Received: 20 June 2021 | Revised: 11 December 2021 | Accepted: 17 January 2022

DOI: 10.1002/stvr.1811

RESEARCH ARTICLE

WILEY

Towards using coupling measures to guide black-box integration testing in component-based systems

Dominik Hellhake[1,2] | **Justus Bogner**[2] | **Tobias Schmid**[1] | **Stefan Wagner**[2]

[1]BMW Group, Munich, Germany

[2]Institute of Software Engineering, University of Stuttgart, Stuttgart, Germany

Correspondence
Dominik Hellhake, BMW Group, Knorrstra?e 147, 80788 Munich, Germany.
Email: dominik.hellhake@bmw.de

Abstract

In component-based software development, integration testing is a crucial step in verifying the composite behaviour of a system. However, very few formally or empirically validated approaches are available for systematically testing if components have been successfully integrated. In practice, integration testing of component-based systems is usually performed in a time- and resource-limited context, which further increases the demand for effective test selection strategies. In this work, we therefore analyse the relationship between different component and interface coupling measures found in literature and the distribution of failures found during integration testing of an automotive system. By investigating the correlation for each measure at two architectural levels, we discuss its usefulness to guide integration testing at the software component level as well as for the hardware component level where coupling is measured among multiple electronic control units (ECUs) of a vehicle. Our results indicate that there is a positive correlation between coupling measures and failure-proneness at both architectural level for all tested measures. However, at the hardware component level, all measures achieved a significantly higher correlation when compared to the software-level correlation. Consequently, we conclude that prioritizing testing of highly coupled components and interfaces is a valid approach for systematic integration testing, as coupling proved to be a valid indicator for failure-proneness.

KEYWORDS

distributed systems, integration testing, software coupling, software faults, software metrics

1 | INTRODUCTION

The development of component-based software systems is a commonly used and widely adopted development approach which supports the design of reusable software components as well as their integration into existing software systems [1]. In literature, a large set of metrics exists to measure and control the quality of component-based systems. Commonly evaluated internal software design properties are *size*, *complexity*, *coupling* and *cohesion* [2]. These attributes influence a system's understandability, testability, maintainability and reusability [3] and can help to identify and eliminate design flaws.

Since individual components of such systems are often developed by independent teams, it is very important to verify the correctness of their integrated functionality once they are assembled into a complete system. This verification is referred to as *integration testing* [4] and is a prerequisite for the final system-level testing. Multiple approaches exist for integration testing of a component-based system like incremental, top down, bottom up or big bang [4]. All these approaches assume that components have already been tested separately in a controlled environment. Integration

testing therefore focuses on the verification of component interplay behaviour to reveal failures caused by erroneous interface implementations or design flaws introduced at the system level. In addition, a test strategy used for integration testing prescribes how to prioritize test case selection to achieve the goals of integration testing in an efficient way with limited time and resources.

In the automotive industry, integration testing is split into multiple stages as a modern vehicle consists of over 200 interconnected electronic control units (ECUs), each of it executing a component-based software system on its own. After software integration testing and testing of each ECU individually, multiple ECUs are put together to form a coherent subsystem of the vehicle during system integration testing. The goal of this step is to verify the interactions of multiple ECUs during execution of a higher-order functionality. In a previous work, we demonstrated how the complete set of ECU interactions can be identified in a given system design based on the usage of interfaces [5]. To accomplish full test coverage, each of the identified component interactions should be verified explicitly. However, this is not feasible for large systems due to the potential combinatorial complexity of functional dependencies, which can lead to substantial testing efforts.

In literature, there are very few formally or empirically validated approaches available for the systematic testing of component interactions in a component-based system. Such approaches become even more important when the software of most ECUs are considered to be black-box, which are, for example, developed by external providers. In this context, code-based approaches to identify failure-prone component interactions are not applicable. Therefore, system integration testing has to be performed solely on interface- and black-box behaviour specifications of each ECU. Because those interface specifications are available early in the development process of a vehicle, we propose an approach based on coupling measures to identify failure-prone component interactions.

By answering the following research questions, we suggest that measures for coupling applicable for black-box components can provide a valuable basis for test case prioritization during system integration testing. We test this hypothesis in an automotive case study, namely, if ECUs with strong coupling or with high interface complexity have an increased probability of containing faults typically found during system integration testing. As a supporting proposition, we assume that due to cognitive overloading, not all information of a software component's context is considered during its development or evolution.

RQ1: Which coupling measures applicable to the black-box *component structure* of a modular system are useful to guide the test case prioritization for system integration testing?

RQ2: Which coupling measures applicable to the black-box *interface structure* of a modular system are useful to guide the test case prioritization for system integration testing?

In the conducted case study, we analysed the relationship between proposed coupling measures found in scientific literature and the failure distribution identified during the system integration testing of a real-world automotive system. By investigating the correlation for each measure, we provide insights about its usefulness for test case prioritization during the integration testing of black-box components. A strong correlation between the occurrences of failures and design measures for software interfaces would greatly contribute to the definition of a systematic method for black-box integration testing. Additionally, our correlation analysis of existing coupling measures provides valuable experiences in applying them to a real-world system from the automotive domain.

This paper is organized as follows: in Section 3, we first provide related work regarding the term coupling and its definition found in literature. We also provide related work regarding the utilization of information about the software architecture to improve testing activities. In Section 2, we define the notion of coupling based on related work. Afterwards, the set of existing measures studied in this work are explained in detail (see Sections 2.2,2.3 and 2.4). In Section 4, the study design is presented: The real world system is introduced, and the data selection for the failure distribution found during integration testing of the system is presented. Finally, the research questions and hypotheses are defined. Section 5 presents the correlations between the measured values and the failure distribution. Lastly, we briefly discuss the observed results in Section 6, mention threats to validity (Section 7) and conclude with a summary of the paper in Section 8.

2 | BACKGROUND

As discussed in the previous section, much work exists defining coupling and complexity metrics for component-based software systems. However, most of these definitions are based on a white-box or at least grey-box perspective onto the structure of the system under test. In automotive industry, these architectural views are in most cases not available which makes most of the defined metrics inapplicable. The architectural views commonly used in the automotive industry are presented in Section 4.1 as part of the context of this study. In this section however, we present a selection of metrics for this correlation study which are applicable even if the software component structure is not available. In Section 2.3, a data-flow-based approach of measuring coupling is presented. In Section 2.3, a dependency-analysis-

based approach of measuring coupling is presented which also works for non-data-flow-based coupling types. Lastly, an information-theory-based approach for measuring coupling is presented in Section 2.4.

2.1 | Coupling in component based systems

Many studies proposed coupling and complexity metrics for component-based software systems [6], but comparatively few works exist which study the correlation of such measures to the fault-proneness of component interactions typically verified during integration testing.

Constantine and Yourdon [7] suggested that modularity of software design can be measured with two qualitative properties: cohesion and coupling. While cohesion focuses on the semantic relatedness of a software module's functionality, coupling describes the degree of interdependence among multiple software modules. Based on these two concepts, a common design goal for a software module is to increase its internal cohesion while keeping its external coupling as low as possible. The terms *software module* and *software component* are commonly used in literature to describe the basic building blocks of structured software design. In this work, both terms are used interchangeably.

One motivation for increasing cohesion while keeping coupling as low as possible is provided by Martin [8], who introduced the term *instability* as the probability that a module has to be changed in response to a previous change of a coupled module. To determine the instability of a software module, Martin distinguishes between efferent and afferent coupling. Efferent coupling C_e exists for a module A if A depends on another module B in terms of a `use` relation to one or more of its provided functionalities. Martin states that a high degree of efferent coupling results in high module instability because any change to one of the functions of module B used by module A may cause subsequent changes in module A. Afferent coupling C_a on the other hand exists for module A if its functions are used by other modules. Consequently, Martin states that a high degree of afferent coupling leads to high module stability due to the responsibility of the module to the rest of the system. The quantification of a software modules instability is given by the ratio between the number of efferent coupling to the overall number of couplings: $I = (C_e/(C_a + C_e))$. The value of I can range from 0 expressing high stability to 1 indicating low stability.

Santos et al. [9] investigated the acceptance of this instability measure in a literature review. They found that the instability measurements stayed relatively constant during the development of the studied systems. They concluded that, among the analysed open-source projects, there would be little awareness of a software modules susceptibility to subsequent changes caused by module coupling. However, they further identified a lack of methods to identify afferent and efferent couplings in an existing software design.

In a work published by Abdellatief, a generic component model which is commonly used in coupling analysis is discussed [10]. This model emphasizes the separation between interfaces and their implementation. It focuses on `use` relations between multiple components and their provided interfaces. In addition, the directness of data-flow-based coupling is shown. With a direct data flow, the sending component communicates the data directly to the receiving component while with an indirect data flow, the sending component communicates the data to the receiver using at least one intermediate component.

The identification of existing coupling in a given software design based on interface specifications is further detailed by Gill and Grover [11], who provides an interface characterization model. Within this model, an interface is described based on its packaging information, a set of constraints, non-functional properties and the interface signature. Furthermore, Gill introduces the term interface complexity to describe a software module's potential to be reused and integrated in a different software design. To identify existing dependencies of a software module, an interface signature provides the set of potential endpoints of coupling in terms of operations, events and properties.

