Universität Stuttgart

# Verifikation softwareintensiver Fahrwerksysteme

Von der Fakultät für Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung


Vorgelegt von

Dominik Hellhake

geboren in Balve, Deutschland


**Hauptberichter:** Prof. Dr. Stefan Wagner

**Mitberichter:** Prof. Dr. Andy Zaidman

**Tag der mündlichen Prüfung:** 03. März 2023


Institut für Software Engineering


2023

# Zusammenfassung

**Kontext:** Die zunehmende Signifikanz von softwarebasierten Funktionen in modernen Fahrzeugen ist der Auslöser vieler Veränderungen im automobilen Entwicklungsprozess. In der Vergangenheit bestand ein Fahrzeug aus mehreren Electronic Control Units (ECUs), welche jeweils individuelle und voneinander unabhängige Softwarefunktionen ausführten. Demgegenüber bilden heute mehrere ECUs funktional kohärente Subsysteme, welche übergreifende und vernetzte Softwarefunktionen wie zum Beispiel Fahrerassistenzfunktionen und automatisierte Fahrfunktionen implementieren. Dieser Trend hin zu einem hochvernetzten Softwaresystem sorgt in der Entwicklung moderner Fahrzeuge für einen hohen Bedarf an geeigneten Architekturmodellen und Entwurfsmethoden. Aufgrund der Entwicklung von ECUs durch verschiedene Entwicklungsdienstleister werden zusätzlich systematische Integrationstestmethoden benötigt, um das korrekte Interaktionsverhalten jeder individueller ECU im Laufe der Fahrzeugentwicklung zu verifizieren. Hierfür stellt Kopplung eine weit verbreitete Messgröße dar, um in komponentenbasierten Softwaresystemen Qualitätseigenschaften wie die Verständlichkeit, Wiederverwendbarkeit, Modifizierbarkeit und Testbarkeit widerzuspiegeln.

**Problembeschreibung:** Während Kopplung eine geeignete Messgröße für die Qualität eines Softwaredesigns darstellt, existieren nur wenig wis-

senschaftliche Beiträge über den Mehrwert von Kopplung für den Integrationstestprozess des aus dem Design resultierenden Systems. Existierende Arbeiten über das Thema Integrationstest beschreiben die schrittweise Integration von White-Box Softwarekomponenten unter Verwendung von Eigenschaften und Messgrößen, welche aus der Implementierung abgeleitet wurden. Diese Abhängigkeit vom Quellcode und der Softwarestruktur sorgt jedoch dafür, dass diese Methoden nicht auf die Entwicklung von Fahrzeugen übertragen werden können, da Fahrzeugsysteme zu einem großen Anteil aus Black-Box Software bestehen. Folglich existieren auch keine Methoden zur Messung der Testabdeckung oder zur Priorisierung der durchzuführenden Tests. In der Praxis sorgt dies dafür, dass lediglich erfahrungsbasierte Ansätze angewendet werden, bei denen signifikante Anteile des Interaktionsverhaltens im Laufe der Fahrzeugentwicklung ungetestet bleiben.

**Ziele:** Um Lösungen für dieses Problem zu finden, soll diese Arbeit systematische und empirisch evaluierte Testmethoden ausarbeiten, welche für die Integrationstests während der Fahrzeugentwicklung angewendet werden können. Dabei wollen wir in erster Linie auch einen Einblick in das Potential bieten, welche Messgrößen Kopplung für die Verwendung zur Testfall-Priorisierung bietet. Das Ziel dieser Arbeit ist es, eine Empfehlung für das systematische Integrationstesten von Fahrzeugsystemen zu geben, welches auf dem Interaktionsverhalten einzelner ECUs basiert.

**Methoden:** Um diese Ziele zu erreichen, analysieren wir im ersten Schritt dieser Arbeit den Stand der Technik, so wie er gegenwärtig bei BMW für das Integrationstesten der Fahrwerkssysteme angewendet wird. Dem gegenüber analysieren wir den Stand der Wissenschaft hinsichtlich existierender Testmethoden, welche auf die Problemstellung der Integration von Fahrzeugsystemen übertragen werden können. Basierend auf diesem Set an wissenschaftlich evaluierten Methoden leiten wir anschließend konkrete Vorgehensweisen für die Messung der Testabdeckung und der Testfall-Priorisierung ab. Im Rahmen dieser Arbeit werden beide Vorgehensweisen empirisch evaluiert basierend auf Test- und Fehlerdaten aus einem Fahrzeugentwicklungsprojekt.

**Beiträge:** Zusammengefasst enthält diese Arbeit zwei Beiträge, welche

wir zu einem zentralen Beitrag zusammenführen. Der erste Bereich besteht aus einer Methode zur Messung der Testabdeckung basierend auf dem inter-komponenten Datenfluss von Black-Box-Komponenten. Die Definition eines Datenfluss-Klassifikationsschemas ermöglicht es, Daten über die Verwendung von Datenflüssen in existierenden Testfällen sowie in Fehlern zu sammeln, welche in den verschiedenen Testphasen gefunden wurden. Der zweite Beitrag dieser Arbeit stellt eine Korrelationsstudie zwischen verschiedenen Messmethoden für Coupling und der Fehlerverteilung in einem Fahrwerkssystem dar. Dabei evaluieren wir die Coupling-Werte von individuellen Software-Interfaces sowie die der Komponenten, welche diese implementieren. Zusammengefasst spiegelt diese Studie das Potential wider, das solche Coupling-Messmethoden für die Verwendbarkeit zur Testpriorisierung haben. Die Erkenntnisse aus diesen Beiträgen werden in unserem Hauptbeitrag zu einer Coupling-basierten Teststrategie für Systemintegrationstests zusammengeführt.

**Fazit:** Der Beitrag dieser Arbeit verbindet zum ersten Mal den Stand der Technik zur Systemintegration von verteilten Black-Box-Softwaresystemen mit dem Stand der Wissenschaft über systematische Ansätze zur Integration von Softwaresystemen. Das Messen der Testabdeckung basierend auf dem Datenfluss ist hierfür eine effektive Methode, da der Datenfluss in einem System das Interaktionsverhalten der einzelnen Komponenten widerspiegelt. Zusätzlich kann das mögliche Interaktionsverhalten aller Komponenten des Systems aus dessen Architektur-Spezifikationen abgeleitet werden. Aus den Studien über die Korrelation von Coupling zur Fehlerverteilung geht außerdem eine moderate Abhängigkeit hervor. Aufgrund dessen ist die Selektion von Testfällen basierend auf die im Testfall erprobten Komponenteninteraktionen und dessen Coupling ein sinnvolles Vorgehen für die Praxis. Jedoch ist die moderate Korrelation auch ein Indiz dafür, dass zusätzliche Aspekte bei der Auswahl von Testfällen für Integrationstests zu berücksichtigen sind.

