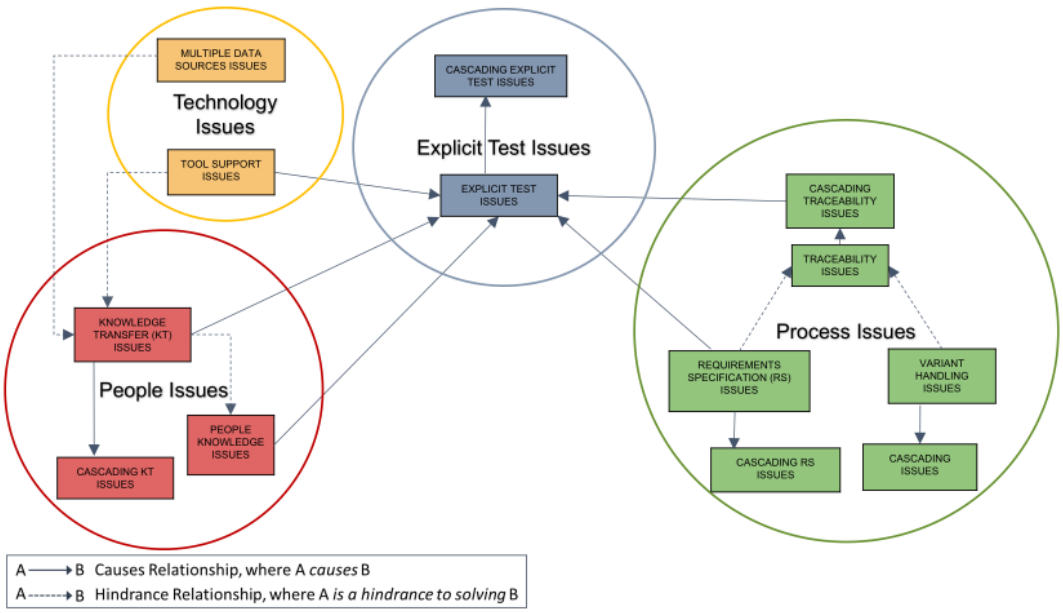


Figure 4.8: Axial code categories

Technology Issues: Technology issues category is used to refer to issues that are related to a lack of adequate tool support for either the test approach itself or the surrounding processes that influence the test approach. For instance, it includes issues like lack of test management tool and lack of requirements management tool.

Explicit Test Issues: Explicit test issues category is used to refer to the set of fundamental test approach issues that are identified to be a consequence of all the issues presented within the previous three categories. For instance, this category includes issues such as 'Inaccurate function test coverage measurement' and 'Lack of means to identify test redundancies and test gaps'.

4.7 Selective Coding - Identifying the Core Category

The last step in the data analysis coding process was to perform selective coding. Here, the four derived higher abstraction axial code categories were further analyzed to study the relationship between them. The objective was to derive the final underlying theory based on deducing the core category that logically relates the other categories. An illustration of the identified relationships among the axial code categories and the final core category is presented in Figure 4.9.

It was identified that there is a very intricate relationship that can be represented through a multi-way dependency among the set of people, process and

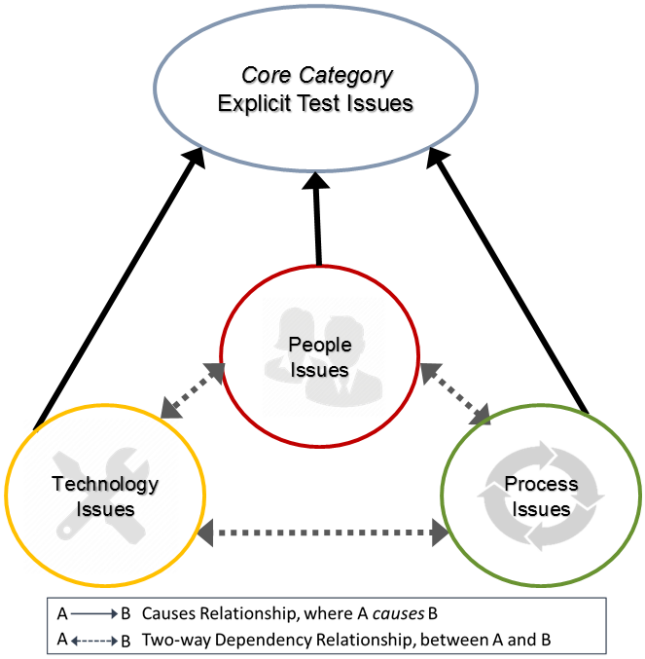


Figure 4.9: Identified core category based on relationships among axial code categories

technology issues. These set of issues together led to the explicit test issues. For instance, let us consider the complex relationship between one set of people, process and technology issues which is presented in Figure 4.10. Hence, it was inferred from the results obtained on performing several steps of data analysis, that an intricate set of people, process and technology issues formed the source for fundamental test issues at the case company.

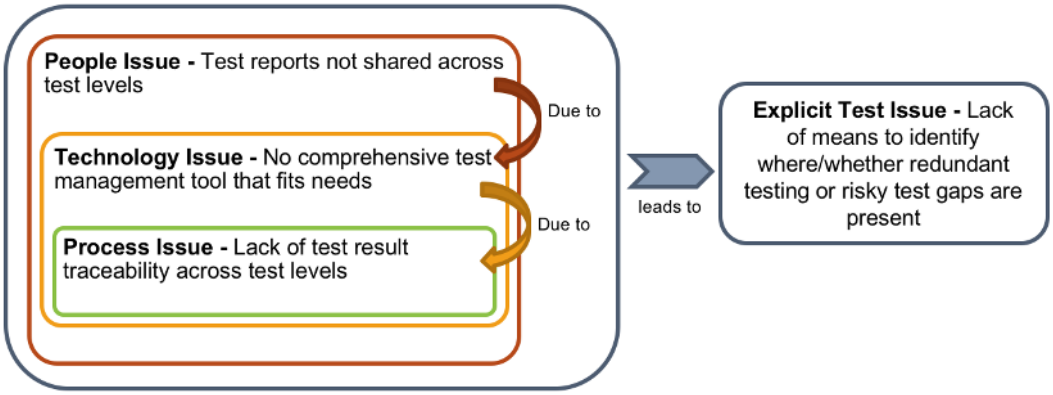


Figure 4.10: An instance of people, process and technology issue relationship

4.8 Triangulation - Confirming the Derived Theory

Data Triangulation: Archived documents and databases have been used as the alternative sources of data collection in order to perform data triangulation [34] and confirm the theory that was generated through the coding process. These alternative data sources were identified through interviews and interactions with case practitioners and included several requirements documents, test reports, test result databases, architectural tools and so on.

For all applicable test issues and issue categories represented in the form of open codes and open code categories, corresponding alternative evidence for their validity from the alternative data sources were identified. For instance, one of the open code category that was generated and inferred to being a part of the issues with the current test approach was 'requirements specification issues'. One of identified issue within this open code category was - 'Incomplete, vague and untestable high-level function requirements specification'. To confirm this interview data analysis result, the function requirements specifications from the internal company database were analyzed to identify if there is an actual generic case of function requirements specifications being incomplete and vague across several functions. Initially, of the 300-odd functions, a sample of 10 functions that differed in terms of their complexity and their ownership (the internal organization departments that handled these functions) were identified and established to be an adequate sample with the help of the industry and university supervisors. Analysis of these functions helped capture the required quantitative data that 9/10 (90%) functions had incomplete specifications in their scenario form. Moreover, it was observed that the textual specification document for the functions was written within structured templates in vague natural language that lead to several untestable requirements. The above observations indicated that there was indeed an issue of incomplete, vague and untestable high-level function requirements specifications for several functions with different complexities. This helped confirm the previously obtained result and strengthen its validity.

User Group Triangulation: While, most of the open codes and open code categories could be confirmed using the data gathered from the archived documents and databases, there were some that could not be aptly triangulated using that approach. For instance, the 'people knowledge issue' of 'ambiguity of test role at function level' was more suited to be studied and analyzed through interaction with people than through analyzing documents and databases. Hence, the confirmation of its existence within the case company was done by conducting interviews with multiple test engineers from the function test level to check whether the ambiguity of test role knowledge was specific to one interview subject

or was a common phenomena among the test engineers of the function test level. Hence, in this way user group triangulation [56] was performed for the suitable open codes and open code categories.

Evaluator Triangulation: In addition to the above approaches, evaluator or facilitator triangulation approach [56] was used to validate the conclusions drawn from the interview data analysis. This was implemented by presenting the data analysis results to industry practitioners who have been working in conditions where they most likely face these issues, and PhD researchers who are conducting ongoing research within similar areas. This ensured other independent researchers' and industry practitioners' confirmation was obtained for the theory that was generated based on data analysis of the interview data.

Chapter 5

Data Analysis

The results obtained on performing an initial detailed analysis of the data collected during the case study are as presented in the previous Chapter 4. These results help answer a subset of research questions RQ1, RQ2 and RQ3, while paving the way for research question RQ4 to be answered. Taking into consideration that RQ1 has been explicitly answered in the previous chapter, this chapter aims to further analyze the case study results to explicitly answer research questions RQ2 and RQ3 while setting the context for answering RQ4.

5.1 Issues with Current Approach to Test Distributed Software Functions at Scania

On analyzing the data from interviews and confirming the results with suitable triangulation approaches, it was inferred that, there exists no comprehensive test coverage information for the distributed software functions across the system, function and vehicle integration test levels at the case company. Moreover, it was identified that this was due to a intricate set of issues with the people, process and technology aspects (refer to Table 5.1) that are involved in testing the distributed software functions across the three test levels. This analysis thereby answers research question RQ2.

At the vehicle integration test level, it was identified that there is inaccurate measurement of test coverage for the distributed software functions. This was mainly due to reporting the test results in terms of the number of scenarios that are tested against the incomplete set of scenarios pertaining to that function. Here, the incompleteness of the scenario-based requirements specification of a function indicates that, the scenarios tested at this level today represent only a small set of several possible execution paths of the function across different vehicle variants and in different real-time execution situations. While on one hand the scenario-based requirements specifications are incomplete, on the other hand, the structured natural language requirements document for the functions are non-implementation specific and too vague to report test results against.

Hence, evidently, there is a clear gap and a need to explore suitable strategies for testing performed at the vehicle integration test level for the distributed software functions. This conclusion is on par with what has been reported in [1] that identified this gap based on a review of the research in test, verification and validation within the automotive domain.

Table 5.1 summarizes the issues pertaining to the approach implemented to test the distributed software functions at the case company, including the issues discussed above. It can be observed that the issues identified at the case company are similar to the issues found in the literature as presented in Section 2.3. It is important to note here that, Table 5.1 presents high-level issue categories and a brief summary of the issues that each category comprises, in order to provide a comprehensive view. A complete list of all the issues in the form of open codes is presented in Figure 4.5 of Chapter 4.

No	**Issue Category**	**Description of Issues**
1	People Issues	Ambiguity of test role at function test level and a lack of bigger picture of function test effort distribution across test levels
		Lack of appropriate means to share knowledge pertaining to test results across the test levels
2	Process Issues	Incomplete, vague and untestable higher level function requirements specification
		Variant-focused testing is highly dependent on each individual tester's knowledge with lack of means to handle function variants through simplistic representation
		Lack of adequate traceability across requirements and test results of the different test levels
3	Technology Issues	Lack of tool support to assist in test management, requirements management and change management
		Complexity of having test artifacts (test results, test reports, test cases, etc.) and function and system requirements spread across multiple locations with limited interoperability

4	Explicit Test Issues	No comprehensive function test coverage information across test levels
		Ambiguous and inaccurate test reports generated at vehicle integration test level
		Lack of test report generation at the function test level
		Lack of appropriate means to identify where or whether redundant testing and risky test gaps are present across the test levels

Table 5.1: Issues identified with the current approach to test distributed software functions at Scania

5.2 Alternative Approaches to Address Test Issues

On identifying the current test approach issues, the next objective was to propose an effective cross-functional verification strategy that would help in addressing these issues. The strategy aims to help provide a means to identify and improve test coverage for distributed software functions at the vehicle integration level, while reducing test redundancies and test gaps across the test levels. In other words, the strategy aims to resolve the explicit test issues by addressing their cause which includes the identified people, process and technology issues.

Below in Sections 5.2.1 and 5.2.2 the possible alternative approaches to address the test issues through an in detail study of the people, process and technology issues are discussed. Following this, in Section 5.2.3, the identified alternative approaches are presented.

5.2.1 Understanding People, Process and Technology

The identified issue categories leading to explicit test issues, namely, the People, Process and Technology issue categories, each in itself represent a complex area of study. In the current case, the existence of issues with high interdependencies within all three challenging areas leads to an extremely complex phenomena. Given the time constraint for the thesis research, providing a comprehensive solution with multiple strategies that aim to resolve all the issues in each complex area, was deemed to be infeasible. Hence, the first step was to identify an appropriate subset of the issues that can be suitably addressed through the proposed verification strategy. It is important to note here that, since the three areas of people, process and technology are highly interdependent as also

recognized by other researchers like in [61], [62] and [63], no individual solution to the subset of issues exists in isolation. Therefore, though the verification strategy proposed in this research aims to provide an *effective* solution for one subset of issues, it would likely reach its full potential in terms of its *efficiency* when it is complemented with solutions that address the remaining subset of issues. Here, the terms 'effectiveness' and 'efficiency' are often used in close association with each other while assessing the results of any approach, process or strategy. *Effectiveness* is best defined as doing things right. It is a study of the value of the results obtained. On the other hand, *Efficiency* is concerned with doing right things. It relates to maximizing the results using optimum resources, that is, least time, effort and cost, once its effectiveness is established [64].

Table 5.2 provides a tabular representation of the identified alternative issue subsets. It includes a brief analysis of the expected value of basing the proposed verification strategy on the alternative approaches that aim to address and resolve each one of the identified issue subsets. This comparative analysis was conducted based on capturing scientific knowledge from literature studying the relationships among people, process and technology. This literature was identified based on performing a literature review according to the guidelines presented in [37]. The results of the analysis were then used as a basis to select one of the issue subsets to be the focus of the verification strategy proposed in this research.

Based on the analysis of alternative issue subsets, it has been concluded that alternative (3), that is, an approach with a focus on addressing the process issues, is most suitable and viable in the current context. This is because it has been identified in literature (refer to Table 5.2) and observed in the case study results (refer to Figure 4.10), that process issues lie at the core of the three identified issue categories. Technology is adopted to fit the process established, and people knowledge and effort is built around the process. Hence, addressing the process issues will help pave the way to studying and addressing the people and technology issues that persist around the established process. Hence, the verification strategy proposed in this research aims to address the process issues in order to resolve the explicit test issues identified in the current approach for testing distributed software functions. It is important to note here that, the proposed strategy will be most efficient when followed by solutions and well established approaches that provide

a) a means of adopting adequate technology support for the proposed effective process, and
b) an overall enhancement of people knowledge around the new process and technology.

No	Issues Subset			Description
	People	**Process**	**Technology**	
1	✓	✓	✓	Efficient people, effective processes and supportive technology have been identified to be the critical success factors for organizations in several disciplines including knowledge management [65] and several aspects of software development including verification [62]. In the context of the automotive industry, improving and integrating these three areas has been reported in [63] to have led to tremendous success for Toyota, an important player within this industry. Hence, it can be inferred that a verification strategy that aims to address the issues pertaining to people, process and technology aspects of testing is relatively the most effective solution [62] as compared to solutions that aim to address issues within any one of the areas in isolation.
2	✓			In an organization, people implement the processes and use the technology to do so in an enhanced manner [61]. There is an increasing focus within organizations to identify people issues and thereby derive suitable solutions to resolve these issues. This is due to recognition that people are a critical success factor for efficient process execution and process improvement [66] and also for the adoption of technology [67]. However, while people issues are very critical and important to resolve, the overall efficiency of people effort might be hindered in case of issues with the process that has been laid down for them to implement or the technology that has been provided for them to use.

3		✓		A process is what has been laid down to ensure the objectives and goals of any task are met. A focus on processes can improve the effectiveness and efficiency of the task under consideration [61]. Technology should be adapted to fit the processes and people [63]. Moreover, people competencies and knowledge should be refined to adapt to the processes and technology [65], in a continuous learning and implementation environment [63]. Hence, approaches that deal with addressing and improving the process lie at the core of people and technology enhancement approaches [62].
4			✓	Technology is used to save time and cost in the act of people executing processes [61]. Hence, for the successful adoption of technology in order to get most value from its investment, a focus on resolving people and process issues is essential [67].

Table 5.2: Alternative people, process and technology issue subsets

5.2.2 Understanding the Fundamental Process Issue Areas

In this section, with an aim to address and resolve the identified process issues, an understanding of the fundamental areas where the process issues lie in the general context and specific context of the automotive industry is presented. These process issue areas include requirements specification, traceability and variant handling as depicted in Figure 4.8. It is interesting to note here that, each of the process issue areas have been identified to be challenges in the related literature pertaining to automotive software testing approach as presented in Chapter 2.

Requirements Specification

Requirements Engineering as defined by IEEE standard [68] is a process that contains all the activities to establish and maintain the requirements that are to be met by the software or the system. Such definitions of requirements engineering in the context of not only software engineering but a more generic definition in the context of systems engineering have been given by several authors like in [69]

and [70]. Laplante in his book [70] defines requirements engineering as a branch of engineering that deals with the goals and constraints of a system and that is concerned with the relationships of these factors to requirements specification and their evolution over time and across families of systems. Hence, requirements engineering as a discipline contains several activities like requirements elicitation, requirements analysis, requirements specification, requirements verification and validation and requirements management.

Requirements specification for a software or a system is one activity within requirements engineering which as defined by the IEEE standard is a "structured collection of the requirements (functions, performance, design constraints, and attributes) of the software/system and its operational environments and external interfaces" [68]. There are several methods for requirements specification that can be most suitably classified into informal, semi-formal and formal approaches [71]. The informal methods are based on using common natural language. Such informal methods have been found to contribute to 79% of specifications in software engineering [72] and also most commonly used in systems engineering [73]. The semi-formal methods are based on using either structured natural language through forms and templates [72] or use-case based requirements engineering [59]. Lastly, the formal methods are based on using formalized language in the form of mathematical notations and formal logic [74]. Many specification languages and methods for informal, semi-formal and formal approaches have been developed to support the requirements specification process, each with its own advantages and disadvantages [74].