The different aspects covered by the six generic types of coupling discussed by Myers [12] and the inconsistent usage of the term coupling in software design literature is discussed in a work presented by Xia [13]. Xia argues that coupling is often only considered between two modules, which would neglect frequent cases in real-world software systems, for example modules connected to other modules and multiple modules connected to one module. To alleviate this, Xia provides a brief definition for coupling given in equation 1, which relies on a single data flow connecting multiple modules. In his definition, coupling is measured as a composition of the shared data complexity $C[d]$, the potential impact on the dynamic behaviour of coupled components $PC[d]$ and the number of modules coupled to d, that is, $N[d]$. Consequently, coupling of a software module is defined as the sum over the module's out-flowing data $MC(m) = \sum_{i=1}^{n} EC[d_i]$.

$$EC[d] = C[d] * PC[d] * N[d] \tag{1}$$

In accordance to the definition of module coupling in equation 1, this work studies different measures for the number of dependencies $N[d]$. The presented related work showed that defining a sound measure for the data complexity C

[*d*] and the degree of impact on the dynamic behaviour of a coupled module *PC*[*d*] is sensitive to the programming paradigm, type of system and the availability of code. As the goal of this work is to study coupling measures applicable for software systems containing black-box modules, the identification of applicable measures for data complexity and program control is left open for future work. Consequently, coupling among multiple modules is considered of the same type and impact strength.

The coupling measures described in this work solely take quantitative aspects of coupling into account while leaving additional facets like strength or type out for future work. The measures have been selected out of existing literature and are used in Section 5 for correlation analysis with a failure distribution. In Sections 2.2 and 2.3, conventional approaches are described to measure coupling based on the counting of software modules or the number of data flows. In Section 2.4, an approach is described which measures coupling based on the amount of entropy created in a system by a certain data flow.

2.2 | Data flow analysis

Data flow analysis is a common approach for identifying dependencies in a structured software design [14,15]. When utilizing a system's inner connectivity in terms of data flow, coupling analysis results in a directed graph in which each edge represents a sender–receiver relation for a certain piece of information communicated between two software modules.

Abdellatief defines **interface coupling** (*IC*) within the context of data flow analysis in equation 2 as the number of out-flowing data (*OF*) over all operations (*p*) of an interface multiplied by the number of receiving components *n*. This definition captures the effect of afferent coupling as mentioned in Section 2. Abdellatief further assumes that a large number of afferently coupled components result in more required context-related information to test and maintain a software module. At the component level, **component coupling** (*CC*) is defined as the sum of all interface coupling factors of all interfaces implemented by that component, that is, $CC = \sum_{i=1}^{n} IC_i$. Similarly, the component-based system coupling *CBSC* is defined as the sum of all component coupling factors for each component contained in the system, that is, $CBSC = \sum_{i=1}^{n} CC_i$.

$$IC = n * \sum_{i=1}^{p} OF_i \tag{2}$$

Henry and Kafura provide a brief discussion of information-flow-based module complexity [16,17]. Similarly, Kumari provides an empirical and theoretical analysis of information-flow-based complexity [18]. The measure introduced in his work is shown in equation 3. Similar definitions are provided by Lakshmi [19] and Kharb [20]. In contrast to coupling measures like *IC*, *CC* and *CBSC*, information flow measures focus on a component's inner dependencies among input and output parameters. A module with a high value for information flow measurement indicates that it is strongly connected to its environment thus indicating a high coupling.

$$IIF = (Fan - in * Fan - out)^2 \tag{3}$$

The **interface information flow** (*IIF*) is calculated based on the squared product of the number of data flowing into the component ($Fan - in$) and the number of data flowing out of the component ($Fan - out$). In this definition, multiplying the number of in-flows and out-flows represents all possible combinations of functional dependencies between input and output parameters. Squaring that number represents the assumption that complexity may not scale with the number of parameters in a linear way. For component and system level complexity, the interface information flow *IIF* values are accumulated to the **component information flow** $CIF = \sum_{i=1}^{n} IIF_i$ and further to the component-based software information flow $CBSIF = \sum_{i=1}^{n} IIF_i$, which is similar to the definition of coupling.

2.3 | Dependency analysis

While data flow analysis is designed to measure coupling based on data shared between multiple software modules, dependency analysis measures are designed to be applicable to any type of dependency discussed in Section 2. Multiple articles exist proposing the use of a matrix model to manage dependencies in a component-based system [21-24]. Furthermore, these articles suggest that any dependency graph of a component-based system can be represented as a dependency matrix. In such a matrix, each component is represented by a column and a row. If a component c_i is dependent

on another component c_j the value at the position ij within the matrix equals to 1 as defined in equation 5. This matrix representation of component interdependencies can be applied to both directed and undirected dependencies. Given such a matrix representation of the system, Li defines the **dependency coefficient** (DC) for a particular component as the sum of its row and column values as shown in equation 6. Because all values per row and column are added up, ingoing and outgoing dependencies are weighted equally. The dependency coefficient of a component captures its coupling to the rest of the system, as it scales with the number of dependent components. However, besides being applicable for directed and undirected dependencies, the dependency coefficient differs further from IC and CC because it does not scale with multiple dependencies between the same two components.

$$DM = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \dots & \dots & \dots & \dots \\ d_{n1} & d_{n2} & \dots & d_{nn} \end{bmatrix} \tag{4}$$

$$d_{ij} = \begin{cases} 1 & \text{if } c_i \rightarrow c_j \\ 0, & \text{otherwise} \end{cases} \tag{5}$$

$$DC(C_k) = \sum_{j=1}^{n} [d_{kj}] + \sum_{j=1}^{n} [d_{jk}] - 2[d_{kk}] \tag{6}$$

2.4 | Information theory

In contrast to counting components, connections and interface compositions, Allen and Anan studied an approach to measure coupling in component-based systems based on information theory [25-28]. According to Allen, a software system can be represented as a graph in different ways to highlight different aspects of its design. Common examples of such design abstractions for object-oriented designs are given in a framework proposed by Briand, Daly and Wüst [29,30], which is used to measure coupling and cohesion based on class inheritance, method invocation or class-attribute references. In summary, Allen states that a more sophisticated measurement than just the count of features of an artefact will be more useful in determining coupling and cohesion [26].

In information theory, coupling and complexity are measured based on the entropy of the distribution of row patterns $H(S)$ [31]. Applied to the original system graph, the entropy depicts the average information per node which can be multiplied by the number of nodes in the system to get the overall amount of information $Size(S)$. To measure the size of a system, it is important to exclude the information added by dependencies to the system's environment, which is denoted in the definition of $Size(S)$ as $-\log p_{l(0)}$. In addition to the system-level size measure, size can also be calculated for a module as well as for a node of the system. While module size can be calculated as defined in Table 1, the size of a node requires the calculation of the node's subgraph S_i, which contains all nodes and modules of the system, but only edges which are connected to the node i. The size of a node subgraph provides the amount of information the node contributes to the system. Understanding the usefulness of the size measure in information theory becomes easier if it is calculated based on the system's data flow graph as described in Section 2.2. Entropy within the system $H(S)$ represents the average amount of data shared by a module, while the system's size $Size(S)$ represents the overall amount of data shared between all modules of the system.

TABLE 1 System- and module-level measures [26]

			Entropy-based coupling measures
1	$H(S)$	=	$\sum_{i=1}^{n_s} p_l(-\log p_l)$
2	$Size(S)$	=	$(n+1)H(S) - (-\log p_{l(0)})$
3	$Size(m_l \mid S)$	=	$\sum_{i \in m_k} (-\log p_{l(i)})$
4	$Complexity(S)$	=	$\sum_{i=1}^{n} Size(S_i) - Size(S)$
5	$Complexity(m_k \mid S)$	=	$\sum_{i \in m_k} Size(S_i) - Size(m_k \mid S)$
6	$Coupling(MS)$	=	$Complexity(MS^*)$

Furthermore, the system's complexity is defined as the excess entropy of the sum of information of each node in relation to the overall information of the system, which can be equal or greater: $\sum_{i=0}^{n} H(S_i) >= H(S)$. This phenomenon is extended to the calculation of complexity for a given system graph S as shown in Table 1. According to Allen, excess entropy represents the average information in relationships, which indicates the strength of coupling for each node.

Coupling of a system graph is calculated based on the complexity of the intermodule relationships and therefore based on the amount of information represented by edges with nodes in different modules. To calculate coupling, the complexity calculation is applied to the system's intermodule edge graph. Due to the fact that the system graph used in this approach is undirected, the coupling measure is not affected by the direction of relationships, which is different to other coupling measures discussed in this work.

3 | RELATED WORK

In this section, we first present related work regarding test case prioritization in general in order to support the identification of the test gap in Section 4.3. The second part of this section provides related work regarding the utilization of coupling and complexity measures to improve testing.

Hao proposes a unified test case prioritization technique which combines the principles of total and additional test case prioritization strategies[32]. Hao first states that most existing research on test case prioritization focus on the prioritization itself and follow one of two overall strategies. The first strategy a *total* strategy sorts test cases according to the number of elements that they cover. The second, *additional* strategy on the other hand repeatedly selects test cases that cover the maximal elements not yet covered by previously prioritized test cases. Based on that, Hao provides his definition of a unified test case prioritization which he evaluates using 40 C and 28 Java programs. His results demonstrate that the fault detection probability of his proposed technique in general reside between those using purely *total* or *additional*.

In a work presented by Elbaum, the trade-offs between fine granularity and coarse granularity prioritization techniques is studied [33]. When performing test case prioritization at a fine granularity, test cases will be selected based on code coverage. An example for a coarser granularity would be a function level coverage analysis at which test cases are selected based on the set of functions examined during execution. Elbaum used 16 different approaches across these granularities in his studies. In his results, Elbaum shows that test case prioritization at both function-level and statement-level showed similar rates of fault detection which makes test case prioritization at a coarse level more cost effective. In addition to the level of granularities, Elbaum studied the effects of incorporating measures of fault proneness onto the fault detection rate of test case prioritization techniques. He found that measures of fault proneness can significantly improve the effectiveness of test case prioritization. However, his results show that the improvement was comparatively small and not consistent when compared to other techniques which suggests that information about fault proneness may not be intuitive and obvious.