# Abstract

**Context:** The growing significance of software-based functionality in a modern vehicle is at the root of changes in the automotive industry. In the past, a vehicle consisted of multiple electronic control units (ECU), each of it executing software functions dedicated to an isolated vehicle function. In a modern vehicle, however, multiple ECUs form coherent subsystems providing higher-order software-based functions like driving assistance or automated driving. For the development of modern vehicles, this transition to a highly interconnected software system caused a high demand for good architectural models and design practices. In addition, because the development of most of the electronic control units is outsourced to different companies, systematic integration testing techniques are required in order to verify the correctness of interactions between multiple ECUs. For component-based software systems, coupling is the degree of interdependence of individual components and a widely adopted indicator for quality attributes of a software architecture reflecting the understandability, reusability, modifiability as well as testability.

**Problem Statement:** However, while coupling provides a valuable measure to track the quality of a software architecture during development, very little scientific research has been conducted studying the potential contribution of coupling measures to the integration testing process of the resulting

system. In general, most existing work describing integration testing techniques focuses on the step-wise integration of white box components based on their implementation. This makes the contributions to this scientific field unavailable for the integration testing of a modern vehicle, which mostly contains black-box or off-the-shelf components for which details about their implementations are not available. Furthermore, there are no applicable approaches to measure test coverage or prioritizing test cases during integration testing of black-box components defined in literature. In practice, this often results in a predominantly experience based integration testing plan with a significant amount of component interaction related faults being found during system testing or in the field.

**Objectives:** To address this, the aim of this research is to provide empirically evaluated techniques to support the integration testing process. In particular, we want to provide insight into the usefulness of coupling measures for the prioritization of test cases as well as a test coverage measurement approach based on component-interaction identified in the systems architecture. Overall, we aim to provide recommendations for systematically testing the integration of distributed software system containing black-box components.

**Methods:** To archive this goal, we first analyzed the industry state of the practice exercised at BMW during the integration testing of the Chassis Control System. We then analyzed the academic state of the art and identified existing techniques applicable to the problem faced at BMW. Based on these findings, we then identified the research gap and designed new approaches for the measurement of test coverage and test case prioritization. Both approaches are empirically evaluated based on real-world test data and software architecture of a project conducted at BMW.

**Contributions:** We provide contributions in two different areas, which we combine to form the central contribution of this work. First, we provide an approach for test coverage analysis applicable for black-box integration testing based on inter-component data-flow. By introducing and evaluating a data-flow classification scheme for test cases and failures, we enable data to be collected about the usage of data-flow in existing test cases and failures

found during integration testing as well as failures not discovered by the integration testing phase. In addition, we demonstrate how such data is used to measure test coverage of existing test cases and how test gaps can be identified.

Second, we provide a correlation study of several coupling measures as defined in literature to the failure distribution of a real world system showcasing their effectiveness of reflecting the fault-proneness of individual component interactions. We applied these measures to the interface- and component level architecture of the studied system to demonstrate the usefulness of interface coupling for integration testing compared to component coupling. Combining these two contributions, we provide the central contribution provided in this dissertation, namely the conceptualization of a Coupling-based Test Strategy for Black-Box System Integration Testing.

**Conclusion:** With our contributions, we are the first to combine the current industry state of practice for integration testing of distributed systems containing black-box components as well as the scientific state of measuring component and interface coupling to form a systematic methodology for integration testing. Data-Flow based test coverage measurement is applicable for black box integration testing, as it effectively reflects the set of tested ECU interactions in contrast to the overall set of ECU interactions. However, coupling-based test case prioritization has shown low to moderate correlation to failure distribution depending on the selected coupling measure. This indicates that additional aspects besides coupling have to be taken into account when selecting test cases for execution during black-box integration testing. In summary, practitioners in the automotive industry can use our proposed approach for coupling-based testing for black-box integration to complement the existing experience-based test-planning.

# Acknowledgments

Conducting research in industry has its own challenges and pitfalls. Therefore, my genuine gratitude goes out to all people from both company and university who have supported me during the course of this work.

In particular I want to thank my PhD advisor Prof. Dr. Stefan Wagner. He greatly supported my understanding on how to conduct empirical research in a correct way which had a direct influence onto the research design and structure of the studies presented in this work. In addition, Stefan helped me by identifying the relevant research questions included in the practical problems I faced at the beginning of my PhD in the department for developing chassis control system at BMW. Even though I was an external PhD student in his group, he fostered collaborations with his other students and made sure I was well connected to the world of research in the best possible way.

I am also grateful for the support of Dr. Céline Laurent-Winter, who had already been the advisor of my Master's thesis, for encouraging me to undertake a PhD and giving me the opportunity to do so at BMW. I also want to thank her for our productive discussions at the beginning of my PhD.

Lastly I want to thank all my fellow colleagues who undertake a PhD in parallel to me. In particular I want to thank Justus Bogner, Kai Mindermann and Tobias Schmid who are also part of the research group at the Institute

of Software Engineering at University of Stuttgart.

# Contents

Chapter 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 [XLKR00]. In literature, a large set of metrics exist to measure and control the quality of component based systems. Commonly evaluated internal software design properties are *size*, *complexity*, *coupling* and *cohesion* [BMB96]. In this section we provide a brief summary of the research field addressed in this thesis. For this, we derive problems faced in the industry. We subsequently formulate our research objective and guiding research questions to address these problems. Lastly, we outline the structure of this thesis.

## 1.1 Motivation

Early software-based solutions in a car were local in their functionality and strictly isolated from each other. Independent functions like the engine control were used to run on single dedicated Electronic Control Units (ECUs) and just a few kilobytes in size. However, over the past decades the amount of software in cars grew exponentially and caused various changes in the design and development of modern vehicles. The most significant change is the vehicles transition from including few individual and unconnected software functions to highly distributed software-based systems implementing functions distributed over several ECUs connected by bus systems as the underlying communication infrastructure. This trend in the development of a vehicle's modular software design has been enabled by suppliers which take care of a significant part of the development effort. As ECUs and their underlying software were developed by these suppliers, the automotive industry still follows the traditional concept of having parts of a vehicle produced by a chain of suppliers while being only responsible for their assembly. However, this idea stems from a purely hardware related point of view. With software becoming the most significant force of innovation in the automotive industry, the responsibility of a cars manufacturer evolved from the assembly of hardware parts during production to the integration of distributed software based functions during development.

Since individual ECUs in a modern vehicle are often developed by independent teams often located at different companies, it is very important to verify the correctness of their composed functionality once they are integrated into a complete system. This verification is referred to as *system integration testing* and is a prerequisite for the final system-level testing at the end of the vehicles series development phase. In general, multiple basic approaches exist for integration testing of a component-based system like incremental, top down, bottom up or big bang [NT08]. All these approaches assume that individual components have already been tested separately in a controlled environment. Integration testing therefore focuses on the verification of component interplay behavior 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 when trying to accomplish full test coverage at component interaction level.