As identified by the Standish Group [75], one of the major factors for failure of projects or products developed through these projects is not technical issues but having incomplete requirements specification. On the other hand, one of the major success factors for projects or products developed through these projects is to have clear requirements specification [69]. Moreover, such requirements specification issues, like ambiguity and incompleteness, directly impact testing that is done against these requirements. This makes the disciplines of requirements specification and testing highly entwined. Therefore, having an effective process for requirements specification has a positive influence on the testing process [76]. Hence, evidently, having incomplete, vague and ambiguous requirements specification for the distributed software functions in the current context is a critical issue to be solved from the test perspective.

Even within the automotive industry, enhancing competency and efficiency in requirements specification and thereby in testing has been identified to be one of the fundamental technological core competencies required for handling automotive software [7]. However, while such issues with requirements specification for the distributed software functions have been found to be a challenge, solutions

derived through research to solve these issues are sparse [23][29].

Hence, to identify a feasible and effective approach to resolve the current requirement specification issues, a comparative analysis of the possible requirements specification methods in the automotive context from the testing perspective is required.

Variant Handling

Software and System Product Line Engineering (PLE) is a discipline that has emerged based on developing not just one software or one system, but a family or portfolio of softwares and systems with variations in their functions and features [77]. In both software and systems engineering, PLE approach has been given a lot of attention in the past decade. Adoption of PLE as a means of offering highly customizable end products has been found to be a promising approach for improved productivity, enhanced quality, reduced costs and competitive advantage [78][79]. However, it adds a new dimension of variant handling complexity to all aspects of software and system development [80][81].

Handling variants efficiently, which is termed variability management, has been found to be a key activity for successful PLE. Many approaches have been developed for variability management [82]. One most widely used variant management approach is through feature models that aim to model variant information in a way that assists in simplifying variant handling through different stages of development [83]. These feature models either represent the structural perspective, the functional perspective or both perspectives of the product [80]. For instance, a structural feature model for automotive vehicles might deal with modelling information pertaining to all the hardware and mechanical components variants of the vehicle on which the functions execute. Whereas, a functional feature model for automotive vehicles might deal with modelling all the variations of the functions that are present in the vehicles.

One popular approach to generating these feature models is through requirements specifications and product description documents [80]. While this process and how to automate this process have been studied and reported in literature [84][85][86], what has not been adequately focused on is how this process can be enhanced through incorporating adequate variant information within the requirements specification in a suitable manner.

In the context of the automotive industry, while several automotive companies have adopted PLE and gained the advantages it offers, they have also been found to be afflicted with the issue of variant handling across their software and system development [87][88]. One such aspect of automotive software and system development that is challenged by variant handling complexity is testing [1][2].

In general testing in PLE has been found to be predominantly concerned with the functional feature models [80]. However, in the current context of testing automotive distributed software functions across the test levels, adequate variant information for the functions is required to be modelled from both the functional and structural perspective.

So it can be inferred that, in order to ensure effective testing of the distributed functions in a PLE context of the automotive industry, it is important to capture both functional and structural variant information of the functions within their requirements specification. This ensures that the test effort is driven based on variant knowledge which is very crucial in PLE. Having such variant information either missing or implicit within the requirements effects both feature modelling and subsequently testing [83].

The variant information can be incorporated in the natural language requirements [84], in semi-formal requirements models [89] or in formal requirements [90]. Hence, to identify a feasible and effective method to capture variant information and resolve the current variant handling issues, a comparative analysis of the possible approaches to incorporate variant information within the function requirements specification in the automotive context from the testing perspective is required.

Traceability

Requirements and Test Traceability provides a means of aligning requirements and testing during software and system development [91][92]. Establishing a link through traceability between these two disciplines has been found to improve the overall outcome of the software and system development process [91]. A forward traceability from requirements to tests helps to determine what requirements are covered by which test. Similarly, backward traceability from tests to requirements helps to find root cause of failed tests and to improve change management. Traceability also helps to identify degree of test coverage of requirements and enables adopting a suitable and accurate metric for test coverage measurement [91]. Having such accurate means of measuring test coverage is crucial for industries [92] and is a test issue we aim to resolve. Hence, focus on traceability from the testing perspective is essential.

Though researchers and industry practitioners identify the need for linking requirement and testing, most often little effort is put into realizing this need, leaving a gap between the two areas [91][93]. Consequently, traceability is one of the challenges in aligning requirements and testing processes [76].

Such is the case even within the context of the automotive industry, where alignment of requirements and testing through traceability has been found to be

an challenge in [1]. Moreover, for large and complex systems like in the case of automotive systems, requirements usually exist at multiple levels of abstraction. Hence, a more complex and integrated traceability solution with *hierarchical* and *forward-backward* traceability is required. Here, hierarchical traceability is established among the requirements and forward-backward traceability is established between the requirements and the test artifacts [69][92].

Effective traceability rarely happens by chance or through ad hoc efforts. To establish effective traceability it is important to explicitly address it in the requirements and testing processes [93] and follow a systematic approach for its implementation [94] . Incorporation of traceability has been studied in the context of several requirements specification and testing approaches. Majority of these studies have been found to be focusing on establishing traceability in formal model-based requirements specification and testing [91]. Other studies have addressed establishing traceability through semi-formal requirements specification and testing based on use cases and scenarios [60][95].

Hence, to identify a feasible and effective means of establishing traceability between requirements specification and test artifacts to resolve the current traceability issues, an analysis of the possible approaches to establish traceability in the automotive context from the testing perspective is required.

5.2.3 Deriving Alternative Process Enhancement Approaches

Based on the study of the process issue areas, the identified alternative process enhancement approaches are presented in Table 5.3.

No	Process Enhancement Approaches
(1)	Enhancing distributed software function requirements specification to obtain a complete, unambiguous and testable set of requirements by setting standardized guidelines, templates and completeness criteria to the existing natural language requirements. Enhancing variant knowledge by adding required functional and structural variant information pertaining to the function to its requirements specification using natural language. Establishing traceability between the natural language requirements across test levels and test artefacts. Obtaining comprehensive test coverage information across the test levels based on coverage of natural language requirements of the function.

(2)	Enhancing distributed software function requirements specification to obtain a complete, unambiguous and testable set of requirements by defining a semi-formal model based on use cases and scenarios to represent the function requirements. Enhancing variant knowledge by adding required functional and structural variant information pertaining to the function to its scenario-based model. Establishing traceability between the semi-formal scenario model across test levels and test artefacts. Obtaining comprehensive test coverage information across the test levels based on coverage of the semi-formal scenarios that represent the function.
(3)	Enhancing distributed software function requirements to obtain a complete, unambiguous and testable set of requirements by defining formal mathematical models to represent the function requirements. Enhancing variant knowledge by adding required functional and structural variant information pertaining to the function to its formal model. Establish traceability between the formal model across test levels and test artefacts. Obtaining comprehensive test coverage information across the test levels based on coverage of the formal model that represents the function.

Table 5.3: Alternative process enhancement approaches

5.3 Comparative Analysis of the Alternative Process Enhancement Approaches

A comparative analysis of the three identified alternative process enhancement approaches was performed to assess the relative effectiveness and feasibility of each alternative. This analysis was initially done based on scientific literature knowledge pertaining to the pros and cons of each of the alternatives in the general context of software engineering, the specific context of embedded system software engineering and targeted context of automotive embedded system software engineering. The scientific knowledge was captured through a literature review conducted according to the guidelines in [37]. In addition, to further strengthen

the basis to present one of the alternatives as the relatively most effective and feasible approach to test automotive distributed software functions, industry expert opinions based on their experience were captured. Here, their estimations of the cost and value of implementing each of the solutions in the automotive industry was collected through interviews. The results of the comparative analysis based on scientific literature are presented in Section 5.3.1, followed by the results based on industry expert opinion in Section 5.3.2. The final comprehensive conclusions drawn from the comparative analysis are presented in Section 5.3.3.

5.3.1 Comparative Analysis based on Scientific Literature

The three process enhancement approaches identified (refer to Table 5.3) suggest taking three different approaches to enhancing requirement specification of the distributed software functions. Enhancing requirements specification in all three of the approaches is based on addressing completeness, testability and ambiguity aspects of the requirements and variant information handling capabilities. This is considered the first step towards improved testing of the distributed software functions across the test levels in the automotive case company.

Approach (1) involves enhancing the current natural language requirements specification for the distributed software functions with structured guidelines and templates to tackle incomplete and ambiguity. Across several industries, either unstructured natural language or structured natural language using templates and guidelines is most widely used requirements specification technique. This is due to its ease of adoption and lack of need of any special skills which implies low cost of implementation [73]. Attempts have been made to address and enhance completeness and unambiguity of natural language textual requirements by providing guidelines and metrics for measuring and improving completeness [71][73] and by extending adequate tool support [72]. Yet, such natural language textual requirements, irrespective of the level of abstraction of the system or software they define, have been found in ample of cases to still be prone to misinterpretation, ambiguity, inconsistencies and inaccuracy [74][96]. Moreover, from the test perspective, though it is a highly recommended practice in almost all the natural language requirements specification guidelines to ensure the requirements are testable, it is rarely ever religiously implemented in practice giving rise to several non-testable requirements [96]. In addition, it has been found through experience and reported in literature that textual natural language requirements are *"only part of the game"* [23]. While, natural language requirements are very important to capture and present certain requirements at any level of the V-model of automotive software development and testing, they are not sufficient. The industry identifies the need for the textual requirements at each level to be supported with adequate ways to capture and present various other complex attributes of the industry like variant information, distributed functionality with several scenar-

ios of execution in real time, etc. These are hard to capture effectively through natural language textual requirements [23]. Hence, generating natural language textual requirements has been found to be only a part of handling requirements of the automotive system software through different phases of development like implementation and testing. The requirements specification should be complimented with suitable system modelling that can help capture the other complex information in an implementable, testable, maintainable and traceable manner [69].

Approach (2) involves enhancing the requirements specification for distributed software functions by implementing semi-formal use case and scenario-based modelling as a means to capture complete and unambiguous set of requirements in the form of function scenarios. One of the major advantages stated in literature of adopting use cases and scenarios for requirements specification is its support for functional testing and test case generation. Moreover, it has been found in [96] to be able to adequately capture requirements for product line variant handling. In addition, it provides an easier means of linking requirements specification to other artifacts like test artifacts [97]. Hence, it is of no surprise that such UML based modelling for requirements has been found to be the most widely used based on a survey conducted by [72] and is recommended by several software development approaches like Unified Software Development Process for requirements specification [59]. Yet, generating scenarios for testing from the use case based requirements specification is still labour intensive with lack of adequate tool support [98]. This makes it comparatively less cost effective than approach (1). UML itself is a modelling language for ensuring suitable models of systems and softwares are developed without ambiguity and inconsistency. Yet, in addition, there is supporting literature providing guidelines for writing use cases and generating the corresponding scenarios to avoid ambiguities and errors and ensure high quality of implementation of this approach in a way that will help capture all of its advantages [59][97][98].

Approach (3) involves adopting a formal approach to specifying requirements of the distributed software functions based on mathematical models using mathematical entities. Formal methods have their obvious advantages in discovering and appropriately addressing incompleteness and ambiguity in requirements specification [99]. Moreover, it is well established in literature that formal methods can be used for effectively verifying requirements and eventually cut down cost of testing through automatic test case generation from the formal models [100] with adequate tool support [101]. Yet adoption of formal methods comes with a requirement of high mathematical skills for their profitable use. It is also perceived to be cumbersome and difficult due to its complexity in dealing with requirements in mathematical and logical notations [71][74]. The above factors indicate that adoption of solution (3) in the current context might require exorbitant initial

investment with a slow rate of value addition to the industry. But on the other hand, there are researchers like Hall [102] who have tried to argue that high cost, need for extraordinary mathematical skills and difficulty to adopt formal methods for requirements specification are myths. The author argues that while the initial cost of implementation is high, formal methods provide a means to cut down the cost of development over time and enhance the effectiveness of the testing process. That aside, for systems and functions with changing requirements and product families with variant handling complexities, like in the safety critical automotive and aircraft industries, formal specifications pose maintainability and traceability complexities along with the other stated factors that sometimes tend to outweigh the advantages it offers in other fronts [103].

On addressing requirements specification and variant information handling, the three alternative process enhancement approaches proceed to address means of handling traceability between the higher level distributed software function requirements and lower level system requirements, and across the test results obtained from the system, function and vehicle integration test levels. This established traceability is then suggested to be used as a means to obtain comprehensive test coverage information for the distributed software functions.

According to a systematic mapping study [91] conducted to identify useful approaches for alignment of requirements specification and testing, it has been found that, majority of the research focus for establishing traceability between these two disciplines is on semi-formal and formal approaches to Model Based Testing (MBT). As the name suggest MBT is based on using models of the software or system under test in order to drive the testing effort. In semi-formal MBT, requirements which are modelled in the form of behavioural models (use cases, scenarios in the form of Message Sequence Charts etc) are aligned with testing by using these models to automatically or semi-automatically generate test cases [91]. Hence, traceability can be established between requirements specification and testing by mapping the behavioural models to the relevant test artifacts. On the other hand, formal MBT focuses on aligning testing with requirements, specified in the form of formal models using formal languages, by generating test cases from these formal models and subsequently establish traces between them [104].

While formal methods have been found to be one of the best options for model based requirements specification, testing and aligning these two disciplines for safety critical systems like the systems within automotive industry [91], it has also been found to be difficult for practitioners to implement. It is expected to require high experience in representing requirements in mathematical and logical notations [71][74]. This makes it is a highly cost-intensive solution. Moreover, formal methods have limited scalability as one of their challenges when dealing

with large and complex systems [91].

On the other hand, focusing on semi-formal methods through use cases and scenarios as a means to model and test the functional behaviour of a software or a system has been found to have their own challenges like the challenge of generating executable test cases for enhancing efficiency of testing [105]. But exploring semi-formal MBT methods and addressing the challenges within it has been found to generate solutions that are more cost effective and scalable as compared to the formal MBT methods. This makes it an interesting area of research with high focus from the industry for whom cost effectiveness and feasibility of solutions are critical [91].

Based on the knowledge acquired from the relevant scientific literature, the comparative assessment of the alternative approaches can be summarized as presented in Table 5.4. Here, the comparative assessment is based on the overall *Cost* in terms of time and effort, certainty of *Return on Investment(ROI)* and *value of the ROI* on adopting and implementing each of the alternative approaches. For this assessment a relative ordinal scale of Highest(H), Medium(M) and Least(L) was used.

Approach	**Criteria for Assessment**		
	Cost	**Certainty of ROI**	**Value of ROI**
(1)	L	M	L
(2)	M	H	M
(3)	H	L	H

Table 5.4: Results of comparative analysis of the alternative process enhancement approaches based on scientific literature

5.3.2 Comparative Analysis based on Industry Expert Opinion

Industry expert opinion on the relative feasibility and effectiveness from the testing perspective of the three alternative process enhancement approaches was captured through interviews of practitioners from within the case company. This was to strengthen the basis for stating one of the alternative approaches as being relatively most effective and feasible. During the interviews, the researcher presented the three alternative approaches at an adequate level of detail to ensure the interviewees captured the essence of each alternative without any ambiguity or misinterpretation of core concepts involved in each. While conducting the interviews, the researcher had been working simultaneously on capturing the pros and cons of each alternative based on scientific literature. Yet, these details were not presented to the interviewees to ensure that their opinions were not influenced

by any other factors and were purely based on their own individual experience.

Here, the practitioners selected as interview subjects are termed 'industry experts' since they have been carefully chosen with the assistance of the industry supervisor to ensure practitioners with vast experience and high knowledge in the relevant areas were selected to share their opinions. A total of six industry experts were identified to be most suitable and interviewed. Their opinion on the relative feasibility and effectiveness of adopting the alternative approaches was captured through their assessment of each approach based on the the following three criteria: *Cost* of implementation in terms of people effort and time, *certainty of ROI* and *Value of the ROI*. The interviewees were requested to provide their comparative assessment of the alternative approaches for each criteria using a relative ordinal scale of Highest(H), Medium(M) and Least(L).

Table 5.5, Table 5.6 and Table 5.7 present the relative assessments provided by the industry experts for alternative approach (1), (2) and (3) respectively.

Alternative Approach (1)			
Assessment Criteria/ Industry Expert	Cost	Certainty of ROI	Value of ROI
Expert 1	L	H	M
Expert 2	L	H	L
Expert 3	L	M	L
Expert 4	L	H	L
Expert 5	L	L	L
Expert 6	L	H	L

Table 5.5: Results of comparative analysis of process enhancement approach (1) based on industry experts

Alternative Approach (2)			
Assessment Criteria/ Industry Expert	Cost	Certainty of ROI	Value of ROI
Expert 1	M	M	H
Expert 2	M	M	H
Expert 3	M	H	M
Expert 4	M	M	M
Expert 5	M	H	H
Expert 6	M	M	H

Table 5.6: Results of comparative analysis of process enhancement approach (2) based on industry experts

Alternative Approach (3)			
Assessment Criteria/ Industry Expert	Cost	Certainty of ROI	Value of ROI
Expert 1	H	L	L
Expert 2	H	L	M
Expert 3	H	L	H
Expert 4	H	L	H
Expert 5	H	M	M
Expert 6	H	L	M

Table 5.7: Results of comparative analysis of process enhancement approach (3) based on industry experts

Evidently, based on the data presented in Table 5.5, majority of the industry experts who have been interviewed believe that adopting approach (1) might cost the least and also have the highest certainty of ROI relative to the other two alternatives. But what is also clear is that majority of them believe the ROI on such an investment will add least value to the industry in terms of solving the test issues at hand.

Based on the industry expert opinion of adopting alternative (2) presented in Table 5.6, it is evident that all industry experts interviewed believe this approach would cost somewhere between what would cost the company to adopt approach (1) and approach (3). While there is no such clear common ground for their assessment of certainty and value of ROI, majority of them believe that the industry can be somewhat certain (medium to high) of the ROI. Moreover, the value of the ROI is expected to be relatively high as compared to alternative (1) and (3) based on their experience.

Finally, analyzing the industry expert opinion of adopting alternative (3) presented in Table 5.7, it can be observed that all of the industry experts interviewed believe this approach would cost the highest of the three alternatives. Moreover, majority of them also believe that there is relatively least certainty of ROI on adopting this approach. Their opinion of the value addition on successfully adopting this approach has no clear indications, which might be pointing towards a trend that within the industry there are contradicting views on the value addition of adopting this approach.