Given the definition of coupling and the available set of applicable measures, the second part of this related work section provides an overview of existing work regarding the utilization of architectural measures for software testing. Mendes et al. conducted a systematic mapping study to investigate existing approaches to use information about the software architecture to improve the testing activities [34]. In their work, the authors included 27 studies ranging from the improvement of software testing management in general to the automatic generation of test cases based on software architecture specifications. They found that most of the studies were published in the last 10 years, suggesting that the research topic only emerged recently. In addition, Mendes et al. point out that out of 27 included studies only two were conducted in industry, which creates the impression that there is little to no adoption of the proposed approaches.

Selby studied the ratio and strength of coupling and its relation to the error-proneness of interacting components of the system [35]. For this, Selby used a real-world software system including 77 subsystems implemented with 148,000 lines of code. Coupling is measured based on the number of data bindings between multiple routines similar to the data-flow-based coupling analysis described in Section 2.2. He found that routines with the lowest coupling ratio showed significantly less errors. In addition, for those routines, existing errors were less costly to fix. The goal of Selby is similar to the goal of this work. However, the included measures for coupling, size and strength as well as the methodology of data collection and analysis are not directly applicable for this work due to the unavailability of the code of most components contained in the system.

Singh et al. conducted an empirical study on different versions of open-source software to analyse the benefit of various object-oriented metrics for the quality of the software [36]. In particular, a technique of test case prioritization based on the fault-proneness of the software modules is presented, which is very similar to the approach presented in this work. However, due to the fact that the presented test case prioritization technique is designed to be applicable for object-oriented software systems, coupling is defined and measured based on a dependency graph showing class level

coupling. In their results, Singh et al. show that object-oriented coupling measures like *number of children* (NOC), *coupling between objects* (CBO) or the *depth of inheritance* (DIT) do affect the software quality and can therefore be used for test case selection and prioritization.

Richardson introduced the chemical abstract machine (CHAM) model, which formally specifies software architecture based on interconnected components, internal states and state-transformation rules to derive a set of architecture-based testing criteria [37]. In a previous work, we defined similar test criteria and evaluated them against the fault distribution in a real-world software system [5].

As demonstrated by Mendes, most related work regarding the use of information about the software architecture to improve software testing activities directly addresses the generation of test cases based on architecture specification. However, very few works exist studying the use of architecture specifications to measure test coverage or to provide guidance for test case selection and prioritization, specifically for integration testing activities. In addition, very few of the existing studies were conducted in industry, leaving the empirical evaluation of the suggested approaches open. In this work, we directly try to address these aspects.

4 | CASE STUDY DESIGN

In this section, we briefly introduce the case study design which is organized according to the case study design and planning structure proposed by Runeson [38]. The overall structure of the case study design is summarized in Figure 1. In Section 4.1, the architectural concepts and design paradigms typically used in automotive software systems are described in short to provide a broad description of the context of this study and to underline the objective of this study. In Section 4.2, we briefly introduce the chassis control system whose development project is used in this study as the case. In Section 4.3, the research questions are defined as the basis for the evaluation. At the end of this section, the units of analysis are presented in Sections 4.4 and 4.5 as well as in Section 4.6.

4.1 | Context of the study

A modern car resembles a distributed software system implementing many software-based functions. These functions are structured into different functional domains of a vehicle, ranging from clusters of classical driving functions up to super-ordinate functions regarding driving assistance and automated driving. A major challenge for the development of modern cars is that these functions are highly dependent on each other. In particular, many functions for driving assistance or automated driving are highly sensitive to the operational state of classical driving functions like brake or steering control. To identify and control these functional dependencies and feature interactions early in development, different structural views on the architecture of cars are used. Broy [39] provided an overview of the different levels of architectural models used for the development of cars as shown in Figure 2.

The usage of the presented architectural views is organized alongside the V-Model, which is widely adopted in the automotive industry. On the highest level, the *function hierarchy* is derived from functional and non-functional requirements. The main purpose of this hierarchy is to describe the software-based functionality implemented in the car to the user which, according to Broy, not only contains the driver but also maintenance and production staff and other individuals potentially interacting with the car.

Based on the functional hierarchy, the system design is derived in its *logical architecture*. At this level, the car is already decomposed into a distributed system of interacting components and subfunctions. One of the main goals of

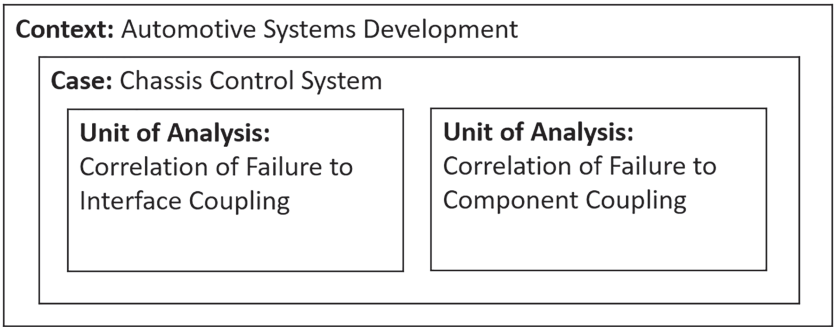

FIGURE 1 Embedded case study design according to Runeson [38]

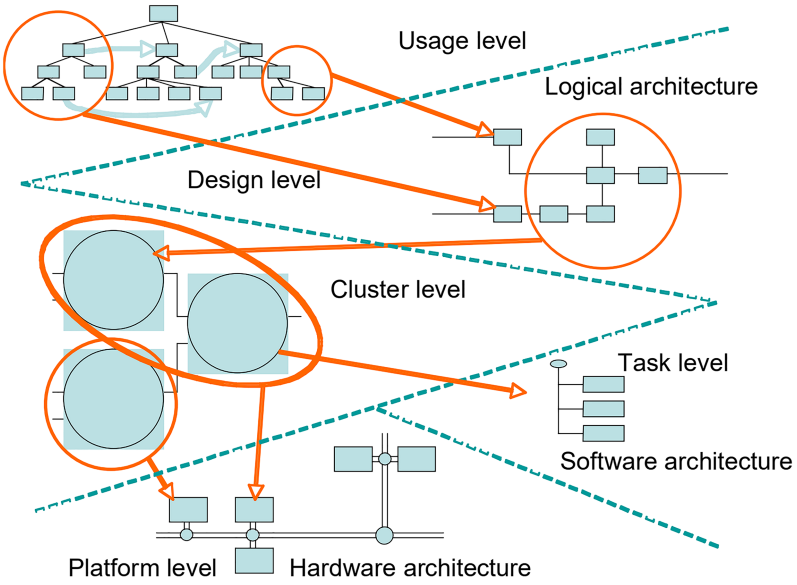

FIGURE 2 Architectural views on automotive systems [39]

this architectural level is to describe the observable behaviour on a functional level based on feature and function interaction while keeping each component and subfunction as independent from its implementation as possible. In a recent work, service-oriented architectural concepts are used at this architectural level to decompose the functional hierarchy into services [40-42].

At the *cluster level*, functional components of the logical architecture are grouped to form coherent functional clusters. The main goal of this clustering is to provide an intermediate step between the implementation (independent from the logical design level) and the highly implementation-dependent software and hardware architecture. The high-level *software architecture* is directly derived from the cluster-level view and its logical components. Furthermore, the software architecture describes the structuring of the operating system including the required hardware drivers and communication stacks as well as the scheduling of tasks.

In contrast to the software architecture, the decomposition of a vehicle in a set of ECUs, sensors and actuators are described by the *hardware architecture*. In addition, the interaction of hardware components is specified by the hardware architecture in terms of the bus systems used for interconnecting each hardware component as well as gateways used for communication. The hardware architecture plays an important role in the development of cars, as most hardware components specified within the hardware architecture are developed by suppliers. Thus, a large portion of the development of hardware and software is outsourced. As a consequence, the requirements specification used for the outsourced development of a certain hardware component is a composition of the logical architecture describing the software functionality as well as the hardware architecture describing the communication and other hardware related requirements.

The outsourced development of most vehicle ECUs plus their software-based functionalities has an additional impact on the test process. As already stated, a test strategy used to verify the software-based functions of a vehicle puts emphasis on integration testing to ensure that the set of features developed by different teams located in different companies do interact as specified. However, due to the size of modern vehicles containing hundreds of ECUs, sensors and actuators implementing several hundred features, a well-founded test selection and prioritization approach is required.

4.2 | Description of the studied system

The information-theory-based approach as well as the dependency analysis and data-flow-based approach of measuring coupling relies on a graph abstraction of the system. As Allen states, multiple graph abstractions can be created for a given system with each focusing on a different aspect of the system's design. Therefore, this section describes how the generic approaches are applied to the chassis control system and which limitations and implications arise due to the nature of including black-box components. All case study data for this system has been automatically collected via proprietary tool chains from architecture documentation or the integration testing result artefacts.

The system graph abstraction created for the information-theory-based measurement of coupling contains 47 modules and 756 nodes, which are connected using 3,512 edges resulting in a size of 7790.98 bits and a complexity value of 18,801.73 bits. For black-box modules, a complete graph of the software's inner structure is not available. Therefore, the graph abstraction of the system automatically represents an intermodule edge graph as described in section 2.4. Since it requires the module's internal inter-dependencies, the cohesion metric cannot be applied for both module- and system-level. In Table 2, the results for $Size(m_l|S)$ and $Complexity(m_k|S)$ are listed for each ECU of the chassis control subsystem. The size of an ECU hereby represents the amount of information added to the system while its complexity represents the amount of information in relationships between the ECU and the rest of the system. It is noticeable that the two modules with the highest values for both size and complexity (ECU7 and ECU8) together contribute over 50% of the system's overall size and complexity.

