CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Leveraging AI for Automated Requirement to Test Alignment in Automotive Embedded Systems

A Design Science Research in the Automotive Industry

Master's Thesis in Computer science and engineering

Filip Sjölander, Jonathan Svantesson

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's Thesis 2026

Leveraging AI for Automated Requirement to Test Alignment in Automotive Embedded Systems

A Design Science Research in the Automotive Industry

Filip Sjölander, Jonathan Svantesson

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Leveraging AI for Automated Requirement to Test Alignment in Automotive Embedded Systems
A Design Science Research in the Automotive Industry
Filip Sjölander, Jonathan Svantesson

Supervisor: Linda Erlenhov, Department of Computer Science and Engineering
Industrial Supervisor: Gongpei Cui, Volvo Cars
Examiner: Hans-Martin Heyn, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Filip Sjölander, Jonathan Svantesson
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

The rapid increase in the complexity of automotive software has made the manual verification of software traceability labour-intensive. Consequently, practitioners can encounter "alignment gaps", where a formal traceability link exists between a requirement and a test case, but the test case fails to exercise the requirement's intended functionality. To address this challenge, this thesis develops a diagnostic artificial intelligence (AI) artefact. The artefact is aimed to automate the assessment of requirement-to-test alignment within the automotive embedded systems domain.

The study employs a three-cycle Design Science Research methodology. In the first cycle, a literature review and practitioner interviews were conducted to identify design attributes for the proposed artefact. Four design attributes were determined: semantic similarity, contextual information, structural consistency, and hierarchical dependencies. During the second cycle, a system was developed utilizing Large Language Models (LLMs) integrated with a Retrieval-Augmented Generation (RAG) architecture.

In the final cycle, the artefact was evaluated through a mutant injection analysis to measure its fault detection capabilities, alongside a case study involving industry professionals to assess its practical utility. The mutant injection analysis demonstrated that the RAG architecture improved its analytical consistency and its ability to detect injected logical defects. Furthermore, the case study, which utilized the NASA-TLX questionnaire and post-task interviews, indicated that the artefact reduced the general workload associated with evaluating test case alignment. However, due to the small sample size and sampling limitations, these findings should be interpreted as suggestive. Ultimately, this thesis concludes that combining RAG architectures with LLMs represents a promising proof-of-concept for transparent decision support in maintaining requirement-to-test alignment in safety-critical automotive environments.

Keywords: Requirements Engineering, Software Testing, Alignment, RAG, LLMs, Automotive, Traceability

Acknowledgements

First and foremost, we would like to thank our supervisor at Chalmers, Linda Erlenhov, for her continuous support, valuable feedback, and guidance throughout this thesis. Linda's contribution has been vital to its success, and this work could not have been completed without her. Additionally, we would like to thank our examiner, Hans-Martin Heyn, whose input helped shape our research and guided us in the right direction. With his feedback and knowledge about the domain, we were able to refine our ideas.

Furthermore, we would like to thank Volvo Cars for the opportunity to conduct our master's thesis there. Their expertise and knowledge were essential to this thesis. We would also like to extend our sincere gratitude to our supervisor at Volvo Cars, Gongpei Cui, for his unwavering support and extensive knowledge of Volvo's systems, way of working and automotive testing.

Filip Sjölander, Jonathan Svantesson, Gothenburg, June 2026

Contents

List of Figures

List of Tables

1

Introduction

Software traceability, defined as the establishment and maintenance of trace links between artefacts (e.g., requirements, test cases, and source code), is essential for maintaining consistency across software development lifecycles and is recognized as a prerequisite for high-quality software production [1], [2], [3]. Its key importance lies in the ability to bidirectionally trace software artefacts and verify that initial project goals remain aligned throughout the software development lifecycle, supporting critical processes such as impact analysis and change management [3], [4]. In practice, however, establishing and maintaining traceability links is labour-intensive and the process remains largely manual and ad-hoc [1], [5]. Thus, manual link creation introduces risks of incomplete or inconsistently maintained traceability, which can harm the overall quality of a software system [1].

Furthermore, in a study by Ruiz et al. [5], a software-practitioner perspective is taken, showing that the importance of traceability is not always clear and that challenges in implementing it are present. Moreover, Fucci et al. [6] also bring up challenges in the industrial implementation of traceability and shed light on problems when traces are absent, pointing to consequences such as insufficient resource allocation and complex defect management. These challenges have justified research in automating traceability, with the goal of reducing manual effort and improving quality.

Current approaches to automated traceability typically leverage information retrieval (IR), machine learning (ML), or deep learning (DL) architectures [7], [8]. However, they are often connected to classifying links between artefacts (e.g., requirements to code) and not specifically assessing the alignment of existing links, which suggests a less explored area [9], [10]. Additionally, labelled traceability data of links in industrial contexts tends to be sparse, making it even harder for existing approaches to be used in places beyond research settings [11]. This thesis, however, bypasses this data barrier by leveraging access to an established industrial repository. This unique foundation is used to investigate the potential of generative AI as a novel solution for assessing the alignment between requirements and test cases within the automotive industry.

1.1 Problem Description

The automotive industry operates under rigorous engineering standards, where software malfunctions can result in significant liability and operational risk [10], [12]. To manage these risks, industry standards like ISO 26262 [13] prescribe the use of the V-model lifecycle. This framework emphasizes traceability between requirements and verification artefact, ensuring that safety-critical functions are systematically tested and validated [14]. In practice, this structured approach is designed to verify that the system meets its intended reliability targets.

However, the rapid evolution of vehicles from purely mechanical machines into complex, distributed computing systems challenges this theoretical reliability. A modern vehicle relies on 70 to 150 Electronic Control Units (ECUs) [15], [16] and operates on over 100 million lines of code [15], [17]. As the volume and complexity of automotive software scales, the traditional, manual approach to Verification and Validation (V&V) [18] has become a bottleneck [12].

This bottleneck is particularly evident in the management of software traceability. A major limitation in current V&V processes is the difference between simply having a traceability link and actually trusting its quality [10]. Because it is practically impossible to manually check the correctness of tens of thousands of links, "alignment gaps" can emerge. In these cases, specific parts of a requirement remain unverified even though a trace link exists in the system. Since manual analysis is still the primary way to find these hidden gaps, the massive scale of modern automotive software makes this traditional approach difficult to scale efficiently, and may affect confidence in verification completeness[10].

1.2 Purpose and Scope of the Study

The primary objective of this research is to develop a diagnostic artefact designed to evaluate the relationships between requirements and test cases within the automotive embedded systems domain. While traditional software traceability focuses on the mere existence of a bidirectional connection between artefacts, this study focuses specifically on the alignment of test cases and requirements. In this context, alignment is defined as the extent of whether an associated test case adequately exercises the functional intent of the requirement to which it is linked. By moving beyond a binary "linked or not status", this research addresses the industry challenge of alignment gaps, where specific aspects of a requirement remain unverified despite the existence of a formal traceability link.

The scope of this study is strictly limited to the assessment and gap identification of existing test cases and requirements and does not encompass the automated generation of new requirements or test scripts. Furthermore, this study does not attempt to provide an overarching assessment of the inherent quality of test cases and requirements. Instead, it evaluates the alignment between specified requirements and the execution logic of their linked test cases, flagging potential anomalies for human

review. The resulting artefact is designed as a decision-support tool that provides a transparent evidence chain for its findings, rather than presenting definitive, automated quality judgments. By identifying these alignment gaps, the system aims to reduce the workload and manual effort required for verification within safety-critical environments.

The findings presented in this thesis reflect general challenges commonly encountered in large-scale automotive software engineering and should not be interpreted as deficiencies specific to the industrial partner.

1.3 Research Questions

RQ1: What attributes should serve as foundational design criteria for an artefact assessing alignment between requirements and their associated test cases in the automotive domain?

This question focuses on identifying the critical characteristics and features necessary to construct an effective diagnostic tool. Establishing evidence-based design criteria is crucial. Without a deep understanding of what industry practitioners actually need and what the literature dictates, any resulting artefact risks being misaligned with safety-critical engineering workflows. This question is addressed by retrieving data from a literature review and a thematic analysis of semi-structured practitioner interviews.

RQ2: To what extent can the proposed AI-based artefact support automated assessment of requirement-to-test alignment?

This question evaluates the reliability, correctness, and technical boundaries of the developed artefact in identifying hidden coverage gaps. This is addressed through an ablation study and mutant injection analyses. Specifically, these analyses explore how integrating different combinations of retrieved context impacts the model's performance at assessing the alignment, allowing for the identification of the optimal artefact setup.

RQ3: How does the implementation of the proposed artefact reduce the workload required by practitioners to identify alignment gaps?

The final question focuses on the practical human utility. High accuracy is fundamentally insufficient if the system's output is too difficult to interpret or fails to improve the engineer's daily workflow. Therefore, it is important to quantify the artefact's actual impact on the user. To address this, a Case Study was conducted, utilizing the standardized NASA-TLX questionnaire alongside post-task interviews to empirically contrast traditional manual reviews against the tool-supported assessment. Additionally, an injected mutant analysis was conducted, simulating common faults in test case creation and how well the artefact detects them.

1.4 Significance of the Study

By designing a decision-support diagnostic artefact tailored for the automotive embedded domain, this research addresses a critical industry challenge: the manual and error-prone process of verifying test sufficiency. The study therefore contributes to the relatively unexplored field of alignment assessment, moving beyond binary traceability to ensure that safety-critical software artefacts are not just linked, but functionally consistent.

This approach supports the rigorous demands of automotive safety standards while reducing the operational overhead associated with manual reviews. Also, this research highlights a domain that we believe will inevitably grow in importance, as code becomes more and more AI-generated rather than human-written, therefore, requiring more reliability and safety.

2

Background

2.1 Traceability

Traceability in software engineering is defined as the ability to track and manage the lifecycle of a requirement across all phases of development [19], [20]. By establishing connections between heterogeneous artefacts, traceability maps the complex relationships across the development lifecycle [21].

The foundational elements of traceability systems are trace artefacts and trace links. Trace artefacts represent the distinct nodes of information, varying in granularity from an overarching specification document down to a discrete element, such as a single requirement, a UML class, or a segment of source code. A trace link is the formal association established between a source artefact and a target artefact. Although these links typically possess a primary logical direction, they operate bidirectionally in practice, enabling engineers to perform both forward and backward relational traversal through the system's architecture [21].

The implementation of robust traceability yields significant benefits for both compliance and general software quality assurance [22]. Within safety-critical domains such as the automotive industry, it is a strict regulatory mandate under standards like ISO 26262 [13]. Beyond compliance, traceability empowers project managers to actively monitor progress by quantifying the implementation status of individual requirements [23].

2.1.1 Traceability Link Recovery

Traceability Link Recovery (TLR) is defined as the automated or semi-automated process of identifying and establishing trace links between software artefacts to reduce the burden of manual maintenance [20]. The primary challenge in TLR is the "knowledge gap", which refers to the semantic and syntactic differences between high-level artefacts (e.g., requirements written in natural language) and low-level artefacts (e.g., source code written in programming languages).

The most established methods to automate the process of traceability link recovery are IR-based methods [7], which treat software artefacts as text documents and utilize statistical models to determine textual similarity. Common models include

the Vector Space Model (VSM) [24], Latent Semantic Indexing (LSI) [25], and Latent Dirichlet Allocation (LDA) [26].

ML approaches represent the second major methodology, typically treating link recovery as a binary classification problem [7]. These methods extract features from various perspectives of the artefacts and employ classifiers, which have consistently demonstrated strong performance across diverse datasets to predict whether a link exists.

More recently, the field has moved toward DL and Large Language Models (LLMs) to leverage representation learning [7], [8]. These techniques, including Word2Vec, Doc2Vec, and transformer-based models like BERT or GPT [27], attempt to capture complex semantic context and logical reasoning to identify dependencies.

2.2 Artificial Intelligence

The term Artificial Intelligence (AI) was first coined in 1956 [28] by emeritus Stanford Professor John McCarthy, who defined it as "the science and engineering of making intelligent machines" [29].

2.2.1 Transformer Models

One of the most prominent AI-architectures is Transformers model. Transformers excel at tasks where sequential order matters, such as natural language processing (NLP) and time series analysis. Historically, such problems were addressed by recurrent neural networks (RNNs). These networks process sequences one element at a time, while utilising hidden states that capture information from previous steps, when processing subsequent elements. However, this sequential processing presents a limitation. It prevents parallelization across time steps within individual sequences. This restriction means that even with large computational resources, RNNs cannot fully leverage modern hardware when processing long sequences, as each time step must wait for the previous one to complete. Transformers address this bottleneck through their self-attention mechanism, which computes relationships between all sequence positions simultaneously, enabling fully parallel processing of the entire sequence. [30]

Furthermore, the majority of neural sequence transduction models use an encoder-decoder architecture, as illustrated in Figure 2.1. As the name suggests, this structure has two key parts, beginning with the encoder [31]. The encoder consists of n identical layers, with each having two sublayers of a multi-head self-attention layer and a position-wise fully connected feed-forward network. Each of these is finalised with a layer normalization, together with residual connections (short cut links that improves gradient flow). Additionally, positional encodings (Adding information of token position) are inserted in the input embeddings. The decoder is built in similar fashion, with two multi-head self-attention layer, but one being masked, and a position-wise fully connected feed-forward network. It also has a positional

encodings injected into the input of the masked self-attention layer, however, the input here is the target sequence, but shifted. The second self-attention layer in the decoder, which take the output of the encoder as input, replicates the encoder mechanism. Both of these sublayers are finalised with layer normalization, and residual connections around the sublayers. Lastly, after the decoder a normal linear layer is added followed up with a softmax activation function to give a probability of upcoming token [30].

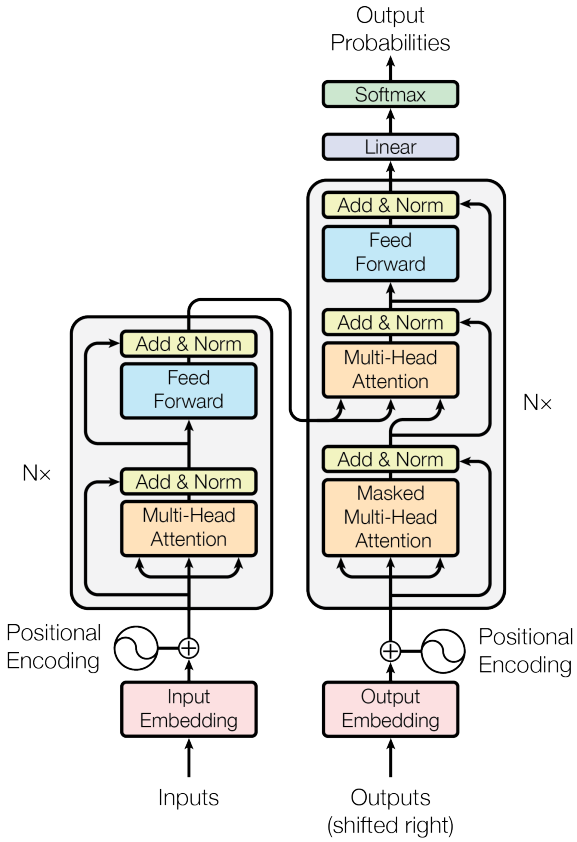

Figure 2.1: The Transformer model architecture [30]. Reproduced with permission from Google.

The attention function used in these models can vary; however, the function described in the original paper is a scaled dot-product attention, formally defined by Vaswani et al. [30] as shown in equation 2.1, where d_k denotes the dimensionality of the key vectors. Moreover, the fundamentals behind the attention function is based on a token to acquire information about the relevance of other tokens in the sequence relative to itself. This is achieved through three components: queries (Q), keys (K), and values (V), which are derived through learned linear transformations of the input embeddings. The query corresponds to the token of interest, the keys correspond to all tokens in the sequence for comparison, and the values contain the information to be aggregated based on the computed attention weights.

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

With the help of several heads (h), the computation is done simultaneously, where different weight matrices are used for different heads, capturing a variety of patterns. The computed attention from every head is then concatenated and multiplied with a shared output weight matrix (W^O), resulting in the output of the multi-head self-attention layer. However, in the masked multi-head self-attention layer, a modification is done. This modification is necessary to prevent each position from attending to subsequent positions. Thus, avoiding leakage of ground-truth information from future tokens and is accomplished by assigning a value of $-\infty$ to the attention logits corresponding to future positions, which the softmax function then effectively converts to zero.

2.2.2 Large Language Model

A Large Language Model (LLM) is a type of AI-model that understands, generates and interacts with natural language [32], [33]. The evolution of these models represent a major shift in NLP, driven by the combination of transfer learning [34] and computational scale [33].

At its core is the transformer architecture [30], the neural network leveraging self-attention and parallel processing capabilities. But also that rather than relying on human-annotated datasets, LLMs are pretrained using self-supervised learning of broad, large datasets of raw text [33]. During this pre-training stage, the models objective is to capture abstract co-occurrence patterns in the data. In autoregressive models, this is achieved by continuously predicting the probability of the next "token" [32]. A token can represent a word, a subword or a single character in a sequence. In other models, this may involve masked language modeling, where the system learns to predict a missing word given its surrounding context [35].

Today, the most prominent models are Generative Pre-trained Transformers (GPTs) [36]. By utilizing massive scale, often spanning hundreds of billions of parameters, these models develop emergent capabilities that were not explicitly programmed into them, such as in-context learning [33], [36]. By leveraging these emergent properties we can go from basic text prediction to provide the cognitive and generative capabilities of modern chatbots, enabling them to follow complex instructions and reason through problems [33].

2.2.3 Prompt Engineering

Prompt engineering is a strategic technique to extend the functional capabilities of Large Language Models [33], [37]. It involves carefully designing task-specific instructions, or "prompts", to guide model outputs and achieve the desired behaviours. The internal structure of prompt engineering typically consists of three

pivotal elements: the instruction, which defines the task; the context, which provides background to steer the model; and the user's input, as seen in Figure 2.2. This approach improves the model performance across different domains without requiring any modifications to the core model parameters or the traditional need for extensive retraining [37]. This technique leverages the model's pre-existing knowledge by providing natural language instructions or learned vector representations that activate relevant parameters to perform specific downstream tasks [33].

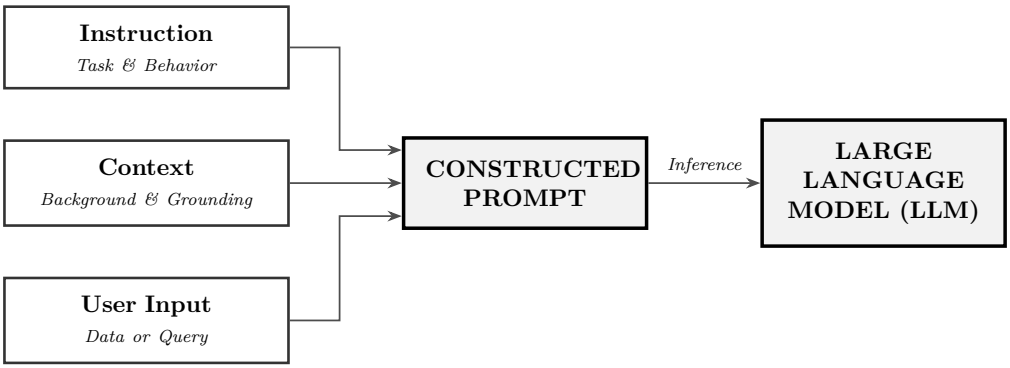

Figure 2.2: Conceptual framework of prompt engineering internal structure.

There is a large variety of techniques when it comes to prompt engineering, ranging from simple methods like zero-shot and few-shot prompting to complex logical frameworks [37]. Zero-shot prompting removes the need for training data by relying on carefully crafted instructions to guide models toward novel tasks using their pre-existing knowledge. Few-shot prompting builds on this by providing a small number of high-quality input-output examples to induce a deeper context for complex tasks. To bridge the gap in complex reasoning, techniques such as Chain-of-Thought (CoT) [38] prompting encourage models to follow coherent, step-by-step logical processes. Other advanced methods like Tree-of-Thoughts (ToT) [39], [40] and Graph-of-Thoughts (GoT) [41] allow for non-linear exploration, backtracking, and the evaluation of multiple reasoning branches to solve intricate problems.

Despite its promise in making AI more adaptable and efficient, prompt engineering still faces significant challenges. Key issues include the presence of biases, factual inaccuracies (hallucinations), and gaps in the interpretability of how the models reach specific outputs [37]. However, it remains a foundational resource for researchers and practitioners, offering a versatile "plug-and-play" mechanism to unlock the full potential of pre-trained models across a wide spectrum of real-world applications.

2.2.4 Retrieval-augmented Generation

Retrieval-augmented Generation (RAG) was first introduced in 2020 by Lewis et al. [42] and is the combination of LLMs and information retrieval. While standalone LLMs stores a lot of factual knowledge within their parameters, they suffer from critical limitations, they are prone to producing "hallucinations"(plausible but factually incorrect outputs), they lack transparency regarding the sources of their

answers, and their static world knowledge is difficult to expand or revise without expensive retraining [42], [43]. By dynamically retrieving relevant information from an external knowledge base to ground the model's responses, RAG effectively reduces hallucinations and significantly improves the reliability, accuracy, and richness of the generated content, particularly for knowledge-intensive tasks [43], [44].

The core architecture of a RAG, often characterised as Naive RAG, operates in three main phases [43], [44]:

- Indexing: This offline preparation phase involves cleaning the external corpus, dividing the documents into smaller, manageable chunks, and encoding these chunks into dense vectors using an embedding model, an encoder-only transformer model. These vectors are then stored in a database to facilitate efficient and frequent searches.

- Retrieval: When a user poses a query, the same encoding model converts the query into a vector representation. The system calculates the semantic similarity between the query vector and the document chunks (often using Maximum Inner Product Search), retrieving the top-K most relevant passages.

- Generation: The original query and the retrieved context are concatenated into a prompt. A generative LLM then formulates a contextually grounded response based on this combined input.