5.3.3 Comprehensive Results of Comparative Analysis

Table 5.8 presents the comprehensive tabulated view of the results from the comparative analysis of the three alternative process enhancement approaches based on scientific literature and industry expert opinion.

Approach	**Estimation based on Scientific Literature**			**Estimation based on Industry Expert Opinion**		
	Cost	**Certainty of ROI**	**Value of ROI**	**Cost**	**Certainty of ROI**	**Value of ROI**
(1)	L	M	L	L	H	L
(2)	M	H	M	M	M	H
(3)	H	L	H	H	L	M

Table 5.8: Comprehensive results of comparative analysis of alternative process enhancement approaches

Based on the captured scientific literature knowledge and opinions of the industry experts interviewed, it can be inferred that the process enhancement approach (2) is expected to be relatively most feasible and effective for implementation and adoption in order to suitably address and enhance the testing of distributed software functions.

Chapter 6

An Effective Cross-Functional Verification Strategy

6.1 Proposed Cross-Functional Verification Strategy

This section presents the proposed cross-functional verification strategy for testing the distributed automotive embedded software functions. The strategy proposes enhancing the requirements specification of the distributed software functions by implementing a semi-formal use case and scenario-based modelling that provides a means to capture complete and testable requirements with adequate function variant information. There after, the strategy proposes adopting a multi-level reuse concept of combining test results from the system, function and vehicle integration test levels by establishing appropriate traceability across the requirements and among the test results. The obtained comprehensive test coverage information pertaining to the distributed software functions can then used to identify test gaps and test redundancies to enhance testing at the vehicle integration test level. In Section 6.1.1, a summary of the previous work related to the proposed strategy is presented. This is followed by the steps for implementation of the proposed verification strategy in Section 6.1.2.

6.1.1 Summary of Previous Work

Modelling requirements for verification is not a new concept. It has been adopted for various verification activities, most popularly for testing and is termed MBT. MBT has been found to help significantly enhance the testing process [91]. Basing the testing effort on model-based requirements specification approaches has been found to identify more errors in requirements and implementation than those hand crafted from textual requirements [106]. A semi-formal approach, based on the use of scenarios for modelling requirements within MBT is being increasingly studied in literature [103][105][107][108]. A large part of the research that focuses on using such semi-formal model-based approach to aligning requirements specification and testing through traceability has been found to involve

proposing solutions but little research focus is on evaluating and validating these proposed solutions [91].

Tsai et al. in [108] explore scenario-based traceability in order to select test cases for impact analysis and regression testing of changes. The authors propose the use of scenarios in regression testing of system software function changes for reduced test effort through improved impact analysis. Fundamentally it is similar to the work in the current research in terms of exploring scenarios for improved testing. It differs in aspects where the current research aims to capture scenarios and propose their suitability in reducing test gaps and test redundancies and improving test effort distribution based on establishing traceability across multiple test levels of the automotive industry V-model.

Thangarajan et al. in [95] explore scenarios in agent-oriented software engineering. They focus on adding additional structure to scenarios in order to facilitate traceability and testing through automatic test case generation, execution and analysis. While yet again, scenarios are proposed to be useful in the context of establishing the desired traceability, the authors focus more on the traceability between the scenarios and implementation code for improved testing. On the other hand, the current research focuses on traceability between scenarios and lower-level systems requirements and test results to obtain comprehensive test coverage information for improved test effort distribution.

Nebut et al. in [103] present work on deriving function tests from use cases in the form of test scenarios through test objectives. The major focus of the presented research, similar to [95] and [108], is on the automation of the process of using scenario-based requirements specification and testing. Where as, in the current research, the focus is on studying and validating the effectiveness of using scenarios for testing and traceability in the automotive industry.

Arnold et al. in [105] explore the concept of modelling requirements using scenarios and contracts for validation of requirements. Here, contracts are preconditions and post conditions to scenarios of the system execution. The authors further discuss their approach based on validation performed for their proposed model. Hence, the authors approach deals with exploring scenarios for validation of requirements and more so a formal approach to defining and implementing scenarios.

Within the context of embedded system software, specifically in the automotive industry, most previous related work in MBT and subsequent study on traceability and alignment of the disciplines of requirements specification and testing is based on formal methods [109][110][111]. While some studies focus on the concept of test scenarios as a means of managing the requirements of the

increasingly complex automotive embedded software, they take a formalized approach to the generation and use of test scenarios like in [5], [112] and [113].

Moreover, most of the testing focus through such formal modelling methods is concentrated at the system integration and lower test levels of the automotive V-model [5][32][114][115][116][117]. While some apply their approaches to distributed functions at the vehicle integration test level like in [109], focus on this area in general is limited. MBT at the vehicle integration test level and more specifically a semi-formal scenario-based MBT at this level of the automotive test process is unknown today but with great potential for improvement [32].

Hence, exploring the application of semi-formal modelling of requirements and variability information in order to generate test scenarios and subsequently tracing the obtained scenarios to lower-level system requirements and testing is an area that is explored little in the automotive industry, but with great potential to add value to the testing process. Such a multi-level reuse of test effort has been recognized for test effort reduction during test case generation in [118] and [119]. Within the current thesis research, this concept of multi-level reuse is dealt with a broader perspective of reducing overall test effort, by identifying where test redundancies and test gaps lie across multiple test levels. Such comprehensive test strategies to cope with the complex aspects of the automotive software testing process like [12], are limited for integration testing of the distributed software functions.

6.1.2 Steps for Implementation of the Proposed Cross- Functional Verification Strategy

Following is a detailed description of the steps for implementation of the proposed verification strategy based on alternative approach (2) presented in Table 5.3. A brief outline of these steps is presented in Figure 6.1.

STEP 1. Enhancing requirements specification
This is the first step in the implementation of the strategy which deals with adopting a use case and scenario-based modelling of the distributed software function requirements. This step is implemented as a means to obtain complete and testable set of requirements of the distributed software functions in the form of scenarios with sufficient function variant information.

STEP 1.1 Identification of function use cases
For the distributed software function under consideration, identify possible use cases. Here, as defined in Chapter 4, each use case of a function describes one of its unique behaviour. Use

cases can be considered to be part functions that describe its different behaviours that together make up the entire function and its comprehensive behaviour. For example, while one of the use cases for a function can be activation of a particular feature, the other use case can be deactivation of the feature. These together with other use cases, if present, make up the entire function. It is important to note here that, no use case models are to be generated. Rather the possible use cases are to be identified.

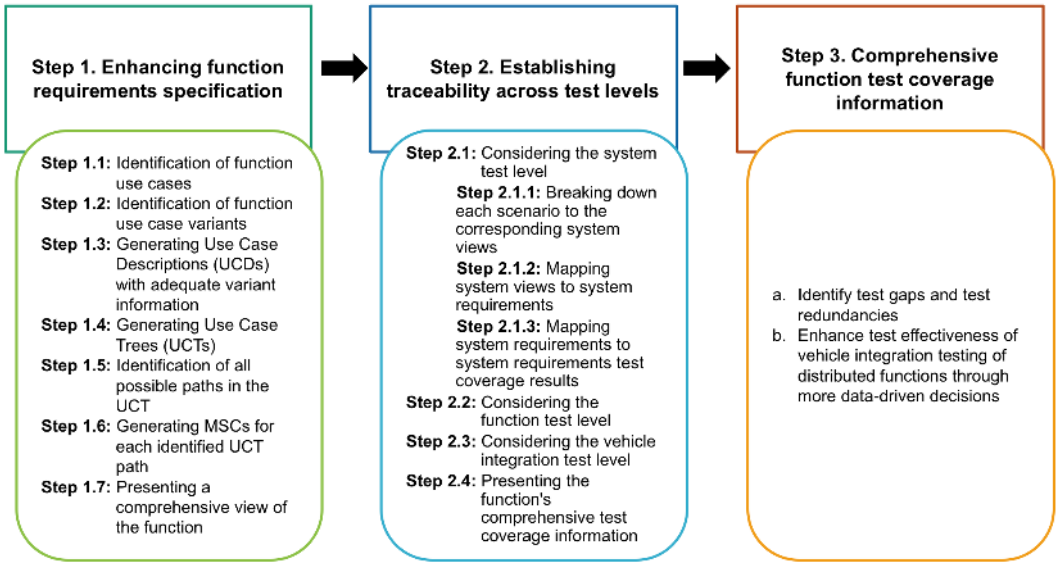


Figure 6.1: Steps for implementation of the proposed verification strategy

STEP 1.2 Identification of function use case variants

For each use case of the function, identify possible use case variants, that is, all possible contexts in which the function use case takes a different sequence of actions to execute the same behaviour described by the use case. For instance, if activation of a feature is one of the use case of a function, its variants can be identified to be all those contexts in which different steps are executed to reach the same outcome. For example, the activation of the feature maybe executed differently in vehicles with liquid engine and in vehicles with gas engine. In this case, the use case variants include vehicles with liquid engine and vehicles with gas engine.

STEP 1.3 Generating Use Case Descriptions (UCDs) with adequate variant information

The next step is to generate an UCD for each use case of the function. An UCD is a step-by-step description of the interaction between the participating systems required to execute the

use case in its operational environment. It is to be written in structured natural language. There are several different templates and guidelines that have been proposed in literature for generating UCDs. One such template is presented by Somé [59] which is the inspiration for the current template proposed. The template is tailored and enhanced to be fit for the automotive domain. An UML representation of an abstract UCD template is as shown in Figure 6.2. A more detailed description of the abstract UCD template with the content to be presented in each of its sections is presented in Figure 6.3.

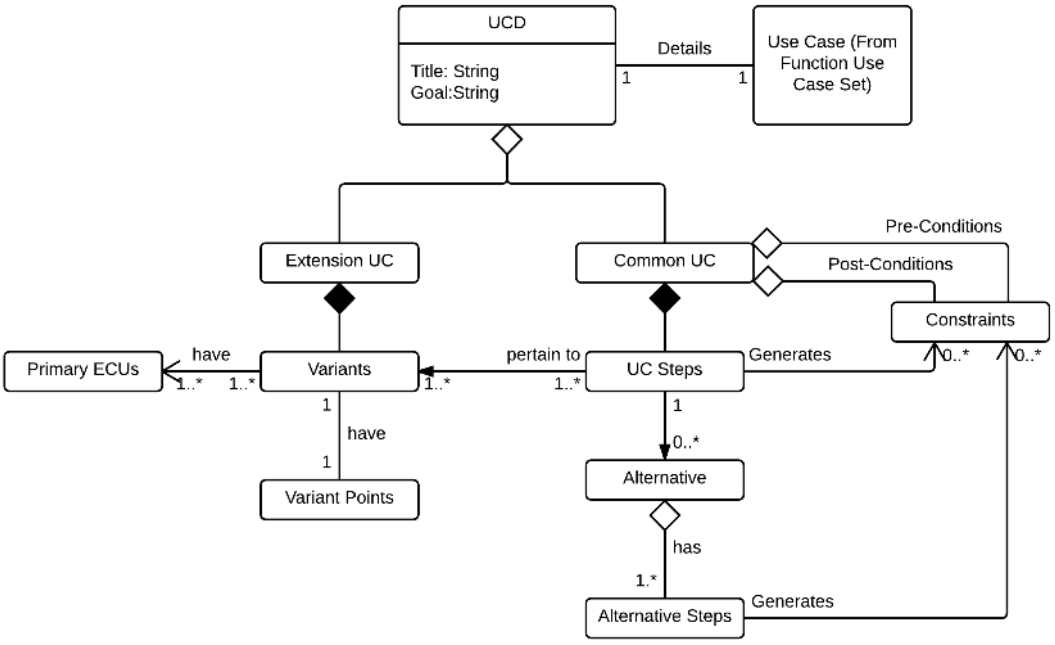


Figure 6.2: UML representation of abstract UCD template

Here, based on the use case variants identified in the previous step, the information pertaining to how the steps of execution of the use case vary in case of each such identified variant is to be incorporated into the UCD using *variant points*. These variant points are inserted at the beginning and ending of the steps of use case execution that are specific to each particular variant.

STEP 1.4 Generating Use Case Trees (UCTs)

In this step, an UCT is to be generated for each use case of the function. An UCT is a graphical tree-structure representation of all possible paths for the use case execution from when pre-conditions are met to reach the post-conditions. Each node of

$UC_{(Function\ number)\_(Use\ Case\ Number)}$ Use Case Description	
Title:	***In this section the title is used to indicate the use case name which relates to one of the overall behavior of the function.***
Variants:	*In this section, variants of the use case as identified in Step 1.2 are specified.*
Primary ECUs:	*In this section the primary ECU family and current ECU version that are involved in the use case's execution for each use case variant are stated.*
Goal:	*In this section, the fundamental goal(s) of the use case is presented. The comprehensive set of goals of all use case's for a particular function form the goals for the entire function.*
Pre-Conditions:	*In this section, the pre conditions that need to exist in the system for the execution of the use case are to be specified.*
Post-Conditions:	*In this section, the post conditions that will be a result of all possible executions of the use case are to be specified.*
Steps:	*In this section, all the Use Case (UC) steps in the positive execution of the use case are stated in the chronological order.* *Variant Point is inserted at the beginning and end of those UC Steps pertaining to each variant.* *All the other steps which are not surrounded by any variant point are common for all variants.* *When a UC step involves sending or receiving a signal from or to a primary ECU, the signal name is to be specified with a non-implementation specific signal value for the first time it appears in the UCD. In the subsequent UC steps that involve the same signal, a constant name referring to that signal can be used.*
Alternatives:	*In this section, for each UC step the possible alternatives are considered. For instance, the alternative may pertain to the case when the positive condition considered by UC Step does not exist.* *For each alternative of each UC Step, the corresponding alternative steps of how the alternative condition is handled by the function is specified in a similar format to the UC steps.*

Figure 6.3: Abstract UCD template

the tree represents one Use Case (UC) step or its alternative. Here, each alternative UC step should be represented as a single node irrespective of how many steps it has under it. Hence, the tree that is generated as a result has several paths, each path containing a unique combination of steps specified in the UCD.

STEP 1.5 Identification of all possible paths in the UCT

In this step, all possible UCT paths are to be identified such that

a) Nodes common to all variants are covered at least once for each variant

b) Nodes specific to a particular variant are covered at least once for that variant

STEP 1.6 Generating MSCs for each identified UCT path

Each UCT path identified as a result of the previous step, is considered to be a 'scenario'. As defined in Chapter 4, a scenario is one single sequence of interaction between the involved systems to execute a use case of the function. Paths covering the primary

UC Steps for each variant are to be represented by the main scenario. All the other paths covering one of more of the alternative UC steps are termed as the alternative scenarios.

Now, MSCs are to be generated to represent each identified scenario. Adequate notes should be used within the MSC to capture important information from the UCD within the scenario models. This is necessary to enhance understandability of the models. Such scenario-based modelling of the requirements enhances the implementation-specific representation of the function requirements and provides testable semi-formal scenarios. Here, the level of implementation-specific details to be incorporated within the scenario models is critical and required to be established. To generate a scenario, the communication between any two systems through signals is to be depicted in the MSC of the scenario, based on the information provided in the UCD. This signal information in the UCD for inter-system communication should be presented as non-implementation specific signal values for implementation specific signal names (stated in Figure 6.3). The motivation behind using non-implementation specific signal values is to provide a means of avoiding a need to redundantly reflect any change to the signal values for all the scenarios in which the corresponding signal is used. Such redundancy would lead to inefficient means of handling changes to the function's inter-system communication signals.

For instance, let us consider the signal *totalfuellevel(value)* which can have a valid input value ranging from 0-100, an invalid Error or Not Available value represented as 'NA/ERR'. Now representing the invalid signal value using *totalfuellevel(NA/ERR)* would mean any change to the corresponding value used to represent the error or not available condition would require substantial effort to reflect this change in all scenarios that use this signal value. Instead, in such a case, if using a non-implementation specific signal value that represents the condition like *totalfuellevel(Error/Not Available)*, would provide a means to store the corresponding implementation-specific value in a central repository and reflect it in the scenarios dynamically. This would ensure any change in the implementation-specific signal values can be handled at one location and be reflected consistently across all scenarios that utilize this signal value. Similar concept can be extended for the signal names. Such a means would support both consistent change management as well as a means to automate the generation of the scenario MSCs from the UCDs using suitable tools.

Since, this thesis work deals with laying down a suitable process for enhancing distributed software function testing and does not consider the surrounding tool and people aspect required, we will address this in the discussion and future work section.

STEP 1.7 Presenting a comprehensive view of the function
On identifying possible UCT paths and representing them as scenarios using MSCs, the next natural step is to present the final set of scenarios of the function for each use case, based on the variant information. This representation is used to provide a comprehensive view of the function as a set of complete and testable function scenarios.

STEP 2. Establishing traceability across test levels
This step can be illustrated as in Figure 6.4. Here, to establish traceability by capturing the requirements test coverage results across the test levels, it is important to identify the test rounds at each test level that correspond to testing the same software version. This is necessary to ensure the correct requirements test coverage information is captured across the test levels.
It is important to note here that since the systems and signals involved in the execution of the main scenario vary in case of different variants, we consider scenarios for each variant individually for establishing traceability.

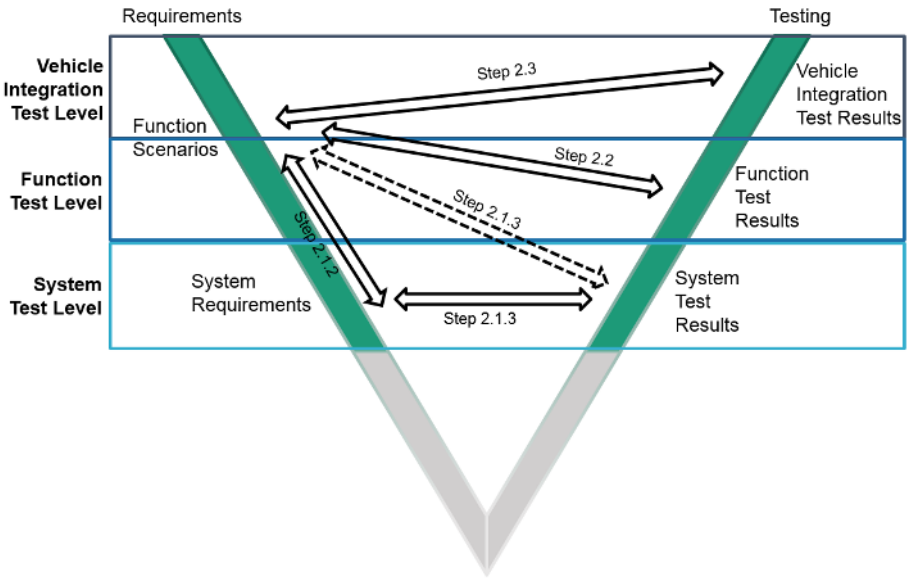


Figure 6.4: Illustration of traceability across system, function and vehicle integration test levels

STEP 2.1 Considering the system test level

The first step is to consider the system test level. Here, there is a need to initially establish traces between the function scenarios and system level requirements. There after, the system level requirements test coverage pertaining to the relevant requirements can be captured against the function scenarios. This provides an understanding of how much of the function is tested at the system test level across all the relevant systems.