In Figure 3, the correlation of size to the failure distribution of all software interfaces of the chassis control systems is shown. The majority of interfaces form groups with equal size which can be seen based on the vertical alignment of nodes on the x-axis. In information theory, the size of a node is calculated based on the node's subgraph S_i, which only contains edges and endpoints incident to that node. When calculating $Size(S_i)$ of multiple node subgraphs of a given system, the result solely depends on the entropy of the graph $H(S_i)$ because the number of nodes n and the probability value of the environment $p_{l(0)}$ remain constant. The entropy of a graph $H(S_i)$ is calculated based on the probability values of the distinct row patterns. In case of a node subgraph, the number of distinct row patterns is a direct result of the number of endpoints connected to the node. Therefore, the size of a node mainly scales with the number of endpoints connected to that node. This is also demonstrated in Figure 4, which shows the size and number of endpoints of each interface in a scatter plot.

TABLE 2 Results of module level complexity measures

Ecu	$Size(m_l\|S)$	$Complexity(m_k\|S)$
ECU7	3553.61	9802.35
ECU8	1756.65	4828.09
ECU6	810.69	1570.60
ECU5	413.84	724.35
ECU1	353.19	532.87
ECU10	317.88	513.65
ECU9	180.27	237.56
ECU4	113.29	157.19
ECU3	94.96	157.31
ECU2	83.30	120.56

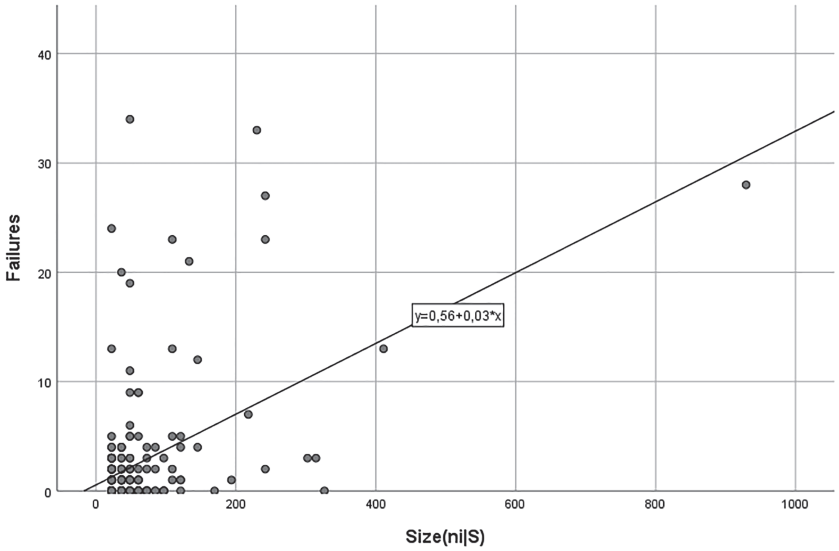

FIGURE 3 Correlation of $Size(S_i)$ of interfaces to interface level failure distribution

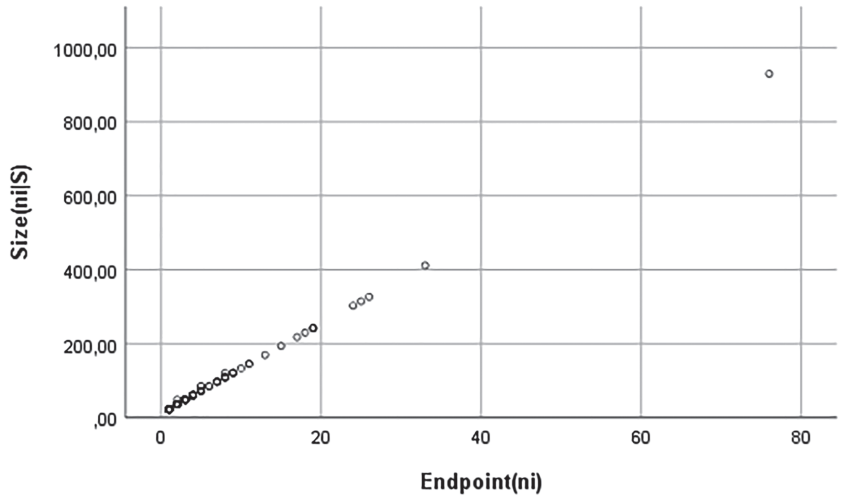

FIGURE 4 Proportion of endpoints to $Size(S_i)$ of interfaces

TABLE 3 Results of component level dependency and data flow measures

ECU	*CC*	*CIF*	*DC*
ECU7	2,574	102,000,390,625	80
ECU8	797	2,202,049,476	51
ECU6	127	91,164,304	27
ECU5	70	1,258,884	23
ECU1	31	1,210,000	9
ECU10	22	173,056	8
ECU3	15	9025	7
ECU4	10	32,400	6
ECU9	8	15,376	7
ECU2	3	400	6

The component model created for the dependency analysis and data-flow-based measurement of coupling contains 47 components and 756 interfaces, which form 3512 dependencies among each other. In Table 3, the component information flow *CIF*, component coupling *CC* and dependency coefficient *DC* are shown.

At system level, the component information flow *CIF* values for ECU 7 and ECU 8 and are noticeably high. This is caused by the fact that these two ECUs do have the highest amount of shared data flowing in and out.

As described in Section 2.3, the dependency coefficient *DC* of a certain component represents the total number of distinct components which are dependent on that component. The component coupling *CC* of a component on the other hand only takes outgoing dependencies into account. The component coupling value therefore represents the afferent portion of the overall couplings captured by the dependency coefficient.

4.3 | Research question

Xia emphasizes the ambiguity of coupling as a concept and the resulting challenge in deriving meaningful measures [13]. He states that the impact of coupling can be distinguished into two factors. First, it can have an impact on the system, as changes to highly coupled components may result in ripple effects and additional changes to other components. Second, when designing or changing a modular system, coupling can have an impact on the developers, as they need a complete understanding of the component's context in terms of its coupling to the rest of the system.

In this work, we study the effects of both impacts of coupling as a potential source of faults introduced into the system and detected during integration testing. As discussed in Sections 4.5 and 4.4, this study focuses on inter-ECU coupling as component coupling within the system as well as the coupling of individual interfaces implemented by these

components. The research questions as a baseline for the hypotheses are derived in accordance to Runeson and Höst [38] and Runeson [43].

RQ1: Which coupling measures applicable to the black-box *component structure* of a modular system are useful to guide the test case prioritization for integration testing?

RQ2: Which coupling measures applicable to the black-box *interface structure* of a modular system are useful to guide the test case prioritization for integration testing?

By focusing on the second type of potential impact of coupling, we provide the proposition that high coupling of a component or interface increases the probability for this component to show fail behaviour during integration testing. For interfaces implemented by each component, we propose the same relationship. These propositions stem from the assumption that during changes to the design or implementation of a component or interface, not all information of its context are considered due to cognitive overloading of the designers or developers. Given that potential impact of coupling onto the failure-proneness of component interactions, we will answer the research questions individually by using a two-step approach. First, we define and test multiple hypotheses regarding the correlation of each selected coupling measure to the failure-proneness for both component and interface coupling. Out of the accepted set of hypotheses and coupling measures which do show positive correlation to failure-proneness we answer the research question based on the observed strength of correlation. The hypotheses tested in this study are formulated for each coupling measure introduced in Section 2, both for component coupling and interface coupling. The list of hypotheses is given in Table 4.

4.4 | Data collection: Interface coupling

The structure of an ECU's software is defined in the software architecture specification. For the chassis control system, each ECU's software implements the AUTOSAR standard for software architecture which defines software components as structural element. A software component is defined as a self-contained unit which encapsulates a certain functionality equivalent to the software module in conventional modular software design. Communication and interaction among multiple software components are specified using port-prototypes, which are instances of Port-Interfaces. For this work, port-interfaces are considered as conventional software interfaces, as their goal is to specify the static structure of information exchange and to enable a design-by-contract workflow [44]. However, a detailed definition of the structural elements relevant to replicate the described study can be found in chapters 3.2 and 4.2 of the respective AUTOSAR Software Component Template specification.

Incoming and outgoing communication of a software component can be categorized as intra- or inter-ECU interactions. Regarding the data flow involved, intra-ECU communication describes data flow between software components located on the same ECU while inter-ECU communication describes data flow among software components distributed over different ECUs using a physical communication bus.

TABLE 4 Our nine hypothesis for the correlation between coupling measurements and failure-proneness

		Alternative hypothesis
Component coupling	H_{c1a}	The failure-proneness of an ECU during system integration testing is positively correlated to its $Size(m_l\|S)$ measure according to Table 1.
	H_{c2a}	The failure-proneness of an ECU during system integration testing is positively correlated to its $Complexity(m_k\|S)$ measure according to Table 1.
	H_{c3a}	The failure-proneness of an ECU during system integration testing is positively correlated to its *CC* measure according to Section 2.2.
	H_{c4a}	The failure-proneness of an ECU during system integration testing is positively correlated to its *CIF* measure according to Section 2.2.
	H_{c5a}	The failure-proneness of an ECU during system integration testing is positively correlated to its $CDC(C_k)$ measure according to Section 2.3.
Interface coupling	H_{i1a}	The failure-proneness of a component interface during system integration testing is positively correlated to its *Size* $(n_i\|S)$ measure according to Table 1.
	H_{i2a}	The failure-proneness of a component interface during system integration testing is positively correlated to its *IC* measure according to Section 2.2.
	H_{i3a}	The failure-proneness of a component interface during system integration testing is positively correlated to its *IIF* measure according to Section 2.2.
	H_{i4a}	The failure-proneness of a component interface during system integration testing is positively correlated to its $IDC(I_k)$ measure according to Section 2.3.