Moreover, Naive RAG has limitations, such as low retrieval precision, context window limitations, and redundant information. This have led to new paradigms, such as Advanced RAG and Modular RAG [43], [45]. Advanced RAG introduces pre-retrieval optimisations to help improve the quality of the query and the indexed data, using methods like fine-grained segmentation, query rewriting and metadata filtering [43]. But also post-retrieval optimisations such as refining the context before passing it to the LLM with methods such as compression and reranking to ensure that the most relevant documents are to be included in the context [43]. Modular RAG moves to a more modular pipeline [45]. Specific functional components, such as targeted web or database search modules, memory modules, and independent validation module to assess retrieval reliability can be dynamically swapped, added, or reconfigured to suit specific real-world applications [43].

2.3 Software Testing

Defective code is a major challenge in modern software engineering. In the automotive domain, however, the stakes are significantly higher. A failure is not merely a deviation from a specification, but often a precursor to a hazard, a potential source of harm to the driver, passengers, or road users [13].

To understand the objectives of testing within this safety-critical context, it is necessary to first establish precise terminology regarding software anomalies. While the terms bug, error and failure are often used interchangeably in casual conversa-

tion, the software engineering literature distinguishes between them to describe the mechanism by which defects propagate [46]:

- **Fault:** A static defect in the software, often referred to as a "bug". This is a mistake in the code or design (e.g., an incorrect logic operator or a missing condition).
- **Error:** An incorrect internal machine state that results from executing a fault. A fault may exist in the code but never lead to an error if that specific code path is never executed or if the specific input does not trigger the problematic state.
- **Failure:** An external, incorrect behaviour with respect to the requirements or expected behaviour. A failure occurs only when an error propagates to the software's output or interface, becoming visible to the user or system boundary [46].

2.3.1 Verification and Validation

In the automotive sector and other safety-critical domains, the software development lifecycle is usually governed by the V-Model, a framework formalized by standards such as ISO 26262 for functional safety and ISO/IEC/IEEE 12207 for systems engineering [13], [47]. Unlike iterative agile methodologies, the V-Model mandates a disciplined symmetry between design and testing, ensuring rigorous traceability from high-level requirements down to final system integration [48].

Central to this framework is the continuous evaluation of the system through Verification and Validation (V&V) [18]. These processes are inseparably linked to Requirements Engineering (RE) and rely heavily on traceability to ensure that every test case is explicitly mapped to the requirements it is designed to evaluate [23], [49]. Relying on the foundational distinction established by Boehm [18], V&V addresses two questions:

- Verification ("Are we building the product right?"): This process evaluates the work products of a specific development phase to ensure they meet the technical constraints and specifications defined at the start of that phase [46], [50]. In the context of RE, verification acts as the check to guarantee that the developed product accurately satisfies the documented functional and quality requirements [49].
- Validation ("Are we building the right product?"): Validation is the high-level evaluation of the system to ensure it fulfils its intended use and meets stakeholder expectations in a real-world environment [46]. Validation begins during the elicitation phase of RE, where stakeholders first confirm that the documented requirements accurately reflect their real-world business goals [49].

Bidirectional trace links allow engineers to track relationships directly between requirements and their associated artefacts, providing the structural backbone for

V&V. This methodology ensures regulatory compliance, accelerates defect tracking, and verifies that the final system accurately reflects stakeholder needs [21].

2.3.2 Hardware-in-the-Loop Testing

Hardware-in-the-Loop (HIL) simulation is a real-time technique used for the comprehensive, cost-effective, and repeatable system-level testing of embedded controllers [51]. In this configuration, the "system under test" (SUT), often an Electronic Control Unit (ECU), is connected to a real-time simulation that acts as the physical plant or operational environment [51], [52].

While pure software simulations (Model-in-the-Loop or Software-in-the-Loop) facilitate rapid prototyping, HIL bridges the gap to full-scale physical testing by incorporating real hardware into the simulation loop [52], [53]. The HIL simulator maintains a bidirectional, closed-loop interaction by monitoring the SUT's actuator commands and injecting synthetically generated sensor signals back into the controller [51], [52].

This technique is particularly effective across several high-stakes scenarios [52]. For safety-critical systems, HIL allows for exhaustive testing where operational failures in a real environment would be unacceptably costly or dangerous [51]. It also enables the simulation of abnormal and faulty conditions, allowing engineers to test embedded code under extremes that might otherwise physically damage the hardware. Furthermore, the platform facilitates precise regression testing and repeatability by exactly duplicating test conditions to ensure software changes do not produce unintended effects [51]. HIL testing is essential for achieving functional safety compliance, facilitating the verification and validation (V&V) processes mandated by the V-model and industry standards like ISO 26262 [52].

2.4 Requirements Engineering

Requirements Engineering (RE) is a fundamental domain within software engineering concerned with the elicitation, analysis, and specification of requirements for a proposed system [54]. It serves as the bridge between stakeholder demands and the technical constraints of the system under development, defining the exact parameters the system must satisfy to meet user needs. The primary objective of RE is to decide precisely what to build. Errors made during this phase severely compromise the resulting system [55]. Furthermore, these defects are the most difficult and expensive to rectify later in the development lifecycle [55].

The traditional RE process is composed of core activities that include requirements elicitation, analysis and negotiation, documentation, validation, and management [56]. During the elicitation and analysis phase, requirements are systematically gathered from stakeholders, who include individuals, groups, or organizations actively involved in, affected by, or capable of influencing the system's outcomes [54]. Analysts must elicit these demands, evaluate them for consistency, feasibility, and

completeness, and translate them into formal requirements [49].

Following this evaluation, the documentation phase produces a requirements specification that acts as an input blueprint for the design and implementation phases [49]. Requirements can be articulated at various levels of abstraction within this document. Most notably, differentiate between domain-level requirements, which describe user activities and workflows in the environment outside the product, and product-level requirements, which specify the exact functional inputs, outputs, and features the software product must provide.

To ensure accuracy, the process relies heavily on validation and verification. Validation is the process by which stakeholders confirm that the documented requirements accurately reflect their real-world needs and business goals [49]. Verification, on the other hand, is the subsequent check to ensure that the final developed product actually satisfies these specified requirements. Furthermore, because business demands and technical constraints naturally evolve, continuous changes throughout the product lifecycle are handled through requirements management [49]. A foundational component of this management is requirements traceability, defined as the ability to describe and follow the life of a requirement in both forward and backward directions, from its origins through its specification, implementation, deployment, and ongoing refinement [21]. Traceability supports critical downstream engineering tasks such as change impact analysis, coverage analysis, and dependency analysis.

Within the specification itself, requirements are broadly categorized into two main dimensions, functional and quality requirements. Functional requirements define the specific behaviours and capabilities of the system, detailing how it records, computes, transforms, and transmits data [49]. In contrast, quality requirements, or non-functional requirements, govern the overall efficacy of the system rather than its specific functional behaviours. They impose strict architectural constraints on attributes such as performance, security, and maintainability [49], [57]. Fulfilling functional specifications is inherently insufficient if the system fails to meet specific quality thresholds. A functionally correct product is in the end unviable if it is unstable or user-hostile. Consequently, a core engineering challenge lies in rigorously balancing these competing non-functional demands.

2.5 Requirements-to-Test Alignment

The wording of requirements-to-test alignment is not as straightforward as it may initially seem. In this thesis, it is defined as the verification of whether an associated test case adequately exercises the functional intent of the requirement to which it is linked. In contrast, Unterkalmsteiner et al. [58] describe alignment as "a goal-directed concept, i.e. the adjustment of requirements engineering (RE) and software testing ST efforts for coordinated functioning and optimized product development," which focuses more broadly on the information linkage and coordination between RE and ST.

By acknowledging these differing perspectives, this thesis focuses on a more concrete and lower level of abstraction, emphasizing whether the intended functionality of a requirement is actually exercised by its corresponding test cases.

2.6 NASA-TLX Questionnaire

To measure mental workload during task performance, the NASA-TLX questionnaire can be used [59]. It is a six-dimensional tool where each dimension is rated on a scale from 0 to 100, completed after a task to assess the workload it required. The questionnaire has been applied across numerous fields and provides a practical, portable alternative to more resource-intensive assessment methods. In Table 2.1 all dimensions are described.

Table 2.1: The six NASA-TLX dimensions. The following definitions are reproduced directly from Hart and Staveland [59].

Dimension	Description
Mental Demand	How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching)? Was the task easy or demanding, simple or complex, exacting or forgiving?
Physical Demand	How much physical activity was required (e.g., pushing, pulling, turning, controlling, activating)? Was the task easy or demanding, slow or brisk, restful or laborious?
Temporal Demand	How much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?
Performance	How successful do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?
Effort	How hard did you have to work (mentally and physically) to accomplish your level of performance?
Frustration	How insecure, discouraged, irritated, stressed, and annoyed versus secure, gratified, content, relaxed, and complacent did you feel during the task?

3

Related Work

3.1 Requirements-Test Traceability

As software systems grow bigger and more complex, establishing comprehensive traceability has become increasingly critical [60]. The foundations of modern traceability were laid by Hamilton and Beeby [61], where they describe traceability as the ability to "discover the history of every feature of a system".

In the past, the effectiveness of Requirements-Test Traceability (RTT) has rested on human intuition. Uusitalo et al. [62] argue that a tester's domain expertise is not merely an asset but a critical success factor. Testers with deep system insight can interpret high-level requirements with minimal supplementary data because they already grasp the underlying "why" and "how" of the architecture.

However, as systems grow, relying solely on human expertise or iterative manual processes becomes untenable. Recent research into LLMs for RTT has introduced a new set of challenges, specifically regarding scalability [63]. While LLMs excel at reasoning through the semantic nuances of individual trace links, the standard "per-link" dialogue approach is neither time- nor cost-efficient for large-scale systems. Because the potential links between requirements and test cases can grow quadratically, this iterative bottleneck currently prevents LLMs from supporting the real-time, continuous traceability updates.

Recent advancements have proposed utilizing Retrieval-Augmented Generation (RAG) to overcome the context limitations of standard LLMs in TLR tasks. For instance, the LiSSA framework [64] demonstrates that combining LLMs with RAG allows for generic trace link recovery across diverse artefact types, such as requirements, documentation, and source code. Studies on this framework indicate that integrating RAG not only makes it feasible to process large-scale datasets that would exceed standard LLM context windows, but also allows these models to significantly outperform traditional, task-specific approaches on various code-related tasks.

Another way to address these limitations is with hybrid frameworks combining LLMs with Heterogeneous Graph Transformers [65]. This approach moves beyond simple textual similarity, which often fails in Natural Language to Programming Language (NL-PL) tasks due to the inherent semantic gap between documentation and ab-

stract code [8]. By using LLMs to generate "Reason nodes" that justify links through technical explanations [66], these models capture the "why" of a connection while using the graph's message-passing mechanisms to model structural dependencies and link transitivity. However, Heterogeneous Graph Transformers require substantial training data. While effective at discovering latent test-to-requirement links that human analysts might overlook, they are insufficiently transparent to serve as the sole source of truth for trusting that a test is sufficient.

3.2 Trace Link Explainability

In addition to recovering and maintaining trace links, an important challenge highlighted in the literature concerns the interpretability of automated traceability results. Several studies emphasize that the effectiveness of traceability techniques depends on whether human analysts can understand why a link has been generated. Guo et al. [63] argue that trace link explanation is a distinct and necessary task, separate from link recovery itself, as it enables analysts to evaluate the correctness and relevance of proposed links, especially in environments where traceability decisions must withstand external scrutiny such as safety-critical or regulated domains [63]. The need for explanations is further amplified when stakeholders lack deep domain knowledge or when system artefacts include highly technical terminology.

Building on the need for interpretability in automated systems, Maro et al. [10] highlight that the automotive domain faces unique hurdles that make "black box" automation particularly difficult to adopt. Their study found that while scientific literature frequently proposes ML and IR for link recovery, these techniques are often rejected by practitioners because they produce incorrect or "false" links that violate the strict safety requirements of ISO 26262. This lack of trust is a primary reason why significant traceability challenges remain unsolved in practice, as developers cannot rely on automated results that they cannot manually verify or understand. To bridge this gap, the authors suggest moving toward semi-automatic maintenance, such as event-based notifications that explain why a link is "suspect" after an artefact change, which would allow human analysts to remain in the loop and maintain the high link quality necessary for safety-critical certification.

3.3 Quality Assessment of Requirement and Test

The theoretical foundations of software test adequacy were formally established by Goodenough and Gerhart [67], who proposed a testing theory centred strictly on reliability, completeness, and validity. Furthermore, the IEEE 829 Standard for Software Test Documentation [68] and its subsequent revision [69] established the first universal frameworks for standardized test documentation. These standards were intended to serve as a formal point of reference, providing a structured methodology that enables organizations to achieve higher levels of consistency and procedural completeness across testing life cycles.

Moreover, the IEEE 830 Recommended Practice for Software Requirements Specifications [70], follows the same lines as the IEEE 829, where the recommendation defines what qualities and content a good requirements specification should have. Criteria such as correctness, unambiguousness and consistency are named as qualities a good software requirements specification (SRS) should have. However, the recommended practice was superseded by IEEE/ISO/IEC 29148:2011 [71], and subsequently by ISO/IEC/IEEE 29148:2018 [72], which today stands as the leading standard for requirements engineering and the specification of high-quality requirements.

Furthermore, highlighting the importance of the quality of requirements, as early as 2001, Boehm and Basili [73] provided Software Defect Reduction Top 10 List. The paper is to some extent an updated version of Industrial Metrics Top 10 List, published in 1987 by Boehm [74]. The authors give ten facts in the form of evidence-based findings, covering topics such as, the disproportionate cost of late defect fixing and the effort spent on avoidable rework.

Additionally, Bertolino [75] equates overall long-term goals with dreams in the domain of testing and the roadmap to them, pointing out challenges and obstacles. A universal test theory is proposed as one of the dreams, with the aim to work as a framework for choosing the most suitable technique for the task at hand. The challenges connected to this dream are many, however some important ones are worth touching upon. The test effectiveness for detecting certain types of faults is often unclear, posing a challenge in selecting an appropriate test technique. A combination or mix of test techniques is proposed as a mitigation. Furthermore, Bertolino raises the challenge of test oracles, where there are problems regarding how to give a verdict on whether a test outcome is acceptable. She highlights that this is a problem that needs to be solved for both test automation and model-based testing to work effectively. [75]

Following the same theme, Tran et al. [76] conducted a tertiary study, investigating the literature reviews done on software testing artefacts and their quality, resulting in a model for test quality. Some interesting findings could be drawn from this study, such as the quality aspects in the context of test case generation. The study brings up the fact that they did not find any quality attributes connected to the test case generation context in the secondary studies they reviewed. Some reasons are brought up, however, more interestingly, Tran et al. point out the important need for analysability and traceability in generated test cases, mostly to secure the possibility to trace the generated tests back to their initial requirements. All in all, the study provides a good overview of how test cases and test quality are measured and defined, resulting in a model for understanding quality in test artefacts [76].

Lastly, to further enlighten the problem of test case alignment assessment, Tran et al. [77] takes the research one step further trying to identify which quality attributes in test cases and test suites are most important for practitioners. As they point out, all the different qualities present in the research domain, described previously by Tran et al. [76], cannot be met. It has to be a trade-off, opening an opportunity

to understand what practitioners prefer. The findings suggest that generally the quality aspect is of large concern and the survey study suggest that Fault Detection, Usability, Maintainability, Reliability, and Coverage are the most important quality attributes. Furthermore, the survey showed problems for the practitioners in quality-attribute definition, measurement, and maintenance, ending with establishing a need for support.

4

Research Methodology

The methodology of this thesis is based upon Design Science Research (DSR). It relies on the foundational principles established by Hevner et al. [78], while explicitly advised by Knauss [79] to adapt these principles to the specific scope of a master's thesis. This adapted framework was selected due to its efficacy in solution-focused environments, making it well-suited for an industry-partnered thesis, aimed at developing a practical solution.

Furthermore, Knauss [79] suggests an iterative three-cycle plan, where each cycle focuses on one research question, lasting roughly one month. These research questions are derived from three focus areas; the problem, the artefact and the evaluation. The first cycle is aimed to understand the problem and how it has been dealt with before, ensuring that the intended artefact solves a real problem. Requirements of the desired artefact is often derived from this phase. However, in our case, cycle one is used not only for understanding the problem, but also for identifying design insights of a potential artefact. The second cycle, or the solution cycle, is where the most progress is done towards creating a viable artefact. Lastly, the evaluation phase is supposed to evaluate the artefact and give a verdict on its usefulness in regards to the intended setting.

Even though the cycles are focusing on different steps, Knauss recommend not to finish them one by one, instead make use of iterations and extend the knowledge gained as the thesis progresses. Stagnating in the problem-understanding phase poses a significant risk of developing a misaligned artefact [79]. This risk is often high in industry settings where the root cause of an issue may be obscured by conflicting stakeholder perspectives or an incomplete initial problem understanding.

Due to the importance of stakeholders in the development of the artefact, significant focus in the methodology was placed on evaluating it from their perspective. This is addressed through a Case Study, consisting of a NASA-TLX Questionnaire and post-task interviews, testing the artefact in its intended setting, following the recommendations of Engström et al. [80]. Simultaneously, optimisation of the key techniques in the artefact was done during Cycle 2, grounded by the findings of Cycle 1.

To maintain transparency, this thesis deviates from Knauss's [79] guidelines by de-

tailing only the finalized architectural decisions rather than every exploratory iteration. For instance, early design cycles involved testing alternative architectures, including Graph Neural Networks (GNNs). However, after brief evaluation, GNNs were deemed insufficient, due to the characteristics of the data we intended to use, namely its limited size and high heterogeneity. While presenting the complete evolution of these discarded design choices holds some theoretical value, prioritizing the final outcomes of each cycle ensures a more coherent and focused thesis.

4.1 Iterative Research Cycles

The different cycles consist of several in-depth steps. These steps are described in detail in their respective sections. As mentioned earlier, the overall approach follows an iterative process to ensure continuous progression and development throughout our study. Consequently, several iterative processes, such as the development of the artefact or the refinement of its techniques, were carried out across all cycles. These actions were highly dependent on the information gained from the results of each cycle. For instance, if novel insights regarding the problem space emerged during Cycle 3, these findings were looped back to refine the artefact, depending on time and resource constraints. This dynamic strategy maximised the integration of valuable, late-stage insights. The approach used in all cycles is visualised in Figure 4.1, which illustrates how each cycle involves three steps performed iteratively to refine and optimise the results. The steps are *Design*, *Implement*, and *Evaluate*, and served as a guiding strategy throughout all cycles.

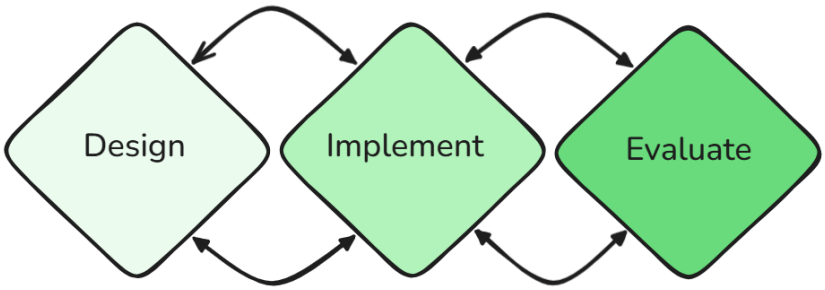

Figure 4.1: Visualisation of the iterative process applied within each cycle, highlighting the three aspects of the DSR approach used in this thesis: *Design*, *Implement*, and *Evaluate*.

5

Cycle One: Problem and Design Understanding

The first cycle focused on the problem and the initial steps toward retrieving foundational knowledge for the design of the artefact, adhering to the first research question. This cycle consisted of the essential steps of a literature review and practitioner interviews, analysed through thematic analysis. The results of the cycle are presented directly after the methodology, to uphold a logical progression.

5.1 Methodology

The methodology of Cycle 1 details the steps of both the literature review and the practitioner interviews, including aspects such as search strategy, thematic analysis, and inclusion and exclusion criteria.

5.1.1 Practitioner Interviews

The primary objective of the practitioner interviews was to establish a deeper understanding of the problem space and to enable an evidence-based approach when identifying the foundational design attributes for the proposed artefact [81].

Interviewees were selected through a voluntary sampling method targeting industry practitioners within the automotive embedded systems domain. This process yielded four participants. Consequently, the high level of expertise among the interviewees was a natural outcome of this voluntary process rather than the result of deliberate, experience-based filtering. The individuals who chose to participate possessed broad professional backgrounds.

To maintain confidentiality, the participants remain anonymous and are identified strictly by their current roles and years of industry experience (see Table 5.1). Given their extensive backgrounds, these practitioners are regarded as domain experts, and their responses serve as the data source for the subsequent thematic analysis.

The semi-structured interviews consisted of nine to eleven baseline questions and lasted approximately 40 minutes each. Three interviews were conducted in English

and one in Swedish. Prior to each session, participants were provided with a brief overview of the thesis and the study's objectives, along with a clear disclaimer stating that they could withdraw their participation at any time.

Table 5.1: Practitioners interviewed with the role and amount of years active in the industry.

ID	Role	Experience
P1	Group Design Leader	9 Years
P2	System Test Engineer	10 Years
P3	System Designer	8 Years
P4	Engineering Manager	15 Years

5.1.1.1 Thematic Analysis

To analyse the gathered interview content, a thematic analysis was conducted, organising the material into different themes. These themes were subsequently used to further understand the problem, provide useful information in the decision of design attributes to base the artefact on and to give their insights in what essential aspects an intended artefact should have.

The thematic analysis was guided by the synthesis framework conceptualized by Cruzes and Dybå [82], adapted to suit the solution-oriented nature of this thesis. The process began with data immersion, where interview transcripts were thoroughly reviewed to establish familiarity before labeling. During this stage, specific text segments were labeled based on their direct relevance to the research objectives. To maintain rigorous alignment with the thesis's overall purpose, the analysis was conducted at the semantic level. It focused on the statements made by the practitioners rather than attempting subjective latent interpretations. This approach ensured that the resulting themes accurately and defensibly reflected the practitioners' stated operational needs.