STEP 2.1.1 Breaking down each scenario to the corresponding system views

To establish traces between the function scenario and system level requirements, each function scenario should be taken and broken down to the system view. Thus, each scenario has a corresponding set of system views of all the systems that contribute to its realization. Here, each system view represents a set of all relevant input and output combinations pertaining to that system's role in the scenario. This is depicted in Figure 6.5.

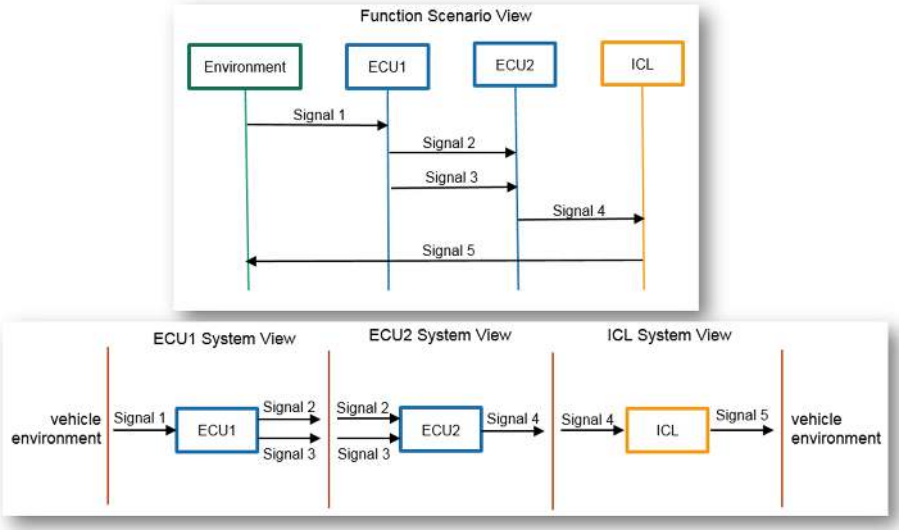


Figure 6.5: Illustration of the concept of function scenario and it's corresponding system views

STEP 2.1.2 Mapping system views to system requirements

For each system view of a function scenario, the requirements of the system that pertain to ensuring the desired output is generated for the provided input i.e, requirements that handle the interface signals under consideration should be mapped to the scenario. This, performed for all the systems

involved in the scenario will produce a set of requirements of each involved system that together make up the system level requirements for that function scenario.

STEP 2.1.3 Mapping system requirements to system requirements test coverage results

To obtain system level test coverage information for the function scenario, the test coverage results for the corresponding system level requirements spread across multiple systems should be gathered and mapped to the function scenario.

The above steps (2.1.1 to 2.1.3) are to be iterated for all function scenarios in order to collect the entire function's system level test coverage information for each test round.

STEP 2.2 Considering the function test level

The next step is to consider the function test level. The test focus at this level, as discussed previously in Section 4.1, is on testing the overall function requirements. Therefore, the test results should be mapped to the corresponding scenarios for each test round. Here, the scenarios and their expected outcomes, represent the function requirements that are to be tested.

STEP 2.3 Considering the vehicle integration test level

The last step is to consider the vehicle integration test level. At this test level, as discussed previously in Section 4.1, the test focus is on interface communication across the systems involved in the execution of the function scenarios. Therefore, at this test level, similar to the case of the function test level, the test results are to be mapped against the function scenarios for each test round.

STEP 2.4 Presenting the function's comprehensive test coverage information

In this step, the obtained comprehensive test coverage information for the function, across the three test levels should be presented accordingly.

On implementing the above steps of the verification strategy, comprehensive function test coverage information across the system, function and vehicle integration test levels for the distributed software functions is expected to be obtained. This information can then be used to identify risky test gaps, test redundancies and make a more data-driven decision during integration testing of the distributed software functions at the vehicle integration test level.

6.2 Validation of the Proposed Cross-Functional Verification Strategy

The proposed verification strategy was validated by implementing it using historic data pertaining to a legacy embedded software function in Scania Trucks, UF18- Fuel Level Display (FLD). This function is distributed across a total of six systems which are system tested across three different departments within the organization. It has two variants, each using a different subset of systems for its execution. Moreover, it is a legacy function. This indicates that it is required to be present on every truck irrespective of the vehicle and fuel type. It can be considered to be a simple function which has mediocre safety critical nature. While there are other more safety critical functions, they are too complex to study within the given time frame of the thesis. Hence, all the above factors provide a multi-fold motivation for the choice of the function used for validating the proposed strategy.

6.2.1 Implementation of the Proposed Verification Strategy on FLD Function

The results of implementing the proposed strategy on the chosen FLD function based on the steps presented in Section 6.1 are as follows.

STEP 1 Enhancing requirements specification
Each sub-step performed to realize the goal of this step, which is to obtain the final semi-formal scenario-based representation of the chosen function, was initially performed based on the researcher's study and analysis of the function's integration and system level requirements and other relevant descriptive documents which assisted in gaining a complete understanding of the function. To further strengthen the completeness and accuracy of the obtained final results, each sub-step results were discussed, reviewed and refined with the function owner who is responsible for the function.

STEP 1.1 Identification of function use cases
For the FLD function there exist two use cases or function parts which make up the whole function. The first use case is to detect and display the fuel level on the Instrument Cluster (ICL) which is the display component of the truck where the driver (considered as part of the truck environment) can view the fuel level in the fuel gauge. The second use case is to detect, activate and deactivate low fuel level warning based on when the corresponding activation and deactivation conditions are met in the truck. The low fuel

level warning is then displayed accordingly on the ICL. Naming the identified use cases based on their primary behaviour, the function's use cases are as presented in Figure 6.6.

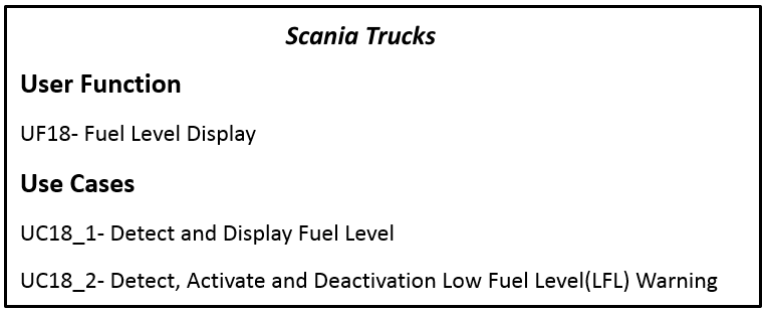


Figure 6.6: FLD function use cases

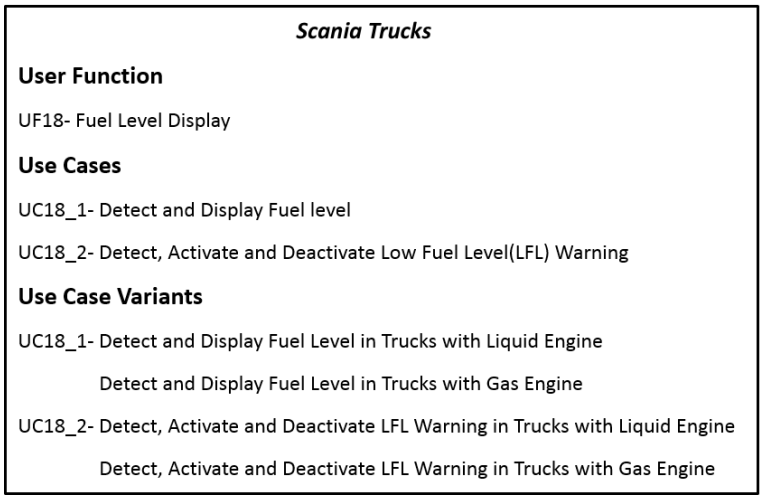


Figure 6.7: FLD function use case variants

STEP 1.2 Identification of function use case variants
The FLD function has two function use case variants, namely, the FLD function execution in trucks with gas engine (Compressed Natural Gas and Liquefied Natural Gas fuel types) and the FLD function execution in trucks with liquid engine (Diesel fuel type). The subset of systems involved in the execution of the function variants vary. Hence, the representation of the use cases of the function can be further enhanced based on the two variants identified as illustrated in Figure 6.7.

STEP 1.3 Generating UCDs with adequate variant information
For each of the two use cases (UC18_1 and UC18_2) an UCD was written by referring to the given UCD template to ensure

the UCD captures all steps in the execution of the use cases for all variants. The final UCDs were reviewed and refined with the help of the function owner. The UCDs for UC18_1 and UC18_2 are presented as Figures D.2 and D.1 respectively in Appendix D. It is important to note here that, due to confidentiality of the logic that is built in the relevant systems for execution of the FLD function, certain requirements have been captured at a higher level of abstraction within the UCDs presented in this report.

STEP 1.4 Generating UCTs

The next step in the natural order of implementing the strategy included generating the UCTs for each use case, based on the corresponding UCDs. Here, the primary UC Steps in the *Steps* section of the UCD (refer to Figures D.2 and D.1) which describe how the function is executed in case of valid conditions/inputs and the corresponding alternative steps which describe how the function is executed in case of invalid conditions/inputs are differentiated for improved understanding based on color coding the nodes accordingly. Here, green color nodes are used to represent the UC steps in case of valid conditions and red color nodes are used to represent the alternative UC steps in case of invalid conditions. Figures E.1 and E.2 in Appendix E represent the UCTs for UC18_1 and UC18_2 respectively.

STEP 1.5 Identification of all possible paths in the UCTs

Now considering the UCTs for each use case separately, there are a total of 5 paths identified for UC18_1 that cover all the nodes based on the conditions stated for the identification of the paths in Section 6.1. Of these, paths 1 and 2 cover the primary UC steps for each variant and hence are part of the UC18_1 main scenario. The remaining 3 paths cover one or more of the alternative UC steps and are hence the alternative scenarios for UC18_1. Similarly, for UC18_2 there are a total of 3 paths identified that cover all the nodes as per the conditions. Of these, paths 1 and 2 cover the primary UC steps for each variant and hence are part of the UC18_2 main scenario. The remaining one path is represented as the alternative scenario for this use case. Hence, a total of 6 scenarios, 4 scenarios of UC18_1 and 2 of UC18_2 represent the entire FLD function.

STEP 1.6 Generating MSCs for each identified UCT path

In this step, each scenario identified was represented using an MSC. The MSCs were generated manually as of today. An example MSC for the alternative scenario 1 of UC18_1 is presented

in Figure 6.8. This scenario depicts how the function is executed and fuel level estimation is calculated when the fuel level sensor signal value is error or not available.

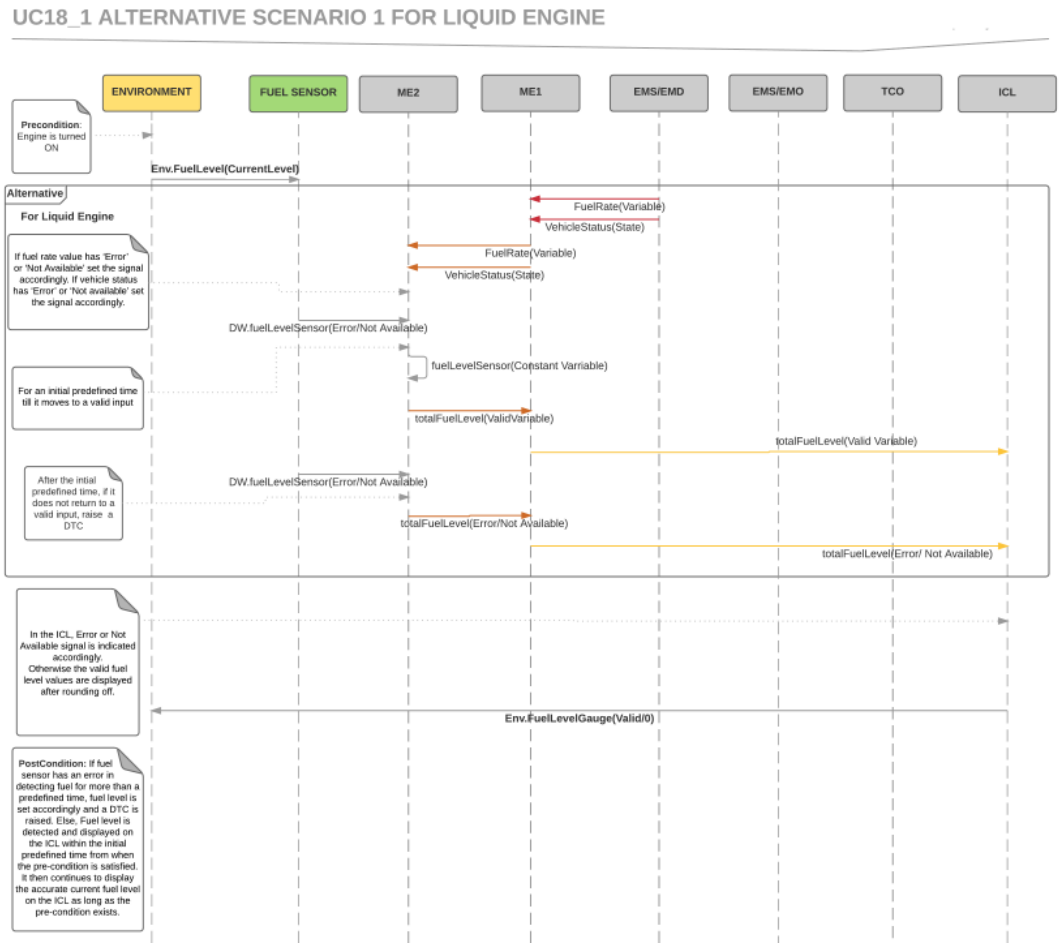


Figure 6.8: MSC for UC18_1 Alternative Scenario 1

STEP 1.7 Presenting a comprehensive view of the function
Each of the 6 function scenarios along with the paths they cover and variants they pertain to can be summarized as illustrated in Figure 6.9.

STEP 2. Establishing traceability across test levels
An essential task while establishing traceability across requirements and test results of the three test levels for the FLD function was to identify the appropriate test rounds across the test levels that contribute to testing the same system software versions. For this, a top down approach was implemented. Here, three of the latest test rounds

User Function	Function Use Cases	Use Case Variants	Use Case Scenarios	Function Scenarios (SCNs)	Paths covered as per the corresponding UCTs
UF18	UC18_1	Liquid and Gas Engine	Main Scenario	SCN1	1-2-3-4p-5-6p-12-13-14p-15-16-17 1-7-8p-9-10-11p-12-13-14p-15-16-17
		Liquid Engine	Alternative Scenario 1	SCN2	1-2-3-4n_1-6n-12-13-14n-15-16-17
			Alternative Scenario 2	SCN3	1-2-3-4n_2-5-6p-12-13-14p-15-16-17
		Gas Engine	Alternative Scenario 3	SCN4	1-7-8n-9-10-11n-12-13-14n-15-16-17
	UC18_2	Liquid and Gas Engine	Main Scenario	SCN5	1-2-3p-4-6-7p-8-9-10 1-2-3p-5-6-7p-8-9-10
			Alternative Scenario 1	SCN6	1-2-3n-6-7n-10

Figure 6.9: Comprehensive set of FLD function scenarios

(spread over a period of three months) at the vehicle integration test level were chosen. The test results for each of these test rounds was reported on test weeks TW1602, TW1606 and TW1610. Here, the software versions of all the relevant systems (systems that are involved in the execution of the chosen FLD function) were identified. This information was then used as a basis to identify the test rounds and test reports at the function and system test level where the corresponding software version was tested.

STEP 2.1 Considering the system test level

In order to consider the system level requirements and test coverage information, in a manner that can be effectively mapped to the function scenarios, the following steps were executed.

STEP 2.1.1 Breaking down each scenario to the corresponding system views

Each scenario was taken and broken down to the corresponding system views of all the systems that were involved in its realization. For instance, let us consider function scenario SCN2 which represents alternative scenario 1 for UC18_1 for liquid engine trucks (refer to Figure 6.9). It has 4 systems that are involved in its execution, namely, Main ECU 2 (ME2) that handles the main fuel level estimation calculation, Engine Management system - Diesel engine (EMD) that sends important and required information needed for the fuel level estimation calculation, Instrument Cluster (ICL) that displays the fuel level in the fuel gauge, and Main ECU 1 (ME1) which in this scenario helps bridge messages in the form of signals across different systems. The scenario and each of the system views for SCN2 (UC18_1 Alternative Scenario 1) are depicted in Figure 6.8 and Figure 6.10 respectively.