If the software architecture specification is available for each ECU contained in a vehicle subsystem, a data flow graph can be assembled which highlights the information flow within the chassis control system. However, this is often not possible in practice because the software development of most ECUs is outsourced based on a given black-box behaviour specification. This does make the software structure of an ECU as a set of interconnected software components unavailable and consequently, the intra-ECU communication is hidden. Therefore, coupling analysis at software level is not possible for a vehicle subsystem like the chassis control system studied in this work. However, software interfaces involved in inter-ECU communication are available as part of the ECU's interface specification. With respect to Xia's data flow centric definition of coupling discussed in Section 2, coupling analysis can directly be performed based on the inter-ECU data flow for ECUs at system level and software interfaces at interface level.

Therefore, within the graph abstraction used for the information-theory-based coupling measurement, nodes represent software interfaces which specify the ECU's external communication and interaction. However, the component model required for dependency and data-flow-based coupling measurement further details each interface according to its set of operations specified by that interface. When applied to software interfaces implementing the AUTOSAR standard for software architecture, an operation represents a certain port of an interface. The set of out-flows and in-flows of an interface can therefore be assembled according to the set of data defined for the provided and required ports. A detailed list of the number of software interfaces implemented by each ECU and the number of shared data assigned as in- and out-flow is given in Table 5.

ECU and interface interactions are modelled using edges in information theory as well as in dependency and data flow analysis. In this work, an edge represents the flow of shared data between two software interfaces implemented by different ECUs. Therefore, edges reflect sender-receiver relations among software interfaces in an undirected way. To enable a comparison of the information-theory-based approach and the dependency analysis and data flow approaches in Sections 2.3 and 2.4, an edge is created for each variable shared between two software interfaces. However, because information-theory-based complexity does not scale with multiple edges connecting the same two nodes, multiple exchanges between two software interfaces can also be represented by a single edge.

4.5 | Data collection: Component coupling

In the automotive industry, the vehicle is a collection of interconnected ECUs, which provide a set of vehicle functions. A subsystem of a vehicle is an interim system containing a subset of the vehicle's ECUs, namely, those implementing a coherent subset of vehicle functions. This study uses the chassis control subsystem, which mainly implements distributed software functions for electronic stability control, adaptive damping control and rear-wheel steering. There are two architectural levels to describe the structure of the system studied in this work.

As each ECU represents a component-based software system in its own right, an automotive software system can be described as a system of component-based software systems (CBSS). The set of ECUs contained in the chassis control system is listed in Table 5 together with the amount of implemented interfaces and data flow. For simplicity reasons, it is assumed that the inter-ECU communication takes place on a single physical communication bus. Because the chassis control system is a subsystem of a vehicle, there is additional communication between chassis control related ECUs and other ECUs of the vehicle. Dependencies of one chassis control related ECU to the rest of the vehicle are taken into account as coupling to the environment.

TABLE 5 Software interface, data flow and failure count for the case system

ECU	Interfaces	Interfaces having out-flow	Faulty interfaces	In-flow	Out-flow
ECU1	35	13	9	44	25
ECU2	8	1	1	20	1
ECU3	9	1	0	19	5
ECU4	11	2	0	18	10
ECU5	39	5	4	66	17
ECU6	78	24	9	18	10
ECU7	348	148	78	625	511
ECU8	170	56	30	237	198
ECU9	17	1	1	31	4
ECU10	30	4	0	52	8
Sum	**745**	**255**	**132**	**1130**	**789**

The information-theory-based approach as well as the dependency- and data-flow-based approach of measuring component coupling relies on a graph-abstraction of the system. On the highest level, the graph abstraction is partitioned into multiple modules. Within the chassis control system, a module represents an ECU. The graph abstraction of the chassis control system therefore contains 10 modules listed in Table 5 and named accordingly.

In multiple studies, Allen suggests that the graph abstraction used for information-theory-based measuring of component coupling should model the environment of the system by using an additional module containing a single node [26,27]. This node is disconnected from the system because coupling and complexity caused by relationships between the system and its environment are not relevant in this case. The chassis control system represents a subsystem of the vehicle. Therefore, chassis control related ECUs interact with non-chassis control system related ECUs, which can be considered as the environment. However, in the context of this work, generalizing those interactions to the system's environment would diminish the understanding of a component's or software interface's context. To take those interactions into account, the system graph contains additional modules for each ECU that interacts with one ECU of the chassis control subsystem.

In this work, we apply the measures discussed in Section 2 to the system-level architecture of the chassis control system. Therefore, each ECU is treated as a component and its coupling to the rest of the vehicle is measured as component coupling.

4.6 | Data collection: Failure distribution

For this study, we used test and fault data of a vehicle series development project covering the system integration test phase of the chassis control subsystem.

During vehicle development, each ECU represents a component-based software system containing hundreds of software modules. In general, testing of an ECU involves software unit, software composition, software integration and isolated ECU testing [45]. The integration of ECUs which are mostly developed and tested externally into a vehicle subsystem is referred to as system integration. The goal of system integration testing is to assure a stable subsystem in a controlled environment such that the subsystem can be tested in its actual environment during vehicle testing [4]. An integration test strategy used for system integration testing prescribes how to verify the interplay effects and functional dependencies among multiple ECUs and which test cases to prioritize to assure that every ECU has consistent assumptions about the semantics and frequency of shared information [45,46].

To identify the data flow involved in a detected failure, we used a defect classification scheme for black-box behaviour observation, which has been introduced in prior work [5]. In short, by using the classification scheme, a failure is broken down into the smallest number of conditions required to trigger its fault as observable failure behaviour. These conditions are then categorized into the following three phases of observation:

Precondition: To trigger a fault to surface as a failure, the system under test has to be in a certain state. Such a state is expressed based on data flowing among participating modules. The data flow is classified as a set of predicates. **Stimulation:** Given the required precondition, a failure can be triggered by stimulating certain data flow in a predefined way. This data flow is classified as stimulation-use. **Verification:** Once a failure is triggered, the actual failure behaviour of the system under test can be observed on one or more data flows as a derivation from the specified behaviour. These data flows are classified as verification-use.

For each of these phases, constraints are documented in a semi-formal way to describe the system state based on the inter-ECU communication. In Table 6, the data flow constraints of a simplified example failure affecting the freewheeling protection software function are given. In this example, the failure behaviour is only triggered in case of an idle vehicle with the electronic park brakes engaged. In addition, the selected gear needs to be neutral. A vehicle speed greater than zero is then applied in order to simulate a freewheeling situation and trigger the actual failure behaviour. The expected behaviour of the chassis control system includes the setup of hydraulic brake pressure to stop the vehicle from moving. However, as shown in the verification constraint, the freewheeling protection did not engage as specified.

The classification scheme's goal is to describe the entire occurrence of a failure including required preconditions and triggers to enable test coverage analysis for black-box integration testing. However, in this study, only constraints on

TABLE 6 Data flow profile of an example failure

Usage	Shared Data	Involved ECU
Precondition	$V\_VEH = 0$ m/s	Brake control system
	$COND\_PBRK = 1$ (fastened)	Parking brake actuator
	$GEAR\_SEL = N$	Gear box controller
Stimulation	$V\_VEH > 0$ m/s	Brake control system
Verification	$HYD\_BRK\_TRQ = 0$ Nm	Brake control system

data flow identified for the verification are utilized, since they reveal the sending software interfaces showing the failed behaviour as a derivation from its specified behaviour.

Because only software interfaces which include out-going interactions can show failure behaviour, the correlation study provided in 5 is only applied to those interfaces, even if the set of measures described in Sections 2.3 2.2, and 2.4 are applied to the complete set of software interfaces. In Table 5, the number of software interfaces implemented by each ECU is listed together with the amount of in- and out-flowing data. In comparison to that, the amount of software interfaces which are involved in at least one failure detected during system integration testing are listed as well.

5 | CASE STUDY RESULTS

To measure to which extent the failure-proneness of a certain component or interface correlates to its coupling, we used Spearman as well as Pearson correlation analysis. By conducting a Pearson correlation analysis, we are able to identify if there is a linear relationship between failure-proneness and coupling. For this, the amount of failures observed at a certain interface or component has to be proportional to its coupling values. However, because we do not expect a potential relationship between the number of detected failures and the coupling values of an interface to be proportional we included a Spearman rank-order correlation analysis as well. Based on the Spearman correlation analysis, we are able to evaluate the monotonicity of a potential correlation between coupling of an interface and its failure-proneness.

Because multiple hypotheses are being tested in this study on a single data set, the possibility of accepting a hypothesis that falsely appears significant increases [47]. To control for this, we use a p-value adjustment method introduced by Benjamini and Hochberg [48]. Alongside its definition, all tests in Tables 7,8,9 and 10 have been ranked according to their p values in an ascending way. In addition, the critical value is calculated based on the equation $(Rank/m_i) * Q_i$. By selecting a false discovery rate of 5% ($Q_i = 0.05$), all tests with critical values lower than the false discovery rate are considered as significant.