5.1.2 Literature Review

The objective of the literature review was to extract information from relevant scientific studies to answer the first research question of this thesis. This process secured a significant amount of information regarding attributes to focus on in the design choices of the artefact in cycle two. Additionally, the review sought knowledge about the strategies used in the more rich nearby research field of TLR. This information was then combined with the findings extracted from the thematic analysis of the interviews.

5.1.2.1 Search Strategy

The search strategy of the literature review used a structured search process across digital research databases to identify attributes valuable for the first research question. Queries were conducted across digital databases, including ScienceDirect, IEEE Xplore, and Google Scholar. The primary search string applied was: `("traceability link recovery" OR "requirements traceability") AND ("requirement*" OR "test-case*" OR "test*") AND ("natural language processing" OR "NLP" OR "LLM" OR "graph neural network*" OR "information retrieval" OR "IR-based")`

Other search terms, such as "requirements-to-test alignment", "requirements-to-test mapping" and "trace link explainability", were used, but a significant amount of interest was placed on the nearby research field of Traceability Link Recovery (TLR). The reason for this was mainly explained by the fact that because the field is substantially more mature and closely related. Because TLR focuses on the automated or semi-automated process of finding and establishing links between software artefacts, the techniques used are highly transferable. Secondary search extensions, such as forward or backward snowballing, were not used in the search strategy [83].

5.1.2.2 Inclusion and Exclusion Criteria

To ensure the quality of the literature pool and exclude irrelevant studies, specific inclusion and exclusion criteria were defined. These criteria were established in accordance with the guidelines proposed by Petersen et al. [84] and informed by related studies by Zou et al. [8] and Hassan et al. [85]. Among the standard criteria, it was required that the papers were written in English and that the full text was accessible. Also, a saturation criterion was established. If significant information about a potential domain of interest for the design choices of the artefact was gathered, further searches were discontinued. The reason for this lies in that the attributes picked in the cycle inherit an experimental rationale and does not claim complete accuracy. To clarify, the main objective of the literature review was not to perform a full-scale evaluation of the field of assessing alignment for trace links, rather seek insights of what attributes should be brought to attention while constructing the artefact. The review should therefore be interpreted as more of a targeted literature review intended to inform artefact design, rather than as a systematic mapping of the field. All the inclusion and exclusion criteria used during the search can be seen in Table 5.2.

Table 5.2: Selection criteria for the literature review, with rationale.

No.	Criterion	Rationale
Inclusion Criteria		
I1	Only studies from 2005 until the day of the search.	Ensures relevance
I2	Both primary studies and secondary studies were included in the search.	Reasonable scope
Exclusion Criteria		
E1	Grey literature (white papers, blog posts, keynote summaries).	Ensures quality
E2	Duplicated papers.	Redundancy
E3	Full text not available.	Availability
E4	Search discontinued once relevance was established for a specific domain.	Efficiency

5.2 Results

To address Research Question 1, we combined a thematic analysis of practitioner interviews with a literature review. The information gathered from both sources formed the rationale behind the design choices made during the development phase of Cycle 2. This results section therefore concludes with a list of important attributes which work as the base for the design choices of the artefact. In addition to identifying relevant attributes, the thematic analysis also provides insights into expectations and requirements for the intended artefact.

5.2.1 Thematic Analysis

The following themes were identified using the analytic procedure described in Section 5.1.1.1. It outlines different high-level themes used to identify design attributes to consider in the development of the artefact.

5.2.1.1 Context

The first theme identified in the interviews is the importance of context. Both the hierarchical, situational and environment-specific information required to understand requirements and test cases. Practitioners described layered requirements trees, where meaning is distributed across levels, reading a single statement in isolation is often insufficient. To understand the context of the systems accurately depends on unspoken domain knowledge gained through experience and familiarity with the product.

> *"There are a lot of dependencies, it's very nested, you need to know the*

context and domain."

— P2

"When it's component requirement, it's easy. One-to-one relationship between the requirement and the test. But when you up levels, it's harder to test"

— P2

5.2.1.2 Agility vs. Traceability

Teams emphasized the strength of combining agile, conversation-driven alignment with structured documentation when needed. Internal discussions help them move quickly and build shared understanding, while collaboration with external partners naturally benefits from clearer requirements and formal handshakes. Finding this balance enables knowledge to flow while still supporting traceability and easing downstream verification.

"We find that having direct discussions is more efficient, but still value written requirements also."

— P3

5.2.1.3 Inconsistent structure

A third theme highlights the challenges of achieving structural consistency. While organizations see potential to a unified framework (e.g., consolidating on a single test platform), in practice test cases have different needs. Also the structure of the requirements and test cases can vary, with different fields specified, such as preconditions, and different code styles might hinder automation.

"Some levels of requirements have clear templates for the associated test cases, while other levels leave more room for the individual test writer's approach."

— P4

5.2.1.4 Manual verification

A fourth theme that was common between all interviews is the reliance on manual work to link requirements and tests and to assess coverage. It is the people working that do the manual verification. Teams reported relying on people to judge completeness and adequacy, with limited automated assurance. Manual review can be nuanced and contextual, but it is labour-intensive and susceptible to inconsisten-

cies across individuals and teams. This is a challenge broadly recognized across the software engineering industry, not unique to any single organization or domain.

> *"It's human-verified. It's a manual process to review if the test covers the requirement."*
>
> — P1

> *"We follow a standardized process for verification, but it still relies on manual review, which means we can never guarantee 100% accuracy."*
>
> — P1

5.2.1.5 Combinatorial complexity

Even with clear requirements and structured test design, coverage is threatened by the exponential growth of variants, configurations, and state transitions. Practitioners described discovering alignment issues when untested combinations surfaced and stressed the importance of exercising state changes, not just steady-state "happy paths". Prioritizing the most risk-bearing combinations across SIL, HIL, and vehicle contexts becomes critical.

> *"You can't test with all variants, it grows exponentially very quickly. A configuration input can be missed."*
>
> — P3

Practitioners therefore seek strategies and tools that surface boundary conditions and variant-sensitive scenarios early, rather than retrofitting them after defects are found at a test stage.

5.2.1.6 Qualities of a good requirement and test case

Another theme that was consistently expressed across interviews was the shared understanding of what constitutes a high-quality requirement and test case. Despite the variability in tools, templates and levels of formality, the alignment of what enables clarity, reproducibility and effective verification.

Interviewees described good requirements as those that present a clearly stated objective or rationale, specify what the system must achieve, and remain interpretable across multiple levels of abstraction. Such requirements embed sufficient contextual information to guide downstream development and test design, while avoiding unnecessary constraints on technical implementation. In parallel, a good test case was repeatedly characterized as one that articulates clear preconditions defining the initial system state, combined with concrete triggers or actions and specific, measurable expected results. Detailed verification criteria, including explicit pass/fail

conditions, were viewed as essential for supporting both manual and automated assessment.

Beyond the normal flow, also testing negative scenarios, boundary conditions and state-change behaviours, such as restarts, which often can expose defects in normal "happy path" testing. Finally, they valued well-defined post-conditions, to return the system to a normal state. This should be done to not affect subsequent tests.

5.2.1.7 Practitioners expectations for the artefact

When asked about the artefact being developed in this thesis, practitioners converged on a need for explainable and transparent coverage assessment, rather than a single opaque score. Specifically, they suggested that the artefact provide:

1. a decomposition of each requirement into clear sub-points, together with the explicit mapping from those sub-points to the corresponding tests.

2. an at-a-glance view of what is covered versus uncovered, accompanied by reasons and pointers to the evidence;

3. a transparent explanation of any 0–100 coverage figure, including the underlying dimensions and their weights.

4. reliability signals (e.g., repeatability/stability or flakiness) that calibrate how much the score can be trusted in practice.

In addition, they asked for diagnostic feedback that separates true coverage gaps from data-quality issues (such as missing preconditions or broken/absent links), and for the ability to ingest supporting artefacts beyond the requirement text to enrich the analysis context.

> *"Not only a score. It should give reasoning, list sub-points covered/not."*
>
> — P1

> *"What does 0–100 mean? Which dimensions is it focusing on? Include flakiness/repeatability to weight trust."*
>
> — P3

This theme points to artefacts that produce actionable transparency. Interpreted as, interpretable metrics, clear rationales, and concrete guidance that practitioners can validate.

5.2.2 Literature Review

The results of the literature review are presented in the following section, outlining areas from which valuable design choices can be derived from. Although many of these are implemented in slightly different research contexts, mostly TLR, the core focus is to identify different areas of characteristics to understand if they can be used in a tool aimed for assessing alignment. While an exact artefact used for our purpose in the automotive domain, is not directly described in the literature, we believe taking inspiration from areas as TLR, can result in valuable insights in the creation of the artefact.

5.2.2.1 Semantic

The semantic similarities between textual entities have proven many times to be a useful tool in text similarity. It is for instance brought up by Wang et al. [9] as one of the edges in a GNN solution, attempting to solve the TLR problem between different artefacts in the traceability spectra. Additionally, the follow-up paper to the Wang et al. paper, Zou et al. [8] simultaneously clearly states that textual similarity is not enough for heterogeneous and homogeneous link predictions in TLR, and suggest auxiliary strategies applied in two different approaches, being LLM and HGT. Even though this could indicate irrelevance of semantic alignment, it is still used by both approaches, and both papers are more emphasising that auxiliary strategies is needed, than saying that textual similarity, such as semantic alignment, is not sufficient.

Furthermore, Guo et al. [63] presented, the now rather standard approach, in using word embeddings to understand the semantic similarity in the traceability domain. It proved, at that time, to be very sufficient and superseded the IR approaches used, further strengthen the selection of the semantic alignment as a ground for our design choices.

Lastly, Lin et al. [11] introduces three different architectures, which takes advantage of BERT [35] to predict links between issues and commits in open source projects, on a semantic level. The challenge of sparse data is also brought up which is solved by a three-fold procedure, consisting of pre-training, intermediate training and fine-tuning.

5.2.2.2 Context

The context driven characteristics of alignment could be seen as intuitive. If alignment should be determined, aspects based on the context of the requirement and the test case should be considered, both fine-grained ones and general ones. However, certain examples of usage are presented to motivate the selection of an attribute related to context.

Wang et al. [7], conducted a systematic literature review in 2024, adhering to the need of a proper SOTA scanning of TLR techniques and methods, and a reliant benchmark of TLR. In this paper, several characteristics in requirement to code

cases are utilized in the task of classifying links. Among these, IR-methods together with contextual dependency information between artefacts, sits as an solution for improved accuracy, further rationalises the choice of incorporating the need for understanding context, in the creation of the artefact.

Moreover, Zhao et al. [86] utilise a knowledge graph to capture contextual information of software artefacts. The graph is constructed by parsing Java source code to extract classes, methods, and their structural relationships, while names and comments are incorporated as textual descriptions for TLR. This further puts light on opportunities in utilising context in TLR, and subsequently motivates the context attribute.

5.2.2.3 Structural

The structural part represents the largest concrete area of assessment, focusing on the common structural characteristics in test cases and requirements, such as preconditions, clear objectives, oracles, and test steps. It is valuable to match these with the non-functional aspects described in most component requirements, as gaps in alignment may result in missing quality verdicts. Additionally, independently assessing the quality of both requirements and test cases can help detect missing aspects that are generally considered important to well written test case and requirement structure.

To demonstrate the need for this, Beer et al. [87] conducted a preliminary experiment analysing the links between requirements and test cases to understand if low-quality requirements impact the writing of functional test cases. The authors also highlight typical requirements smells, such as “passive voice without naming the agent” and “vague pronouns”. Ultimately, Beer et al. conclude that the early detection of requirements smells has a direct impact on the functional test cases derived from poor-quality requirements, making this a potentially valuable area of study.

Furthermore, documents of the cooperating company were also a finding during the literature review, however, not directly found thanks to the search terms or criteria. The documents are regarded as guidelines for writing test cases and requirements. For data privacy reasons, the content will not be talked about explicitly but rather a rationale to include the structural assessment in the list of areas to consider.

5.2.2.4 Dependencies

As described in the semantic paragraph, Zou et al. [8] discuss auxiliary strategies in TLR tasks. One of these strategies is utilising the fact that textual similar code artefact are probably inferred to the same source artefact. In other words, if test case one is already linked to requirement 1 and have high textual similarities with test case two, but fails to find similarities to requirement one, even though the ground truth says they are linked, the textual similarities between the two test cases can be used to determine a link. This is something worth applying in the artefact, more precisely links connected to the same parent requirement, and the lack of it.

Additionally, in the case of Wang et al. [9], as discussed in the Semantic section above, shows that extending and including dependencies between code artefacts actually increased the precision of the link prediction in the HGT architecture. While this can be seen as a good indication to incorporate dependencies between artefacts, it is worth noting that these dependencies are very specific for GNNs and code-to-code dependencies. However, it still opens up an opportunity to examine the value, due to the requirements linkage to more abstract requirements.

Moreover, Gao et al. [88] utilise different code dependencies among code artefacts to construct a dependency graph, which is later used to boost or penalise IR-based candidate links based on a small amount of user feedback. Their approach combines traditional IR methods with closeness analysis on captured dependencies. This combination of dependency information and feedback is concluded as an improvement of the accuracy.

5.2.3 Identified Design Attributes

In combination with the literature review and themes derived from the semi-structured interviews, the attributes guiding the development of the artefact in Cycle 2 are presented in Table 5.3. Each attribute is describe individually in the table.

Table 5.3: Attributes aimed to work as foundation of the design choices made in cycle 2.

Attribute	**Description**
Semantic	Both the thematic analysis and the literature review indicate that semantic similarities represent an important attribute to consider. The interviews highlight that higher-level requirements can be difficult to semantically link, but lower-level requirements often align almost one-to-one. Furthermore, the literature shows that semantic similarity is already present in several TLR studies, suggesting that semantic comparison should be considered in the implementation of the artefact.
Context	Contextual information about test cases and requirements is key when assessing alignment. This is particularly emphasized in the interviews with industry practitioners, where, for instance, nested connections and an overview of the entire test system is needed. The literature review also highlights these aspects, making it clear that context must be integrated into designing of the artefact.
Dependencies	Information from both the interviews and the literature review indicates that understanding how requirements are linked to one another, and how higher-level, abstract requirements cascade down into low-level requirements and test cases, can provide valuable insights for assessing alignment. It is also derived from the interviews that automotive testing comes with a lot of complexity, thus introducing more reasons to take dependencies into account and incorporate holistic information to the artefact.
Structure	When it comes to structural alignment, the near one-to-one correspondence between component requirements creates a strong opportunity to analyse the presence and consistency of structural subparts within both the requirements and the test cases. This includes aspects such as condition consistency and the absence of structurally important elements. The rationale behind these structural checks is emphasised in both the interviews and the literature review.

6

Cycle Two: Design and Development of Artefact

The primary objective of Cycle 2 was the technical realisation and iterative refinement of the artefact. Building upon the design attributes established in Cycle 1, this phase concentrated on optimizing the system's core assessment capabilities. This was achieved by iteratively analysing the outputs of the alignment assessments for requirement–test case pairs. Additionally, an ablation study was conducted using both aligned and deliberately misaligned pairs, with the results presented in the later part of the chapter.

6.1 Methodology

The methodology of Cycle 2 details the design choices for the artefact and how they connect with the design attributes discovered in Cycle 1. It also specifies the technical implementation of the artefact, as well as the necessary information about the ablation study.

6.1.1 Design Choices

To answer Research Question 2, the findings of Cycle 1 were used to base the system's architectural design. When evaluating potential artificial intelligence architectures, standard LLMs were initially considered but found to have limitations. As established by Lewis et al. [42], purely parametric language models struggle to precisely access and process factual knowledge, are prone to "hallucinations", and cannot easily update their internal memory without computationally expensive retraining. Furthermore, while LLMs possess strong natural language understanding, they inherently lack domain-specific context and are restricted by input length limitations. Given the vast scale of data needed for it to understand the whole domain, forcing a standard LLM to directly process an entire software repositories introduces severe computational overhead and is often mathematically restricted by these context windows [64].

To overcome these context limitations, alternative advanced methodologies, such as Heterogeneous Graph Transformers (HGT), were evaluated. However, these struc-

tural approaches demand massive, richly annotated datasets to effectively learn complex representations [8]. Given the relatively small surface area of the selected ECUs available for analysis in this study, such data-heavy approaches were deemed unfeasible.

Consequently, based on the specific attributes identified in Cycle 1 and the practical constraints of the engineering environment, a Retrieval-Augmented Generation (RAG) pipeline was selected as the final architectural solution. RAG effectively bypasses the data bottleneck of graph architectures; by relying on dynamic retrieval rather than parametric weight updates, the system can leverage the reasoning power of an LLM without requiring an extensive training corpus. Simultaneously, it solves the inherent flaws of standard LLMs by integrating a parametric memory with a non-parametric memory by retrieving from a vector database of raw text. This integration significantly reduces hallucinations, produces more factual outputs [42], and allows the system to scale efficiently by dynamically retrieving relevant chunks of information right before generation [64]. Recent empirical evaluations explicitly support this choice, demonstrating that RAG architectures can significantly outperform existing state-of-the-art models on code-related traceability tasks [64].

In the context of this thesis, this architecture is implemented as a web application that effectively leverages the reasoning capabilities of LLMs while ensuring that the artefact's alignment assessments remain transparent, inspectable, and grounded in the dynamically retrieved context. This structural implementation directly addresses the expectations gathered during the Cycle 1 practitioner interviews, where engineers emphasized the strict need for an explicit, explainable evidence chain rather than an opaque, unverifiable coverage score.

6.1.2 Data Gathering

The data gathering process for the RAG-system was conducted within the active engineering environment of the industrial partner, utilizing their established database of test cases and requirements. This data serves as their central repository for systems architecture and requirements management. To maintain a focused and analytically manageable dataset, the extraction scope was restricted to Electronic Control Units (ECUs) governing interior and exterior vehicle functions, such as interior/exterior lighting. The primary objective of this extraction was to analyse the traceability links between component-level requirements and their corresponding test cases.

To maintain a high-quality knowledge base, the data was filtered to retain only artefacts reflecting current engineering domain. Items that were outdated, no longer in active use, or linked to an excessively high number of requirements were excluded, as they would be difficult to verify that they were correct. The extraction process also gathered manually written guidelines that explain how specific test commands behave.

6.1.3 Preprocessing and Inference

The system operates through a two-phased architecture divided into offline data preparation and online inference and consists of five components. Table 6.1 details the core responsibility of each component, while the structural relationships and data flows between them are illustrated in Figure 6.1.

Table 6.1: Description of the key components utilized within the system architecture.

No.	Component	Description
1	Signals Parser	Responsible for building the signal catalog.
2	Test Case Parser	Parses test cases from VTT files.
3	Requirement Extractor	Enriches data from the requirements management system.
4	Analyse API	An API service that performs alignment analysis using RAG.
5	Analyse Frontend	A dashboard for visualizing and managing real-time alignment.

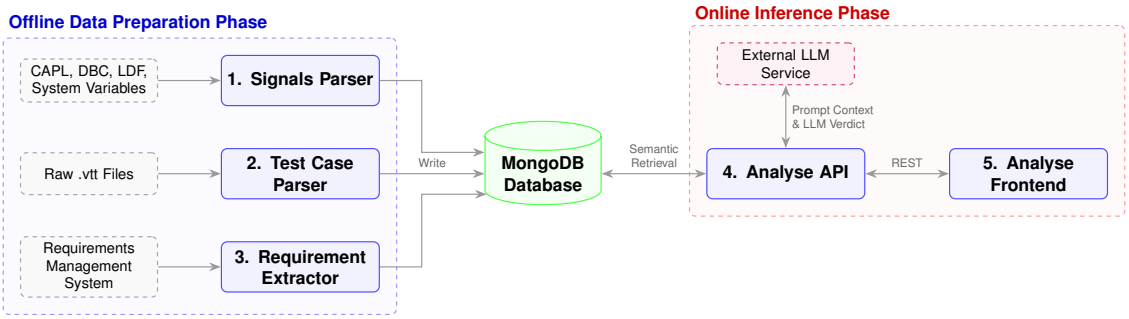

Figure 6.1: High-level system architecture of the artefact, illustrating the separation between the offline data preparation phase and the online inference phase.

The system architecture separates offline data preparation from online inference. Because the production knowledge base consists of continuously evolving artefacts, a static snapshot was created specifically for the analysis in this thesis to ensure consistent evaluation. While this snapshot served as a fixed baseline, the underlying system retains the capability to update the knowledge base dynamically as needed.

All components share a single MongoDB database as their integration point. The data models are document-shaped JSON, which allows each component to write and read self-describing documents without requiring a rigid relational schema.

6.1.3.1 Component 1: Signals Parser

Vehicle network signal identifiers appear throughout CAPL test scripts as the mechanism by which test logic interacts with ECUs. Without the underlying network

database, an identifier such as `BusName::MessageName::SignalName` is limited in meaning for both a human reviewer and an LLM.

This confusion is a concrete example of the *Semantic* design attribute. Without knowing what a signal means, neither a human reviewer without domain knowledge nor an LLM can assess whether the test case correctly exercises the behaviour specified in the requirement. The Signals Parser addresses this by constructing a structured, semantically enriched signal catalogue from the network database files present in the repository. At inference time the catalogue is used to inject signal descriptions into the RAG context, transforming vague identifiers into interpretable evidence.

Because raw signal identifiers and their associated metadata provide minimal insight into a signal's underlying intent, relying on them directly is sub-optimal. To enrich this context while strictly minimizing the inference-time workload of the primary alignment LLM, a pre-computation strategy was utilised. During an offline data preparation phase, concise natural-language descriptions were generated for all signals lacking human-authored documentation. These semantic enrichments were then persisted to the database, ensuring they were available for retrieval during the active assessment phase.