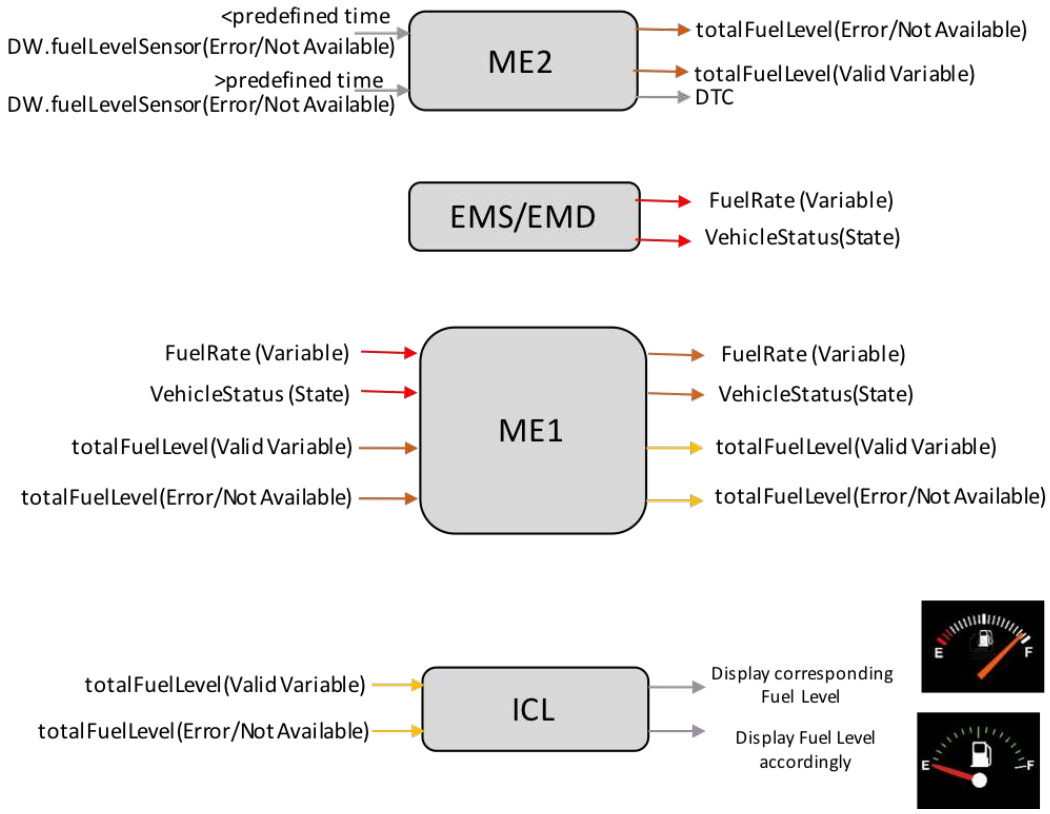


Figure 6.10: Set of system views for FLD function SCN2 (UC18_1 Alternative Scenario 1)

STEP 2.1.2 Mapping system views to system requirements

For each system view generated for the function scenario, the requirements of the system that pertain to ensuring the desired output is generated for the given input were identified and mapped to the function scenario. For instance, taking the case of function scenario SCN2, for each of the 4 systems the corresponding system level requirements were identified. Among the three systems ME2, ME1 and ICL, ME2 and ME1 belonged to one organizational department and the ICL belonged to another organizational department. This lead to a slight variation in the naming patter used for the system level requirements for these systems. Yet, the level of abstraction of the requirements remained consistent which ensured the system level requirements could be captured accurately for all 3 systems. Coming to the case of the EMD system, it was identified that the requirements for this system were being formulated and was a work in progress. This

is one of the major gaps identified in successfully implementing the proposed strategy that will be discussed in the later sections. To handle this missing information, the researcher captured the requirements of the system in all the applicable scenarios and reviewed and revised the captured requirements with the organizational team handling the EMD system development and testing. This ensured the requirements were stated precisely. A similar means was employed in handling the missing requirements at system level for the Engine Management system - Gas engine (EMO) which belonged to the same engine management organizational department. On performing this for each function scenario of each use case variant, a comprehensive set of system level requirements pertaining to the FLD function were identified. This is as presented in Figure 6.11.

Use Case Variant	Func -tion SCN	System Level Requirements					
		ME2	ME1	ICL	EMD	EMO	GSC
Liquid Engine	SCN1	AER201_2 AER201_5 AER201_3 AER201_10 AER201_4 AER201_11	AER343_1 AER343_3 AER343_4 AER343_5 AER343_6 AER343_7 AER343_8	SIR002_01_D_01 SIR002_17_A SIR002_18_A	CAN Specification indicating relevant signals are sent from the EMD to ME2 as and when required	-	-
	SCN2	AER201_14 AER201_35 AER201_33 AER201_36 AER201_34 AER201_37		SIR002_01_D_01 SIR002_17_A SIR002_01_D_02 SIR002_18_A			
	SCN3	AER201_12 AER201_15		SIR002_01_D_01 SIR002_17_A SIR002_18_A		-	-
	SCN5	AER201_2 AER201_5 AER201_3 AER201_6		SIR002_5_D SIR002_10_B SIR002_13_B SIR002_14_B		-	-
	SCN6	AER202_4		SIR002_15_A SIR002_16_A	-	-	-
Gas Engine	SCN1	AER201_30		SIR002_01_D_01 SIR002_17_A SIR002_18_A	-	EMO_SFR1	GSC_SFR1 GSC_SFR2 GSC_SFR3
	SCN4	AER201_31		SIR002_01_D_01 SIR002_17_A SIR002_01_D_02 SIR002_18_A	-	EMO_SFR1	GSC_SFR3 GSC_SFR4
	SCN5	AER201_7 AER201_3 AER201_5 AER201_6		SIR002_5_D SIR002_10_B SIR002_13_B SIR002_14_B	CAN Specification indicating relevant signals are sent from the EMD to ME2 as and when required	-	-
	SCN6	AER202_4		SIR002_15_A SIR002_16_A	-	-	-

Figure 6.11: FLD function scenarios mapped to system level requirements

STEP 2.1.3 Mapping system requirements to system requirements test coverage results

On capturing system level requirements across the systems involved in the execution of the FLD functions scenarios, the corresponding test results from the relevant test rounds were captured and mapped accordingly. The test results captured from the different test rounds at the system level are reported against the chosen test weeks of the integration test level (TW1602, TW1610, TW1614) as presented in Figure 6.12. The overall coverage of a scenario at the system test level for a test round is calculated as a percentage value obtained from dividing the number of system level requirements tested by the total number of system level requirements identified. For instance, Figure 6.13 presents the system test level coverage of SCN2 for TW1606.

Variant	Func -tion SCN	Coverage of Requirements at System Level for TW1602						Coverage of Requirements at System Level for TW1606						Coverage of Requirements at System Level for TW1610					
		ME2	ME1	ICL	EMD	EMO	GSC	ME2	ME1	ICL	EMD	EMO	GSC	ME2	ME1	ICL	EMD	EMO	GSC
Liquid	SCN1	6/6	3/7	0/3	2/2	-	-	6/6	3/7	3/3	2/2	-	-	6/6	3/7	2/3	2/2	-	-
	SCN2	6/6	3/7	0/4	2/2	-	-	6/6	3/7	3/4	2/2	-	-	6/6	3/7	3/4	2/2	-	-
	SCN3	2/2	3/7	0/3	2/2	-	-	2/2	3/7	3/3	2/2	-	-	2/2	3/7	2/3	2/2	-	-
	SCN5	4/4	3/7	0/4	2/2	-	-	4/4	3/7	4/4	2/2	-	-	4/4	3/7	0/4	2/2	-	-
	SCN6	1/1	3/7	0/2	-	-	-	1/1	3/7	1/2	-	-	-	1/1	3/7	1/2	-	-	-
Gas	SCN1	1/1	3/7	0/3	-	1/1	3/3	1/1	3/7	3/3	-	0/1	0/3	1/1	3/7	2/3	-	0/1	0/3
	SCN4	1/1	3/7	0/4	-	0/1	0/2	1/1	3/7	3/4	-	0/1	0/2	1/1	3/7	3/4	-	0/1	0/2
	SCN5	3/4	3/7	0/4	1/1	-	-	4/4	3/7	4/4	1/1	-	-	4/4	3/7	0/4	1/1	-	-
	SCN6	1/1	3/7	0/2	-	-	-	1/1	3/7	1/2		-	-	1/1	3/7	1/2	-	-	-

Figure 6.12: System level requirements coverage across test rounds TW1602, TW1606 and TW1610

Use Case Variant	Function SCN	System Level Requirements				Coverage of Requirements at System Level for TW1606			
		ME2	ME1	ICL	EMD	ME2	ME1	ICL	EMD
Liquid	SCN2	AER201_14 AER201_35 AER201_33 AER201_36 AER201_34 AER201_37	AER343_1 AER343_3 AER343_4 AER343_5 AER343_6 AER343_7 AER343_8	SIR002_01_D_01 SIR002_01_D_02 SIR002_17_A SIR002_18_A	CAN Specification indicating relevant signals are sent from the EMD to ME2 as and when required	6/6	3/7	3/4	2/2
Overall Coverage of SCN2 at System Test Level for TW1606						14/19			
						73.68%			

Figure 6.13: System level requirements coverage for FLD function SCN2 for test round TW1606

STEP 2.2 Considering the function test level
At the function test level for the FLD function, while there is a black-box testing of the function scenarios that is performed by the function owner, there are no test reports generated or test results stored. Hence, the test results for the considered test rounds were captured based on interaction with the function owner regarding when and what was tested during these test rounds. The consequence of unavailability of test results and test data from this test level on the proposed strategy and on the overall testing process is discussed in the later sections.

STEP 2.3 Considering the vehicle integration test level
At the vehicle integration test level, the test results reported against the previous incomplete scenarios of the function were captured and mapped against the newly generated complete set of scenarios for each of the test rounds considered. The results of this step are presented in Table 6.14.

Test Round	Total number of Trucks Tested	Liquid Engine Trucks	UF18 SCNS covered	Gas Engine Trucks	UF18 SCNS covered
TW1602	11	11	SCN 1 SCN5	0	-
TW1606	12	10	SCN 1 SCN5	2	SCN 1
TW1610	9	9	SCN 1 SCN5	0	-

Figure 6.14: Vehicle integration test level scenario coverage results

STEP 2.4 Presenting the function's comprehensive test coverage information
As a result of the above steps, comprehensive test coverage information for the FLD function across the three test levels was obtained for each of the three test rounds considered. The results obtained for test round TW1606 are presented in Figure 6.15. Similar comprehensive function test coverage information for test rounds TW1602 and TW1610 are presented in Figures F.1 and F.2 in Appendix F.

The obtained comprehensive function test coverage information for FLD function across the system, function and vehicle integration test levels for the considered three test rounds helped identify the following:

Use Case Variant	Function SCN	TW1606		
		System Test Level	Function Test Level	Integration Test Level
Liquid	SCN1	77.78%	100%	100%
	SCN2	73.68%	Not Tested	Not Tested
	SCN3	71.42%	Not Tested	Not Tested
	SCN5	76.47%	100%	100%
	SCN6	50%	Not Tested	Not Tested
Gas	SCN1	46.67%	Not Tested	100%
	SCN4	46.67%	Not Tested	Not Tested
	SCN5	75%	Not Tested	Not Tested
	SCN6	50%	Not Tested	Not Tested

Figure 6.15: Comprehensive FLD function scenario test coverage across the three test levels for test round TW1606

- From the data presented in Figures 6.15, F.1 and F.2, it is evident that most of the test effort across the test levels for testing the FLD function is focused on liquid engine trucks. This exposes the risky test gap in testing the function adequately across test levels for its implementation in gas engine variant of the trucks.
- There is redundant testing of the main function scenarios, SCN1 and SCN5, at the function and vehicle integration test levels. On obtaining comprehensive function test coverage information across test levels, such redundancies can be avoided by ensuring the function and vehicle integration test levels cover different scenarios of the function.
- On further studying the data presented in Figure 6.12, it is evident that most of the test gaps at the system level for testing the FLD lie in ME1, EMO and GSC systems.
- With the data now present at the vehicle integration test level, any change can be traced from the system it is pertaining to, to the appropriate function scenarios that it effects. This reduces the effort in analyzing the change impact at the vehicle integration level.

- The strategy presents a clear need to report the test results from each of the test levels. Any unknown test effort at any test level will be a barrier in identifying the test gaps and test redundancies, and thereby a barrier to performing more informed testing at the higher vehicle integration test level.

Another important result obtained on implementation of the proposed verification strategy was the identification of a potential error in the function design for one of its variant which was unnoticed previously.

Chapter 7

Discussion and Limitations

7.1 Discussion

The research conducted helped identify the current approach to test distributed automotive embedded software functions as described in literature [1][2] and as implemented at the case company (refer to Section 4.1), in order to answer research question RQ1. This helped capture knowledge and gain a comprehensive understanding of the state-of-the-art approach used to test distributed software functions across the test levels of the automotive industry V-model. In addition, the issues with the current test approach implemented at the case company have been identified to answer research question RQ3. Based on the analysis of the data collected, it was deduced that the issues exist within three fundamental areas. These areas are the *people, process* and *technology* aspects of the current test approach that in turn lead to the explicit test issues as presented in Section 5.1. The people issues have been found to be profoundly dealing with lack of adequate knowledge transfer across the different test levels and consequently among the different departments of the organization that deal with testing activities. Moreover, it was also found that the current test approach is hindered by a lack of bigger picture of test effort across the test levels for all involved test engineers. There was also ambiguity in the test role at the function test level which was identified. Under technology issues, it was found that the efficiency of the current test approach was effected significantly due to the lack of suitable tools to assist and reduce manual effort in test management, requirements management and change management. Each of these disciplines are highly intertwined with the test approach and hence have an impact on its effectiveness and efficiency. Further, the process issues were found to be incomplete and untestable function requirements which inefficiently captured function variant information. In addition, there was a lack of established traceability along the requirements and across the requirements and test results of the V-model that was identified.

The explicit test issues so caused due to the above mentioned set of people, process and technology issues include: *lack of comprehensive function test coverage information across test levels*, *ambiguous and inaccurate test reports generated*

at vehicle integration test level, *lack of test report generation at function test level* and *lack of appropriate means to identify where or whether redundant testing and risky test gaps are present*. The study of these explicit test issues helped recognize that there was in fact no comprehensive test coverage information of how much the distributed software functions are tested across the different test levels at the case company. This helped answer research question RQ2. Another observation was that the overall people, process, technology and explicit test issues identified at the case company were similar to the ones identified in literature presented in Section 2.3. Of the three set of issues, process issues have been found to be comparatively most widespread in terms of their existence and knowledge of the hindrance they are causing to the effectiveness of the current test approach as presented in Section 4.5.

There after, the general areas of people, process and technology were further studied in Section 5.2.1 It was then identified based on scientific literature knowledge and analysis of the case study results that, the process issues lie at the core of the three identified issue categories. Technology is adopted to fit the process established, and people knowledge and effort is built around the process. Hence, addressing the process issues was deduced to pave the way to further studying and addressing the people and technology issues that persist around the established effective process. Thus, initially to propose a cross-functional verification strategy, a study of the alternative process-enhancement approaches and their comparative analysis based on scientific literature knowledge and industry expert opinion was conducted. This helped draw conclusions that a semi-formal use case and scenario-based approach to aligning requirements specification and testing can be deemed relatively most feasible and effective solution to address and enhance the process of the current test approach.

Hence, the proposed cross-functional verification strategy presents an approach of use case and scenario-based semi-formal modelling of requirements specification and testing at the vehicle integration test level of the automotive industry V-model. There after, a means to establish adequate traces across the system, function and vehicle integration test levels is provided. Based on the results obtained on implementing the verification strategy on a legacy function at Scania, it was inferred that the proposed strategy proves to be an effective means of obtaining multi-level comprehensive test coverage information for distributed software functions in a practical real world setting. This information was found to assist in identifying test redundancies and test gaps across the test levels. Therefore, it provides a means to enhance test effectiveness of integration testing of the distributed software functions based on more informed decisions driven by the comprehensive test coverage information. This study thus helped answer the final research question RQ4.

Moreover, the implementation of the proposed verification strategy helped identify a potential function design error. This opens up the possibility of more extensively investigating the value of the proposed strategy for enhancing the function designs through the proposed means of identification and rectification of possible design errors across the function variants.

The strategy presents a manner in which the requirements and testing data across test levels can be comprehensively tied together to aid in effectively testing the distributed software functions. This consequently led to identifying the two critical gaps or missing links which have the capability to potentially hamper the effectiveness of the strategy. These include missing system-level requirements and missing test reports at function test level. These gaps or missing links should be filled to ensure the strategy can be realized to its full potential and obtain maximum value.

While the strategy's effectiveness is validated based on implementation, its efficiency will likely reach its highest potential with the incorporation of suitable tool support and right people knowledge with this strategy as identified based on the study presented in Section 5.2.1.

During the implementation of the strategy, it was noted that considerable effort was invested in manually generating the function scenario models. A similar or more effort is foreseen for generating the executable test cases from these function scenarios manually. However, the area of implementation of the strategy that took up most of the effort was collecting valuable and critical information in the form of requirements and test results from all concerned test levels to establish the desired traces across the test levels. This information was spread across several departments within multiple data sources. Hence, these aspects that account to maximum effort in terms of time and cost being spent on implementing the strategy present areas where the efficiency of the strategy can most likely be further enhanced.

Study of suitable approaches and complementing tools that can be adopted to enhance the efficiency by reducing the manual effort through semi-automation or complete automation of the above mentioned areas of implementation presents a promising platform for future research. There has been research conducted to study how the processes of generating scenario models from restrictive language text and generating executable test cases from scenarios can be automated like in [95][103][108]. Hence, exploring this area more extensively would help identify suitable means of how the data can be most effectively and efficiently represented, stored, accesses and maintained across multiple data sources. There after, it can also help identify potential tools that can assist in considerably reducing the manual effort invested in implementing this aspect of the verification strategy in the

automotive industry.

The other area of implementation which accounted to maximum manual effort as mentioned above, is for gathering and establishing the required traces across requirements and test results. Handling this step, which is predominantly manual as of today, would require exploring potential Requirements Management and Test Management tools which support cross-communication between them. Moreover, it also requires studying the people knowledge that needs to be established around using these tools for enhancing the execution of the process that has been laid down. While, any such tool will still require adequate manual effort, it will help manage the traces and extract information regarding the distributed software function test coverage across the test levels more efficiently. Hence, this area provides a great platform for research into suitable requirements management and test management tools, and people aspect of adopting those tools that can support the complexity that software in automotive industry poses.

In essence, the process issues identified with the current test approach have been addressed using the proposed verification strategy. The strategy's feasibility and effectiveness was then established as result of an initial study in a real world setting of the automotive industry. The efficiency of this strategy can be addressed by studying the people knowledge and technology tool support aspects that are needed to complement the process-based strategy. Thereby, this would provide a comprehensive people, process and technology-based solution to the test approach issues identified. The current study therefore sets the process-focused platform for moving towards a comprehensive solution for the issues with the current approach to test distributed software functions in the automotive industry.