At the component level, we tested if $Size(mk|S)$, $Complexity(mk|S)$, component coupling (CC), component information flow (CIF) and component dependency coefficient (CDC) are correlated with the number of failures related to a component. In Tables 7 and 8, the results of the correlation analysis are shown for Pearson and Spearman correlation, respectively. All tests resulted in significant p values, and therefore, the corresponding hypotheses (H_{c1a} to H_{c4a}) listed in 4 can be accepted. However, due to the high p value, the test for CIF is considered as not significant, and therefore, H_{c5a} is rejected. When comparing the Pearson correlation coefficient (r_p) to the Spearman correlation coefficient (r_s) for the component level tests, both show similar strength. The strongest correlation is tested for the component dependency coefficient (CDC) which indicates that 72.6% of the variation in failures found at a certain component of the system can potentially be explained by this coupling measure.

At the interface level, we tested if $Size(n_i|S)$, interface coupling (IC), interface information flow (IIF) and interface dependency coefficient (IDC) are correlated with the number of failures related to an interface. By accepting a false discovery rate of 5%, the result of the interface information flow (IIF) is considered to be not significant for both Spearman and Pearson correlation; thus, the hypotheses H_{i1a}, H_{i2a} and H_{i4a} can be accepted while the hypothesis H_{i3a} is

TABLE 7 Results of component level Pearson

Measure	r_p	r_p^2	P_p	$Rank_p$	Linear regression
CDC	0.898	0.806	0.000425	1	$y = 18.2 + 3.41 * x$
$Complexity(m_k\|S)$	0.853	0.728	0.002	2	$y = 10.21 + 0.03 * x$
$Size(m_k\|S)$	0.846	0.716	0.002	2	$y = 2.56 + 0.07 * x$
CC	0.722	0.521	0.018	3	$y = 30.03 + 9.2E-3 * x$
CIF	0.55	0.303	0.094	4	$y = 41.07 + 1.63E-9 * x$

TABLE 8 Results of component level Spearman correlation

Measure	r_s	r_s^2	P_s	$Rank_s$
CDC	0.852	0.726	0.002	1
$Complexity(m_k\|S)$	0.853	0.728	0.002	1
$Size(m_k\|S)$	0.853	0.728	0.002	1
CIF	0.816	0.666	0.004	2
CC	0.779	0.607	0.008	3

TABLE 9 Results of interface level Pearson correlation

Measure	r_p	r_p^2	P_p	$Rank_p$	Linear regression
IDC	0.548	0.300	1.99E-21	1	$y = 0.19 + 0.73 * x$
$Size(n_i \mid S)$	0.476	0.227	7.84E-16	2	$y = 0.56 + 0.03 * x$
IC	0.367	0.135	1.50E-09	3	$y = 1.9 + 0.03 * x$
IIF	0.201	0.040	0.001	4	$y = 2.2 + 0.26 * x$

TABLE 10 Results of interface level Spearman correlation

Measure	r_s	r_s^2	P_s	$Rank_s$
IDC	0.303	0.092	8.09E-07	1
$Size(n_i \mid S)$	0.253	0.064	0.00004	2
IC	0.243	0.059	0.0009	3
IIF	0.043	0.002	0.497	4

rejected. In Tables 9 and 10, the results of the correlation analysis are shown for Pearson and Spearman correlation, respectively. When comparing the Pearson correlation coefficient (r_p) to the Spearman correlation coefficient (r_s), the Pearson correlation values are significantly higher over all tested measures. This indicates that there is a stronger linear trend than the rank correlation. The highest correlation is tested for the interface dependency coefficient (*IDC*). Its squared correlation coefficient r_p^2 indicates that 30% of the variation in failures found at a certain interface during integration testing can potentially be explained by this coupling measure.

When comparing the correlations for interface coupling and component coupling measures, we notice that the correlation of the coupling measures is substantially stronger when applied at the component level.

6 | DISCUSSION

The data-flow-based coupling measures tested in this study were generally weakly correlated for both interface and component coupling. When applied to the chassis control system introduced in Section 4, measures for interface coupling and component coupling showed limited applicability. The definition of interface coupling in equation 2 assumes that all out-flowing data of an interface are consumed by interfaces coupled to it. Because component coupling is defined as the sum of interface coupling of each interface implemented by the component, this assumption is further carried over to the component-level coupling measurement. For the system studied in this work, only a subset of outflow is used by a coupled interface or component as in-flow in most cases. The definition for interface and component coupling therefore leads to an overestimation of the actual coupling, which may affect the correlation in this study.

For interface and component information flow measures, we found an even stronger limitation in applicability. According to its definition in equation 3, information flow is measured as the squared product of the number of in- and out-flowing data. However, some of the interfaces of the chassis control system either only contain out-flow (e.g., for requesting a status) or in-flow (e.g., for diagnostic control routine). For those interfaces, the measure for information flow automatically results in a value of zero, which may be the reason why the information flow metric shows the weakest correlation of all measures tested in this study, both for interfaces and components.

The interface and component dependency coefficient measure showed the best performance for both interface and component coupling. According to the definition given in equation 6, these measures could directly be applied to the system design of the chassis control system. However, when using the dependency coefficient as a coupling measure, some implications arise which may be counter-intuitive within the context of a real-world system. Similar to the information theory measures, the dependency coefficient mainly scales with the number of coupled components. The amount of distinct messages shared between coupled interfaces or components is not covered. In addition, the direction of coupling is not used, and therefore, a differentiation between afferent and efferent coupling is not taken into account.

The information-theory-based measures also achieved good performance in our test. For the Spearman analysis of the component-level correlation, $Size(mk \mid S)$ and $Complexity(mk \mid S)$ achieved the strongest correlation. Similar to the dependency coefficient, the measures for $Size(mk \mid S)$ and $Complexity(mk \mid S)$ mainly reflect the number of coupled components or interfaces. This is also demonstrated in Figure 4. Unlike the dependency coefficient measure, which solely reflects the number of coupled components or interfaces, the measures for $Size(mk \mid S)$ and $Complexity(mk \mid S)$ account for redundancy and patterns in a system design [25].

However, these measures are designed around concepts which make their application to a real-world system not an easy task. In information theory, an additional node representing the system environment is added to the system graph. When applied to a real-world system, this requires that coupling to the system environment has to be generalized, which may lead to an underestimation of the component or interface coupling. Furthermore, the calculation of coupling based on the undirected representation of a system's interconnectivity does not conform to Martin's definition of component instability [8] and may therefore not account for all notions of coupling in a real-world system. Another concept of information-theory-based coupling measurement is the fact that these measures do not scale with multiple messages shared between components or interfaces. According to the assumptions made in Section 4.3 about cognitive overloading, we expect the probability of introducing a fault during maintenance to increase even if data flow is added to already coupled interfaces or components. However, this expectation is not captured by any of the measures for component or interface coupling used in this study.

When combining the test results listed in Tables 7 and 8 with the accepted hypotheses in the context of the research questions, we do believe that, out of the tested measures, the dependency coefficient and the information-theory-based approach provides the best guidance for system integration testing activities at both interface and ECU level.

6.1 | Practical implications

Our proposition for the steps required for the practice of system integration testing in the automotive industry involves the development process of the *system architecture and design* as well as the *system integration testing*. In the context of the test data required for data-flow-based test coverage and coupling-based test case prioritization explained in previous sections, this section describes the activities and steps relevant for practice. In Figure 5, the activities addressed for coupling-based system integration testing of the V-Model are highlighted as a continuation of the V-Model development process described in Section 4.1. Within this section, these activities are explained in detail alongside an activity diagram shown in Figure 6.

The most important activity for coupling-based system integration testing is the *system architecture design* as well as the *software design* since these provide a formal description of the system structure in terms of interacting software and hardware components. In a first step, the *graph abstraction* of the system under test has to be derived based on its *system- and software architecture* specifications. These design specifications are typically available at the beginning of the series development of a vehicle in a machine-readable format, which can be converted to a system graph in a straightforward way. Even though the graph abstraction used in the studies provided in this work has been generated out of a database export without manual steps involved, the system design tools were proprietary to the company at which the study has been conducted. This has an impact on the necessary effort of generating the graph abstraction, as no existing and publicly available tool chain can be used. Once the *graph abstraction* has been generated for the system under test,

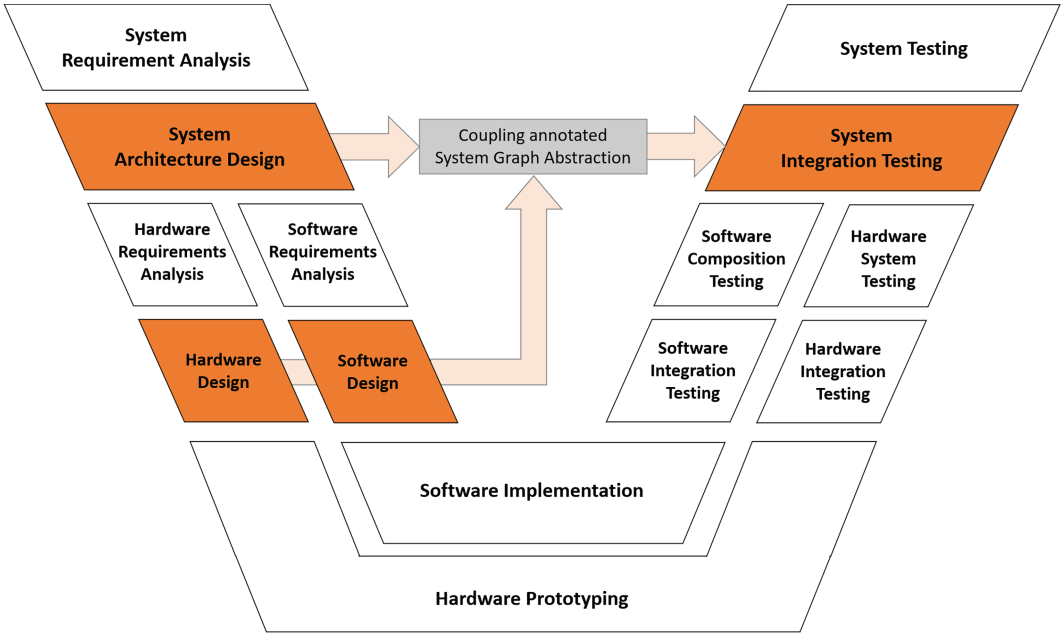

FIGURE 5 Collaborating activities of the V-Model involved in the proposed approach of coupling-based system integration testing