In the automotive industry, integration testing is split into multiple stages as a modern vehicle consists of over 200 interconnected electronic control units each of it executing a component based software system on its own. After software integration testing and testing of each ECU individually is done by the supplier, 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 all potential interactions between multiple ECUs during execution of a higher-order functionality. To accomplish such a coverage of component interactions, each of the identified component interactions should be considered for explicit verification. However, this is not feasible for large systems due to the potential combinatorial complexity of functional dependencies, which can lead to substantial testing efforts.

## 1.2 Problem

In literature, there are very few formally or empirically validated approaches available for the systematic testing of component interactions in a component-based system [SCK10]. Such approaches become even more important when the software of most ECUse are considered to be black-box, due to the fact that development has been done by external teams. In this context, code-based approaches for identifying failure-prone component interactions are not applicable. Therefore, system integration testing has to be performed solely on interface- and black-box behavior 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.

In today's state of the practice of system integration testing of an auto-

motive system, test design, implementation and selection of test cases is commonly done based on experience from past series development projects. For this, test data is analyzed regarding groups of similar or reoccurring defects which are then used to guide the implementation of new test cases and the planning of their execution during the different test phases. For the integration testing of an automotive system, such experience based test strategies have proven effective due to the high reuse of ECUs and software among different vehicle models and vehicle generations as well as the high similarity in a vehicle's system and software architecture. However, due to the growing number of newly added software based functions to modern vehicles, the change in system design and architecture over time is becoming significant. This increasing development progress between two consecutive generations of vehicles causes the experience from past projects to be not fully applicable anymore.

In summary, the state of the art of system integration testing done in the automotive industry does not include systematic approaches for test coverage measurement or test case selection and prioritization. In academia, there is a sound understanding of such holistic strategies for integration testing on software level. However, due to the fact that these strategies are built-up on the availability of code and software architecture documentation they are not directly applicable for the problem of system integration testing faced in the automotive industry.

## 1.3 Research Objective

We formulate our overarching research objective to address the described problems in the following way:

*Provide a systematic approach for system integration testing of automotive subsystems including methods and techniques which allow the finding of component interaction related faults early in development.*

To define a systematic testing technique for system integration testing, we derive two major research topics for this thesis. First, a coverage measure for component interaction which is considered directly on component level and in addition on interface level while a component implements multiple interfaces. Second to that, an approach for test case selection and prioritization based on a measure of fault-proneness of component interactions. In the following section, these two research directions are presented alongside their research questions.

- **RQ1:** How effective is inter-component based data-flow classification in capturing the system's dynamic behavior relevant for test coverage analysis and test gap identification?

Because the goal of system integration testing is to prescribe how to verify the interplay effects of each component in order to assure that every component has consistent assumptions about the semantics and frequency of shared information [Int11], a coverage measure should identify all potential component interactions based on the system's design. Furthermore it should identify which component interactions are verified by a certain test case to consequently provide an overview of tested and untested component interactions.

- **RQ2:** 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?
- **RQ3:** 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?

A test case selection and prioritization technique however should be based on a sound measure for fault-proneness which utilizes the characteristics of component interactions to derive the potential of holding a fault during development. By answering the research questions **RQ2** and **RQ3**, we provide the proposition 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. This proposition stems from the assumption, that due to cognitive overloading, not all information of a software component's context is considered during its development or evolution.

## 1.4 Contribution

The central and unique contribution of this thesis is the conceptualization of a **Coupling based Test Methodology for Black-Box Integration Testing**. For this, we are the first to combine the current industry state of practice for integration testing of distributed systems containing black-box components as well as the scientific state of measuring component and interface coupling to form a systematic methodology for integration testing. We show how a coupling measurement of the system under test can be derived from the specification documents commonly available in a vehicle series development in a fully automatic way. In addition, we provide a definition of a generic process model which describes which activities of the development process of a modern vehicle is involved in the proposed coupling based system integration testing approach. This main contribution is derived from our two research directions described in the previous section and therefore combines the following individual contributions.

By first introducing a data flow classification scheme for test cases and failures, we utilize knowledge about the data flow induced into the system during test execution to obtain the overall **data flow based coverage** of an existing test suite and reveal untested data flow. Second, we analyze the observed data flow during the occurrence of a failure to determine to which ratio it is covered by the detecting test case. Finally, we evaluate the usefulness of the identified data-flow based approach of measuring test coverage for system integration testing by comparing the usage of data flow

of failure reports created by other testing activities (defect slip-through) to the coverage of data flow of the existing test cases. By showing that traces of failures which slipped by the integration test phase have significantly lower similarity to the traces of executed test cases compared to the traces of failures found during execution we demonstrated the effectiveness of data flow coverage analysis for black-box integration testing. In addition to the classification scheme, we introduced basic **data flow based coverage criteria** specifically designed to measure the utilization of component interaction based on data flow. Lastly, we demonstrated how the data flow classification of existing test cases and failures slipped by the integration testing phase can be used to identify potential test gaps in terms of unconsidered component interactions.

In the conducted case study, we analyzed 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 system 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. As part of a correlation study, we applied various coupling measures as defined in literature for software level coupling measurement to interface- and component level architecture of a real world system. The failure distribution over interfaces and components of the system has been collected based on the data-flow involved during occurrence of the failure.

## 1.5 Pre-Published

The above mentioned contributions have been introduced into the scientific discourse in the form of publications for each of the two research direction. The following list consists of the publications that are created during the course of this thesis and the presented studies.

a) D. Hellhake et al. “Towards using coupling measures to guide black-box integration testing in component-based systems.” In: *Software Testing, Verification and Reliability* 32.4 (Mar. 2022). URL: https://doi.org/10.1002/stvr.1811

   In this article, we analyze the relationship between different component and interface coupling measures found in literature and the distribution of failures found during integration testing of a automotive system. The presented correlation study provides the empiciral basis for the conceptualization of the test case prioritization approach in chapter 4.

b) D. Hellhake, T. Schmid, and S. Wagner. “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)*. Apr. 2019, pp. 420–429

   In this article, we introduce a data-flow based observation scheme which captures the interplay behavior of involved ECUs during test execution and failure occurrences. In addition, we introduce a data flow-based coverage criterion designed for black box integration. The classification scheme as well as the coverage criteria is used in section 3 to formalize a test coverage approach applicable for black-box integration testing.

c) D. Hellhake and S. Wagner. *Kommunikationsfluss-orientiertes Testen von Softwarefunktionen im Steuergeräteverbund*. Tech. rep. 7. FACHKONFERENZ AUTOTEST, 2018. URL: https://fkfs-veranstaltungen.de/fileadmin/4_AutoTest/pdf/AutoTest_2018/

In this article, we analyze common types of failures found during system integration testing or system level testing. In addition, we provide a discussion on the applicability of data-flow to such failure types. In section 3 the applicability of data-flow classification to common faults found during system integration testing provides the basis to define the limitations of the presented approach.

d) M. Golagha et al. “Aletheia: A Failure Diagnosis Toolchain.” In: *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. 2018, pp. 13–16

This article provides an automated approach for developers of ECU software to reduce time required for fault localization of failures detected during software composition and integration testing. The approach has been evaluated in the context of the series development project described in section 2.3 which is also used for the evaluation of the approaches provided in this work.