7.2 Threats to Validity

It is an essential step to determine the threats to validity of a research study and its results as a means of judging the quality of the study. The criticality of this step multiplies in studies implementing empirical research methods that are prone to a wide magnitude of possible threats [120]. The chosen research method - Case study is one such empirical method. Hence, identifying validity threats and specifying appropriate mitigation strategies employed is essential to determine the quality of the research. For empirical quantitative studies such as the one chosen for this thesis, there are 4 major types of validity threats as identified in [120]. These include - construct validity, conclusion validity, internal validity and external validity/generalisability. A description of these major validity threats and others deemed to be relevant to the current research, along with mitigation strategies adopted to avoid them to the best of the researcher's ability are presented in this section. The mitigation strategies employed are based on

existing scientific literature [33][34][40][121].

7.2.1 Construct Validity

Construct validity deals with the extent to which the operational measures studied accurately represent what is being investigated [34]. In the conducted research, there were two possible threats to construct validity. One was a possibility of designing interview questions that may be misinterpreted by the interviewees and lead to collecting data irrelevant to what was being studied. This risk was mitigated in two ways. Firstly, the designed interview questions were reviewed and revised with the help of the thesis supervisors to ensure their suitability and validity. There after, the data collected was analyzed simultaneously while conducting the interviews so as to re-adjust the interview questions if deemed necessary based on discussion with the supervisors. Moreover, data triangulation was employed where in the theory generated was confirmed by collecting data from multiple sources of evidence.

Another critical threat to construct validity was the use of terms that were considered to be generic but might mean different things in different contexts and sometimes according to different perspectives. Hence, to mitigate this threat, the terms that posed this threat were defined when and where necessary in the report to ensure a common understanding of the terms is maintained as suggested in [121].

7.2.2 Internal Validity

Internal validity deals with how causal relationships are examined and relevant conclusions are drawn [34]. In the conducted research, there is a possible risk of poor data analysis leading to incorrect conclusions since the researcher is novice. There were multiple counter measures adopted to tackle this risk. Firstly, the results of data analysis were elaborately discussed with the thesis supervisors as well as with the senior engineers and other participants of the case study. Moreover, the results of data analysis were backed with adequate literature support to validate the conclusions drawn. In addition, data collection and data analysis were conducted simultaneously, which is one of the most important strategies of ensuring internal validity according to [40]. However, there still is a possible internal validity threat that exists in this research due to the consideration of limited test result data (test results of three test rounds considered) for drawing conclusions from the implementation of the proposed strategy. This threat can be mitigated by focusing future work on studying the results of implementation of the verification strategy on more extensive test result data spread across several test rounds.

There also exists a possible threat to internal validity due to conducting a literature review for the comparative analysis performed and reported in Section 5.2.1 and Section 5.3.1 rather than a systematic literature review [122]. This presents a risk that some relevant scientific literature was not considered for the study.

7.2.3 External Validity/Generalisability

External validity deals with the extent to which the research findings can be generalized and are viable and of interest outside the investigated case [34]. Since the research was conducted within a single automotive company, it might encounter such an external validity threat where the findings cannot be generalized to other companies. This risk of generalisability was mitigated to a great extent by explaining in sufficient detail the context within which the study was conducted and the characteristics of the case being investigated. In addition, sufficient literature pertaining to the automotive domain is referred in order to establish links between the current work and other related work, as a means of addressing its generalisability.

Another major external validity threat lies in the implementation of the proposed strategy for only one distributed software function. This threat exists in this research work and can be mitigated by focusing future work on strengthening the validity of the proposed strategy by extending its implementation to a large number of more complex distributed functions.

7.2.4 Repeatability

Repeatability deals with whether the study conducted is repeatable, hence making it reliable [34]. For this research, there was a risk that the case study conducted will not be adequately documented to make it repeatable. This risk was mitigated by using the case study protocol to design the research. Moreover, the actions undertaken throughout the research were discussed with and reviewed by the thesis supervisors with the help of weekly thesis status meetings. Such an auditing has been found to help mitigate this risk [33].

7.2.5 Scope

Owning to the complexity of the problem domain that was considered for this research, there is a risk that the scope of the thesis project may be misinterpreted. To mitigate this risk, the scope of the project was adjusted with the assistance of the thesis supervisors according to the needs and possibilities at the case company and according to the consideration of the contribution of this work to the research body of knowledge. The decision on whether to and how to reduce the project

scope was based on a consensus between the researcher, industry supervisor and university supervisor. The decided scope is specified in Chapter 1 in the report to avoid misinterpretation.

Chapter 8

Conclusion and Future Work

8.1 Summary and Conclusion

With the advent of software for the realization of several functions of a vehicle like fuel level display and advanced emergency braking, there has been and continues to be several challenges that the automotive industry faces to which it was unfamiliar a few decades ago [7][8][23][24][26]. This is discussed based on relevant scientific literature and presented in Chapter 2 and identified at the case company and reported in Section 5.1. One such identified facet of the software-related challenges in the automotive industry is in the area of system software testing [1][2].

The automotive embedded software functions are increasingly distributed in nature, implying that the software modules for the realization of the functions are distributed across several ECU systems of the vehicle. This adds to the complexity of integration testing the distributed software functions at the vehicle integration test level of the automotive V-model [2].

There exists research that focuses on several aspects of system software testing both in the general context and within the automotive industry as presented in Chapter 2 and Section 6.1.1. However, there is limited scientific literature focusing on the challenges in integration testing of the distributed software functions [1][2] and possible effective and feasible strategies to tackle these challenges within the automotive industry.

Hence, the current thesis research contributes to the area of integration testing of distributed automotive embedded software functions. Initially, the current approach for testing the distributed software functions, along with the challenges with the current approach are studied, both in literature and the case company as presented in Chapter 2, Section 4.1 and Section 5.1. There after, an effective cross-functional verification strategy is proposed. This strategy provides a means to solve the process-based challenges of the current test approach and thereby enhance the effectiveness of integration testing of the distributed software functions.

Implementation of the proposed strategy on a legacy automotive distributed software function, Fuel Level Display, has given promising results. The broad concept of multi-level reuse of test results across the system, function and vehicle integration test levels of the automotive V-model was found to provide an effective and feasible means of capturing comprehensive test coverage information pertaining to the distributed software functions. Consequently, it has been identified that there is a critical link between requirements and testing that needs to be established across all test levels for implementing the multi-level reuse concept. Such 'traceability' links provide a means to identify test redundancies and test gaps across the test levels for testing the distributed software functions. It hence helps take more data-driven decisions at the vehicle integration test level for integration testing of the functions.

Moreover, the proposed approach to use case and scenario-based function requirements specification has been found to provide a means to capture test-driven requirements in the form of function scenarios. This was found to make them suitable for establishing the desired traceability between requirements specifications and testing across the test levels. Identifying the different execution scenarios of the function also helped suitably capture the variant information pertaining to the function's execution. In addition, the implementation of the strategy helped identify a potential function design error. This opens up the interesting possibility to more extensively study the effectiveness of the proposed strategy for enhancement of distributed automotive embedded software function design through use case and scenario-based requirements specification.

8.2 Future Work

The future work for the thesis research includes:

- Strengthening the claim of effectiveness and feasibility of the proposed verification strategy by implementing it on a wider scale with more number of complex distributed functions and by capturing more extensive test result data spread across several test rounds.
- Strengthening the claim of effectiveness and feasibility of the proposed verification strategy by studying the results of its implementation within the wider context of several automotive companies.
- Exploring potential approaches, tools and the required people knowledge surrounding these tools that can possibly help reduce the manual effort and thereby optimize the cost and effort in the implementation of the proposed verification strategy.

- More extensively studying the value of the proposed verification strategy for enhancement of the distributed automotive embedded software function designs through use case and scenario-based requirements specification.

References

[1] A. Kasoju, K. Petersen, and M. V. Mäntylä, "Analyzing an automotive testing process with evidence-based software engineering," *Information and Software Technology*, vol. 55, no. 7, pp. 1237–1259, 2013.

[2] D. Sundmark, K. Petersen, and S. Larsson, "An exploratory case study of testing in an automotive electrical system release process," in *6th IEEE International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2011, pp. 166–175.

[3] A. M. Pérez and S. Kaiser, "Top-down reuse for multi-level testing," in *17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS)*, 2010, pp. 150–159.

[4] M. Adenmark, "Scania Test Levels, Scania Internal Document (REST08012)," 2008.

[5] M. Conrad, "A systematic approach to testing automotive control software," *Proceedings to International Congress on Transportation Electronics (Convergence '04)*, pp. 297–308, 2004.

[6] F. Saglietti, "Testing for dependable embedded software," in *36th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2010, pp. 409–416.

[7] K. Grimm, "Software technology in an automotive company: major challenges," in *Proceedings of the 25th International Conference on Software Engineering*. IEEE Computer Society, 2003, pp. 498–503.

[8] B. Katumba and E. Knauss, "Agile development in automotive software development: Challenges and opportunities," in *Product-Focused Software Process Improvement*. Springer, 2014, pp. 33–47.

[9] J. S. Her, S. W. Choi, J. S. Bae, S. D. Kim, and D. W. Cheun, "A component-based process for developing automotive ecu software," in *Product-Focused Software Process Improvement*. Springer, 2007, pp. 358–373.

[10] F. Franco, M. Mauro, S. Stevan, A. B. Lugli, and W. Torres, "Model-based functional safety for the embedded software of automobile power window system," in *11th IEEE/IAS International Conference on Industry Applications (INDUSCON)*, 2014, pp. 1–8.

[11] M. Conrad, "Verification and validation according to ISO 26262: A workflow to facilitate the development of high-integrity software," *Proceedings to 6th European Congress on Embedded Real Time Software and Systems (ERTS2)*, 2012.

[12] S. S. Barhate, "Effective test strategy for testing automotive software," in *International Conference on Industrial Instrumentation and Control (ICIC)*. IEEE, 2015, pp. 645–649.

[13] R. Awédikian and B. Yannou, "A practical model-based statistical approach for generating functional test cases: application in the automotive industry," *Software Testing, Verification and Reliability*, vol. 24, no. 2, pp. 85–123, 2014.

[14] J. McDonald, L. Murray, P. Lindsay, and P. Strooper, "Module testing embedded software-an industrial pilot project," in *Proceedings to the 7th IEEE International Conference on Engineering of Complex Computer Systems(ICECCS)*. IEEE, 2001, pp. 233–238.

[15] O. Praprotnik, M. Gartner, M. Zauner, and M. Horauer, "A test suite for system tests of distributed automotive electronics," in *2nd International Conference on Advances in Circuits, Electronics and Microelectronics(CENICS)*. IEEE, 2009, pp. 67–70.

[16] G. Dhadyalla, N. Kumari, and T. Snell, "Combinatorial testing for an automotive hybrid electric vehicle control system: a case study," in *IEEE 7th International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 2014, pp. 51–57.

[17] "IEEE Standard for System and Software Verification and Validation," *IEEE Std 1012-2012 (Revision of IEEE Std 1012-2004)*, pp. 1–223, May 2012.

[18] J. S. Collofello, "Introduction to software verification and validation," Software Engineering Institute Curriculum Module (SEICM), Carnegie Mellon University, Tech. Rep. SEI-CM-13-1.1, 1988.

[19] D. R. Wallace and R. U. Fujii, "Software verification and validation: an overview," *IEEE Software*, vol. 6, no. 3, p. 10, 1989.

[20] *ISO 26262-1:2011 Road vehicles Functional safety - Part 1: Vocabulary*, International Organization for Standardization(ISO), Geneva, Switzerland, November 2011.

[21] L. Freedman, *Strategy: A history.* Oxford University Press, 2013.

[22] A. Sherer, J. Rose, and R. Oddone, "Ensuring functional safety compliance for ISO 26262." Proceedings of the ACM/EDAC/IEEE 52nd Annual Design Automation Conference(DAC), 2015, p. 98.

[23] M. Weber and J. Weisbrod, "Requirements engineering in automotive development- Experiences and challenges," in *Proceedings of the IEEE Joint International Conference on Requirements Engineering*, 2002, pp. 331–340.

[24] F. Fabbrini, M. Fusani, G. Lami, and E. Sivera, "Software engineering in the European automotive industry: Achievements and challenges," in *32nd Annual IEEE Computer Society International Conference on Computers, Software and Applications(COMPSAC)*, 2008, pp. 1039–1044.

[25] M. Broy, "Automotive software and systems engineering," in *Proceedings of the 3rd ACM and IEEE International Conference on Formal Methods and Models for Co-Design(MEMOCODE)*, 2005, pp. 143–149.

[26] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th International Conference on Software Engineering.* ACM, 2006, pp. 33–42.

[27] M. Broy, I. H. Kruger, A. Pretschner, and C. Salzmann, "Engineering automotive software," *Proceedings of IEEE*, vol. 95, no. 2, pp. 356–373, 2007.

[28] G. Hurlburt and J. Voas, "Software is driving software engineering?" *IEEE Software*, vol. 33, no. 1, pp. 101–104, Jan 2016.

[29] P. Braun, M. Broy, F. Houdek, M. Kirchmayr, M. Müller, B. Penzenstadler, K. Pohl, and T. Weyer, "Guiding requirements engineering for software-intensive embedded systems in the automotive industry," *Computer Science- Research and Development*, vol. 29, no. 1, pp. 21–43, 2014.

[30] J.-L. Boulanger and V. Dao, "Requirements engineering in a model-based methodology for embedded automotive software," in *IEEE International Conference on Research, Innovation and Vision for the Future in Computing Communication Technologies(RIVF)*, 2008, pp. 263–268.

[31] A. Puschnig and R. T. Kolagari, "Requirements engineering in the development of innovative automotive embedded software systems," in *Proceedings of the 12th IEEE International Requirements Engineering Conference.* IEEE, 2004, pp. 328–333.

[32] E. Bringmann and A. Kramer, "Model-based testing of automotive systems," in *1st IEEE International Conference on Software Testing, Verification, and Validation*, 2008, pp. 485–493.

[33] R. K. Yin, *Case study research: Design and methods*. Sage Publications, 2013.

[34] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, pp. 131–164, 2008.

[35] P. Rossi, J. Wright, and A. Anderson, *Handbook of Survey Research*, ser. Quantitative Studies in Social Relations. Elsevier Science, 2013.

[36] C. Robson and K. McCartan, *Real world research*. Wiley, 2002.

[37] J. Rowley and F. Slack, "Conducting a literature review," *Management Research News*, vol. 27, no. 6, pp. 31–39, 2004.

[38] (Accessed: January 24, 2016) Scania. [Online]. Available: http://www.scania.se/

[39] M. N. Marshall, "Sampling for qualitative research," *Family practice*, vol. 13, no. 6, pp. 522–526, 1996.

[40] J. M. Morse, M. Barrett, M. Mayan, K. Olson, and J. Spiers, "Verification strategies for establishing reliability and validity in qualitative research," *International Journal of Qualitative Methods*, vol. 1, no. 2, pp. 13–22, 2002.

[41] J. Rowley, "Conducting research interviews," *Management Research Review*, vol. 35, no. 3/4, pp. 260–271, 2012.

[42] D. W. Turner III, "Qualitative interview design: A practical guide for novice investigators," *The qualitative report*, vol. 15, no. 3, p. 754, 2010.

[43] C. B. Seaman, "Qualitative methods in empirical studies of software engineering," *IEEE Transactions on Software Engineering*, vol. 25, no. 4, pp. 557–572, 1999.

[44] S. Looso, R. Borner, and M. Goeken, "Using grounded theory for method engineering," in *5th International Conference on Research Challenges in Information Science (RCIS)*, 2011, pp. 1–9.

[45] S. Adolph, W. Hall, and P. Kruchten, "Using grounded theory to study the experience of software development," *Empirical Software Engineering*, vol. 16, no. 4, pp. 487–513, 2011.

[46] G. Coleman and R. O'Connor, "Using grounded theory to understand software process improvement: A study of Irish software product companies," *Information and Software Technology*, vol. 49, no. 6, pp. 654–667, 2007.

[47] J. Lawrence and U. Tar, "The use of grounded theory technique as a practical tool for qualitative data collection and analysis," *The Electronic Journal of Business Research Methods*, vol. 11, no. 1, pp. 29–40, 2013.

[48] O. Badreddin, "Thematic review and analysis of grounded theory application in software engineering," *Advances in Software Engineering*, vol. 2013, p. 4, 2013.

[49] J. M. Corbin and A. Strauss, "Grounded theory research: Procedures, canons, and evaluative criteria," *Qualitative sociology*, vol. 13, no. 1, pp. 3–21, 1990.

[50] B. G. Glaser, A. L. Strauss, and E. Strutzel, "The discovery of grounded theory: Strategies for qualitative research." *Nursing Research*, vol. 17, no. 4, p. 364, 1968.

[51] A. Cooney, "Choosing between Glaser and Strauss: An example," *Nurse researcher*, vol. 17, no. 4, pp. 18–28, 2010.

[52] A. Strauss and J. Corbin, *Basics of qualitative research: Grounded theory procedures and techniques*. Sage Publications, 1990.

[53] S. Sarker, F. Lau, and S. Sahay, "Building an inductive theory of collaboration in virtual teams: An adapted grounded theory approach," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*. IEEE, 2000, pp. 1–10.

[54] J. W. Drisko, "Using qualitative data analysis software," *Computers in Human Services*, vol. 15, no. 1, pp. 1–19, 1998.

[55] A. Atherton and P. Elsmore, "Structuring qualitative enquiry in management and organization research: A dialogue on the merits of using software for qualitative data analysis," *Qualitative Research in Organizations and Management: An International Journal*, vol. 2, no. 1, pp. 62–77, 2007.

[56] C. E. Wilson, "Triangulation: the explicit use of multiple methods, measures, and approaches for determining core issues in product development," *Interactions*, vol. 13, no. 6, pp. 46–47, 2006.

[57] "OMG Unified Modeling Language Specification - Version 2.5," OMG, March, 2015. [Online]. Available: http://www.omg.org/spec/UML/2.5/

[58] "OMG Systems Modelling Language Specification version 1.4," OMG, June, 2015. [Online]. Available: http://www.omg.org/spec/SysML/1.4/

[59] S. S. Somé, "Supporting use case based requirements engineering," *Information and Software Technology*, vol. 48, no. 1, pp. 43–58, 2006.

[60] L. Naslavsky, T. A. Alspaugh, D. J. Richardson, and H. Ziv, "Using scenarios to support traceability," in *Proceedings of the 3rd International Workshop on Traceability in emerging forms of Software Engineering*. ACM, 2005, pp. 25–30.