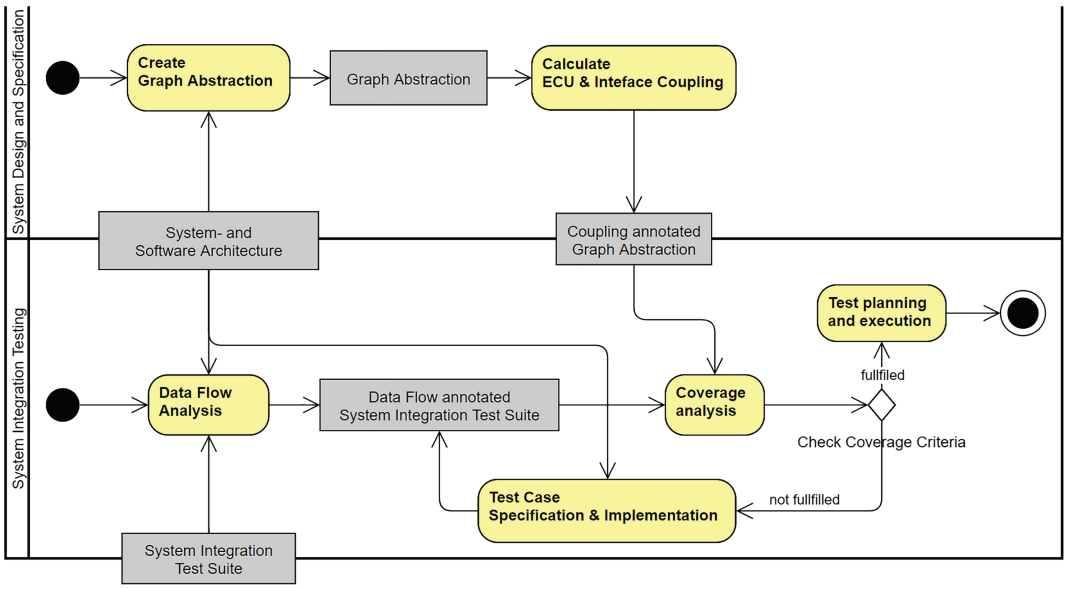

FIGURE 6 Proposed activities for coupling-based system integration testing

the coupling measures are collected for inter-ECU coupling as well as the coupling of each interface implemented by each ECU. As a result, individual coupling values for each ECU as well as individual values for each external interface implemented by these ECUs are added to the *graph abstraction* to create a *coupling-annotated system graph abstraction*. This annotated graph abstraction is then provided to the system integration testing process, as its coupling values indicate the probability of each element containing faults according to the correlation measured in Sections 2.

The process of *system integration testing* starts with a *data flow analysis* of the existing test cases to identify the data flow used to (a) describe the system state for test case preconditions, (b) describe the stimulation applied to the system during execution and (c) verify the system behaviour. A brief description of how component and interface interactions are identified in test cases is provided in more details in previous work [5]. The data flow analysis of these existing test cases results in the *data-flow-annotated system integration test suites*, which is then used with the *coupling-annotated graph abstraction* for coverage analysis. If the existing test cases do not satisfy the coverage criteria, information about uncovered data flow is used as input for the implementation of additional test cases. As discussed in Section 2, the overall space of observable data-flow-based system behaviour can become very large in a distributed system with moderate complexity. Achieving full test coverage is in practice often not feasible. To implement additional test cases which have a high chance of finding faults during execution, a prioritization technique is required. For this, a coverage measure with high correlation to failure occurrence for the system under test is used according to the results shown in Section 5. As a result, component behaviour which implements high coupling is considered for verification in additional test cases. Once the desired coverage is achieved, test planning and execution takes place according to existing and widely used test planning techniques, like testing of newly added or implemented features.

In summary, the described activities are repeated over the course of the series development project for each test phase or milestone. Given the identification of test gaps and the prioritization of test cases according to error proneness identified using coupling measures, the set of test cases considered for system integration testing evolves and improves over time.

7 | THREATS TO VALIDITY

There are several threats to validity drawn from the setup and context of this study, which have been addressed as follows.

7.1 | Conclusion validity

The conclusion validity of this work is mainly affected by the statistical power of the experiment. At first, data used for the experiments stem from only one vehicle development project. In addition to that, the component level correlation analysis

lacks statistical power for both Pearson and Spearman correlation. This is caused by the relative low number of ECUs contained in the studied chassis control subsystem. Due to the fact that the research question is highly relevant for the automotive industry, repetition studies are required to assure that the results provided in this work are reliable and valid.

7.2 | Internal validity

A major threat to the validity of this paper's experiments is the manual data flow classification of failure-reports used for the correlation analysis. As described in Section 4.6, a classification scheme has been used which was introduced in a prior work. Even though the classification scheme's effectiveness of describing the relevant data flow of a certain failure-report has been studied in a dedicated work, manual steps are required to apply the scheme to real world failures. To address the reliability of the classification, it was ensured that the team who performed the failure classification did not know the association between the failure reports and the test case which lead to its finding during execution. The goal of this measure is to reduce the bias between the test scenario and the failure classification. Furthermore, it was ensured that the data flow classification of a test case and its associated failure reports was not done by the same team member. However, all team members knew about the purpose of their work and had a deep understanding of the coverage criteria being the subject of this study during classification.

In addition to that, we selected all ECUs contained in the chassis control subsystem of a vehicle, which are further connected to other ECUs contained in different subsystems. The ECUs contained in the chassis control system implement external software interfaces, which form dependencies based on the inter-ECU data flow. The graph abstraction generated for this system therefore only reflects inter-ECU communication. However, in the context of the case study design, this may affect our hypothesis that strongly coupled software interfaces are more prone to show failures during system integration testing. An external software interface can also be coupled to other internal software components, which we did not analyse in this study.

The second threat affecting the *internal validity* is related to the failure distribution analysis. For this, we used the test results from the system integration testing phase performed during all milestones of the series development of the chassis control system. Overall, 132 failures have been detected, which we assigned to the respective software interfaces. Since one fault may cause multiple failures, the fault distribution would have provided a potentially more comprehensive base for this study. However, due to the black-box nature of an automotive development project, information about the fault which caused a certain failure is not available. In addition, it is important to mention that the selected set of defects does not only contain failed behaviour in terms of a deviation from specified behaviour but also defects caused by incorrect or incomplete specification and other types of defects [4]. Furthermore, taking every milestone of the series development project into account may also have affected the results of this study. During early milestones, not all functionality is fully implemented, which potentially can cause failed tests. In this study, this threat has been mitigated by excluding failures detected by testing incomplete functionalities. However, incomplete implementations can cause failures in fully implemented functionalities in the form of subsequent effects, which cannot be fully controlled in a real-world software development project.

7.3 | Construct validity

A threat to the validity of this paper's experiments stems from the set of selected coupling measures. Due to the fact that the AUTOSAR standard for embedded software does not inhibit common software design paradigms like object orientation or caller–callee related inter-component communication, the goal of the metric selection performed for this study is to select metrics which are directly applicable to the software architecture concepts commonly used in the automotive industry and described in Section 4.1. This may lead to the threat that the theory of coupling is not clear enough to be measured by the selected coupling measures for the given system used in this study. In addition to that, the information-theory-based metrics provided in Section 2.4 do lack statistical evidence that coupling is actually measure. However, the impact of this threat is limited due to the superordinate goal of finding correlations between the selected set of metrics and the software quality in terms of reliability.

7.4 | External validity

The *external validity* is also affected by a limited generalizability of the presented results, mainly because this study has been conducted in a company based on a single development project. But also the nature of the studied chassis control system limits the generalizability of the presented results to other non-automotive software systems. The AUTOSAR standard for embedded software implemented by each ECU defines unique mechanisms for interconnectivity of multiple software components, which greatly affects the generalizability of this study to component-based software systems in general. We mitigated this threat by providing broad insight into the derivation of the system graph abstraction of

the AUTOSAR-based software architecture. We transformed the software and system architecture design into a graph abstraction, which is then used for all coupling measures. The goal of the graph abstraction is to abstract the automotive software specific design paradigms and to provide a general basis to compare systems implementing different architecture patterns or programming paradigms.

8 | CONCLUSION AND FUTURE WORK

System integration testing of automotive subsystems is an extensive task, as multiple independently developed of-the-shelf ECUs are integrated to implement a coherent set of software functions. This work studied the correlation between several coupling measures proposed in scientific literature and the number of observed failures at two architectural levels: the component level, which shows the interconnectivity of ECUs as components of the chassis control system, and the software interface level, which represents the external software interfaces implemented by each ECU.

Our results provide the empirical foundation for potential approaches to guide test case selection by utilizing coupling measures. We evaluated the correlation between such coupling measures proposed in literature and the failure distribution of a chassis control subsystem. Our contribution is a first step towards an approach for systematic selection of test cases during integration testing of a distributed component-based software system with black-box components.

By testing five different coupling measures at both component and interface level, we found that measures which reflect the number of elements coupled to a certain component or interface correlate best with the distribution of failures found during integration testing. Results indicate that over 70% of the failure distribution at the component level can potentially be explained by these measures. We therefore believe that the number of coupled elements is an indicator for failure-proneness and can be used to guide test case prioritization during system integration testing.