## 1.6 Thesis Structure

The remainder of this thesis is structured in the following way alongside the major research directions and contributions. First, chapter 2 provides the theoretical background for the coupling measures studied in this work as well as the state of the practice of developing and testing modern vehicles. In chapter 3, our main contribution in the research direction of data flow based test coverage measurement is provided including the definition of the data flow based classification scheme, coverage criteria and their evaluation in a study containing real world test and failure data. In chapter 4, our second major contribution in the direction of coupling based test case prioritization are presented. The insights derived from our studies in the conclusion of chapter 3 and 4 are then combined to form our contribution to the practice of system integration testing presented in chapter 5. Lastly, Chapter 6 briefly summarizes our work, broadly discusses its implications and limitations, and finally closes with an outlook on promising future research directions. The outline of this thesis is also shown in figure 1.1.

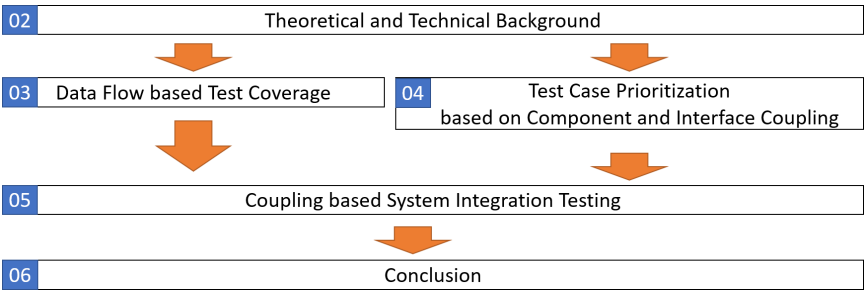



Figure 1.1: Thesis outline and structure

Chapter 2

# Theoretical and Technical 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 gray-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 not directly applicable.

In this chapter, we first present the architectural view commonly used in the automotive industry in section 2.1. These architectural views are widely used in this work. They provide the basis for identifying data-flow in section 3 and consequently the individual existing component interactions. In addition, they are also used in section 4 to calculate component and interface coupling. We then present the state of the practice of system integration testing as a continuation of the development process in order to highlight activities relevant for the context of this thesis. These process descriptions provide the baseline for the conceptualization of a coupling

based integration testing process in section 5. Followed by that, we introduce in section 2.3 the real-world system which provides the context for all data collected and studied during the course of this work. Finally, we present the theoretical foundation for the set of metrics selected for the study in section 4. In section 2.4.1 a data flow based approach of measuring coupling is presented. In section 2.4.2, 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 2.4.3.

## 2.1 Model Driven Development

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. An overview of the different levels of architectural models established in the automotive industry for the development of cars is provided by [Bro03] and shown in figure 2.1.

The usage of the presented architectural views is organized alongside the V-Model, which is widely adopted in the automotive industry. One of the main design activities relevant for the architectural views discussed in this section is the *System Architecture Design* which aims to describe the vehicle as a composition of hardware and software components. In addition to that, the activities for *Hardware Design* as well as *Software Design* focus onto the specification of individual components contained in the hardware- and software architecture. In figure 2.2, the activities of the V-Model are highlighted which are relevant for the architectural views described in this sections and studied in this thesis.

On the highest level, the software based functions are 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 [Bro03]. This level of architectural view provides an *Usage* point of view by focusing onto the human-machine interface and is structured in a hierarchical manner. Higher level functions are brought down

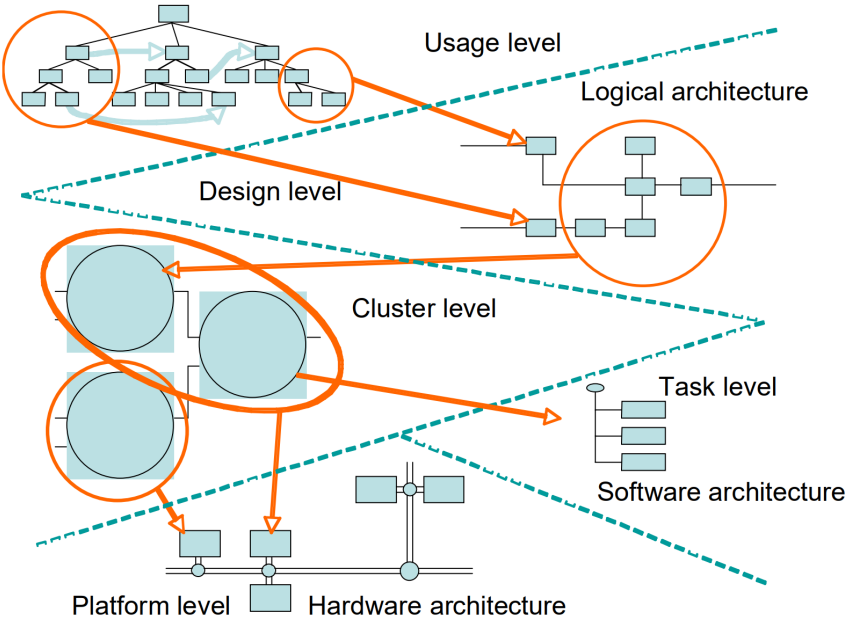



Figure 2.1: Architectural Views on Automotive Systems [Bro03]

into interconnected sub-functions.

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 sub-functions. One of the main goals of this architectural level is to describe the observable behavior on a software-structure level based on feature and function interaction while keeping each component and sub-function as independent from its implementation as possible. In recent work, service-oriented architectural concepts are used at this architectural level to decompose the functional hierarchy into services [GKJ16], [BFL08], [WZM11].

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 (indepen-

dent from the hardware design level and the partitioning of functionality onto different ECUs) 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 which is one of the main subjects of study in this thesis.

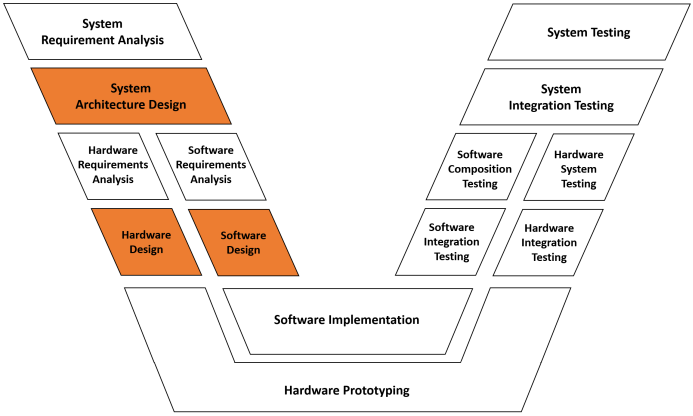



Figure 2.2: System- and Software Design relevant Activities of V-Model

## 2.2 System Integration Testing

In order to provide the foundation for the introduction of a Coupling based System Integration Testing process in section 5 we present existing work regarding system integration testing as practiced in the automotive industry. In theory, a test strategy used for integration testing prescribes how to verify the interplay effects and functional dependencies among multiple components and which test cases to prioritize to efficiently assure that every component satisfies the correct semantics of shared information [Int11][Int13]. In [NT08], Naik provides a general overview of the different aspects of System Integration Testing. However, in this section we revisit the state of the practice for integration testing of an automotive subsystem and provide related work for each covered aspect of the integration testing process.