6.1.3.2 Component 2: Test Case Parser

In standard automotive verification workflows, test cases are frequently authored using specialized environments such as Vector vTESTstudio (VTT)[1]. The raw `.vtt` files produced by this tool are proprietary XML documents. While they are inherently machine-readable, their structure and reliance on raw CAPL (Communication Access Programming Language) syntax make them problematic for LLM ingestion. To ensure the LLM can accurately reason about the test logic without inferring intent from code-level syntax, each file is processed through a four-step pipeline before entering the knowledge base:

1. XML parsing and schema validation
2. Step normalisation and deduplication
3. Assembly of a unified per-test-case document
4. Markdown generation per test case

This conversion significantly optimizes the data payload. An empirical evaluation across a dataset of 957 test cases demonstrated an average token reduction of 81.3%. Specifically, the average footprint decreased from 4,060 tokens in the raw XML encoding to 759 tokens in the Markdown representation. We have not formally tested whether the parsing process is a strictly 100% lossless conversion, meaning we cannot guarantee that the generated Markdown could be perfectly round-tripped back

[1] `https://www.vector.com/int/en/products/products-a-z/software/vteststudio/`

into the exact original XML syntax. However, in practice, the conversion works well for our analysis. The Markdown representation gives clear visual and structural separation between setup steps, actions, and assertions. This transforms the pipeline from a programming language-to-natural language, into a natural language-to-natural language interaction, an approach demonstrated to enhance underlying model performance [8].

The initial set of 957 test cases was both manually reviewed and cleaned using Python scripts to remove instances unsuitable for the evaluation scope of this thesis. Items were excluded if they were duplicates, structurally incomplete relative to the analysis objectives, shorter than 15 words, or linked to more than eight requirements. These were thresholds chosen to ensure analytical manageability and after this scoping process, 367 test cases were used in the thesis.

6.1.3.3 Component 3: Requirement Extractor

The industrial partner's requirements management system is the source of truth for requirements and traceability links. The Requirement Extractor connects the test cases with the requirements, addressing the *Context* and *Dependencies* design attributes directly. These attributes suggest that the artefact has access to the full hierarchical container structure of each requirement and to all linked artefacts such as parent and child requirements, linked test cases, and associated design guidelines. Raw exports from the requirements management system do not expose this full hierarchy in a retrieval-ready form, it must be reconstructed by joining several different export artefacts. The extractor performs this reconstruction and stores the enriched documents in MongoDB, where they are available to the retrieval pipeline at inference time.

6.1.3.4 Component 4: Analyse API

The Analyse API serves as the functional core of the artefact. For any given requirement-to-test-case pair, it generates an alignment verdict supported by retrieved context and linked test cases or requirements. To execute this, the architecture uses the *RAG* paradigm, combining semantic retrieval with structured metadata filtering and LLM-based generation.

6.1.3.5 Component 5: Analyse Frontend

To visualise the analysis, a frontend React application was developed. Its design is directly motivated by the practitioner expectations identified in Cycle 1, which emphasized that the artefact must deliver *explainable* and *transparent* alignment assessments. Rather than providing a single, opaque score, the system presents a decomposed result set that practitioners can readily validate and act upon.

A central element of this framework is the step-level semantic evaluation. To overcome the limitations of aggregate scoring, the system classifies each line of the test case using a four-tier schema: *green* (direct requirement validation), *yellow* (ambiguous or partial validation), *red* (contradictory or irrelevant code), and *neutral*

(infrastructure, comments, and unknown steps). This discrete, line-by-line mapping enables practitioners to rapidly identify and isolate coverage gaps at their exact point of execution.

6.1.4 Context Selection Rationale

The selection of contexts to include in the prompt was directly guided by the four design attributes identified in Cycle 1: *Semantic*, *Structure*, *Context*, and *Dependencies*. Each context category was deliberately chosen to address one or more of these attributes.

6.1.4.1 Linked Requirements and Test Cases

The practitioner interviews consistently highlighted that no single requirement or test case can be assessed in isolation, requirements carry implicit parent-child constraints, and test cases inherit assumptions from their sibling tests in the same suite. This reflects the *Dependencies* and *Context* attributes. Providing the LLM with requirements that are formally linked to the most structurally similar test cases preserves the verified traceability structure of the knowledge base. Similar test cases additionally give the LLM a concrete structural reference, they show what a correctly implemented test for a related requirement looks like. This anchors the line-by-line comparison to an empirically grounded baseline rather than to the model's generic prior knowledge, utilising case-based reasoning.

6.1.4.2 Guidelines

Design guidelines are component-specific guidelines that describe, in more detail, domain-specific subsystems, providing the LLM with valuable information about the requirement and test case under assessment. Concurrently, all documents classified as general guidelines are retrieved unconditionally. These general guidelines provide information about the different commands used in the test case markdown, along with boolean clarifications. Both types of guidelines correspond to the *Context* attribute, while the design guidelines can also be tied to the *Dependency* attribute.

6.1.4.3 Signal Descriptions

Vehicle network signal identifiers appear throughout test cases as one of the primary mechanism by which test logic interacts with ECUs. Without the signal catalogue, these identifiers are semantically vague to both human reviewers without domain knowledge and LLMs. This directly connects to the *Semantic* design attribute. By resolving each identifier to a structured human-readable description, the pipeline enables the LLM to evaluate whether the signals exercised by the test case are the correct ones given the behaviour specified in the requirement.

6.1.4.4 Function Descriptions

Test library functions, are domain-specific vocabulary that carries implicit preconditions and side effects not visible in the test case Markdown. A function name

alone is insufficient to determine whether the test correctly exercises the specified behaviour, such as, the function's parameter space and return contract. This is a second example of the *Semantic* attribute. Including function descriptions does not only allow the LLM to assess whether the right function was called, but also whether it was called with the correct arguments and in the correct sequence, relative to the requirement's specified trigger and behaviour.

6.1.4.5 Snippet Examples

Even with rich retrieved context, LLM verdicts on alignment can be inconsistently calibrated, two structurally similar pairs may receive different scores because the model's internal threshold for what alignment means is underdetermined. To address this, one aligned and one subtly misaligned (mutant) example retrieved from the snippet collection are included as implicit few-shot calibration anchors. The mutant is intentionally structurally close to the aligned example. For instance, the same test pattern with a single incorrect signal assertion, to make the distinction between alignment and misalignment as precise as possible.

6.1.5 Ablation Study Design

To isolate the exact contribution of the RAG context and determine the optimal artefact architecture, an iterative ablation study was designed. Each requirement–test-case pair was evaluated under multiple configurations by enabling and disabling individual prompt sections (e.g., full context, no context, functions + signals + snippets + guidelines).

The no-context condition served as the baseline, with the LLM receiving solely the requirement text and the test case. The objective was to empirically identify which configuration produced the most reliable evaluation without causing generative instability.

6.1.5.1 Ablation Study Dataset

The dataset used for evaluation consisted of 46 requirement–test case pairs drawn from the industrial test repository. These were extracted from a total of 367 test cases and requirements, described in 6.1.3.2. The evaluation-set, therefore, represented approximately one-eighth of the total, broadly in line with the standard 80/20 or 80/10/10 train–validation–test splits used in machine learning. The reason for the split, which is not common in RAG architectures, was to prevent retrieving the same test case or requirement during evaluation, as well as to limit the evaluation to a more manageable size. Each pair was processed three times to account for the stochastic, non-deterministic nature of the underlying language model.

6.1.5.2 Evaluation Metrics

We evaluate a dataset P comprising $n = 46$ requirement–test-case pairs. To account for the generative variance inherent to LLMs, each pair is evaluated across $K = 3$ independent runs.

Verdict quality index (VQI)
To quantitatively evaluate the model's performance, we convert its verdicts into an ordinal 1–5 scale. We define this mapping, $\phi : \mathcal{V} \to \{1, \ldots, 5\}$, as follows:

$$\phi(\text{Excellent}) = 5, \quad \phi(\text{Good}) = 4, \quad \phi(\text{Sufficient}) = 3$$

$$\phi(\text{Poor}) = 2, \quad \phi(\text{No coverage}) = 1$$

For each pair, we identify the modal verdict across the K runs to establish a consensus score. We then calculate and report the mean VQI across all pairs for each condition. This enabled us to determine whether introducing RAG context (such as signal and function definitions) systematically improves the overall alignment assessment compared to the baseline.

Verdict consistency (VC)
Given that LLMs are inherently non-deterministic, identical inputs can produce slightly different outputs across multiple evaluations.

Verdict consistency measures the inter-run agreement of the assigned verdicts. It calculates the percentage of test case pairs for which the model assigns the exact same overall verdict across all K runs:

$$\text{VC} = \frac{\left|\{c \in C : v_{c,1} = v_{c,2} = \cdots = v_{c,K}\}\right|}{n} \times 100$$

Statistical hypothesis testing
To determine whether differences between each RAG-augmented condition and the No Context baseline are statistically reliable, a two-sided Wilcoxon signed-rank test is applied to both VQI and VC scores at the pair level ($n = 46$). The null hypothesis (H_0) states that the median difference in scores between a given RAG condition and No Context is zero; the alternative hypothesis (H_1) states that the median difference is non-zero. A significance threshold of $\alpha = 0.05$ is used throughout.

6.2 Results

The artefact developed to address Research Question 2 is a web application implementing a RAG pipeline for alignment assessment between requirements and test cases. Its architecture, components, and design are described in full in this result section, also establishing the core analysis pipeline. Additionally, an ablation study was done to determine contextual importance.

6.2.1 Alignment Analysis Pipeline

To demonstrate the system's end-to-end functionality, the `/analyse` endpoint serves as the primary entry point for the pipeline. This endpoint triggers a nine-step execution flow that maps directly onto the four core design attributes established in Cycle 1.

6.2.1.1 Step 1: Requirement Description Generation

Before retrieval begins, the analysis service constructs a unified runtime requirement description. This step synthesises the primary requirement with all supporting requirements linked to the test case under review. These elements are merged into a single, standardized Markdown document via an LLM call. This semantic transformation turns the requirement(s) text into a more consistent format, resulting in a more interpretable format for the downstream alignment LLM. The format is based upon best practices for requirement structure, where the requirement is tied to *Context/Preconditions*, *Trigger*, *Behaviour* and *Outcome*.

6.2.1.2 Step 2: Case-based Test Case Semantic Ranking

The pipeline first transforms both the requirement and the test case under assessment into dense vector representations of 3072 dimensions, using the text-embedding-3-large[2] embedding model. By projecting these artefacts into a shared semantic space, the system evaluates their underlying intent rather than relying on superficial keyword matching. Subsequently, the system utilises metadata to constrain the search space. If the queried test case is associated with a specific component with a relatively high abstraction level, the repository is pre-filtered to include only test cases within that same abstraction tree. Otherwise, it defaults to the entire repository if no component is specified. Then, the test case vector is compared against the remaining documents using cosine similarity to measure semantic proximity. The system then extracts the top two highest-scoring documents ($k = 2$). These retrieved artefacts serve as primary case-based context, providing the main generation model with highly relevant, structurally similar context to guide its final alignment verdict.

The rationale for ranking test cases first, rather than requirements, reflects a key empirical insight from iteratively refining the artefact. Test cases in the knowledge base often share direct structural similarity with the query test case. The same functions and the same signal patterns. Selecting by test case similarity first therefore produces a more contextually coherent candidate set, than selecting by requirement similarity.

6.2.1.3 Step 3: Linked Requirement Retrieval

Rather than ranking requirements independently by semantic similarity, the pipeline retrieves requirements that are *directly linked* to the case-based test cases. This traversal strategy preserves the traceability structure of the knowledge base and avoids selecting requirements that are merely topically adjacent but not formally associated with the evidence test cases.

6.2.1.4 Step 4: Function Context Enrichment

Function names referenced in the test case Markdown are extracted and matched against the function catalogue. The resulting candidates are passed to a joint LLM

[2] `https://developers.openai.com/api/docs/models/text-embedding-3-large`

relevance judge (described in Step 7) rather than injected directly, preventing irrelevant functions from occupying context window space.

6.2.1.5 Step 5: Guideline Retrieval with Metadata Filtering

The retrieval of design guidelines is executed via firstly metadata filtering and secondly semantic search. Initially, the system queries the guidelines collection using the target requirement's component and module metadata to isolate component-specific rules. The retrieved guidelines are also restricted with a maximum word count of 100 words, reducing the risk for noise in the prompting.

Alongside this targeted search, the architecture unconditionally retrieves all context designated as *general guidelines*. This approach ensures that even if no component-specific guideline matches, the prompt receives general guidelines to understand the test case structure, such as command definitions.

6.2.1.6 Step 6: Snippet Example Retrieval

To further improve the prompt, the system retrieves two snippet examples via semantic search against the target test case: one *aligned* example and one *mutant* (tagged misaligned) example. This retrieval is strictly gated by a cosine similarity threshold of 60%. The choice of the specific threshold was based on trial and error, with different thresholds tested to identify an appropriate value. This was possible because we created the material ourselves and therefore had some understanding of the scenarios in which certain snippets should be retrieved.

Furthermore, if no snippets were over the minimum, the system intentionally omits these examples to prevent the LLM from being misdirected by irrelevant guidance. These snippets serve as calibration anchors in the prompt. By presenting the LLM with a chunked specific portion of the test case which aligns to a small trait of the requirement, the system provides few-shot guidance to the LLM.

6.2.1.7 Step 7: Joint LLM Relevance Judgement for Signals and Functions

Signal candidates are retrieved via semantic search against the signals collection rather than by exact identifier lookup, like the functions retrieval. Embedding-based retrieval is beneficial for signal gathering, as signals are not always strictly identical (e.g., due to suffix variations), yet could still have valuable information that contributes to the understanding of the test case.

Both candidate lists are then jointly submitted to a single LLM relevance judgement call together with the test case Markdown and the requirement(s) combined Markdown. The LLM returns only the subset of signals and functions that are genuinely relevant to the analysis. Only those that pass this relevance gate are injected into the final prompt context.

6.2.1.8 Step 8: Prompt Assembly

The assembled context is serialised into a structured prompt. All prompt sections are independently toggleable, enabling the ablation experiments described in Section 6.1.5.

6.2.1.9 Step 9: Result Generation

The response carries the verdict and the evidence chain that the LLM produced. This design decision is motivated by the explainability requirement identified consistently in Cycle 1 and in the literature. Practitioners will not act on an opaque verdict from an automated system in a safety-critical context. By exposing the retrieved context, and the combined runtime requirement description, alongside the verdict, the system allows practitioners to interrogate the reasoning process. This allows for an opportunity to identify cases where retrieval produced misleading context.

6.2.2 Prompt Engineering Design

To ensure the LLM operates accurately, some prompting strategies were tested. The prompting technique that produced the most reliable results was subsequently integrated into the artefact. The GPT-5.4 nano model[3] was selected for this task due to its cost-effectiveness, low latency, industrial partner constraints and large context window. The evaluation of the prompting techniques was conducted through human assessment by us, the authors.

The alignment analysis is driven by a single, structured prompt submitted to the LLM for every requirement–test-case pair. As visualized in Figure 6.2 and detailed in Table 6.2, the prompt is divided into a *system message* and a *user message*. The system message encodes the invariant role, task framing, and output schema. Conversely, the user message carries the specific pair under evaluation alongside its dynamically retrieved context.

Within these messages, individual prompt sections are further classified as either static or dynamic. Static sections consist of fixed, hand-crafted instructions that enforce domain knowledge and output constraints; these remain identical across all API requests. Dynamic sections, however, are populated at runtime by the retrieval pipeline and are unique to each evaluation. This architectural separation facilitates isolated ablation testing, as each dynamic section can be toggled independently to measure its specific impact on the final verdict.

At the core of the static instructions, the system message directs the LLM to assume the persona of a specialized quality assurance analyst. Its primary objective is to grade the alignment and coverage between automotive component requirements and HIL/SIL test cases. To ensure a systematic and reproducible evaluation, the message establishes explicit rules, requiring the model to score the alignment across four distinct dimensions to determine the final verdict. These dimensions are

[3] `https://developers.openai.com/api/docs/models/gpt-5.4-nano`

Functional Coverage (max 20), which assesses whether the test actively executes the required behaviour and covers all relevant scenarios; Validation Completeness (max 10), which evaluates whether proper checks, assertions, and verification functions are present; Outcome Verification (max 10), which determines if the expected outcomes dictated by the requirement are correctly verified; and Setup Adequacy (max 10), which verifies that all necessary preconditions and initial states are properly established prior to execution. In addition to these aggregate dimensions, the prompt enforces the four-tier line-level status labeling schema (green, yellow, red, and neutral) discussed previously, ensuring the model provides both a high-level quantitative score and granular, step-by-step diagnostic feedback.

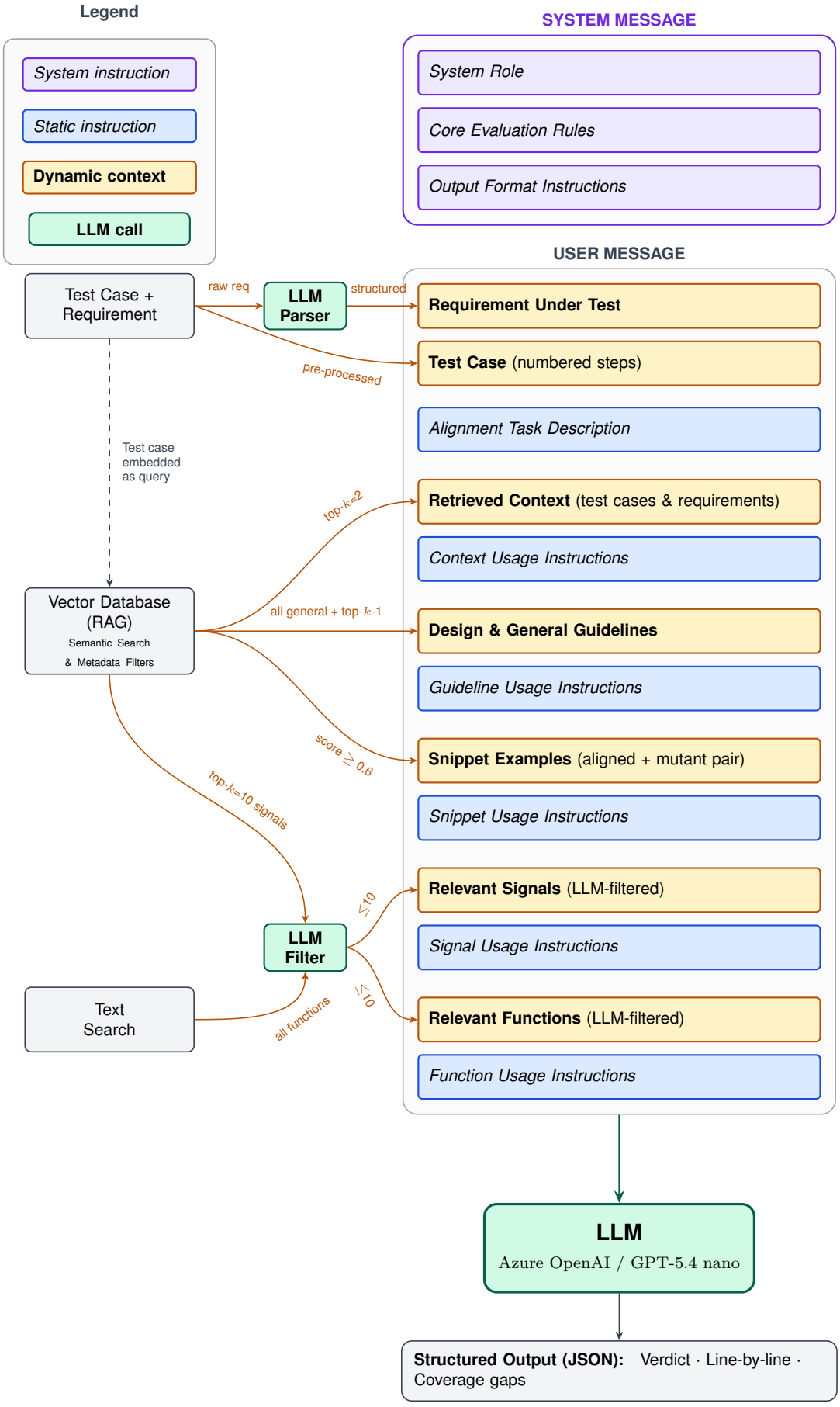

Figure 6.2: Prompt instructions and architecture

Table 6.2: Section-by-section breakdown of the alignment-analysis prompt, covering all static template text and the retrieval mechanism for each dynamic context source.