We further found that data-flow-based coupling measurements do not capture the nature of an automotive software system and are therefore not applicable. Lastly, our study showed that measuring ECU coupling results in a substantially stronger correlation with the failure distribution than the interface coupling metrics. We therefore believe that having a grey box model showing the internal component structure for each externally developed ECU can improve system integration testing, as component coupling can be measured for each component with highly coupled internal subcomponents.

In future work, we will study the potential benefits of grey box models for of-the-shelf ECUs in comparison to conventional black-box models. Furthermore, we highly agree with Xia's definition of coupling as a composite measure combining the number of dependencies, the complexity of shared information as well as the impact on dynamic behaviour. According to his definition, we will focus on different types of coupling and their impact on the dynamic behaviour of coupled elements.

ACKNOWLEDGEMENT

Open Access funding enabled and organized by Projekt DEAL.

DATA AVAILABILITY STATEMENT

Data sharing not applicable to this article as no data will be available.

ORCID

Dominik Hellhake https://orcid.org/0000-0001-6475-5603

REFERENCES

1. Xia Cai, Lyu MR, Kam-Fai W, Roy K. Component-based software engineering: technologies, development frameworks, and quality assurance schemes. In *Proceedings seventh Asia-Pacific software engineering conference. apsec 2000*. 2000;372–9.
2. Briand LC, Morasca S, Basili VR. Property-based software engineering measurement. IEEE Trans Softw Eng. 1996;22(1):68–86.
3. Patel S, Kaur J. A study of component based software system metrics. In *2016 international conference on computing, communication and automation (iccca)*. 2016;824–8.
4. Naik K, Tripathy P. 2008. System integration testing. In Software testing and quality assurance. John Wiley & Sons, Ltd; 158–91. https://doi.org/10.1002/9780470382844.ch7
5. Hellhake D, Schmid T, Wagner S. Using data flow-based coverage criteria for black-box integration testing of distributed software systems. In *2019 12th ieee conference on software testing, validation and verification (icst)*. 2019;420–9.
6. Abdellatief M, Md Sultan AB, Abdul Ghani A, Jabar AM. A mapping study to investigate component-based software system metrics. J Syst Softw. 2013;86:587–603.
7. Yourdon E, Constantine LL. Structured design: fundamentals of a discipline of computer program and systems design (1st edn.) Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1979.
8. Martin R. Oo design quality metrics, 1997. October.
9. Santos D, Resende A, de Castro Lima E, Freire A. Software instability analysis based on afferent and efferent coupling measures. J Softw. 2017; 12:19–34.

10. Abdellatief M, Md Sultan AB, Sultan M, Abdul ghani A, Abd Ghani A, Jabar AM. Component-based software system dependency metrics based on component information flow measurements. In *The sixth international conference on software engineering advances*, 2018.
11. Gill NS, Grover PS. Few important considerations for deriving interface complexity metric for component-based systems. ACM Spec Interes Group Softw Eng. 2004;29(2):4–4. https://doi.org/10.1145/979743.979758
12. Myers GJ. Reliable software through composite design. J Am Soc Inf Sci. 1977;28:303–303.
13. Xia F. On the concept of coupling, its modeling and measurement. J Syst Softw. 2000;50:75–84.
14. Ural H, Yang B. Modeling software for accurate data flow representation. In *Proceedings of the 15th international conference on software engineering*, ICSE '93. IEEE Computer Society Press: Los Alamitos, CA, USA. 1993;277–86. http://dl.acm.org/citation.cfm?id=257572.257633
15. Ince D, Shepperd M. An empirical and theoretical analysis of an information flow based design metric. In *Proceedings of the european conference on software engineering*. 2006;86–99.
16. Henry S, Kafura D. Software structure metrics based on information flow. IEEE Trans Softw Eng. 1981;SE-7(5):510–18.
17. Kafura D, Henry S. Software quality metrics based on interconnectivity. J Syst Softw. 1981;2(2):121–31. https://doi.org/10.1016/0164-1212(81)90032-7
18. Kumari U, Upadhyaya S. An interface complexity measure for component-based software systems. Int J Comput Appl. 2011;36:46–52.
19. Narasimhan L, Hendradjaya B. Some theoretical considerations for a suite of metrics for the integration of software components. Inf Sci. 2007; 177:844–64.
20. Kharb L, Singh R. Complexity metrics for component-oriented software systems. ACM Spec Interes Group Softw Eng. 2008;33(2):4:1–3. https://doi.org/10.1145/1350802.1350811
21. Li B. Managing dependencies in component-based systems based on matrix model. In *Proc. of net.object.days 2003*. 2003;22–5.
22. Sharma A, Grover P, Kumar R. Dependency analysis for component-based software systems. ACM SIGSOFT Softw Eng Notes. 2009;34:1–6.
23. Gill NS, Balkishan. Dependency and interaction oriented complexity metrics of component-based systems. ACM Spec Interes Group Softw Eng. 2008;33(2):3:1–5. https://doi.org/10.1145/1350802.1350810
24. Sengupta S, Kanjilal A, Bhattacharya S. Measuring complexity of component based architecture: a graph based approach. ACM SIGSOFT Softw Eng Notes. 2011;36:1–0.
25. Allen EB, Khoshgoftaar TM. Measuring coupling and cohesion: an information-theory approach. In *Proceedings sixth international software metrics symposium (cat. no.pr00403)*. 1999;119–27.
26. Allen EB. Measuring graph abstractions of software: an information-theory approach. In *Proceedings eighth ieee symposium on software metrics*. 2002;182–93.
27. Allen EB, Khoshgoftaar TM, Chen Y. Measuring coupling and cohesion of software modules: an information-theory approach. In *Proceedings seventh international software metrics symposium*. 2001;124–34.
28. Anan M, Saiedian H, Ryoo J. An architecture-centric software maintainability assessment using information theory. J Softw Maint. 2009;21:1–8.
29. Briand LC, Daly JW, Wust JK. A unified framework for coupling measurement in object-oriented systems. IEEE Trans Softw Eng. 1999;25(1): 91–121.
30. Briand LC, Daly JW, Wust J. A unified framework for cohesion measurement in object-oriented systems. In *Proceedings fourth international software metrics symposium*. 1997;43–53.
31. Gray RM. Entropy and Information Theory. 2nd ed. Springer US, 2007.
32. Hao D, Zhang L, Zhang L, Rothermel G, Mei H. A unified test case prioritization approach. ACM Trans Softw Eng Methodol. 2014;24(2): 1–31. https://doi.org/10.1145/2685614
33. Elbaum S, Malishevsky AG, Rothermel G. Test case prioritization: a family of empirical studies. IEEE Trans Softw Eng. 2002;28(2):159–82.
34. Mendes N, Dias D, Oliveira L, Lana C, Nakagawa E, Maldonado J. Exploring together software architecture and software testing: a systematic mapping. In *International conference of the chilean computer science society*. vol. 35, 2016.
35. Selby RW, Basili VR. Analyzing error-prone system structure. IEEE Trans Softw Eng. 1991;17(2):141–52.
36. Singh A, Bhatia R, Singhrova A. Object oriented coupling based test case prioritization. Int J Comput Sci Eng. 2018;6:747–54.
37. Richardson D, Wolf A. Software testing at the architectural level, 1997. International Software Architecture Workshop, Proceedings, ISAW.
38. Runeson P, Höst M. Guidelines for conducting and reporting case study research in software engineering. Empir Softw Eng. 2008;14(2):131. https://doi.org/10.1007/s10664-008-9102-8
39. Broy M. Automotive software engineering. In *Proceedings - international conference on software engineering*. 2003;719–20.
40. Gopu GL, Kavitha KV, Joy J. Service oriented architecture based connectivity of automotive ECUs. In *2016 international conference on circuit, power and computing technologies (iccpct)*. 2016;1–4.
41. Bocchi L, Fiadeiro JL, Lopes A. Service-oriented modelling of automotive systems. In *2008 32nd annual ieee international computer software and applications conference*. 2008;1059–64.
42. Wagner M, Zbel D, Meroth A. An adaptive software and systems architecture for driver assistance systems based on service orientation. Int J Mach Learn Comput. 2011;1:359–66.
43. Runeson P, Host M, Rainer A, Regnell B. Case Study Research in Software Engineering: Guidelines and Examples (1st edn.) Wiley Publishing, 2012.
44. AUTOSAR. Application interfaces user guide. https://www.autosar.org/fileadmin/user_upload/standards/classic/4-2/AUTOSAR_EXP_AIUserGuide.pdf, https://www.autosar.org
45. International Standard. ISO 26262: road vehicles—functional safety, 2011. http://www.iso.org
46. International Standard. ISO/IEC/IEEE 29119: software and systems engineering—software testing, 2013. https://www.iso.org
47. Blakesley RE, Mazumdar S, Dew MA, Houck PR, Tang G, Reynolds CF, Butters MA. Comparisons of methods for multiple hypothesis testing in neuropsychological research. Neuropsychology. 2009;23(2):255–64. https://doi.org/10.1037/a0012850
48. Benjamini Y, Hochberg Y. Controlling the false discovery rate: a practical and powerful approach to multiple testing. J R Stat Soc: Ser B (Methodol). 1995;57(1):289–300. https://doi.org/10.1111/j.2517-6161.1995.tb02031.x

How to cite this article: Hellhake D, Bogner J, Schmid T, Wagner S. Towards using coupling measures to guide black-box integration testing in component-based systems. Softw Test Verif Reliab. 2022;32:e1811. https://doi.org/10.1002/stvr.1811