In the automotive industry, integration testing is split into multiple stages as a modern vehicle consists of over 200 interconnected electronic control units each of it executing a component based software system on its own. After software unit testing and software integration testing, each ECU is tested individually using stand-alone ECU test benches as part of the software and hardware integration testing. The next intermediate step of integration

towards final vehicle testing involves multiple ECUs which are put together to form a coherent subsystem of the vehicle [SBH15]. The set of ECUs is assembled according to functional domains which are derived from the system architecture definition as described in section 2.1. In automotive industry, this step is called system integration testing and its goal is to verify the interactions of multiple ECUs during execution of a higher-order functionality.

In [HSS+13], Heidrich *et. al.* provide a classification model for hardware-in-the-loop (HiL) test benches commonly used for testing of such automotive subsystems starting from single stand-alone ECU test benches and ranging up to the complete integrated vehicle. The proposed model captures the *Object Level*, *Integration Level* and the *Connection Level* of a HiL test bench. The first Object Level describes the system under test which can either be an ECU, a certain Hardware Unit or the complete vehicle. The second Integration Level describes the degree of integration being subject of the testing. This can range from a single object to multiple functionally dependent or independent objects. The third Connection Level however describes the degree of interconnection of the HiL test bench ranging from a monolithic unit executing all test cases to a distributed HiL system where tests are executed from multiple units which are either remotely or directly connected to each other. In context of the architectural models described in section 2.1, these three aspects of a test-bench describe to which degree the individual parts of the vehicle can be tested on a certain test-bench. Consequently the *Object Level* describes which part of the architecture is covered by the test bench while the *Integration Level* captures which part of the architecture is present by its target implementation and which parts are simulated using mock-ups and abstracted behavior models. In figure 2.3, the classification model is shown. The HiL test benches typically used for system integration testing of automotive subsystems are highlighted within the classification model. These types of test benches are commonly used to conduct the test approach presented in this work.

In [NT08], Naik and Tripathy provide an overview of generic integration testing techniques commonly used also in non-automotive software related

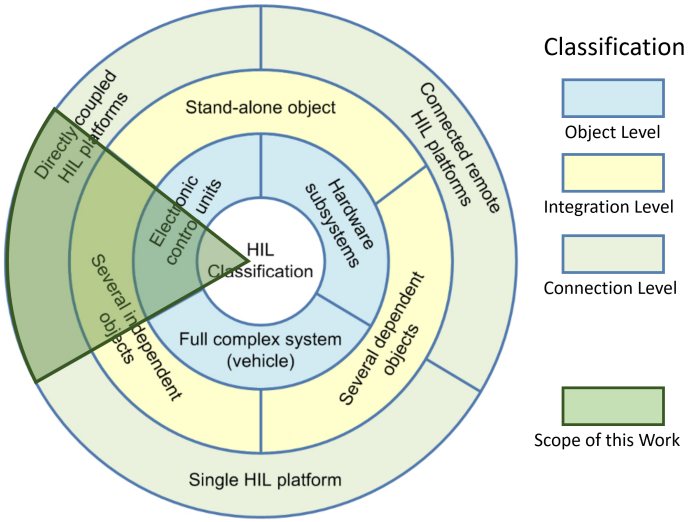



Figure 2.3: Topological classification of HIL test platforms commonly used in automotive industry [HSS+13]

integration testing. These are namely **Incremental**, **Top Down/Bottom Up**, **Sandwich** and **Big bang** integration testing and describe the step-wise integration of components until the desired composition of components is achieved. In the context of system integration testing of a vehicles subsystem, such step-wise integration approaches are applied as well. During the early stages of a vehicles series development project, the software for some ECUs contained in the subsystem under test become available later in time than others. In order to start system integration testing in such early states of development, abstract behavior models are used to stub ECUs for which no software is available or which don't fulfill the software maturity required for integration testing. Due to the fact that the ECUs contained in an automotive subsystem are arranged in a non-hierarchical system structure, the step-wise replacement of behavior models with the target ECU and its software functionality is done in an *incremental* manner during the different phases of development. Recent work exists covering the use of continuous integration

testing techniques for the development of vehicles. In a work presented by Boyang *et. al.* [DAM+21], a test framework is provided which describes a continuous integration testing approach ranging from SIL based tests up to the target ECU as the system under test. However, such approaches are mostly considered for HiL test benches which only cover a single ECU due to the fact that ECUs and their software are often developed by individual suppliers. Including multiple ECUs in an integration testing approach therefore comes which the challenge of introducing a common integration testing framework for multiple suppliers.

As described in section 2.1, 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 [Int11]. The motivation for integrating ECUs which are mostly developed and tested by different supplier companies into a subsystem of the vehicle is to assure a stable subsystem in a controlled environment. Consequently this assures that the subsystem can safely be tested in its actual environment during vehicle testing which is important for safety relevant software based functions like the steering and braking functions implemented by the chassis control system. In practice, test cases used for system integration testing of an automotive subsystem can be grouped according to the scope of testing or the subject of verification. The following scopes of testing are commonly used for system integration testing in the automotive industry:

- **Interface Integrity** An important part of system integration testing is the verification of outgoing and incoming message formats in order to find interface errors. For this, internal communication between two ECUs contained in the subsystem under test is considered as well as external communication to systems not contained in the subsystem under test. In order to verify external communication, the test bench simulates the external system using an environmental behavior model which contains a set of stubs for each ECU not contained in the subsystem under test. In detail, communication is checked for the

number and order of parameters, their size and data type as well as their representation/semantic.

- **End-to-End Protection** In automotive software, which implements the AUTOSAR standard for software design, safety critical communication is typically equipped with end-to-end communication protection mechanisms [AUTa]. Beside others, these mechanisms include alive-counters which are used to signal the correct functioning of the sending unit to all receivers as well as cyclic redundancy check used for message-level integrity. In addition, modern vehicles include a message-level encryption protecting the inter-ECU communication against manipulation. The correct functioning of these mechanisms is verified during system integration testing by simulating errors and checking for the correct detection in the right amount of time.
- **Function Verification** Software based functions which heavily rely on ECU interactions are included in system integration testing to ensure their correct functioning prior to vehicle testing. The selection of functions for the verification during system integration testing is done based on experience from past test phases.
- **System Endurance** An automotive subsystem is expected to stay up for longer periods of time without showing issues. Important endurance tests performed during system integration include continuous start-up/shutdown testing as the order to which ECUs are shutting down is specified based on dependencies and timing constraints. In addition there are also vehicle level test scenarios executed continuously over a longer period of time like driving maneuvers.