Prompt Section	Type	Content / Purpose
SYSTEM MESSAGE		
System Role	SYSTEM	Establishes the LLM persona and task domain. `"You are a specialized quality assurance analyst. Your purpose is to grade alignment/coverage between automotive component requirements and HIL/SIL-test cases."`
Core Evaluation Rules	SYSTEM	Defines the four scored evaluation dimensions, the per-dimension maximum scores, and the four line-level status labels used in the markup output. *Evaluation dimensions (max score):* • **Functional Coverage** (max 20): does the test execute the required behaviour and cover all relevant scenarios? • **Validation Completeness** (max 10): are proper checks/assertions/functions present? • **Outcome Verification** (max 10): are the expected outcomes correctly verified? • **Setup Adequacy** (max 10): are preconditions properly established? *Line-status labels:* • **green**: step directly validates a requirement element (context, trigger, behaviour, or outcome) with clear, verifiable checks; includes necessary steps for test execution. • **neutral**: infrastructure, comments, or irrelevant code with no test value. • **yellow**: step is incomplete or only partially validates the requirement; use when validation is unclear, loosely related, or functionally deviates. • **red**: step is incorrect or contradicts the requirement; checks the wrong behaviour, uses incorrect signals/values, or ignores the requirement. Also instructs the model not to enforce strict literal variable matching (`"Interpret the logical intent of the variables and assertions"`).

continued on next page

(continued from previous page)

Prompt Section	Type	Content / Purpose
Output Format Instructions	SYSTEM	Constrains the response to a single valid JSON object. Defines the mandatory schema: `"{"verdict": "Functional coverage score: ⟨N⟩, Validation completeness score: ⟨N⟩, Outcome verification score: ⟨N⟩, Setup adequacy score: ⟨N⟩", "line_by_line": [{"line_no", "text", "status", "comment"}], "missing_or_incomplete_steps": [{"description", "recommended_addition"}]}"` Rules: output valid JSON only; all fields required; arrays may be empty; `missing_or_incomplete_steps` capped at 2 items; scores are integers within each dimension's maximum.
USER MESSAGE		
Requirement Under Test	DYNAMIC	*Fetched from the request.* Contains the full structured requirement document for the component under test (id, description, purpose, context, trigger, behaviour, outcome).
LLM Requirement Parser	LLM CALL	*Invoked at request time before injection.* Combines and cleans the raw requirement fields from the payload into a single coherent document and inserted into the **Requirement Under Test** block.
Test Case (numbered steps)	DYNAMIC	*Fetched from the request.* The test case is pre-processed into numbered line objects `{line_no, text}` so the model can reference individual lines. Supports both single and multi-test-case payloads.

continued on next page

(continued from previous page)

Prompt Section	Type	Content / Purpose
Alignment Task Description	STATIC	Provides the structured analysis workflow the model must follow. `"This requirement and test case are traceability linked. Analyze whether the test case adequately validates the requirement's Context/Preconditions, Trigger, Behavior, and Outcome."` *Six-step procedure:* 1. Analyse relevant signals, functions, and applicable design guidelines (signal and function descriptions convey functionality beyond their names). 2. Review contextual requirement–test-case examples to understand alignment patterns. 3. Use the case-based chunks to calibrate how alignment/-coverage should be judged. 4. Evaluate the requirement under test and the test case to identify coverage gaps based on the core rules. 5. Provide the line-by-line assessment using the defined labels (green/yellow/red/neutral). 6. Give final dimension scores based on the core rules and maximum-score definitions; consider the severity of any coverage gaps.
Retrieved Context	DYNAMIC	*Retrieved from the Vector Database via semantic (embedding) search, top-k = 2 test cases.* Contains test cases retrieved by cosine similarity against the current query. Gets the requirements linked to the retrieved test cases. Serves as case-based reasoning examples to calibrate the model's alignment judgement.
Context Usage Instructions	STATIC	`"Use the retrieved test cases and requirements as references for case-based reasoning to understand how similar pairs are successfully aligned. Focus on the REQUIREMENT UNDER TEST for your analysis. Do not use the retrieved context to fill in missing or incomplete information; rely on the context solely to guide your understanding of structure and reasoning."`

continued on next page

(continued from previous page)

Prompt Section	Type	Content / Purpose
Design & General Guidelines	DYNAMIC	*Design guidelines are first retrieved from software module text search, then retrieve from the Vector Database via semantic search against the requirement text.* Two sub-types are injected: • **Design guidelines**: component-specific guidelines matched to the requirement's software module. • **General guidelines**: organisation-wide rules that apply independently of component context.
Guideline Usage Instructions	STATIC	Two instruction blocks are emitted (one per sub-type): Design guidelines: `"These are requirement-matched design guidelines tied to the relevant software component. Use them to understand the overall design of higher abstraction levels of the requirement."` General guidelines: `"These are general guidelines that always apply, independent of component/module context. Use them to understand the meaning of the test steps."`
Snippet Examples	DYNAMIC	*Retrieved from the Vector Database (snippet collection) via cosine similarity on pre-computed embeddings (min score = 0.6).* Always exactly two slots: one **aligned** (non-mutant) and one **non-aligned** (mutant) requirement–test-case pair, each annotated with a verdict, rationale, and `teaching_point`. Used as few-shot calibration examples.
Snippet Usage Instructions	STATIC	`"These are reference examples for calibration. You get one aligned (non-mutant) and one not-aligned (mutant) chunked pair. Use their assessment and teaching_point to guide your judgment style."`
LLM Relevance Filter	LLM CALL	*Single LLM call that jointly ranks two candidate pools.* Receives signal candidates (top-k = 10, via semantic search from the Vector Database) and function candidates (via text search), then scores each candidate for relevance against both the requirement and the test case in one pass. Outputs a filtered list of at most 10 signals and at most 10 functions for injection into the prompt.
Relevant Signals	DYNAMIC	*Output of the LLM Relevance Filter (signal stream).* Each entry includes signal name and and human-readable description. At most 10 signals are forwarded to the prompt.

continued on next page

(continued from previous page)

Prompt Section	Type	Content / Purpose
Signal Usage Instructions	STATIC	`"This is resolved signal evidence, and treat it as high-priority technical context. It can help you understand the technical details and the intent of the test steps."`
Relevant Functions	DYNAMIC	*Output of the LLM Relevance Filter (function stream).* Each entry contains the function name and human-readable description, providing the model with the actual implementation behind opaque function calls in the test steps. Capped at 10 functions.
Function Usage Instructions	STATIC	`"These functions are part of the test case steps and are important for understanding its functionality. They provide insight into the technical implementation. Treat it as high-priority technical context."`

6.2.3 Web Application

Figure 6.4 illustrates the full analysis dashboard for a synthetic aviation-domain requirement-to-test-case pair. Because the industrial partner's data is highly sensitive, this mock example is utilized to safely demonstrate the artefact's capabilities. The frontend is specifically engineered to address the practitioner expectations identified in Cycle 1. For instance, moving beyond opaque aggregate scores to provide a highly transparent, decomposed alignment assessment. Structurally, the dashboard is divided into distinct functional areas to seamlessly guide the user's analytical workflow.

6.2.3.1 Input Overview

The uppermost section presents the raw data under assessment, including the requirement ID, original description, and the test case Markdown. Immediately below this, the system provides a "Combined Requirement Description," synthesizing complex requirements into a single, cohesive runtime definition. This ensures that both the human reviewer and the AI operate from the exact same contextual baseline before proceeding to the detailed assessment. Also when an assessment analysis is complete, the verdict of the alignment is shown here to give a high-level overview of the assessment.

6.2.3.2 Line-by-Line Analysis

The central component of the interface is the test case execution map, which decomposes the test case into individual, numbered steps. To bypass the limitations of aggregate scoring, the system classifies each line into a four-tier schema: green

(aligned), yellow (partial/warning), red (misaligned), or neutral (infrastructure). This visual colour-coding is important as it intends to reduces the practitioner's cognitive load, instantly drawing their attention to problematic lines without requiring them to manually parse the entire script.

6.2.3.3 Targeted Rationale and Explanations

As detailed in Figure 6.3, users can hover over any flagged step to reveal an expanded rationale tooltip. In this example, the system explicitly justifies why a step received a "Warning" status (e.g., noting that a 150 ms wait time fails to validate a strict $\leq$ 100 ms latency requirement). This transparency is vital for establishing trust, empowering the engineer to independently verify the AI's logic rather than blindly accepting a black-box verdict.

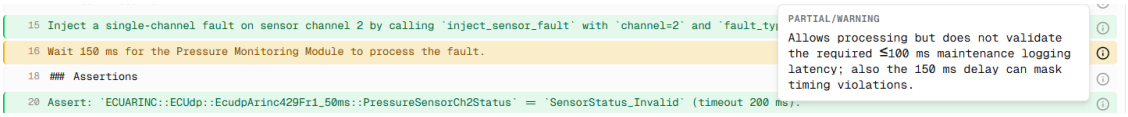

Figure 6.3: Expanded line-by-line rationale for a test step flagged with a warning status. The requirement and test case used in this demonstration are AI-generated for illustrative purposes.

6.2.3.4 Missing or Incomplete Steps

Rather than solely evaluating existing lines, this section identifies macro-level coverage gaps, explicitly highlighting functionality demanded by the requirement that is entirely absent from the test case. To facilitate correction, the system generates concrete, actionable recommendations (e.g., "Add an assertion on the regulation result").

6.2.3.5 Supporting Context

The lower portion of the dashboard exposes the data retrieved by the RAG pipeline. It displays the specific signals, functions, case-based test cases and linked requirements, and guidelines the LLM utilized to form its assessment. Presenting this evidence chain is critical in safety-critical domains, allowing practitioners to fully inspect the tool's reasoning and ensure the assessment is grounded in accurate, approved documentation rather than AI hallucinations.

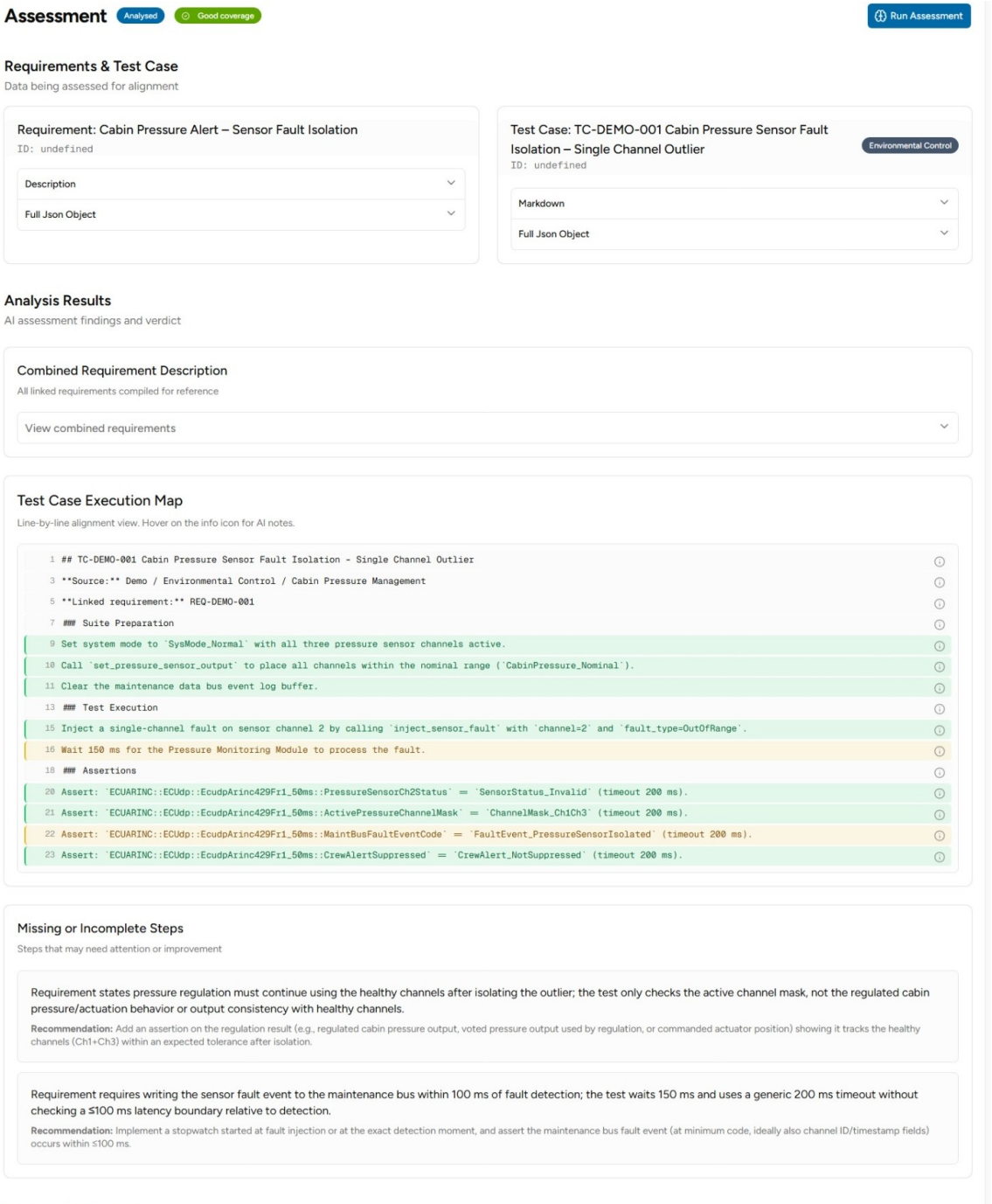
Assessment
Analysed
Good coverage
Run Assessment
Requirements & Test Case
Data being assessed for alignment
Requirement: Cabin Pressure Alert – Sensor Fault Isolation
ID: undefined
Description
Full Json Object
Test Case: TC-DEMO-001 Cabin Pressure Sensor Fault Isolation – Single Channel Outlier
Environmental Control
ID: undefined
Markdown
Full Json Object
Analysis Results
AI assessment findings and verdict
Combined Requirement Description
All linked requirements compiled for reference
View combined requirements
Test Case Execution Map
Line-by-line alignment view. Hover on the info icon for AI notes.
1 ## TC-DEMO-001 Cabin Pressure Sensor Fault Isolation - Single Channel Outlier
3 **Source:** Demo / Environmental Control / Cabin Pressure Management
5 **Linked requirement:** REQ-DEMO-001
7 ### Suite Preparation
9 Set system mode to `SysMode_Normal` with all three pressure sensor channels active.
10 Call `set_pressure_sensor_output` to place all channels within the nominal range (`CabinPressure_Nominal`).
11 Clear the maintenance data bus event log buffer.
13 ### Test Execution
15 Inject a single-channel fault on sensor channel 2 by calling `inject_sensor_fault` with `channel=2` and `fault_type=OutOfRange`.
16 Wait 150 ms for the Pressure Monitoring Module to process the fault.
18 ### Assertions
20 Assert: `ECUARINC::ECUdp::EcudpArinc429Fr1_50ms::PressureSensorCh2Status` == `SensorStatus_Invalid` (timeout 200 ms).
21 Assert: `ECUARINC::ECUdp::EcudpArinc429Fr1_50ms::ActivePressureChannelMask` == `ChannelMask_Ch1Ch3` (timeout 200 ms).
22 Assert: `ECUARINC::ECUdp::EcudpArinc429Fr1_50ms::MaintBusFaultEventCode` == `FaultEvent_PressureSensorIsolated` (timeout 200 ms).
23 Assert: `ECUARINC::ECUdp::EcudpArinc429Fr1_50ms::CrewAlertSuppressed` == `CrewAlert_NotSuppressed` (timeout 200 ms).
Missing or Incomplete Steps
Steps that may need attention or improvement
Requirement states pressure regulation must continue using the healthy channels after isolating the outlier; the test only checks the active channel mask, not the regulated cabin pressure/actuation behavior or output consistency with healthy channels.
Recommendation: Add an assertion on the regulation result (e.g., regulated cabin pressure output, voted pressure output used by regulation, or commanded actuator position) showing it tracks the healthy channels (Ch1+Ch3) within an expected tolerance after isolation.
Requirement requires writing the sensor fault event to the maintenance bus within 100 ms of fault detection; the test waits 150 ms and uses a generic 200 ms timeout without checking a ≤100 ms latency boundary relative to detection.
Recommendation: Implement a stopwatch started at fault injection or at the exact detection moment, and assert the maintenance bus fault event (at minimum code, ideally also channel ID/timestamp fields) occurs within ≤100 ms.

Figure 6.4: The frontend analysis page displaying an alignment assessment for an AI-generated aviation-domain requirement–test case pair. The upper half shows the overall verdict and the combined runtime requirement description; the lower half presents the line-by-line execution map with per-step status indicators and the retrieved context panel. The requirement and test case are AI-generated for illustrative purposes.

6.2.4 Ablation Study Results and Configuration Selection

To understand the impact of different contexts and finalize the artefact's design, we measured the artefact's Verdict Quality Index (VQI), Verdict Consistency (VC), and line-by-line status percentage distribution across multiple ablation configurations.

6.2.4.1 Verdict Quality and Consistency

To comprehensively assess the system's performance, we evaluated verdict quality and consistency under both aligned (normal) and misaligned conditions. This dual approach evaluates whether the artefact assigns higher verdicts to aligned test logic and lower verdicts to deliberately misaligned links.

6.2.4.2 Performance on Aligned Data

This section details the artefact's performance when evaluating normal, aligned data. As shown in Table 6.3, the baseline model (No Context) achieved a Mean VQI of 2.52 (95% CI [2.29, 2.76]) and a mean inter-run consistency of 81.89% (95% CI [77.04%, 86.74%]). The introduction of structured context systematically improved both of these metrics. Notably, the combination of Functions, Signals, and Snippets (F + S + Snip) achieved the highest overall consistency (89.13%, 95% CI [84.57%, 93.70%], $\Delta = +7.24$ pp) and shared the highest Mean VQI of 2.69 (95% CI [2.48, 2.90]). Furthermore, under the Full Context condition, the Mean VQI was 2.65 (95% CI [2.43, 2.87]), with consistency at 85.51% (95% CI [80.68%, 90.34%]).

In the test dataset, introducing RAG context was associated with higher overall verdict consistency compared to the No Context baseline (81.89% to approximately 85.5–89.1%), suggesting that retrieved evidence may help stabilize the LLM's judgments.

The Wilcoxon tests in Table 6.3 show that F + S + Snip is the only configuration that reaches significance on *both* metrics simultaneously: VQI ($W = 92.5$, $p = 0.006$) and Consistency ($W = 52.5$, $p = 0.025$). S + F + G reaches significance on VQI ($p = 0.024$) but not on Consistency ($p = 0.553$), suggesting its higher mean score may not reflect a reliable behavioural shift. The 95% confidence intervals for No Context ([77.04%, 86.74%]) and F + S + Snip ([84.57%, 93.70%]) overlap only marginally (by approximately 2.2 pp), and the Wilcoxon signed-rank test confirmed that this configuration improved inter-run agreement (p=0.025).

The reason for choosing the context combinations we did, where mostly made by looking at prompt length and how much helpful information they add compared to extra, unnecessary text. Case-Based Context introduces the most text, offering rich details but increasing the risk of adding too much to the prompt. On the other hand, Signals and Functions add very little text. Mixing these allowed us to study the trade-off between giving the model detailed context and keeping the prompt clean. To avoid bias, we tested a broad range of these combinations.

Table 6.3: Summary metrics per condition (Normal/Aligned Setting). Mean VQI and consistency are reported with 95% confidence intervals (CI). p_{VQI} and p_{Cons} report Wilcoxon signed-rank p-values for Mean VQI and Verdict Consistency respectively, each relative to the No Context baseline ($n = 46$ pairs). *F = Functions, S = Signals, Snip = Code Snippets, G = Design Guidelines, CB = Case-Based context.*

Condition	$\overline{\text{VQI}}$	95% CI	Δ VQI	p_{VQI}	Cons.(%)	95% CI	Δ Cons.	p_{Cons}
F + S + Snip	2.69	[2.48, 2.90]	+0.17	0.006*	89.13	[84.6, 93.7]	+7.24	0.025*
F + S + Snip + G	2.61	[2.40, 2.81]	+0.09	0.148	87.68	[82.6, 92.8]	+5.79	0.162
Full Context	2.65	[2.43, 2.87]	+0.13	0.067	85.51	[80.7, 90.3]	+3.62	0.297
CB + Snip + G	2.51	[2.29, 2.73]	-0.01	0.974	85.51	[80.7, 90.3]	+3.62	0.297
S + F + G	2.69	[2.49, 2.89]	+0.17	0.024*	85.51	[79.9, 91.1]	+3.62	0.553
No Context	2.52	[2.29, 2.76]	0.00	—	81.89	[77.0, 86.7]	0.00	—

*$p < 0.05$, Wilcoxon signed-rank test (two-sided).

6.2.4.3 Performance on Misaligned Data.

In the misaligned setting (Table 6.4), the interpretation of verdict quality is reversed. Because the misaligned set functions as a negative control rather than an alternative condition to be compared, a *lower* Mean VQI now indicates better performance: the expected behaviour of the system is to recognize the lack of coverage and assign a poor verdict. Across all configurations, the Mean VQI successfully dropped to approximately 1.06–1.15, and consistency was exceptionally high (93.48%–97.83%). The model is therefore not only stable but also consistently converges on the expected low-VQI judgments when presented with complete misalignment—precisely the behaviour the negative control is designed to elicit.

Table 6.4: Summary metrics per condition (Misaligned Setting). *F = Functions, S = Signals, Snip = Code Snippets, G = Design Guidelines, CB = Case-Based context.*

Condition	$\overline{\text{VQI}}$	Δ VQI	Cons. (%)	Δ Cons.
F + S + Snip	1.15	+0.01	97.83	+0.73
F + S + Snip + G	1.09	-0.05	97.83	+0.73
Full Context	1.07	-0.07	97.83	+0.73
CB + Snip + G	1.06	-0.08	94.20	-2.90
S + F + G	1.12	-0.02	93.48	-3.62
No Context	1.14	0.00	97.10	0.00

6.2.4.4 Line-by-line status distributions

When it comes to the test case execution map, Figure 6.5 visualises the proportional distribution of assigned line statuses (Green, Neutral, Yellow, Red) across the evaluated configurations. This detailed, step-by-step breakdown provides insight into how the LLM evaluates individual test lines when provided with varying levels of RAG context, under both normal (aligned) and misaligned conditions.

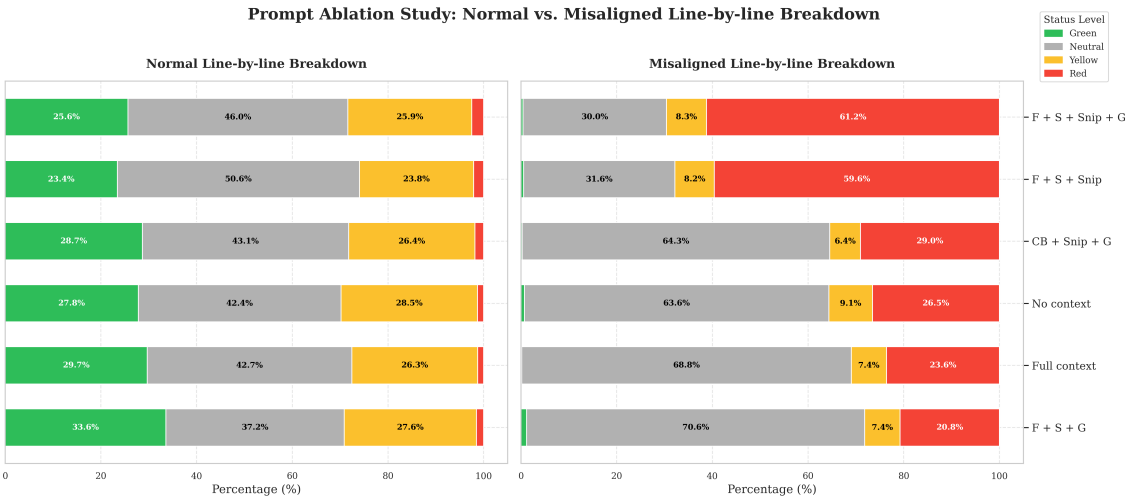

Figure 6.5: Line-by-line status distributions under normal and misaligned execution. *F = Functions, S = Signals, Snip = Code Snippets, G = Design Guidelines, CB = Case-Based context.*

During normal execution, the status distributions remain highly stable across all configurations. The majority of steps are consistently classified as Neutral (typically representing setup, infrastructure, or logging commands), Green (representing active, valid verification steps) and Yellow (partially verified). Crucially, the proportion of Red statuses, indicating severe misalignment or contradiction, remains exceptionally low (between 1.2% and 2.5%). This demonstrates that introducing structured context does not artificially inflate the system's strictness or trigger excessive false positives on standard, healthy codebases.