[61] Y. Zhang, J. Zhang, and J. Chen, "Critical success factors in IT service management implementation: People, process, and technology perspectives," in *International Conference on Service Sciences (ICSS)*. IEEE, 2013, pp. 64–68.

[62] A. Khodabandeh and P. Palazzi, "Software development: People, process, technology," CERN ECP Report 95/5 at CERN School of Computing, Sopron, Hungary, Tech. Rep., 1994.

[63] K. Radeka, "The toyota product development system: integrating people, process and technology by james m. morgan and jeffrey k. liker," *Journal of Product Innovation Management*, vol. 24, no. 3, pp. 276–278, 2007.

[64] R. Biloslavo, C. Bagnoli, and R. Rusjan Figelj, "Managing dualities for efficiency and effectiveness of organisations," *Industrial Management & Data Systems*, vol. 113, no. 3, pp. 423–442, 2013.

[65] G. D. Bhatt, "Knowledge management in organizations: examining the interaction between technologies, techniques, and people," *Journal of knowledge management*, vol. 5, no. 1, pp. 68–75, 2001.

[66] M. Korsaa, J. Johansen, T. Schweigert, D. Vohwinkel, R. Messnarz, R. Nevalainen, and M. Biro, "The people aspects in modern process improvement management approaches," *Journal of Software: Evolution and Process*, vol. 25, no. 4, pp. 381–391, 2013.

[67] M. D. Konrad, "Attention to process and people are key to technology adoption," in *Proceedings of the 20th International Computer Software and Applications Conference(COMPSAC)*. IEEE, 1996, p. 436.

[68] "Systems and software engineering – Life cycle processes –Requirements engineering," *ISO/IEC/IEEE 29148:2011(E)*, pp. 1–94, Dec 2011.

[69] E. Hull, K. Jackson, and J. Dick, *Requirements Engineering*. Springer Science & Business Media, 2010.

[70] P. Laplante, *Requirements Engineering for Software and Systems*, 2nd ed., ser. Applied Software Engineering Series. CRC Press, 2013.

[71] U. S. Shah and D. C. Jinwala, "Resolving ambiguities in natural language software requirements: A comprehensive survey," *ACM SIGSOFT Software Engineering Notes*, vol. 40, no. 5, pp. 1–7, 2015.

[72] M. Luisa, F. Mariangela, and N. I. Pierluigi, "Market research for requirements analysis using linguistic tools," *Requirements Engineering*, vol. 9, no. 1, pp. 40–56, 2004.

[73] A. Ferrari, F. dell'Orletta, G. O. Spagnolo, and S. Gnesi, "Measuring and improving the completeness of natural language requirements," in *Requirements Engineering: Foundation for Software Quality*. Springer, 2014, pp. 23–38.

[74] M. Von der Beeck, T. Margaria, and B. Steffen, "A formal requirements engineering method for specification, synthesis, and verification," in *Proceedings of the 8th Conference on Software Engineering Environments*. IEEE, 1997, pp. 131–144.

[75] T. Clancy, "The standish group report chaos, retrieved Apr 10, 2016 from https://www.projectsmart.co.uk/white-papers/chaos-report.pdf," Chaos Report, Tech. Rep., 1995.

[76] G. Sabaliauskaite, A. Loconsole, E. Engström, M. Unterkalmsteiner, B. Regnell, P. Runeson, T. Gorschek, and R. Feldt, "Challenges in aligning requirements engineering and verification in a large-scale industrial context," in *Requirements Engineering: Foundation for Software Quality*. Springer, 2010, pp. 128–142.

[77] C. W. Krueger, "Industry trends in systems and software product line engineering," in *15th International Software Product Line Conference (SPLC)*. IEEE, 2011, pp. 360–360.

[78] L. Brownsword and P. Clements, "A case study in successful product line development," Software Engineering Institute(SEI), Carnegie Mellon University, Tech. Rep. CMU/SEI-96-TR-016, 1996.

[79] J. D. McGregor, D. Muthig, K. Yoshimura, and P. Jensen, "Guest editors' introduction: Successful software product line practices," *IEEE Software*, vol. 27, no. 3, pp. 16–21, 2010.

[80] N. Itzik and I. Reinhartz-Berger, "Generating feature models from requirements: Structural vs. functional perspectives," in *Proceedings of the 18th International Software Product Line Conference*. ACM, 2014, pp. 44–51.

[81] S. Bühne, K. Lauenroth, and K. Pohl, "Modelling requirements variability across product lines," in *Proceedings of the 13th IEEE International Conference on Requirements Engineering*, 2005, pp. 41–50.

[82] S. Kato and N. Yamaguchi, "Variation management for software product lines with cumulative coverage of feature interactions," in *Proceedings of the 15th International Software Product Line Conference (SPLC)*. IEEE, 2011, pp. 140–149.

[83] S. Thiel and A. Hein, "Modeling and using product line variability in automotive systems," *IEEE Software*, vol. 19, no. 4, p. 66, 2002.

[84] J.-M. Davril, E. Delfosse, N. Hariri, M. Acher, J. Cleland-Huang, and P. Heymans, "Feature model extraction from large collections of informal product descriptions," in *Proceedings of the 9th Joint Meeting on Foundations of Software Engineering*. ACM, 2013, pp. 290–300.

[85] N. Niu and S. Easterbrook, "Extracting and modeling product line functional requirements," in *Proceedings of the 16th IEEE International Requirements Engineering(RE) Conference*, 2008, pp. 155–164.

[86] N. Weston, R. Chitchyan, and A. Rashid, "A framework for constructing semantically composable feature models from natural language requirements," in *Proceedings of the 13th International Software Product Line Conference(SPLC)*. Carnegie Mellon University, 2009, pp. 211–220.

[87] C. Tischer, A. Müller, M. Ketterer, and L. Geyer, "Why does it take that long? establishing product lines in the automotive domain," in *Proceedings of the 11th International Software Product Line Conference(SPLC)*. IEEE, 2007, pp. 269–274.

[88] S. Baumgart, X. Zhang, J. Froberg, and S. Punnekkat, "Variability management in product lines of safety critical embedded systems," in *Proceedings of the International Conference on Embedded Systems (ICES)*. IEEE, 2014, pp. 98–103.

[89] I. Hajri, A. Goknil, L. C. Briand, and T. Stephany, "Applying product line sse case modeling in an industrial automotive embedded system: Lessons learned and a refined approach," in *Proceedings of the ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2015, pp. 338–347.

[90] P. Asirelli, M. H. ter Beek, S. Gnesi, and A. Fantechi, "Formal description of variability in product families," in *Proceedings of the 15th International Software Product Line Conference(SPLC)*. IEEE, 2011, pp. 130–139.

[91] Z. A. Barmi, A. H. Ebrahimi, and R. Feldt, "Alignment of requirements specification and testing: A systematic mapping study," in *Proceedings of the 4th IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, 2011, pp. 476–485.

[92] S. M. Ooi, R. Lim, and C. C. Lim, "An integrated system for end-to-end traceability and requirements test coverage," in *Proceedings of the 5th IEEE International Conference on Software Engineering and Service Science (ICSESS)*, 2014, pp. 45–48.

[93] P. Rempel, P. Mader, and T. Kuschke, "An empirical study on project-specific traceability strategies," in *Proceedings of the 21st IEEE International Requirements Engineering (RE) Conference*, 2013, pp. 195–204.

[94] J. Kukkanen, K. Vakevainen, M. Kauppinen, and E. Uusitalo, "Applying a systematic approach to link requirements and testing: a case study," in *Proceedings of the Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2009, pp. 482–488.

[95] J. Thangarajah, G. Jayatilleke, and L. Padgham, "Scenarios for system requirements traceability and testing," in *Proceedings of the 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 1*. International Foundation for Autonomous Agents and Multiagent Systems(IFAAMAS), 2011, pp. 285–292.

[96] L. Bass, J. K. Bergey, P. C. Clements, P. F. Merson, I. Ozkaya, and R. Sangwan, "A comparison of requirements specification methods from a software architecture perspective," Software Engineering Institute(SEI), Carnegie Mellon University, Tech. Rep., 2006.

[97] C. Denger, B. Paech, and B. Freimut, "Achieving high quality of use-case-based requirements," *Informatik-Forschung und Entwicklung*, vol. 20, no. 1-2, pp. 11–23, 2005.

[98] A. G. Sutcliffe, N. A. Maiden, S. Minocha, and D. Manuel, "Supporting scenario-based requirements engineering," *IEEE Transactions on Software Engineering*, vol. 24, no. 12, pp. 1072–1088, 1998.

[99] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[100] J. P. Gibson, "Formal requirements models: simulation, validation and verification," Department of Computer Science, National University of Ireland, Maynooth, Tech. Rep. NUIM-CS-2001-TR-02, 2001.

[101] C. Heitmeyer, J. Kirby, and B. Labaw, "Tools for formal specification, verification, and validation of requirements," in *Proceedings of the 12th Annual Conference on Computer Assurance (COMPASS)*. IEEE, 1997, pp. 35–47.

[102] A. Hall, "Seven myths of formal methods," *IEEE Software*, vol. 7, no. 5, pp. 11–19, 1990.

[103] C. Nebut, F. Fleurey, Y. Le Traon, and J.-M. Jezequel, "Automatic test generation: A use case driven approach," *IEEE Transactions on Software Engineering*, vol. 32, no. 3, pp. 140–155, 2006.

[104] K. Han, J. Youn, and J. Cho, "A functional requirements traceability management methodology for model-based testing framework of automotive embedded system," in *Proceedings of the 3rd International Conference on Advances in Vehicular Systems, Technologies and Applications (IARIA)*, 2014, pp. 46–51.

[105] D. Arnold, J.-P. Corriveau, and W. Shi, "Modeling and validating requirements using executable cotnracts and scenarios," in *Proceedings of the 8th ACIS International Conference on Software Engineering Research, Management and Applications (SERA)*. IEEE, 2010, pp. 311–320.

[106] A. Pretschner, W. Prenninger, S. Wagner, C. Kühnel, M. Baumgartner, B. Sostawa, R. Zölch, and T. Stauner, "One evaluation of model-based testing and its automation," in *Proceedings of the 27th International Conference on Software Engineering*. ACM, 2005, pp. 392–401.

[107] R. Tommy, N. Prasannakumaran, K. Sumithra, and S. Kesav, "Test scenario modeling: Modeling test scenarios diagrammatically using specification based testing techniques," in *Proceedings of the Third International Conference on Computer, Communication, Control and Information Technology (C3IT)*. IEEE, 2015, pp. 1–7.

[108] W.-T. Tsai, X. Bai, R. Paul, and L. Yu, "Scenario-based functional regression testing," in *Proceedings of the 25th Annual International Computer Software and Applications Conference(COMPSAC)*. IEEE, 2001, pp. 496–501.

[109] S. Siegl, K.-S. Hielscher, R. German, and C. Berger, "Formal specification and systematic model-driven testing of embedded automotive systems," in *Proceedings of the Europe Conference & Exhibition on Design, Automation & Test(DATE)*. IEEE, 2011, pp. 1–6.

[110] M. Broy, S. Chakraborty, S. Ramesh, M. Satpathy, S. Resmerita, and W. Pree, "Cross-layer analysis, testing and verification of automotive control software," in *Proceedings of the International Conference on Embedded Software (EMSOFT)*. IEEE, 2011, pp. 263–272.

[111] R. Marinescu, M. Saadatmand, A. Bucaioni, C. Seceleanu, and P. Pettersson, "A model-based testing framework for automotive embedded systems," in *Proceedings of the 40th EUROMICRO Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2014, pp. 38–47.

[112] H. Post, C. Sinz, F. Merz, T. Gorges, and T. Kropf, "Linking functional requirements and software verification," in *Proceedings of the 17th IEEE International Requirements Engineering(RE) Conference*, 2009, pp. 295–302.

[113] M. Conrad, I. Fey, and S. Sadeghipour, "Systematic model-based testing of embedded automotive software," *Electronic Notes in Theoretical Computer Science*, vol. 111, pp. 13–26, 2005.

[114] K. Lamberg, M. Beine, M. Eschmann, R. Otterbach, M. Conrad, and I. Fey, "Model-based testing of embedded automotive software using Mtest," in *SAE World Congress*. Citeseer, 2004, pp. 8–11.

[115] H. Shokry and M. Hinchey, "Model-based verification of embedded software." *IEEE Computer*, vol. 42, no. 4, pp. 53–59, 2009.

[116] T. Bauer, F. Bohr, D. Landmann, T. Beletski, R. Eschbach, and J. Poore, "From requirements to statistical testing of embedded systems," in *Proceedings of the 4th International Workshop on Software Engineering for Automotive Systems*. IEEE Computer Society, 2007, p. 3.

[117] E. Bringmann and A. Krämer, "Systematic testing of the continuous behavior of automotive systems," in *Proceedings of the International Workshop on Software engineering for Automotive Systems*. ACM, 2006, pp. 13–20.

[118] A. M. Perez and S. Kaiser, "Integrating test levels for embedded systems," in *Proceedings of the Testing: Academic and Industrial Conference- Practice and Research Techniques(TAIC PART)*. IEEE, Sept 2009, pp. 184–193.

[119] C. Pfaller, A. Fleischmann, J. Hartmann, M. Rappl, S. Rittmann, and D. Wild, "On the integration of design and test: a model-based approach for embedded systems," in *Proceedings of the International Workshop on Automation of Software Test*. ACM, 2006, pp. 15–21.

[120] R. Feldt and A. Magazinius, "Validity threats in empirical software engineering research-an initial survey." in *Proceedings of the Conference on Software Engineering and Knowledge Engineering(SEKE)*, 2010, pp. 374–379.

[121] R. Valerdi and H. L. Davidz, "Empirical research in systems engineering: challenges and opportunities of a new frontier," *Systems Engineering*, vol. 12, no. 2, pp. 169–181, 2009.

[122] B. Kitchenham, "Procedures for performing systematic reviews," *Keele, UK, Keele University*, vol. 33, no. 2004, pp. 1–26, 2004.

Appendices

Appendix A

Interview Questionnaires

Interview questions for SYSTEM TEST ENGINEERS

1. As a system tester what are your responsibilities towards the system and the software modules of the function that the system contains? What kind of *experience* do you have in this role?
2. At the system level, what is your test focus and what are the primary test objectives?
3. What are the broad steps of testing (test approach) that you undertake to meet those objectives? Which test environment do you test in?
4. In this approach, how do you handle variants (function variants, system variants, software variants, hardware variants)?
5. Are these test steps same even when addressing change requests? If not what testing approach is used to handle change requests?
6. Now, as part of the testing activity, do you test against any form of requirements of the system or each software modules of the function? If not, how do you keep track of how much has been tested at the level you are at (system level)? If yes, what kind of requirements?
7. If yes, is there traceability between the requirements you test against and the test artefacts?
8. How are *test cases generated*? Manual vs automated.
9. Are source documents or models used to generate test cases (test case generation based on models/ based on requirements documents/ based on tester's experience with the system and understanding of its functionality or other)?
10. So the system software that you test contains modules that together with modules from other systems help in the execution of entire functions which is tested in integration testing. Is there any *traceability between the requirements at the lower level and the higher level (system level implementation specific requirements and function level requirements)*? If yes, to what extent and how are they maintained? If not, has this ever been discussed? Do you consider it to be important? Have there been any issues that have been faced due to lack of such a traceability (directly like in change management or indirectly)?
11. For the testing you perform as a system tester, is there any *test coverage* measurement that you use to identify how much of the system has been tested?
12. Since the effort to test a function is distributed across your level, function level and the integration level (each testing the function with a different focus) is there some means of combining all the results to establish how much of the entire function has been tested across the test levels? If not, does that indicate a possibility of test redundancies or test gaps, or is there some way you work around it to avoid that?
13. What *test artefacts* are generated as a result of the testing you perform (Test cases, test reports, test results, etc.)?
14. Where are these test artefacts stored and how are they maintained over time (Excel, perforce, internal document folders)?
15. They say there is always scope for improvement, in that case, what would you say is *lacking/problematic/a hindrance* in the current test approach for testing the functions?
16. How do you think the current test approach can be improved to address this problem?

Figure A.1: Interview questionnaire for system test engineers

Interview questions for FUNCTION OWNERS

1. As a function owner what are your responsibilities towards the software function? What kind of *experience* do you have in this role?
2. As defined by the Scania V model, one of the responsibilities of the function owner is to perform a part of the integration testing. So, from the perspecting of testing the function, what is your test focus and what are the primary test objectives?
3. What are the broad steps of testing (test approach) that you undertake to meet those objectives? Which test environment do you test in?
4. In this approach, how do you handle variants (function variants, system variants, software variants, hardware variants)?
5. Are these test steps same even when addressing change requests? If not what testing approach is used to handle change requests?
6. Now, as part of the testing activity, do you test against any form of requirements of the function? If not, how do you keep track of how much has been tested at the level you are at (function test level)? If yes, what kind of requirements?
7. If yes, is there traceability between the requirements you test against and the test artefacts?
8. How are *test cases generated* to test the function? Manual vs automated.
9. Are source documents or models used to generate test cases (test case generation based on models/ based on requirements documents/ based on tester's experience with the system and understanding of its functionality or other)?
10. So the function you are responsible for is spread across multiple ECU systems and hence is tested as separate modules at the lower system level. Is there any *traceability between the requirements at the lower level and the higher level (system level implementation specific requirements and function level requirements)*? If yes, to what extent and how are they maintained? If not, has this ever been discussed? Do you consider it to be important? Have there been any issues that have been faced due to lack of such a traceability (directly like in change management or indirectly)?
11. For the testing you perform as a function owner, is there any *function test coverage* measurement that you use to identify how much has been tested?
12. Since the effort to test a function is distributed across the system test level, your level and the integration level (each testing the function with a different focus), is there some means of combining all the results to establish how much of the entire function has been tested across the test levels? If not, does that indicate a possibility of test redundancies or test gaps, or is there some way you work around it to avoid that?
13. What *test artefacts* are generated as a result of the testing you perform (Test cases, test reports, test results, etc.)?
14. Where are these test artefacts stored and how are they maintained over time (Excel, perforce, internal document folders)?
15. They say there is always scope for improvement, in that case, what would you say is *lacking/problematic/a hindrance* in the current test approach for testing the functions?
16. How do you think the current test approach can be improved to address this problem?