An additional aspect of integration testing are the methods used for test design and implementation. During system integration testing of an automotive system, test design, implementation and selection is commonly done based on experience from past series development projects. For this, test data is analyzed regarding groups of similar or reoccurring defects which are then used to guide the implementation of new test cases and the planning of their execution during the different test phases. For the integration testing of an automotive system, such experience based test strategies have proven effective due to the high reuse of ECUs and software among different vehicle models and vehicle generations as well as their high similarity in system and software architecture. However, due to the growing number of newly added software based functions to modern vehicles, the change in system design and architecture as well as its implementation between two consecutive generations is becoming significant. This increasing development progress causes the experience from past projects to be not fully applicable anymore.

In [SN13], Sobotka and Novák also addresses the need for test design and implementation which directly aims at the failure prone aspects of the system under test. In addition, they state that the commonly used method for test design and implementation is labor intensive and error prone due to the required manual steps. Caused by the fact that most automotive subsystems are distributed reactive real-time systems, Sobotka proposes the introduction of a formal specification of the system under test covering the systems architecture and the interaction behavior of individual components. Furthermore, such formalism should also cover aspects like time dependencies between different components, multiple time domains and parallelism as these are the main aspects of the target environment of automotive subsystems. Based on formal descriptions of the system under test, Sobotka and Novák then demonstrates the introduction of automated test-case generations based on an interlinked tool chain.

The need for a formal specification of component interaction related behavior is also pointed out by Shashank *et al.*in a systematic literature survey of integration testing [SCK10]. In their work, Shashank *et al.*categorized existing work about integration testing according to the used testing technique.

They found that most existing work focuses on formal models like *UML based models*, *Contract based Testing* or *Component Testing Model* which goal is to specify component interaction. Furthermore, Shashank *et al.*pointed out that in the absence of a formal approach, scenario based testing techniques are discussed like *State Machine based Testing* or *Feedback-Random Testing*. In addition to the applied testing technique, Shashank *et. al.*also grouped the existing work according to the addressed key challenges. They found that similar to the challenges addressed in this work, most existing work about integration testing of component based systems is limited to issues like *Lack of Source code* as well as *Difficulty in identifying the dependencies*.

In summary, much work exist studying the problems faced in the practice of integrating and testing automotive systems. However, there is a need for formal specifications of the aspects relevant specifically for system integration testing of black-box components. The work presented in this chapter in combination to the architectural models described in section 2.1 serves as the theoretical context to study aspects relevant for system integration testing in section 4 as well as the formalization of a systematic system integration testing process in section 5.

## 2.3 Description of the Studied System

As already mentioned in section 2.1 the overall set of software based functions implemented by a car is structured in functional hierarchies and software clusters. To further support the development of a car, this structure is carried over through all level of architectural views discussed to this point. For the studies presented in this work, we focus onto the chassis control subsystem of a vehicle which is the result of grouping components within the hardware architecture according to the functional domain of chassis control functions. The chassis control subsystem mainly implements distributed software functions for electronic stability control, adaptive damping control and rear-wheel steering.

In order to introduce and describe the chassis control system studied in

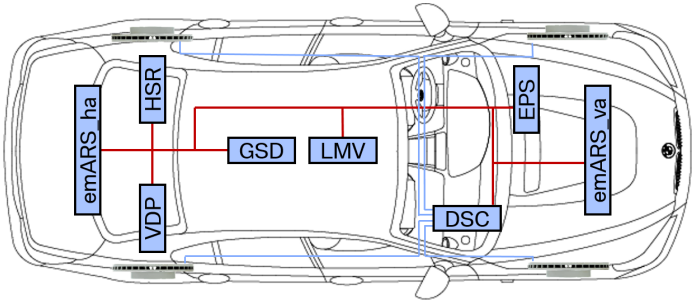



Figure 2.4: Illustration of ECUs used in Chassis Control Subsystem.

this thesis, there are two architectural views provided. First, the hardware architecture which describes the structure of the chassis control system in terms of ECUs, sensors and actuators. An overview of the hardware architecture is shown in figure 2.4 alongside a short description of each component in table 2.1. 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. In the context of this work, dependencies of one chassis control related ECU to the rest of the vehicle are taken into account as coupling to the environment which will play a major role in measuring the coupling of components and interfaces in section 4.4.

As each ECU represents a component-based software system in its own right, the second architectural level relevant to describe the studied system is the logical architecture. As described in section 2.1 the logical architecture provides the specification of observable interaction behavior implemented by different software component partitioned on different ecus. The list of ECU interactions in terms of data flow specified as part of interface specifications is provided in table 4.4. The set of ECUs contained in the chassis control system combined with the amount of interfaces implemented by each ECU will later in this thesis be used to locate failures found during its development in section 3.

Table 2.1: ECUs contained in the case system

| ECU | Description |
|---|---|
| HSR | Rear-Wheel Steering Controller |
| VDP | Vertical Dynamic Controller |
| emARS_ha | Anti-Roll System Controller (rear axle) |
| emARS_va | Anti-Roll System Controller (front axle) |
| GSD | Limited-Slip Differential Controller |
| LMV | Longitudinal Torque Distribution Controller |
| DSC | Brake Control System |
| EPS | Electronic Power Steering Controller |

## 2.4 Existing Coupling Measures

Coupling includes many aspects which need to be taken into account when defining a measurement. In [RNS20], Rizwan provides a theoretical evaluation of the aspects relevant in existing coupling measures as they are defined in literature. However, in this section we focus on coupling measures applicable for distributed component based software system containing black box software as these are typically present in the development of modern vehicles. Therefore, this section provides the theoretical foundation of the coupling measures which have been considered for the study provided in section 4. However, the selection of measures is provided in section 4.3.

In section 2.4.1, we introduce basic data flow based coupling measures which mainly cover the amount of shared data communicated between two coupled components. We then introduce a dependency based measurement of coupling which covers the amount of coupling pairs existing for a certain component of the system in section 2.4.2. Finally, we introduce information theory based measurement of coupling which aims to combine both the amount of data included in a coupling as well as the amount of coupling present for a certain component.

### 2.4.1 Data Flow Analysis

Data flow analysis is a common approach for identifying dependencies in a structured software design [UY93] [IS06]. 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[AMS+18] defines **Interface Coupling** ($IC$) within the context of data flow analysis in equation 2.1 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 which exists for any module in the system under test if its functions are used by other modules. 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. This assumption also serves as one of the key aspects of our proposition for fault-proneness caused by coupling in the research design of the study provided in section 4.2. At the component level, **Component Coupling** ($CC$) is defined as the sum of all interface coupling factors of all interfaces implemented by that component, i.e. $CC = \sum_{i=1}^{n} IC_i$. Similarly the component-based system coupling $CBSC$ is defined as the sum of all component coupling factors for each component contained in the system, i.e. $CBSC = \sum_{i=1}^{n} CC_i$.

$$IC = n * \sum_{i=1}^{p} OF_i \tag{2.1}$$

In [HK81] and [KH81], Henry and Kafura provide a brief discussion of information flow based module complexity. Similarly, Kumari provides an empirical and theoretical analysis of information flow based complexity [KU11]. The measure introduced in his work is shown in equation 2.2. Similar definitions are provided by Lakshmi [NH07] and Kharb [KS08]. 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 as an additional aspect. A module with a high value for information flow measurement indicates that it is strongly connected to its environment thus indicating a high coupling.

$$IIF = (Fan-in * Fan-out)^2 \tag{2.2}$$

The **Interface Information Flow** ($IIF$) as discussed by Lakshmi [NH07] and Kharb [KS08] 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.