On the other hand, the distributions shift dramatically under misaligned execution. When evaluating misaligned test-cases, the proportion of Green steps collapses to near-zero, while Red and Yellow classifications expand significantly as the system flags the erroneous logic.

7

Cycle Three: Evaluation

Cycle 3 focuses on evaluating the artefact to assess its value and effectiveness in its intended industrial setting. The evaluation consists of a NASA-TLX questionnaire and post-task interviews conducted as part of a broader Case Study, alongside a Mutant Injection Analysis. The case study aligns well with the design evaluation methodologies defined by Hevner et al. [78]. Collectively, these methods are used to answer RQ3.

7.1 Methodology

The Case Study and the Mutant Injection Analysis serve as the two primary methods for evaluating the artefact in this cycle. The following sections describe each method in detail, outlining their purpose and how they are applied.

7.1.1 Mutant Injection Analysis

To obtain evidence of the artefact's fault detection capability, three classes of mutant test cases were constructed from the test set of 46 aligned pairs, by introducing controlled defects:

- **Near-Miss mutants**: subtle semantic deviations that preserve the surface structure of the test case while altering its logical correspondence to the requirement (e.g., a slightly different expected value or a reworded action).
- **Function mutants**: replacement of function calls.
- **Signal mutants**: substitution of signals for incorrect alternatives. (However, if there were not enough assertion commands in the markdown, mutated functions were injected instead.)

Each mutant was derived from one of the 46 aligned pairs and evaluated under the same three-run protocol and under full context, no-context and functions + signals + snippets condition, yielding $46 \times 3 = 138$ rows per mutant type and condition. Each row carries exactly three mutated line positions, so the detection rate denominator $M = 138 \times 3 = 414$ line-level observations per bar.

7.1.1.1 Detection rate (DR)

To quantify how effectively the artefact identifies injected faults, mutants, we establish a binary detection threshold based on line-status degradation. First, an ordinal rank is assigned to each line-status label: green = 3, neutral = 2, yellow = 1, and red = 0.

A fault is considered successfully detected if the injected mutation causes the system to assign a more severe (lower-ranked) status to the mutated line compared to its original aligned baseline. For a mutated step at position i in pair p, we define the detection indicator $d(p, i)$ as:

$$d(p,i) = \begin{cases} 1, & \text{if } \text{rank}\left(s_{p,i}^{\text{mut}}\right) < \text{rank}\left(s_{p,i}^{\text{aln}}\right) \\ 0, & \text{otherwise} \end{cases}$$

where $s_{p,i}^{\text{mut}}$ and $s_{p,i}^{\text{aln}}$ represent the assigned statuses for the mutant and original versions, respectively.

The overall detection rate (DR) is then calculated as the proportion of successfully detected faults out of the total number of mutated steps (M) evaluated:

$$DR = \frac{1}{M} \sum_{(p,i) \in \text{Mutants}} d(p,i)$$

This metric provides a clear, aggregate measure of how consistently the artefact correctly flags mutated test logic.

7.1.2 Case Study

To evaluate the utility of the artefact, a Case Study was conducted. This was done by allowing practitioners, gathered from convenience sampling, to do a task related to the problem the artefact was meant to support. The procedure was done by letting the practitioners asses the alignment of one group of requirements and test cases, focusing on the overall alignment of the link and how well it covers the requirement. Moreover, different professionals did the same test, however half the group was allowed to use the created artefact. Additionally, the assessment was done by injecting the links with four faults to find.

The participants represented a range of roles, levels of experience, and domain knowledge related to the specific test case and requirements. Detailed information about the participants is provided in Table 7.1. Each individual session lasted approximately 30 minutes and all but one of the session were done remotely. There was also a test session with a ninth participant use to evaluate the task. The session revealed that doing the task both with and without the artefact with the same person, took too much time, thus the splitting into two groups were decided. Due to this session worked as an evaluating ground, this session were dismissed.

Table 7.1: Participant background and roles. ID = Participant identifier; Role = professional role; Dom = domain knowledge (Yes/No); Req/Test = involvement in writing requirements or test cases as primary work (Yes/No); Experience = years of professional experience

ID	Role	Dom	Req/Test	Experience
P1	Software Engineer	N	N	10
P2	Software Engineer	N	N	15
P3	Software Engineer	N	N	4
P4	Software Engineer	N	N	5
P5	Senior Test Developer Consultant	Y	Y	4
P6	System Test Engineer	N	Y	10
P7	Software Engineer	Y	Y	1
P8	Group Design Leader	N	N	9

7.1.2.1 NASA-TLX Questionnaire

The NASA-TLX questionnaire was used in this thesis to give practitioners their own verdict on how large the workload is in manually asses alignment of links between requirements and test cases. It is also used to establish how large the mitigated workload is, when using the created artefact, or if there is any at all. Thus, this assessment focused on the evaluation of the actual usefulness of the artefact in the daily work of the practitioners. The questionnaire was answered by the practitioners after the task was completed.

The choice of using the raw version of the NASA-TLX questionnaire was motivated by the rather small sample size of the practitioners in the Case Study, and the fact that this was a subpart of a complete evaluation assessment. The important distinctions between the both are the use of weighted subclasses. The original one use them to reduce the different perceptions of workload, where the participant initially evaluate the subclasses based on their perspective. The newer version, brought up in the retrospective analysis of NASA-TLX usage by Hart [89], simply takes the average of the scores to give a verdict on the average workload. In the same paper, Hart brings up papers concluding an equal, greater and lesser differentiation in terms of sensitivity between the original version and the newer version, resulting in a rather equal sensitivity between the two.

Furthermore, the raw NASA-TLX questionnaire is consisting of six different subclasses which are rated by the participants, however, in the case of this thesis, the physical demand was removed. The reason for this was due to the only physical activity done during the tests was using an ordinary computer. This demand was therefore deemed irrelevant.

7.1.2.2 Post-Task Interviews

The evaluation interviews were semi-structured and done with all of the practitioners. For those who did the task manually, the same test case and requirement pair was assessed through the artefact, to give them insight of the artefact and its features. The interviewees were then asked to answer three questions about the output and the artefact. The questions can be seen in table 7.2. These post-task interviews can directly be connected to a specific type of controlled experiment, where insights of the usage of the artefacts is derived through interviews done in connections to the task, also proposed as a suitable way to evaluate design [78].

Table 7.2: Questions asked after using the artefact, regarding the usefulness of the artefact.

No.	Question
1	What do you think is positive about the artefact with respect to its use in the daily workflow?
2	How accurate do you feel the outcome is?
3	What do you think is negative about the artefact with respect to its use in the daily workflow?

The responses were analysed using a small-scale qualitative content analysis, inspired by Bengtsson [90]. Beneficial and concerning aspects were derived as categories, along with corresponding subcategories, supported by quotes from the interviews. No formal coding scheme was introduced, which is motivated by the relatively lightweight nature and small scale of the interviews.

7.2 Results

This section presents the evaluation results of Cycle 3. First, it details the artefact's fault detection capabilities, providing quantitative evidence of its sensitivity to logic errors. Second, it presents the findings from the Case Study, including the NASA-TLX workload assessment and the qualitative insights from the post-task interviews.

7.2.1 Mutant Injection Analysis Results

To evaluate the artefact's fault detection rate, we analysed its response to three classes of injected test-case mutations: *Near-Miss* (subtle semantic changes), *Functions* (altered function calls), and *Signals* (swapped signals).

As established in our detection logic, a fault is successfully flagged if the injected mutation causes the assigned line status to degrade to a lower rank (e.g., from Green/3 to Yellow/1) compared to the normal baseline. Table 7.3 illustrates the average status score drops across the mutated lines.

In every condition and across all mutant types, the average status score of the mutated runs dropped compared to the normal baseline. The highest sensitivity was observed in the detection of swapped *Signals* under the Full Context condition, which triggered a severe average rank drop of 0.86. Even for subtle *Near-Miss* mutations, the artefact successfully degraded the status score (drop of 0.51 in Full Context). These negative rank shifts indicate that the artefact often assigned more severe statuses to mutated lines, suggesting sensitivity to injected faults.

Building upon these rank shifts, Figure 7.1 details the Detection Rate (DR) across the different mutant types and context configurations. Each bar aggregates the same number of observations: 138 mutant runs (46 base pairs $\times$ 3 repeated runs), each contributing 3 mutated lines, for $138 \times 3 = 414$ line-level detection outcomes. Because lines within a run and runs within a pair are correlated (ICC $\approx$ 0.23–0.31, design effect $\approx$ 2.8–3.5), these observations are not independent; the clustering adjustment reduces the effective sample size to ≈ 133, comparable to the number of independent runs (138). We therefore report a single conservative clustering-adjusted 95% margin, computed at $p = 0.5$ and applied uniformly across bars ($\pm$8.5 pp). Because all detection rates fall near 50%, per-bar margins would differ by less than 0.3 pp, so a uniform value is shown. The Detection Rate quantifies the percentage of injected faults that triggered a status degradation.

Consistent with the severity of the score drops, the artefact exhibited its highest detection rates when identifying swapped *Signals* under the Functions + Signals + Snippets context (63.5%, 95% CI [55.0%, 72.0%]) and under Full Context (59.9%, 95% CI [51.4%, 68.4%]), as these represent explicit and easily verifiable logic violations. For altered *Functions*, Full Context reached 54.8% (95% CI [46.3%, 63.3%]). *Near-Miss* mutations, which capture subtle semantic deviations, were the hardest to detect, ranging from 36.2% to 45.4% across conditions.

Because the per-condition intervals are wide and overlap substantially, and because we did not test the pairwise differences directly, we do not claim that any single configuration detects faults more reliably than another. Nevertheless, the consistent directional pattern of RAG augmented conditions showing higher detection rates than No Context across nearly all mutant types. This offers indicative evidence that retrieved context improves fault detectability. Functions Signals + Snippets shows higher point estimates than Full Context for both Signal and Near-Miss mutants, though the overlapping intervals mean this ordering is suggestive rather than established.

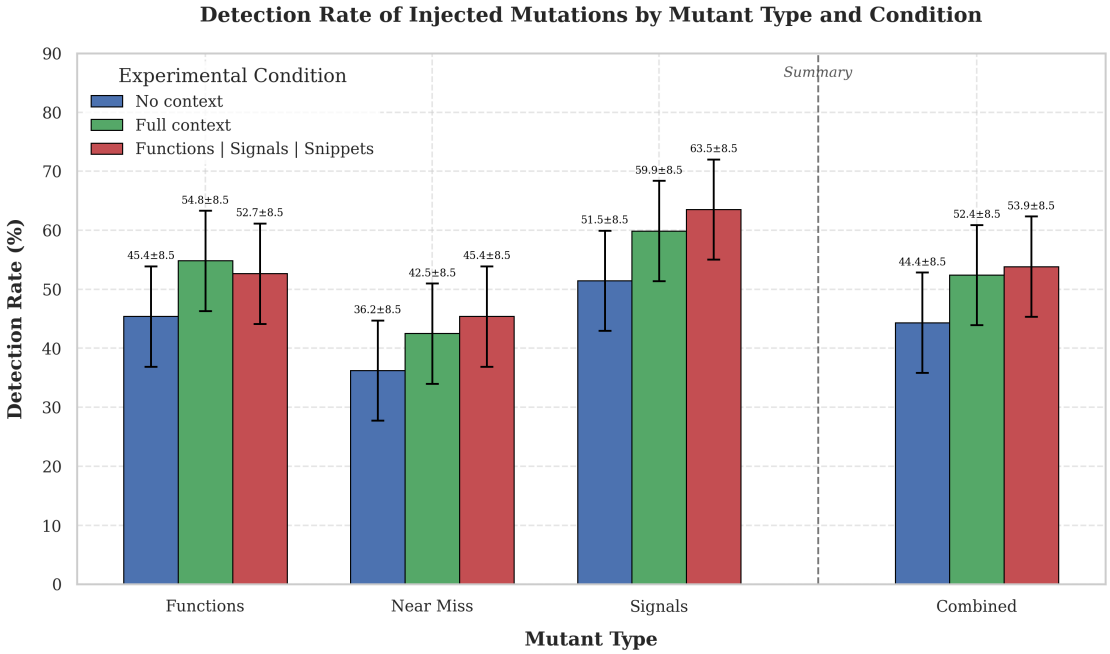

Figure 7.1: Detection rate of injected mutations per condition and mutant type. All conditions use the same number of observations (414 line-level points per bar). Error bars show a single conservative clustering-adjusted 95% margin (±8.5,pp; computed at $p = 0.5$, pooled effective $n \approx 133$, design effect ≈ 2.8–3.5), applied uniformly across bars.

Table 7.3: Average status scores of mutated lines (Green = 3, Neutral = 2, Yellow = 1, Red = 0)

Condition	Mutant Type	Normal	Mutated	Drop (Δ)
No context	Functions	1.7126	1.2874	0.4251
	Near Miss	1.7971	1.4758	0.3213
	Signals	1.8140	1.1401	0.6739
Full context	Functions	1.8575	1.2271	0.6304
	Near Miss	1.9324	1.4179	0.5145
	Signals	1.9300	1.0628	0.8671
Functions + Signals + Snippets	Functions	1.8043	1.1908	0.6135
	Near Miss	1.9203	1.3913	0.5290
	Signals	1.8551	1.0362	0.8188

7.2.2 NASA TLX Questionnaire

The NASA TLX questionnaire's intended goal was to give a verdict on the actual workload the artefact reduced for the end-user in the work of assessing alignment between requirements and test cases. Eight participants were selected, with different

roles, experience and understanding of the domain. This section goes through and dissects the results of the questionnaire.

Table 7.4: Results of participants' workload. ID = Participant identifier; Art = Artefact usage (Yes/No); MD = Mental Demand; TD = Temporal Demand; Perf = Performance (1 = best); Eff = Effort; Frus = Frustration.

ID	Art	MD	TD	Perf	Eff	Frus
P1	Y	70	60	60	55	15
P2	N	20	70	30	40	65
P3	N	95	100	95	100	95
P4	Y	45	70	90	35	45
P5	Y	60	40	20	15	10
P6	N	55	55	25	55	20
P7	N	75	85	45	80	60
P8	Y	55	65	30	70	60

Table 7.5: Mean and standard deviation (±) of workload measures grouped by artefact usage. Δ denotes the difference between conditions (Artefact = Yes minus Artefact = No).

Metric	Artefact = Yes	Artefact = No	Δ (Y–N)
Mental Demand	57.50 ± 10.41	61.25 ± 31.98	−3.75
Temporal Demand	58.75 ± 13.15	77.50 ± 19.36	−18.75
Performance	50.00 ± 31.62	48.75 ± 31.98	+1.25
Effort	43.75 ± 23.94	68.75 ± 26.58	−25.00
Frustration	32.50 ± 23.98	60.00 ± 30.82	−27.50
Overall	48.50 ± 21.95	63.25 ± 27.22	−14.75

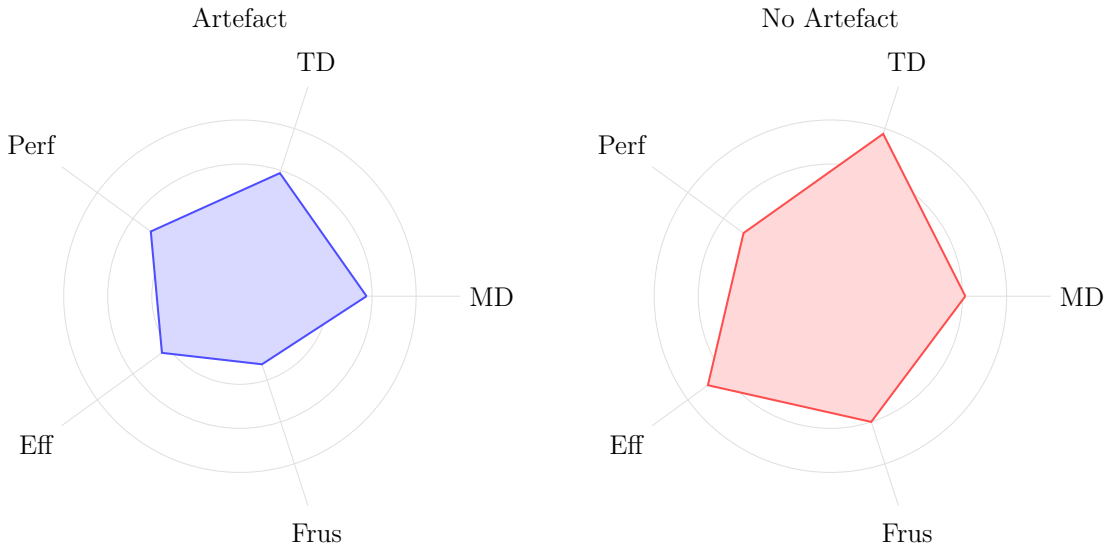

Figure 7.2: Radar plots of mean workload scores (0–100 scale) across Mental Demand (MD), Temporal Demand (TD), Performance (Perf), Effort (Eff), and Frustration (Frus) for artefact and non-artefact conditions.

Several aspects can be drawn from the results of the questionnaire, however, a large disclaimer needs to be established before diving deeper into the findings. The limited number of participants reduces the reliability of our Case Study significantly, together with the diversity in roles and levels of experience among the participants. Efforts have been made to mitigate this by assigning participants with domain knowledge or testing as their primary work to separate groups (manual task vs. artefact-supported task), with the goal of balancing the groups as much as possible.

Furthermore, as can be seen in Figure 7.2 and in Table 7.5 and Table 7.4, all dimensions, apart from performance (where lower values indicate better performance), suggest a greater workload when performing the task without the artefact compared to when using it. The largest difference can be observed in the dimension of frustration, indicating that the artefact reduces the frustrating nature of manually assessing requirements and test cases. At the same time, performance remains largely unchanged between the two conditions.

It is also worth noting that the standard deviation is generally higher in the group that performed the task without the artefact. This indicates a larger spread in the responses within that group, often suggesting more heterogeneous participants and making the mean less representative. As previously mentioned, this variability is partly expected, however, it also provides an indication that the task may be less challenging when using the artefact.

7.2.3 Post-Task Interviews

To analyse the results of the semi-structured interviews, a small-scale content analysis was conducted. The goal of the content analysis was to capture overall opinions of the artefact and its results, as well as to retrieve valuable insights regarding its potential use. The content analysis is divided into two main categorical groups, beneficial and concerning aspects, which together highlight two more concrete aspects under these titles.

7.2.3.1 Beneficial Aspects

Two different content categories were constructed from the interviews. These are *Perceived usefulness for detecting errors* and *Value as an assistant rather than replacement*. Each of these is described below, supported by quotes and examples from the interviews.

Perceived usefulness for detecting errors

Some of the participants argued that the line-by-line assessment in the test execution map can be very useful when it comes to pinpointing irregularities and identifying faults in test cases. As they highlight, it is easy to overlook small details in a test case that may be decisive for the overall test execution. At the same time, it is also easy to introduce small errors when writing test cases, and therefore having an aiding artefact can be beneficial.

> **P1:** "It is sometimes easy to miss things, especially if you are not fully focused or not paying close attention to specific variables, names, and similar details."
>
> **P2:** "I am very impressed by how easily these things can be identified, because we as humans make a lot of errors. Even when you understand that things are not aligned with the document, it is very helpful. If I have to write or understand a large test case, there is a high possibility that something can go wrong."
>
> **P7:** "Maybe if something is wrong, you can check specific lines more easily. Yeah, it is really good."

Value as an assistant rather than replacement

Some of the participants argued that the artefact could be used as an assistant for testers and software engineers, providing valuable information about what might be wrong when assessing the alignment between requirements and test cases. In this sense, it is better used as a supportive tool during the workflow to check for smaller mistakes, rather than as a fully decisive unit, which aligns with the initial purpose of the artefact.

> **P2:** "I mean, this is something we sometimes cannot see with our naked eyes, but I can use it as an assistant. I can let it do the initial review for me, and then I can go through the results myself. Since we still have a human in the loop, it makes my life easier to review those things, so it is helpful."
>
> **P8:** "I like that it explains why it reaches a conclusion and provides recommendations. For every step in the test case, you get an assessment, which is very good."

7.2.3.2 Concerning Aspects

This sections provide the two most prominent concerning aspects derived from the short semi-structured interviews. These are *Accuracy and trust concerns* and *Difficulty understanding without domain knowledge*, based with valuable and important quotes from the actual interviews.

Accuracy and trust concerns

Several participants expressed concerns when asked about the accuracy, as well as the case-based reasoning applied in the RAG system. Firstly, despite the system providing recommendations and a line-by-line rationale, some participants felt that the accuracy is difficult to determine. This concern was raised by both participants with and without domain knowledge. Additionally, even when rationale is provided, participants reported that they are not always able to judge whether the reasoning itself is correct.

Secondly, concerns were raised regarding the case-based reasoning and the snippets provided. As is broadly acknowledged in the RAG literature, any retrieval-based system is inherently dependent on the quality of its underlying knowledge base. One participant illustrated this using an analogy, noting that the reliability of the artefact's assessments is contingent on the correctness of the indexed artefacts. This is a consideration that applies to RAG systems generally, and which further reinforces the importance of treating the artefact as a decision-support tool rather than a fully autonomous assessment system.