Figure A.2: Interview questionnaire for function owners

Interview questions for INTEGRATION TEST ENGINEERS

1. As an integration tester what are your responsibilities towards the functions the vehicle contains? What kind of *experience* do you have in this role?
2. At the integration level, what is your test focus and what are the primary test objectives?
3. What are the broad steps of testing (test approach) that you undertake to meet those objectives? Which test environment do you test in?
4. In this approach, how do you handle variants (function variants, system variants, software variants, hardware variants)?
5. Are these test steps same even when addressing change requests? If not what testing approach is used to handle change requests?
6. Now, as part of the testing activity, do you test against any form of requirements of the functions? If not, how do you keep track of how much has been tested at the level you are at (integration level)? If yes, what kind of requirements?
7. If yes, is there traceability between the requirements you test against and the test artefacts?
8. How are *test cases generated*? Manual vs automated.
9. Are source documents or models used to generate test cases (test case generation based on models/ based on requirements documents/ based on tester's experience with the system and understanding of its functionality or other)?
10. So the functions you test are spread across multiple ECU systems and hence is tested as separate modules at the lower system level. Is there any *traceability between the requirements at the lower level and the higher level (system level implementation specific requirements and function level requirements)*? If yes, to what extent and how are they maintained? If not, has this ever been discussed? Do you consider it to be important? Have there been any issues that have been faced due to lack of such a traceability (directly like in change management or indirectly)?
11. For the testing you perform as an integration tester, is there any function *test coverage* measurement that you use to identify how much of the function has been tested?
12. Since the effort to test a function is distributed across the system level, function level and your level (each testing the function with a different focus) is there some means of combining all the results to establish how much of the entire function has been tested across the test levels? If not, does that indicate a possibility of test redundancies or test gaps, or is there some way you work around it to avoid that?
13. What *test artefacts* are generated as a result of the testing you perform (Test cases, test reports, test results, etc.)?
14. Where are these test artefacts stored and how are they maintained over time (Excel, perforce, internal document folders)?
15. They say there is always scope for improvement, in that case, what would you say is *lacking/problematic/a hindrance* in the current test approach for testing the functions?
16. How do you think the current test approach can be improved to address this problem?

Figure A.3: Interview questionnaire for integration test engineers

Appendix B

Interview Invitation

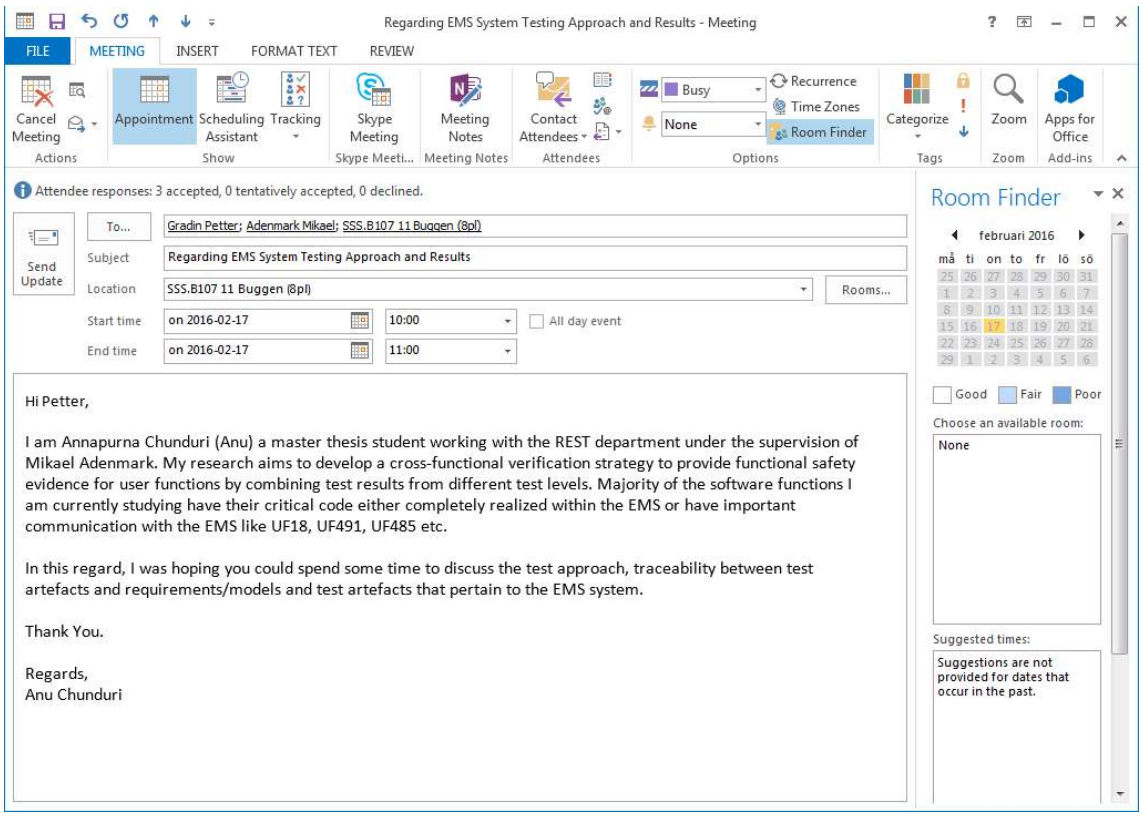


Figure B.1: Snapshot of an interview invitation sent via Microsoft Outlook

Appendix C

Frequency of Occurrence of Open Code Categories

Open Code Category	**No.of interviews where issue was identified (x)**	**Total no.of interviews (y)**	**Frequency of occurrence x/y*100)**
Traceability Issues	13	13	100
Requirements Specification Issues	12	13	92.31
Explicit Test Issues	10	13	76.92
Variant Handling Issues	10	13	76.92
Tool Support Issues	9	13	69.23
Knowledge Transfer Issues	7	13	53.85
People Knowledge Issues	5	13	38.46
Multiple Data Sources Issues	4	13	30.77

Table C.1: Frequency of occurrence of open code categories in the interviews conducted at the case company

Appendix D

UCDs for UC18_1 and UC18_2

UC18_2 Use Case Description	
Title:	***Detect, Activate and Deactivate Low Fuel Level Warning in Trucks***
Primary ECUs:	ME2 ME1 EMS/EMD ICL→ Liquid Engine Trucks ME2 ME1 EMS/EMD ICL → Gas Engine Trucks
Variants:	*a. Detect, Activate and Deactivate low fuel level warning in trucks with liquid engine* *b. Detect, Activate and Deactivate low fuel level warning in trucks with gas engine*
Goal:	*This use case should be present on all trucks and should detect, activate and deactivate the low fuel level warning irrespective of whether the truck has a liquid or gas engine.*
Pre-Conditions:	*Engine is turned ON.* *lowFuelLevelWarningParam is set to Enabled.*
Post-Conditions:	*Low fuel level warning is detected and displayed on the ICL when activation conditions are met.* *The warning is deactivated when deactivation conditions are met.* *In case of an error, any indications of the low fuel level warning in the ICL are removed.*
Steps:	1. Driver turns the ignition ON and *lowFuelLevelWarning signal is set to Deactivated at startup.* 2. Calculation of the output total fuel level signal for truck (UC18_1 Description). 3. An algorithm within ME2 receives valid total fuel level values through internal signal totalFuelLevel (Variable) and monitors them for low fuel level conditions. Begin Variant Point 1: Liquid Engine 4. ME2 detects low fuel level condition if fuel level drops below a predefined percent of total liquid fuel volume and activates low fuel level warning by setting signal LowLevelFuelWarning(Activated) and transmits it to ME1. End Variant Point 1: Liquid Engine Begin Variant Point 2: Gas Engine 5. ME2 detects low fuel level condition if fuel level drops below a predefined value and activates low fuel level warning by setting signal LowLevelFuelWarning(Activated) and transmits it to ME1. End Variant Point 2: Gas Engine 6. The ME1 sends low fuel level warning signal to the ICL. 7. The ICL receives low fuel level warning activated signal and displays this warning as a telltale. 8. Based on vehicle status received by ME2, another algorithm to check if refuel is taking place is activated. 9. If algorithm detects increase in fuel volume above a predefined value or ECU system shutdown, the low fuel level warning signal is deactivated. 10. End
Alternatives:	3n. If the value of total fuel level signal received by the algorithm has 'Error' or 'Not Available'. 1. Set low fuel level warning signal accordingly and transmit to ME1. 2. Move to step 6. 7n. ICL receives low fuel level warning signal as 'Not Available'. 1. Deactivate any low fuel level warning telltale. 2. Move to step 10.

Figure D.1: UCD for UC18_2

UC18_1 Use Case Description	
Title:	***Detect and Display Fuel level in Trucks***
Primary ECUs:	ME2 EMS/EMD ME1 ICL → Liquid Engine Trucks ME2 EMS/EMO GSC ME1 ICL → Gas Engine Trucks
Variants:	*a. Detect and Display fuel level in trucks with liquid engine* *b. Detect and Display fuel level in trucks with gas engine*
Goal:	*This use case should be present on all trucks and should detect and display the fuel level irrespective of whether the truck has a liquid or gas engine.*
Pre-Conditions:	*Engine is turned ON.*
Post-Conditions:	*Fuel level is detected and displayed on the ICL within the initial predefined time from when the pre-condition is satisfied. It then continues to display the accurate current fuel level on the ICL as long as the pre-condition exists.* *If fuel sensor has an error in detecting fuel for more than a predefined time, fuel level is set accordingly and a DTC is raised.*
Steps:	1. Driver turns the ignition ON. *Begin Variant Point 1 : Liquid Engine* 2. Fuel sensor detects fuel height and transmits it to the ME2 through DW.fuelLevelSensor (Variable). 3. Parallel to step 2, ME1 receives fuel rate and vehicle status from EMD through signals FuelRate (Variable) and VehicleStatus (State) and transmits them to ME2. 4. ME2 receives valid signal values for the fuel height, fuel rate and vehicle status. 5. ME2 filters the fuel height value avoid rapid changes caused by movement of the fuel in the tank. 6. Based on the inputs received, ME2 uses the appropriate variation of the algorithm to estimate a valid total fuel level signal as totalFuelLevel (Variable) and move to step 12. *End Variant Point 1 : Liquid Engine* *Begin Variant Point 2 : Gas Engine* 7. Fuel sensor detects fuel level/fuel pressure in fuel tank for gas engine and transmits it to the GSC through DW.HighPressureSensor(Variable)/DW.TankLevelSensor(Variable). 8. GSC receives valid fuel level or fuel pressure value, filters it and estimates a valid gas fuel level value. 9. GSC transmits the estimated gas fuel level to EMO using an internal CAN FuelLevel(Variable). 10. EMO transmits the gas fuel level signal to the ME1 using the signal FuelLevel (Variable). 11. ME1 sends this valid gas fuel level signal to the ME2 where it is set to the output total fuel level signal. *End Variant Point 2 : Gas Engine* 12. ME2 transmits the total fuel level signal to ME1. 13. ME1 sends this total fuel level signal to ICL. 14. The ICL receives a valid estimated fuel level value and rounds it off based on the conditions set and the fuel level is displayed according to the corresponding segment behavior table. 15. Back to step 2 till ignition is turned OFF. When ignition/power supply is OFF continue to next step. 16. Store last estimated value for total fuel level signal in the ME2. 17. End
Alternatives:	4n$_1$. Fuel sensor's fuel height signal has 'Error' or 'Not Available' and fuel rate and vehicle status signals are either valid or invalid. 1. If fuel rate value has 'Error' or 'Not Available' set the fuel rate signal accordingly. If vehicle status has 'Error' or 'Not available' set the vehicle status signal accordingly. 2. For an initial predefined time till it moves to a valid input, the fuel height signal value is kept at a constant and move to step 6p. 3. After the predefined time, if the value remains 'Error' or 'Not Available' then raise a DTC, skip filtering and move to step 6n. 4n$_2$. Signal for Fuel rate or Vehicle Status has 'Error' or 'Not Available' and fuel height signal is valid. 1. If fuel rate value has 'Error' or 'Not Available' set the fuel rate signal accordingly, If vehicle status has 'Error' or 'Not available' set the vehicle status signal accordingly and continue to the next step. 6n. Fault estimating total fuel level signal value due to 'Error' or 'Not Available' set for input fuel height signal for more than predefined time. 1. Set total fuel level signal accordingly and continue to step 10. 8n. The fuel level/fuel pressure signal received by the GSC has 'Error' or 'Not Available'. 1. Set value of estimated fuel level accordingly and continue to next step. 11n. The gas fuel level signal received by ME2 has 'Error' or 'Not Available'. 1. Set total fuel level signal accordingly and continue to next step. 14n. ICL receives estimated total fuel level signal value as 'Error' or 'Not Available' 1. Set total fuel level accordingly and continue to next step.

Figure D.2: UCD for UC18_1

Appendix E

UCTs for UC18_1 and UC18_2

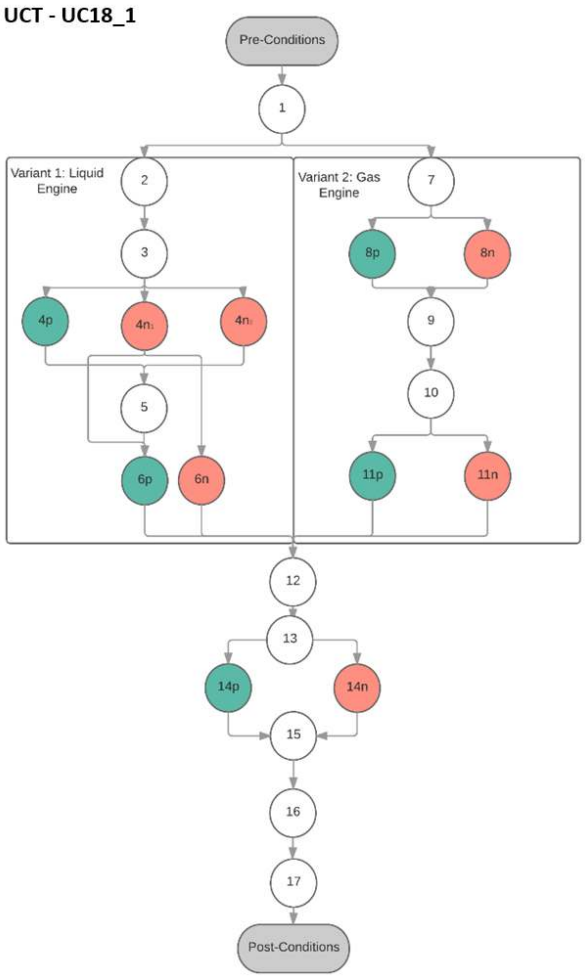


Figure E.1: UCT for UC18_1

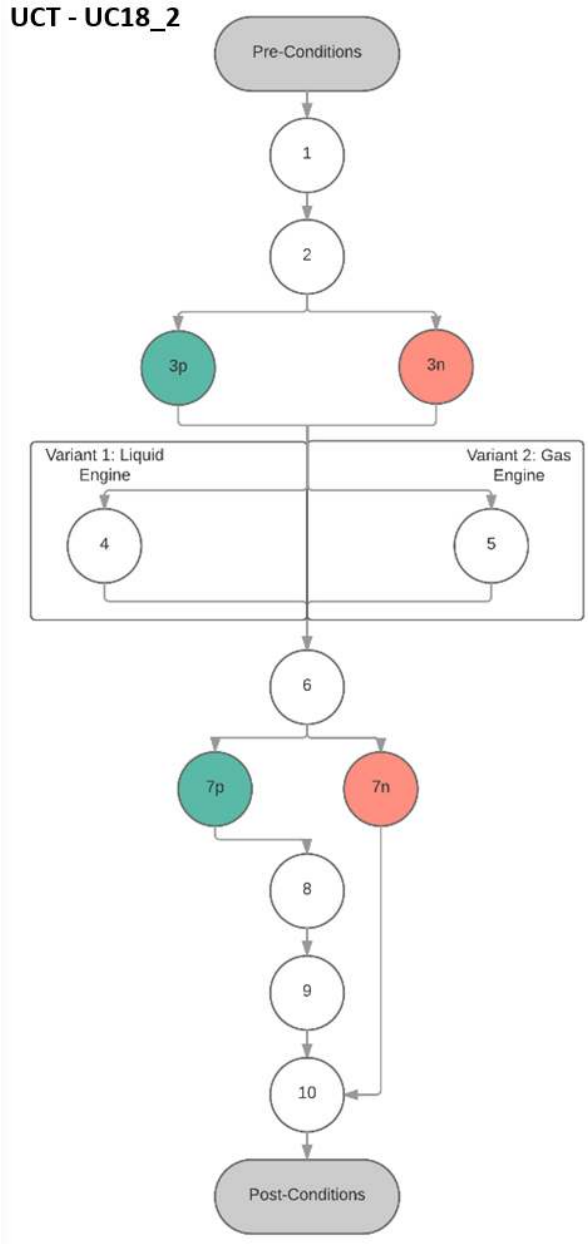


Figure E.2: UCT for UC18_2

Appendix F

Comprehensive Test Coverage Information

Use Case Variant	Function SCN	TW1602		
		System Test Level	Function Test Level	Integration Test Level
Liquid	SCN1	61.11%	100%	100%
	SCN2	57.89%	Not Tested	Not Tested
	SCN3	50%	Not Tested	Not Tested
	SCN5	52.94%	100%	100%
	SCN6	40%	Not Tested	Not Tested
Gas	SCN1	53.33%	Not Tested	100%
	SCN4	26.67%	Not Tested	Not Tested
	SCN5	43.75%	Not Tested	Not Tested
	SCN6	40%	Not Tested	Not Tested

Figure F.1: Comprehensive FLD function scenario test coverage across the three test levels for test round TW1602

Use Case Variant	Function SCN	TW1610		
		System Test Level	Function Test Level	Integration Test Level
Liquid	SCN1	72.22%	100%	100%
	SCN2	73.68%	Not Tested	Not Tested
	SCN3	64.28%	Not Tested	Not Tested
	SCN5	52.94%	100%	100%
	SCN6	50%	Not Tested	Not Tested
Gas	SCN1	40%	Not Tested	100%
	SCN4	46.67%	Not Tested	Not Tested
	SCN5	50%	Not Tested	Not Tested
	SCN6	50%	Not Tested	Not Tested

Figure F.2: Comprehensive FLD function scenario test coverage across the three test levels for test round TW1610