### 2.4.2 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 4.3. In [Li03],[SGK09], [GB08] and[SKB11], an approach to manage dependencies in a component-based system using a matrix model is proposed which suggests 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 2.4. This matrix representation of component interdependencies can be applied to both directed and undirected depen-

dencies. 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 2.5. Because all values per row and column are added up, in- 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.

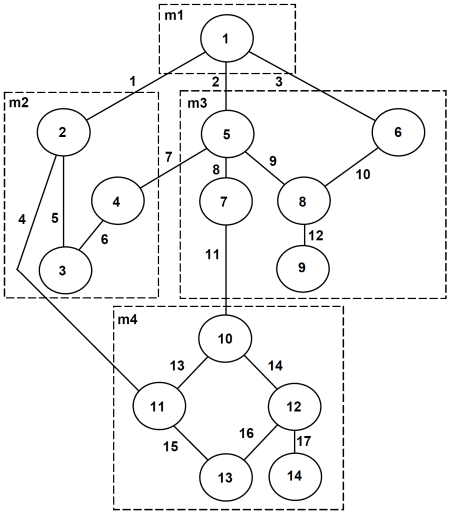
$$DM = \begin{bmatrix} d_{11} & d_{12} & \dots & d_{1n} \\ d_{21} & d_{22} & \dots & d_{2n} \\ \dots & \dots & \dots & \dots \\ d_{n1} & d_{n2} & \dots & d_{nn} \end{bmatrix} \tag{2.3}$$

$$d_{ij} = \begin{cases} 1 & \text{if } c_i \rightarrow c_j \\ 0, & \text{otherwise} \end{cases} \tag{2.4}$$

$$DC(C_k) = \sum_{j=1}^{n} [d_{kj}] + \sum_{j=1}^{n} [d_{jk}] - 2[d_{kk}] \tag{2.5}$$

### 2.4.3 Information Entropy

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 [AK99][All02][AKC01][ASR09]. According to Allen, a software system can be represented as a graph in different ways to highlight different aspects of its design[All02]. Common examples of such design abstractions for object-oriented designs are given in a framework proposed by Briand *et. al.*, which is used to measure coupling




| Module | Node | Edges | $p_{L(i)}$ |
|---|---|---|---|
| Environ. | 0 | 0000000000000000 | 2/15 |
| $m_1$ | 1 | 1110000000000000 | 1/15 |
| $m_2$ | 2 | 1001100000000000 | 1/15 |
| | 3 | 0000110000000000 | 1/15 |
| | 4 | 0000011000000000 | 1/15 |
| $m_3$ | 5 | 0100001110000000 | 1/15 |
| | 6 | 0010000001000000 | 1/15 |
| | 7 | 0000000100100000 | 1/15 |
| | 8 | 0000000011010000 | 1/15 |
| | 9 | 0000000000010000 | 1/15 |
| $m_4$ | 10 | 0000000000101100 | 1/15 |
| | 11 | 0001000000001010 | 1/15 |
| | 12 | 0000000000000101 | 1/15 |
| | 13 | 0000000000000011 | 1/15 |
| | 14 | 0000000000000000 | 2/15 |

Figure 2.5: Undirected graph abstraction of an example system [All02]

and cohesion based on class inheritance, method invocation or class-attribute references[BDW99][BDW97]. In [All02], Allen states that a more sophisticated measurement than just the count of features of an artifact will be more useful in determining coupling and cohesion.

In his work, Allen provides an example of a modular system which is shown in figure 2.5 in a graphical representation as well as in a table listing nodes and edges. The system contains 14 nodes, which are partitioned into four modules and connected by 17 edges. To apply information theory based measures, an additional unpartitioned node is added to the graph, which represents the system's potential dependency to its environment. In the system's $node \times edges$ table, each node is represented as a row while edges are represented as columns. Each cell therefore indicates if the node is an endpoint of that edge. As a consequence, all cells of a row form a binary-pattern depicting the design decisions of the node's coupling to the rest of the system. $P_{l(i)}$ represents the probability mass function based on the proportion of distinct row patterns.

Table 2.2: Working graph abstractions [All02]

| Symbol | Name | Definition |
|---|---|---|
| $S$ | System | Graph abstraction of the software system |
| $S^{\#}$ | Edges-only graph | Edges in $S$ and end points |
| $S_i$ | Node subgraph | Nodes in $S^{\#}$ and edges incident to node $i$ |
| $MS$ | Modular system graph | $S$ partitioned into modules |
| $m_k$ | module k | Nodes in a module and their incident edges |
| $MS^*$ | Intermodule edges graph | Nodes in $S$ and intermodule edges |
| $MS^\circ$ | Intramodule edges graph | Nodes in $S$ and intramodule edges |

Given the graphical and tabular representation of a system graph $S$, Allen introduces a set of operations to create working copies of the system graph. A subset of the definitions used in this work are listed in table 2.2. An intermodule edge graph $MS^*$ contains all nodes and modules of $S$ but only edges with endpoints in different modules. An intramodule edge graph $MS^\circ$ complementary contains all nodes and modules of $S$, but only edges with endpoints in the same module. The intermodule edge graph of the example system $S$ is shown in figure 2.6. An edges-only graph $S\#$ and node subgraph $S_i$ are generated as described in table 2.2.

In information theory, coupling and complexity are measured based on the entropy of the distribution of row patterns $H(S)$ [Gra07]. 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 2.3, the size of a node requires the calculation

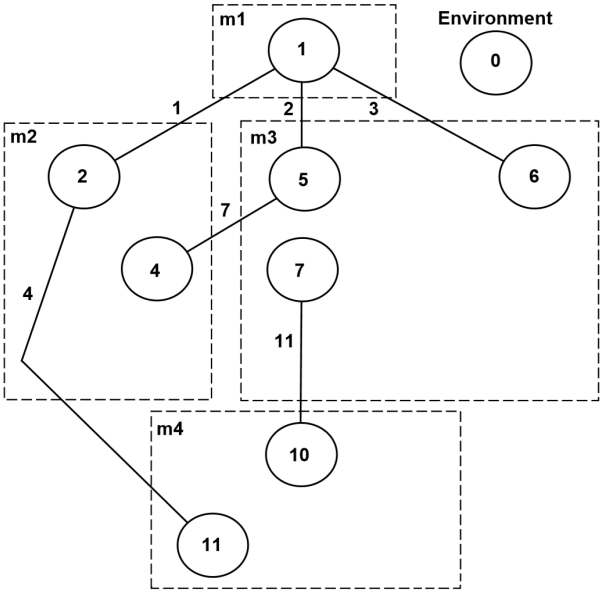



Figure 2.6: Intermodule edge graph $MS^*$ [All02]

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 2.4.1. 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.