> **P4:** "The first thing is that if the accuracy could be almost perfect, it would be much better. The higher the accuracy, the more we can trust the tool. If it gives me feedback and I'm not sure it's correct, I need to spend more effort or more time to determine whether the result is good or not."

> **P5:** "We also need to know whether the AI is actually retrieving the correct links or not. I'm not really sure here if these are the correct links or not."

Difficulty understanding without domain knowledge

Some of the participants expressed concerns that users without domain knowledge of the specific test case and requirement may struggle to understand whether the results are correct. Even though the system provides recommendations and highlights functions or signals together with a rationale, there remains a risk that users may be misled by the output. One participant also pointed out that verifying the results itself may introduce a significant amount of additional workload.

With these considerations, there is a clear concern that, even with the transparency of the information provided of the RAG system, users who are less invested or lack domain knowledge may not be able to effectively use the artefact. Thus, the artefact may be more suitable for users who possess some prior understanding of what the test case and requirement are intended to evaluate.

> **P4:** "Yeah, and that's one of it. And I'm not sure if maybe someone with domain knowledge can see the result and understand the difference quickly. For me, it's too much text — I need time to read and understand it."

> **P8:** "The only thing I question is: is it a correct result? Either I need to spend time carefully understanding and reading it, or I have to talk with the test case owner to confirm whether it fully covers the requirement."

> **P5:** "Basically, for me, because I know this stuff, I understand it directly. But for someone new, this may cause problems."

7.2.4 Summarising Aspects

To summarise the results from the content analysis, the interviews show that the artefact is generally seen as a helpful support tool, especially for spotting small errors and inconsistencies that are easy to miss. Additionally, participants appreciated the detailed, line-by-line feedback and found it useful for guiding reviews and saving effort.

At the same time, there are some clear concerns. Many participants were unsure how much they could trust the results, especially when it came to judging whether the reasoning was actually correct. Also, some noted that it can be hard to use without domain knowledge, meaning that users may need extra time or experience to fully understand and verify the output.

8

Discussion

Before conclusions can be drawn from the derived results, some discussion points need to be acknowledged. Firstly, results of each cycle and caveats connected to the specific cycle are presented, along with mitigations done towards validity threats. Secondly, a more detailed account, towards explaining the threats of validity in general for the entirety of the thesis, is presented. Lastly, the chapter reports how AI was used during the writing of the thesis.

8.1 Cycle One: Problem and Design Understanding

The selection of interviewees for the thematic analysis in cycle one followed a volunteer sampling approach, which introduces several limitations that must be acknowledged. Volunteer sampling carries the risk that participants who choose to contribute are either particularly invested in or hold strong opinions about the problem. This may lead to bias or skewed results, ultimately reducing the reliability of the findings.

In this thesis, the primary risk lies in the potential amplification of certain aspects, where the interviewees' perspectives may reflect their immediate work environment rather than the broader industry context. For example, perceptions of what constitutes a good requirement or test case may vary depending on individual preferences and local practices. Also, expectations of the artefact may be influenced by what participants perceive as most effective within their own teams, potentially addressing highly specific or niche problems rather than generalizable ones. Additionally, the volunteer sampling resulted in an unintentional selection of predominantly highly experienced participants. This risks overlooking perspectives from less experienced practitioners, which may have offered valuable and different insight into the design of the artefact.

Furthermore, by not solely relying on the thematic analysis, some potential bias is mitigated. The literature review serves as a counterweight in deciding the attributes in Cycle 1. The attributes themselves are also generally broad and do not directly correspond to the participants' specific answers, though this still leaves some room for potential bias.

The themes derived from the interviews were entirely identified by us, the authors, who have limited industry knowledge of the subject matter. Therefore, we must acknowledge that more accurate themes might have been drawn from the sessions by industry experts, which could have resulted in somewhat skewed findings. Yet again, the literature works as a secondary base to mitigate this kind of threat.

Also, the literature review is primarily directed toward the domain of TLR, which may be seen as somewhat ill-advised, as it does not fully align with the core aspect of the final artefact, a RAG pipeline. However, it still addresses a closely related problem, requirement-to-test-case link recovery, which aims to determine the unknown links between different artefacts in traceability. Furthermore, validating requirement-to-test-case relationships in automotive testing remains a largely underexplored area, which motivated us to draw insights from neighbouring domains.

Another methodological limitation within the first cycle is the use of a saturation criterion during the literature review. Because the search was discontinued once sufficient information regarding a potential domain of interest was found, there is a risk that the identified design attributes reflect our specific literature sample rather than a definitive set of attributes for the broader domain. However, this approach was chosen deliberately. The literature review was intended to inform targeted artefact design rather than to serve as an exhaustive systematic mapping of the field. Given the time-constrained nature of the project, proceeding to artefact development was a necessary priority. Had a relevant design attribute been discovered later, the DSR methodology would have allowed for iterative refinement, though significant architectural changes remained constrained by the project deadline.

8.2 Cycle Two: Design and Development of Artefact

When it comes to cycle 2 and the design choices of the artefact, there are several discussion points that need to be addressed.

First and foremost, the decisions regarding the design are based on the results from Cycle 1, however, this does not guarantee that the choice of using a RAG is the best approach. One could argue that several other AI approaches might have been more suitable for the task. Nevertheless, the decision was made based on the insights obtained from cycle 1, in combination with the characteristics of the data. The relatively small dataset, after cleaning consisting of approximately 367 test cases with linked requirements, imposed limitations on approaches such as fine-tuning or training a deep learning model. At the same time, the data exhibits a high degree of heterogeneity, with test cases and requirements originating from different parts of the vehicle and different teams within the cooperating company, making it difficult to identify common patterns. Some early attempts were made to construct a Heterogeneous Graph Neural Network (HGNN), which could have been well suited to the attributes identified in Cycle 1, however, as mentioned, the data limitations rendered this approach infeasible.

In addition, the indexing of the data, which is typically a central preprocessing step in RAG systems, was conducted somewhat differently in the artefact developed in this thesis. For instance, there is no standard chunking of the linked test cases or requirements. Instead, they are parsed independently as complete units. The only true form of chunking arises from the manual creation of snippets, consisting of 24 segments in total, twelve mutant and twelve aligned (non-mutant) examples, along with the inclusion of signals and functions. The pairs used for case-based reasoning are therefore preserved entirely in their parsed format.

One could argue that dividing requirements and test cases into smaller chunks would yield more precise and valuable context, aligning more closely with standard RAG practices. While this may indeed be true, the risk of performing chunking improperly is significant, particularly since we, as authors, do not possess the required domain knowledge and experience regarding the specific test cases and requirements. Moreover, if the artefact had focused on a single domain, this approach might have been more feasible, however, since the test cases and requirements span multiple domains, the risk of constructing incorrect chunks is high. The manually created snippets represent an attempt to address this, but are limited to examples that are sufficiently clear for us as novice students in this domain.

8.3 Cycle Three: Evaluation

As is common in RAG systems, the retrieval process, the generative results, and the end-to-end performance should all be evaluated, which can often be a challenging task [44]. However, in our case, due to the complexity of the data, evaluating all these components separately would have required significant effort. In an ideal scenario, such an approach would have enabled a more robust evaluation, but it would have demanded substantially more resources, and the purpose of this thesis is not to develop state-of-the-art methods within the RAG domain. Therefore, we consider the evaluation used in the thesis, appropriate for an exploratory DSR thesis, while acknowledging that a more comprehensive evaluation would require expert-labelled benchmarks and larger-scale studies.

The lack of explicit evaluation of the retrieval process is also partially mitigated by incorporating two filtering steps for each context category, within the retrieval pipeline. For instance, only test cases and linked requirements that share the second-highest abstraction level with the test case and requirement under assessment are eligible for retrieval through semantic search, i.e., via metadata filtering. Additionally, semantically retrieved signals and matched retrieved functions are assessed by a language model, which evaluates their suitability with respect to the targeted test case and requirement.

Furthermore, when it comes to the misaligned test cases and requirements used in the evaluation process, these are linked in an almost completely random manner, with the only constraint being that a test case cannot be paired with its original requirement(s). This introduces some problematic aspects. The approach is some-

what unrealistic, as industry practitioners are unlikely to randomly assign test cases and requirements incorrectly. Slight misalignments between similar requirements and test cases would therefore be more realistic and would have provided a better ground for evaluating the artefact. However, determining whether test cases and requirements are similar enough to be identified as a plausible misalignment requires experience and domain knowledge. Thus, while the initial approach represents a limitation, refining it would require additional effort and expertise.

In addition, a similar procedure is applied to the mutants, where pools of signals and functions are retrieved and subsequently injected, with the only constraint being that they must differ from the original ones. This approach may result in some mutants and misaligned pairs being more difficult to detect than others, leading to a varied level of detectability, and thus affect the evaluation results.

Moreover, the categories derived from the post-task interviews were identified entirely by us, which introduces a risk that they may not be the most accurate. However, given that the interview questions were directed toward fairly distinct clusters of responses, we are confident that the categories we selected are appropriate.

Lastly, the use of convenience sampling for participants in the final stage of cycle 3 enabled an efficient approach to data collection. However, one of the main drawbacks of this sampling technique is its vulnerability to bias. In our case, relying on convenient participants can introduce a risk of overly positive responses, as participants may be inclined to encourage us as students and support the project as a whole. Additionally, two of the participants had previously taken part in the interview phase during the first cycle, where volunteer sampling was the chosen strategy. This further amplifies the risk identified in the discussion of that cycle, carrying it forward into this cycle.

8.3.1 Detection Rate in an ISO 26262 Context

A critical question raised by the mutation injection results is whether a maximum combined detection rate of approximately 44–54% (computed as the average detection rate across the three mutant types per condition; with 95% CI upper bound reaching 57%) is acceptable in the automotive safety context governed by ISO 26262 [13]. The short answer is that it is not on its own sufficient to be used as a primary verification mechanism for safety-critical requirements. A detection rate of roughly 50% implies that approximately half of injected test-case faults go undetected by the artefact.

However, this conclusion must be interpreted in the context of the artefact's intended role. The system is explicitly designed as a *decision-support* tool, meant to assist human reviewers in identifying potential misalignments, not to autonomously certify test cases as compliant. The artefact's 44–54% combined detection rate still offers a meaningful preliminary filter: for signal mutations specifically, detection rates reach 63.5% under the best-performing configuration (95% CI [55.0%, 72.0%]), meaning the artefact correctly flags the majority of these explicitly verifiable signal violations.

Near-miss mutations remain harder to detect (36–45%), reflecting the inherent difficulty of capturing subtle semantic deviations without deep domain knowledge.

Additionally, the low detection rates in the mutant analysis warrant discussion. There might be several reasons why the detection rate ended up rather low, but the main reasons we believe lie in the retrieved context and the initial verdict given on aligned requirements and test cases. The retrieved context is not chunked in an effective way, and has been addressed already in this discussion, which might be the largest reason for undesirable results, due to the noise that ineffective or absent chunking introduces. This also impacts the assessment of aligned pairs, which is therefore linked to the second main problem. What we mean by this is that if the lines in the test execution map are already at a low level (e.g., red or yellow), there is less room to actually see a change, and thus it can affect the result. Some mitigation of this could be achieved by the improvements suggested in future work, presented in Section 9.1.

Furthermore, the mutation injection study used randomly sampled fault classes rather than domain-expert-crafted faults representative of real defects encountered in automotive verification. The study therefore provides a conservative lower bound on real-world utility.

To summarise, the detection rates observed in this thesis do not qualify the artefact as a sole verification measure under ISO 26262 for safety-critical systems. Rather, the results support the artefact's positioning as an intelligent screening layer that reduces the manual review burden by surfacing likely misalignments, while human engineers retain responsibility for final verification decisions.

8.4 Selection of Large Language Model

Throughout this thesis, the GPT-5.4 nano model was selected as the underlying LLM for all inference tasks, requirement parsing, relevance filtering, and alignment analysis. This choice was driven by practical constraints, as the model offers low latency, predictable cost, and company constraints. However, a natural question follows: *would a more capable model have produced meaningfully better results, and what does that imply for the findings?*

The alignment analysis task is cognitively demanding. It requires the model to simultaneously interpret structured requirement fields, follow a multi-step evaluation procedure across four scored dimensions, produce consistent line-level rationale, and integrate heterogeneous context sources such as signals, functions, guidelines, and retrieved examples. These demands push toward the upper end of what a compact model handles reliably.

Therefore, the reported results should be interpreted as an initial operational baseline. While the designed RAG pipeline, semantic filtering, and structured prompting are intended to be model-agnostic, their full efficacy when paired with a more advanced reasoning model, remains a subject for future research.

This observation also points to the importance of the retrieval and prompt engineering layers even when stronger models are available. A capable model without grounding context still risks hallucinating signal semantics or misinterpreting function intent. The infrastructure built here addresses those failure modes independently of model size. Consequently, the findings provide initial support for the architecture as a plausible approach, although further evaluation with other models is needed.

8.5 Threats of Validity

The threats to validity for this research are categorized into four primary areas: conclusion validity, construct validity, internal validity, and external validity. The following subsections detail the methodological constraints encountered throughout all the Design Science Research cycles.

8.5.1 Conclusion Validity

A fundamental threat to the reliability of the measurements in our study is the use of Generative AI, specifically the GPT-5.4 nano Large Language Model. Due to the probabilistic nature of LLMs, identical inputs can yield slightly varying outputs across different inference runs. This means the exact rationale or line-by-line assessment provided by the artefact may fluctuate, introducing inherent variance into the tool's performance. While mitigation strategies such as strict JSON output constraints and a RAG pipeline were implemented to ground the model, absolute consistency cannot be guaranteed.

Furthermore, a major constraint on the reproducibility and verification of these conclusions is the strict confidentiality agreement with the industrial partner. Because our study relies on proprietary engineering data, including raw requirements, test scripts, and internal system architecture, neither the dataset nor the artefact's source code can be made publicly available. This limitation prevents external researchers from directly inspecting the system's implementation details, replicating the exact RAG pipeline, or independently verifying the findings on the same data corpus. Additionally, the vector database represents a snapshot of a live, continuously evolving engineering repository. Exact replication of the retrieval context would require access to the exact database state at the time of extraction.

Finally, a significant threat to conclusion validity is the very small sample size in the evaluation phase (eight participants). This severely limits the statistical power of our study, meaning that the quantitative results derived from the NASA-TLX questionnaire cannot be used to draw definitive, statistically significant conclusions regarding the artefact's overall impact on cognitive workload across the wider engineering workforce.

8.5.2 Construct Validity

A threat to the construct validity of this thesis was the "reduction of human cognitive workload", which was measured using the NASA-TLX questionnaire. However, a notable threat is that the majority of participants in the evaluation phase lacked specific domain knowledge or testing experience. Consequently, the reported cognitive workload may reflect the participants' unfamiliarity with the automotive testing domain rather than the inherent difficulty of the alignment assessment task itself.

Furthermore, the evaluation of the artefact's accuracy relied on either humanly or scripted injected mutants. If these injected faults do not accurately represent the complex alignment gaps actually encountered by practitioners in real-world scenarios, the system's measured performance may not perfectly translate to tasks related to daily workflows.

8.5.3 Internal Validity

One internal threat arises from the volunteer and convenience sampling methods used to recruit participants for both Cycle 1 and Cycle 3. As discussed in the respective cycle evaluations, these sampling strategies introduce a risk of positive response bias, particularly as participants may have pre-existing connections to the researchers and may therefore be inclined to provide overly favourable feedback toward the artefact.

A further internal validity threat involves the potential lack of equivalence between the two groups compared in the workload evaluation (Artefact versus Non-Artefact). Given the small sample size and the varying levels of domain knowledge among participants, there is a risk that the observed reduction in cognitive load was influenced by the uneven distribution of prior experience rather than the artefact itself. To mitigate this confounding variable, deliberate efforts were made during the study design to manually balance the groups by evenly distributing participants with relevant domain knowledge and test-writing experience across both conditions. While it is difficult to definitively isolate the artefact as the sole cause of the workload reduction in such a small sample, the variance in the results provides additional insight, which gives some transparency in the matter.

Additionally, the thematic analysis conducted in Cycle 1 was performed by us, the researchers, who possessed limited prior industry knowledge. This introduces the risk of researcher bias, where the derived themes may be influenced by subjective interpretation rather than purely objective data. To mitigate this, the qualitative findings were continuously cross-referenced with established literature to ensure a more objective baseline.

8.5.4 External Validity

The scope of the data utilized in the knowledge base was restricted to Electronic Control Units (ECUs) governing interior and exterior vehicle functions. The struc-

tural nuances of these specific test cases, along with their corresponding requirement hierarchies, may not accurately represent other automotive sub-domains. Likewise, this limits the certainty that the artefact would generalize effectively to other safety-critical industries, like aerospace or medical devices, without substantial modification.

Finally, the sample sizes used for human evaluation, four industry professionals in Cycle 1 and eight participants in Cycle 3 are small. While this yields valuable qualitative insights suitable for the iterative steps of a Design Science Research approach, these groups are not statistically representative. Therefore, the broader generalizability of the practitioners' feedback and workflow insights to the wider software engineering community remains limited.

8.6 Usage of AI

In the writing of this thesis, LLM models such as Microsoft 365 Copilot have been used. Their usage has been restricted to refining already written text, such as correcting grammar or sentence structure. Text has not been generated entirely from simple prompts. We acknowledge the risks associated with this and have, in all cases, reviewed and verified the refined text. Everything presented in this thesis is therefore something we, as authors, stand behind.

9

Conclusion

This thesis investigated the automation of requirement-to-test alignment within the safety-critical automotive embedded systems domain. Moving beyond binary traceability link recovery, our study formulated and evaluated an automated diagnostic decision-support artefact using a Design Science Research (DSR) methodology spanning over three cycles.

Through practitioner interviews and a literature review, four foundational architectural attributes were established as design pillars for constructing the artefact. These were: semantic similarity, structural consistency, project-wide context, and hierarchical dependencies (**RQ1**). To apply these attributes in practice, an RAG system was constructed, incorporating five different types of retrievable context.

The integration of RAG context successfully improved overall model verdict consistency from 81.89% to up to 89.13% (a gain of +7.24 percentage points, Wilcoxon signed-rank $p = 0.025$, $n = 46$ pairs) compared to the baseline (**RQ2**). The same Functions + Signals + Snippets configuration also yielded the largest improvement in mean Verdict Quality (VQI), rising from 2.52 to 2.69 (+0.17, $p = 0.006$), with the Signals + Functions + Guidelines condition matching this VQI gain ($p = 0.024$). Additionally, the RAG-system demonstrated a directional improvement in mutant fault detection rates, with retrieved context associated with increased sensitivity to logic defects and small structural mutations across nearly all conditions. Detection rates peaked at 63.5% (95% CI [55.0%, 72.0%]) for swapped *Signals* and reached 54.8% for altered *Functions*, while *Near-Miss* mutations ranged from 36.2% to 45.4%, though the differences were not statistically distinguishable at the individual bar level, given the clustering corrected confidence intervals (±8.5 pp).

Furthermore, a Case Study involving eight industry professionals indicated that the tool achieved its objective of mitigating manual workload (**RQ3**). However, due to several risks of bias and a relatively small sample size, the findings are indicative rather than definitive, limiting their broader generalizability.

Altogether, the results suggest that a RAG-based architecture with domain-specific contextual data is a promising approach, for supporting automated requirement-to-test alignment assessment. The findings further suggest that providing structured, domain-specific evidence can make language-model-based assessments more consis-

tent and better grounded than relying on the model's parametric knowledge alone. However, the evaluation of the artefact lacks the scale and statistical power to draw exact conclusions about its general industrial effectiveness. Future improvements in retrieval quality and model reasoning capabilities may further increase the practical usefulness of the approach.

9.1 Future Work

Even though some insights have been provided on how an artefact can be designed to address the relatively unexplored problem of requirements-to-test alignment in the automotive industry, further work is needed. Three different perspectives are therefore presented for future investigation.

9.1.1 Evaluating State-of-the-Art Large Language Models

The first area for future research involves moving from the compact baseline model, GPT-5.4 nano, utilised in our study. Instead, a more advanced, state-of-the-art LLM should be evaluated. Future research should evaluate models with stronger reasoning capabilities. This would help determine whether larger models can resolve line-level ambiguities and reduce uncertainty flags, or whether they introduce new contextual hallucinations that could affect reliability in safety-critical environments.

9.1.2 Optimising Chunking and Retrieval

A second necessary progression is the refinement of the artefact's data processing pipeline, specifically focusing on optimizing chunking and retrieval strategies. In its current iteration, it processes complete requirements and test cases as single chunks to preserve their structural integrity. However, standard RAG best practices dictate that breaking these documents into smaller, more granular, and semantically coherent chunks can drastically improve retrieval precision. Future work should investigate more in detail chunking methodologies tailored specifically to automotive test scripts. By refining this process, the system can ensure that the LLM is fed only the most relevant contextual chunks rather than entire requirements and test cases, thereby reducing noise, reducing contextual overload, and improving overall computational efficiency.

9.1.3 Expanding the Case Study for Statistical Significance

Finally, to validate the artefact's impact, future research should expand the human evaluation phase through larger and more comprehensive case studies. The current Case Study relied on a convenience sample of eight participants. While valuable for gathering qualitative feedback and initial proof-of-concept validation, this limits the broader generalizability of the findings. Future studies should therefore involve a larger and more representative sample of domain-specific software engineers and testers. By scaling up the NASA-TLX workload assessments and tracking work-

load across a wider demographic, researchers can more reliably quantify the tool's effectiveness in reducing workload and confirm its practical usefulness.

Bibliography

[1] J. Cleland-Huang, O. C. Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software Traceability: Trends and Future Directions," in *2014 on Future of Software Engineering*, Hyderabad, India: ACM, May 2014, pp. 55–69, ISBN: 978-1-4503-2865-4/14/05. DOI: 10.1145/2593882.2593891.

[2] O. Gotel and A. Finkelstein, "An Analysis of the Requirements Traceability Problem," in *Proceedings of the 1st IEEE International Conference on Requirements Engineering (ICRE)*, IEEE, 1994, pp. 94–101. DOI: 10.1109/ICRE.1994.292398.

[3] N. Alturayeif, J. Hassine, and I. Ahmad, "Machine learning approaches for automated software traceability: A systematic literature review," *Journal of Systems and Software*, vol. 230, p. 112 536, 2025, ISSN: 0164-1212. DOI: 10.1016/j.jss.2025.112536. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121225002043.

[4] B. Wang, Z. Wang, H. Wan, X. Li, and Y. Deng, "An Empirical Study on Data Balancing in Machine Learning Based Software Traceability Methods," in *2023 International Joint Conference on Neural Networks (IJCNN)*, 2023, pp. 1–8. DOI: 10.1109/IJCNN54540.2023.10191386.

[5] M. Ruiz, J. Y. Hu, and F. Dalpiaz, "Why Don't We Trace? A Study on the Barriers to Software Traceability in Practice," *Requirements Engineering*, vol. 28, no. 4, pp. 619–637, 2023, ISSN: 1432-010X. DOI: 10.1007/s00766-023-00408-9.

[6] D. Fucci, E. Alégroth, and T. Axelsson, "When traceability goes awry: An industrial experience report," *Journal of Systems and Software*, vol. 192, p. 111 389, 2022. DOI: 10.1016/j.jss.2022.111389.

[7] B. Wang, Z. Zou, H. Wan, Y. Li, Y. Deng, and X. Li, "An Empirical Study on the State-of-the-Art Methods for Requirement-to-Code Traceability Link Recovery," *Journal of King Saud University - Computer and Information Sciences*, vol. 36, p. 102 118, 2024, ISSN: 1319-1578. DOI: 10.1016/j.jksuci.2024.102118.

[8] Z. Zou, B. Wang, P. Liang, T. Bi, and H. Jin. "Natural Language-Programming Language Software Traceability Link Recovery Needs More than Textual Similarity." arXiv: 2509.05585. [Online]. Available: https://arxiv.org/abs/2509.05585.

[9] B. Wang, Z. Zou, X. Liang, H. Jin, and P. Liang, "HGNNLink: recovering requirements-code traceability links with text and dependency-aware hetero-

geneous graph neural networks," *Automated Software Engineering*, vol. 32, Article No. 55, May 2025. DOI: 10.1007/s10515-025-00528-2.

[10] S. Maro, J.-P. Steghöfer, and M. Staron, "Software traceability in the automotive domain: Challenges and solutions," *Journal of Systems and Software*, vol. 141, pp. 85–110, 2018, ISSN: 0164-1212. DOI: 10.1016/j.jss.2018.03.060. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121218300608.

[11] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability Transformed: Generating More Accurate Links with Pre-Trained BERT Models," in *Proceedings of the 43rd IEEE/ACM International Conference on Software Engineering (ICSE)*, IEEE, 2021, pp. 324–335. DOI: 10.1109/ICSE43902.2021.00040.

[12] K. Juhnke, M. Tichy, and F. Houdek, "Challenges concerning test case specifications in automotive software testing: assessment of frequency and criticality," *Software Quality Journal*, vol. 29, no. 1, pp. 39–100, Mar. 1, 2021, ISSN: 1573-1367. DOI: 10.1007/s11219-020-09523-0.

[13] International Organization for Standardization, "ISO 26262-1:2018: Road vehicles — Functional safety," Standard, Dec. 2018. [Online]. Available: https://www.iso.org/standard/68383.html.

[14] Global Harmonization Task Force (SG3), "Quality Management Systems - Process Validation Guidance," GHTF/SG3/N99-10:2004, version Edition 2, Jan. 2004. [Online]. Available: https://www.imdrf.org/sites/default/files/docs/ghtf/final/sg3/technical-docs/ghtf-sg3-n99-10-2004-qms-process-guidance-04010.pdf.

[15] M. Mihailovici. "When software writes software," Porsche Newsroom, Accessed: Dec. 7, 2025. [Online]. Available: https://newsroom.porsche.com/en_US/2021/technology/porsche-engineering-when-software-writes-software-25367.html.

[16] eeNews Automotive. "Number of automotive ECUs continues to rise," eeNews Automotive, Accessed: Dec. 7, 2025. [Online]. Available: https://www.eenewsautomotive.com/news/number-automotive-ecus-continues-rise.

[17] D. Zax, "Many Cars Have a Hundred Million Lines of Code," *MIT Technology Review*, Dec. 3, 2012. Accessed: Dec. 7, 2025. [Online]. Available: https://www.technologyreview.com/2012/12/03/181350/many-cars-have-a-hundred-million-lines-of-code/.

[18] B. W. Boehm, "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software*, vol. 1, no. 1, pp. 75–88, Jan. 1984, ISSN: 0740-7459. DOI: 10.1109/MS.1984.233702.

[19] T. W. W. Aung, H. Huo, and Y. Sui, "A Literature Review of Automatic Traceability Links Recovery for Software Change Impact Analysis," in *Proceedings of the 28th International Conference on Program Comprehension*, ser. ICPC '20, Seoul, Republic of Korea: Association for Computing Machinery, 2020, pp. 14–24, ISBN: 9781450379588. DOI: 10.1145/3387904.3389251.

[20] S. J. Ali, V. Naganathan, and D. Bork, "Establishing Traceability Between Natural Language Requirements and Software Artifacts by Combining RAG and LLMs," in *Conceptual Modeling*, W. Maass, H. Han, H. Yasar, and N.

Multari, Eds., Cham: Springer Nature Switzerland, 2025, pp. 295–314, ISBN: 978-3-031-75872-0.

[21] O. Gotel et al., "Traceability Fundamentals," in *Software and Systems Traceability*, J. Cleland-Huang, O. Gotel, and A. Zisman, Eds. London: Springer London, 2012, pp. 3–22, ISBN: 978-1-4471-2239-5. DOI: 10.1007/978-1-4471-2239-5_1.

[22] R. Torkar, T. Gorschek, R. Feldt, M. Svahnberg, U. Raja, and K. Kamran, "Requirements traceability: A systematic review and industry case study," *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, no. 03, pp. 385–433, 2012. DOI: 10.1142/S021819401250009X.

[23] B. Ramesh and M. Jarke, "Toward reference models for requirements traceability," *IEEE transactions on software engineering*, vol. 27, no. 1, pp. 58–93, 2001. DOI: 10.1109/32.895989.

[24] Antoniol, Canfora, Casazza, and D. Lucia, "Information retrieval models for recovering traceability links between code and documentation," in *Proceedings 2000 International Conference on Software Maintenance*, 2000, pp. 40–49. DOI: 10.1109/ICSM.2000.883003.

[25] A. Marcus and J. I. Maletic, "Recovering documentation-to-source-code traceability links using latent semantic indexing," in *25th International Conference on Software Engineering, 2003. Proceedings.*, IEEE, 2003, pp. 125–135.

[26] H. U. Asuncion, A. U. Asuncion, and R. N. Taylor, "Software traceability with topic modeling," in *2010 ACM/IEEE 32nd International Conference on Software Engineering*, vol. 1, 2010, pp. 95–104. DOI: 10.1145/1806799.1806817.

[27] A. D. Rodriguez, K. R. Dearstyne, and J. Cleland-Huang, "Prompts matter: Insights and strategies for prompt engineering in automated software traceability," in *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, IEEE, 2023, pp. 455–464. DOI: 10.48550/arXiv.2308.00229.

[28] R. Solomonoff, "The time scale of artificial intelligence: Reflections on social effects," *Human Systems Management*, vol. 5, no. 2, pp. 149–153, 1985. DOI: 10.3233/HSM-1985-5207. eprint: https://journals.sagepub.com/doi/pdf/10.3233/HSM-1985-5207. [Online]. Available: https://journals.sagepub.com/doi/abs/10.3233/HSM-1985-5207.

[29] J. McCarthy, *What is Artificial Intelligence?* Stanford University Computer Science Department, 2007. [Online]. Available: http://www-formal.stanford.edu/jmc/whatisai.pdf.

[30] A. Vaswani et al., "Attention is all you need," in *Advances in neural information processing systems*, vol. 30, 2017, pp. 5998–6008.

[31] K. Cho, B. van Merrienboer, C. Gulcehre, F. Bougares, H. Schwenk, and Y. Bengio. "Learning phrase representations using RNN encoder-decoder for statistical machine translation." arXiv: 1406.1078. [Online]. Available: https://arxiv.org/abs/1406.1078.

[32] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, et al. "Language Models are Few-Shot Learners." arXiv: 2005.14165. [Online]. Available: https://arxiv.org/abs/2005.14165.

[33] R. Bommasani, D. A. Hudson, E. Adeli, R. B. Altman, S. Arora, S. von Arx, et al. "On the Opportunities and Risks of Foundation Models." arXiv: 2108.07258. [Online]. Available: https://arxiv.org/abs/2108.07258.

[34] S. Thrun, "Lifelong learning algorithms," in *Learning to learn*, Springer, 1998, pp. 181–209.

[35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." arXiv: 1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805.

[36] S. Minaee et al. "Large Language Models: A Survey." arXiv: 2402.06196. [Online]. Available: https://arxiv.org/abs/2402.06196.

[37] P. Sahoo, A. K. Singh, S. Saha, V. Jain, S. Mondal, and A. Chadha. "A systematic survey of prompt engineering in large language models: Techniques and applications." arXiv: 2402.07927. [Online]. Available: https://arxiv.org/abs/2402.07927.

[38] J. Wei et al., "Chain-of-Thought Prompting Elicits Reasoning in Large Language Models," in *Advances in Neural Information Processing Systems*, S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, Eds., vol. 35, Curran Associates, Inc., 2022, pp. 24 824–24 837.

[39] S. Yao et al., "Tree of thoughts: Deliberate problem solving with large language models," *Advances in neural information processing systems*, vol. 36, pp. 11 809–11 822, 2023.

[40] J. Long. "Large language model guided tree-of-thought." arXiv: 2305.08291. [Online]. Available: https://arxiv.org/abs/2305.08291.

[41] Y. Yao, Z. Li, and H. Zhao. "Beyond chain-of-thought, effective graph-of-thought reasoning in language models." arXiv: 2305.16582. [Online]. Available: https://arxiv.org/abs/2305.16582.

[42] P. Lewis et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," in *Adv. Neural Inf. Process. Syst.*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 9459–9474.

[43] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, et al. "Retrieval-augmented generation for large language models: A survey." arXiv: 2312.10997. [Online]. Available: https://arxiv.org/abs/2312.10997.

[44] H. Yu, A. Gan, K. Zhang, S. Tong, Q. Liu, and Z. Liu, "Evaluation of retrieval-augmented generation: A survey," in *CCF Conference on Big Data*, Springer, 2024, pp. 102–120.

[45] Y. Gao, Y. Xiong, M. Wang, and H. Wang. "Modular RAG: Transforming RAG Systems into Lego-like Reconfigurable Frameworks." arXiv: 2407.21059. [Online]. Available: https://arxiv.org/abs/2407.21059.

[46] P. Ammann and J. Offutt, *Introduction to Software Testing*, 2nd ed. Cambridge, England: Cambridge University Press, Dec. 2016.

[47] "ISO/IEC/IEEE International Standard - Systems and software engineering – Software life cycle processes. "[Online]. Available: https://ieeexplore.ieee.org/document/11481698.

[48] K. Forsberg and H. Mooz, "The Relationship of Systems Engineering to the Project Cycle," *Engineering Management Journal*, vol. 4, no. 3, pp. 36–43, 1992. DOI: 10.1080/10429247.1992.11414684.

[49] S. Lauesen, *Software Requirements: Styles and Techniques*. London: Addison-Wesley, 2002, ISBN: 0201745704.

[50] IEEE, "IEEE Standard for System, Software, and Hardware Verification and Validation," Institute of Electrical and Electronics Engineers, Standard IEEE 1012-2024, Aug. 2025. [Online]. Available: https://standards.ieee.org/ieee/1012/7344/.

[51] J. A. Ledin, "Hardware-in-the-loop simulation," *Embedded Systems Programming*, vol. 12, pp. 42–62, 1999.

[52] F. Mihalič, M. Truntič, and A. Hren, "Hardware-in-the-Loop Simulations: A Historical Overview of Engineering Challenges," *Electronics*, vol. 11, no. 15, 2022, ISSN: 2079-9292. DOI: 10.3390/electronics11152462. [Online]. Available: https://www.mdpi.com/2079-9292/11/15/2462.

[53] D. Bullock, B. Johnson, R. B. Wells, M. Kyte, and Z. Li, "Hardware-in-the-loop simulation," *Transportation Research Part C: Emerging Technologies*, vol. 12, no. 1, pp. 73–89, 2004, ISSN: 0968-090X. DOI: 10.1016/j.trc.2002.10.002. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0968090X03000792.

[54] R. Guizzardi, G. Amaral, G. Guizzardi, and J. Mylopoulos, "Ethical Requirements for AI Systems," in *Advances in Artificial Intelligence*, C. Goutte and X. Zhu, Eds., Cham: Springer International Publishing, 2020, pp. 251–256, ISBN: 978-3-030-47358-7.

[55] A. Aurum and C. Wohlin, *Engineering and managing software requirements*. Springer, 2005, vol. 1, ISBN: 9783540250432.

[56] I. Inayat, S. S. Salim, S. Marczak, M. Daneva, and S. Shamshirband, "A systematic literature review on agile requirements engineering practices and challenges," *Computers in Human Behavior*, vol. 51, pp. 915–929, 2015, Computing for Human Learning, Behaviour and Collaboration in the Social and Mobile Networks Era, ISSN: 0747-5632. DOI: 10.1016/j.chb.2014.10.046. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S074756321400569X.

[57] R. Berntsson Svensson, T. Gorschek, B. Regnell, R. Torkar, A. Shahrokni, and R. Feldt, "Quality Requirements in Industrial Practice—An Extended Interview Study at Eleven Companies," *IEEE Transactions on Software Engineering*, vol. 38, no. 4, pp. 923–935, 2012. DOI: 10.1109/TSE.2011.47.

[58] M. Unterkalmsteiner, R. Feldt, and T. Gorschek, "A taxonomy for requirements engineering and software test alignment," *ACM Trans. Softw. Eng. Methodol.*, vol. 23, no. 2, Apr. 2014, ISSN: 1049-331X. DOI: 10.1145/2523088.

[59] S. G. Hart and L. E. Staveland, "Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research," in *Human Mental Workload*, ser. Advances in Psychology, P. A. Hancock and N. Meshkati, Eds., vol. 52, North-Holland, 1988, pp. 139–183. DOI: 10.1016/S0166-4115(08)62386-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166411508623869.

[60] R. Kasauli, E. Knauss, J. Horkoff, G. Liebel, and F. G. de Oliveira Neto, "Requirements engineering challenges and practices in large-scale agile system development," *Journal of Systems and Software*, vol. 172, p. 110 851, 2021, ISSN: 0164-1212. DOI: 10.1016/j.jss.2020.110851. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121220302417.

[61] V. L. Hamilton and M. L. Beeby, "Issues of Traceability in Integrating Tools," in *Proc. IEE Colloquium Tools and Techniques for Maintaining Traceability during Design*, Dec. 1991.

[62] E. J. Uusitalo, M. Komssi, M. Kauppinen, and A. M. Davis, "Linking Requirements and Testing in Practice," in *2008 16th IEEE International Requirements Engineering Conference*, 2008, pp. 265–270. DOI: 10.1109/RE.2008.30.

[63] J. L. C. Guo, J.-P. Steghöfer, A. Vogelsang, and J. Cleland-Huang, "Natural Language Processing for Requirements Traceability," in *Handbook on Natural Language Processing for Requirements Engineering*, A. Ferrari and G. Ginde, Eds. Cham: Springer Nature Switzerland, 2025, pp. 89–116, ISBN: 978-3-031-73143-3. DOI: 10.1007/978-3-031-73143-3_4.

[64] D. Fuchß et al., "LiSSA: Toward Generic Traceability Link Recovery Through Retrieval-Augmented Generation," in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 2025, pp. 1396–1408. DOI: 10.1109/ICSE55347.2025.00186.

[65] Z. Hu, Y. Dong, K. Wang, and Y. Sun. "Heterogeneous Graph Transformer." arXiv: 2003.01332. [Online]. Available: https://arxiv.org/abs/2003.01332.

[66] H. Jin, Y. Yuan, B. Wang, H. Wan, and Z. Zou, "HGTRTracer: Advancing Requirements-Code Traceability with LLM-Augmented Attribution Reasoning and Heterogeneous Graph Transformer," in *2025 32nd Asia-Pacific Software Engineering Conference (APSEC)*, 2025, pp. 678–689. DOI: 10.1109/APSEC66846.2025.00070.

[67] J. B. Goodenough and S. L. Gerhart, "Toward a Theory of Test Data Selection," *IEEE Transactions on Software Engineering*, vol. SE-1, no. 2, pp. 156–173, Jun. 1975. DOI: 10.1145/390016.808473.

[68] IEEE. "IEEE Standard for Software Test Documentation." IEEE Std 829-1983. [Online]. Available: https://ieeexplore.ieee.org/document/3151.

[69] IEEE. "IEEE Standard for Software Test Documentation." IEEE Std 829-1998. [Online]. Available: https://ieeexplore.ieee.org/document/741968.

[70] IEEE. "IEEE Recommended Practice for Software Requirements Specifications." IEEE Std 830-1998. [Online]. Available: https://ieeexplore.ieee.org/document/720574.

[71] ISO/IEC/IEEE. "Systems and Software Engineering — Life Cycle Processes — Requirements Engineering." ISO/IEC/IEEE 29148:2011(E). [Online]. Available: https://ieeexplore.ieee.org/document/6146379.

[72] ISO/IEC/IEEE. "Systems and Software Engineering — Life Cycle Processes — Requirements Engineering." ISO/IEC/IEEE 29148:2018(E). [Online]. Available: https://ieeexplore.ieee.org/document/8559686.

[73] B. W. Boehm and V. R. Basili, "Software Defect Reduction Top 10 List," *Computer*, vol. 34, no. 1, pp. 135–137, Jan. 2001. DOI: 10.1109/2.962984.

[74] B. W. Boehm, "Industrial software metrics top 10 list," *IEEE Software*, vol. 4, pp. 84–85, 1987. [Online]. Available: https://api.semanticscholar.org/CorpusID:237072980.

[75] A. Bertolino, "Software Testing Research: Achievements, Challenges, Dreams," in *Future of Software Engineering (FOSE '07)*, Minneapolis, MN, USA: IEEE Computer Society, 2007, pp. 85–103. DOI: 10.1109/FOSE.2007.25.

[76] H. K. V. Tran, M. Unterkalmsteiner, J. Börstler, and N. bin Ali, "Assessing test artifact quality—A tertiary study," *Information and Software Technology*, vol. 139, p. 106 620, 2021, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2021.106620. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584921000938.

[77] H. K. V. Tran, N. b. Ali, M. Unterkalmsteiner, J. Börstler, and P. Chatzipetrou, "Quality attributes of test cases and test suites–importance & challenges from practitioners' perspectives," *Software quality journal*, vol. 33, no. 1, p. 9, 2025.

[78] A. R. Hevner, S. T. March, J. Park, and S. Ram, "Design science in information systems research," *MIS quarterly*, vol. 28, no. 1, pp. 75–106, 2004.

[79] E. Knauss, "Constructive Master's Thesis Work in Industry: Guidelines for Applying Design Science Research," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering Education and Training (ICSE-SEET)*, 2021, pp. 110–121. DOI: 10.1109/ICSE-SEET52601.2021.00021.

[80] E. Engström, M.-A. Storey, P. Runeson, M. Höst, and L. Baldwin Minku, "How software engineering research aligns with design science: a review," *Empirical Software Engineering*, vol. 25, pp. 2630–2660, 2020. DOI: 10.1007/s10664-020-09818-7.

[81] 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, Dec. 2008, ISSN: 1573-7616. DOI: 10.1007/s10664-008-9102-8.

[82] D. S. Cruzes and T. Dyba, "Recommended Steps for Thematic Synthesis in Software Engineering," in *2011 International Symposium on Empirical Software Engineering and Measurement*, 2011, pp. 275–284. DOI: 10.1109/ESEM.2011.36.

[83] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '14, London, England, United Kingdom: Association for Computing Machinery, 2014, ISBN: 9781450324762. DOI: 10.1145/2601248.2601268.

[84] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, 2015, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2015.03.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584915000646.

[85] S. Hassan, Q. Li, K. Aurangzeb, A. Yasin, J. A. Khan, and M. S. Anwar, "A systematic mapping to investigate the application of machine learning techniques in requirement engineering activities," *CAAI Transactions on In-*

telligence Technology, vol. 9, no. 6, pp. 1412–1434, Dec. 2024, ISSN: 2468-6557, 2468-2322. DOI: 10.1049/cit2.12348. [Online]. Available: https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/cit2.12348.

[86] G. Zhao, T. Li, and Z. Yang, "An Extended Knowledge Representation Learning Approach for Context-Based Traceability Link Recovery: Extended Abstract," in *2020 IEEE Seventh International Workshop on Artificial Intelligence for Requirements Engineering (AIRE)*, 2020, pp. 22–22. DOI: 10.1109/AIRE51212.2020.00010.

[87] A. Beer, M. Junker, H. Femmer, and M. Felderer, "Initial Investigations on the Influence of Requirement Smells on Test-Case Design," in *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, 2017, pp. 323–326. DOI: 10.1109/REW.2017.43.

[88] H. Gao et al., "Using Consensual Biterms from Text Structures of Requirements and Code to Improve IR-Based Traceability Recovery," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Rochester, MI, USA: IEEE, 2022. DOI: 10.1145/3551349.3556948.

[89] S. Hart, "Nasa-task load index (Nasa-TLX); 20 years later," vol. 50, Oct. 2006. DOI: 10.1177/154193120605000909.

[90] M. Bengtsson, "How to plan and perform a qualitative study using content analysis," *NursingPlus Open*, vol. 2, pp. 8–14, 2016, ISSN: 2352-9008. DOI: 10.1016/j.npls.2016.01.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2352900816000029.