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 2.3. According to Allen, excess entropy represents

Table 2.3: System- and Module-level measures [All02]

| | | | 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 | $Size(m_l\|S)$ | = | $\sum_{i\in m_k}(-\log p_{l(i)})$ |
| 4 | $Complexity(S)$ | = | $\sum_{i=1}^{n} Size(S_i)-Size(S^{\#})$ |
| 5 | $Complexity(m_k\|S)$ | = | $\sum_{i\in m_k} Size(S_i)-Size(m_k\|S^{\#})$ |
| 6 | $Coupling(MS)$ | = | $Complexity(MS^*)$ |

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.

The results of the presented measures of size, complexity and coupling for the example system graph provided by Allen are listed in table 2.4. Even though the goal of Allen's work is to contribute to the design phase of a system by providing a comprehensive software quality model, we believe that such a model can be used to predict failure prone component interaction based on the assumption in chapter 1.3 that due to cognitive overloading, not all information of a software component's context is considered during its development or evolution.

In summary, the information theory approach of measuring coupling in a component based system is built up around information entropy and the amount of information added to the interconnected system by individual modules. This approach has been selected for the studies provided in section 4 due to the fact that it is directly applicable to the architectural views commonly used in the automotive industry and described in section 2.1.

Table 2.4: Resulting measures for example graph [All02]

| Scope | Size | Complexity | Coupling |
|---|---|---|---|
| $S$ | 53.7 bits | 170.1 bits | 50.2 bits |
| $m_1 \mid S$ | 3.9 bits | 15.8 bits | 12.9 bits |
| $m_2 \mid S$ | 11.7 bits | 38.0 bits | 12.6 bits |
| $m_3 \mid S$ | 19.5 bits | 62.6 bits | 17.0 bits |
| $m_4 \mid S$ | 18.5 bits | 53.8 bits | 7.7 bits |

This applicability is mainly caused by the fact that the definition of a *node* and *module* is independent from the programming paradigm used in development of the studied system. As a consequence, coupling based on information entropy can be calculated for the functional architecture or hardware architecture in addition to just the software architecture.

Chapter 3

# Data Flow based Test Coverage

The first contribution chapter comprises our studies in the field of measuring test coverage during integration testing of black-box components using data flow based behavior observation. We introduce a data flow classification scheme which allows the identification of data flow usage within existing test cases and failure reports in section 3.3. Knowing which data flow in a system plays a dedicated role in the execution of test cases or the occurrence of a failure at run-time enables us to determine the data flow coverage of any given test set which represents the foundation of the coupling based system integration testing process proposed in section 5. Given the data flow classification, we further discus a concept of identifying degrees of similarity in the usage of data flow when comparing different test cases and failures in section 3.4. Comparing the usage of data flow in a test case to the usage in failures allows us to evaluate the usefulness of data flow based test coverage measurement in section 3.6. In order to complete the definition of a data flow based test coverage measurement approach for system integration

testing we also define data flow based coverage criteria in section 3.5. Lastly, we evaluated the suggested approach in a real world development project and provided a step-by-step example of the test gap identification using the introduced classification scheme and coverage criteria in section 3.6. The work presented in this chapter extends the following publications and puts them into the perspective of this work:

D. Hellhake, T. Schmid, and S. Wagner. "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)*. Apr. 2019, pp. 420–429

D. Hellhake and S. Wagner. *Kommunikationsfluss-orientiertes Testen von Softwarefunktionen im Steuergeräteverbund*. Tech. rep. 7. FACHKONFERENZ AUTOTEST, 2018. URL: `https://fkfs-veranstaltungen.de/fileadmin/4_AutoTest/pdf/AutoTest_2018/`

## 3.1 Related Work

The topic of integration testing of component-based software systems has already been studied in previous work [SCK10][Spi95] and more recently in [KE]. However, most of the related work proposes design-based methods which rely on software architecture descriptions and structured design diagrams. As this work is heavily inspired by code-based methods like interprocedural data-flow testing [FW88][HS91][MCT08] and their extension to coupling-based methods [MS12] this section provides a recap of the topics closely related to this work in order to support the identification of the research gap in section 3.2. The provided work can be grouped according to the identification of component interactions either based on the systems design or based on the systems dynamic behavior.

In [AO00] and [WCO03] the interaction of components of a software system is described using UML collaboration diagrams. The authors present an approach for automatically generating test input data based on these UML collaboration diagrams to check interaction-related behavior for potential

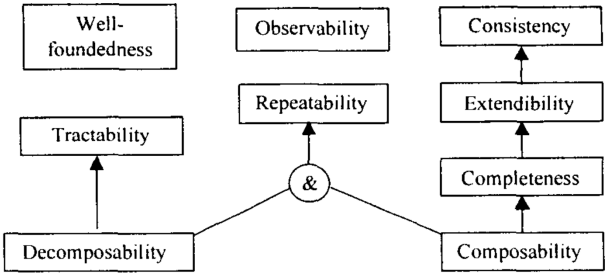



Figure 3.1: Quality Model for Observation schemes [ZH01]

verification during integration testing. These diagrams are created based on the high-level design notations of the system. Even though the approach is presented using an object-oriented software system, describing the composed behavior of multiple components using interaction sequences is very similar to the black-box integration problem subject of this study. The paper also clarifies the advantage of using high-level design notations of the system under test as they are available in early stages of a development project which supports test implementation and planning.

In a work provided by Zhu et. al., an observation theory for component-based software development is formalized and applied to white-box integration testing[ZH01]. The observation theory introduces the set of observable behaviors of a software system and an observation scheme which associates a certain test suite to a collection of phenomena which are observed by executing the test suite on the software system. Zhu *et. al.* further provide a brief definition of properties which have to be satisfied for any well-defined observation scheme. In particular, these properties and the relation among each other shown in figure 3.1 do form a quality model for observation schemes in terms of its fault detection abilities. The contributions provided by Zhu *et. al.* did heavily inspire the definition of the data flow based classification scheme in section 3.3. Furthermore, the definition of a phenomenon alongside different phases of the system's dynamic behavior are directly derived from the presented observation theory.

An approach for learning the behavior model of a component for integration testing is provided by Elsafi *et. al.* [EJA14] who also identified the lack of available formal methods for performing integration testing of components for which the source code is not available. In their work, Elsafi *et. al.* refine the approach of actively learning the behavioral structure of components which has been introduced by Shahbaz and Groß [SG14]. In its essence the proposed approach records a component's behavior in terms of a set of output parameters generated by the component based on a given set of input parameters. These transitions are captured in an observation table during execution of the components either during isolated component tests or as part of the integrated system. Based on a complete or partially complete observation table a Mealy machine is defined formally which consists of an observed component-state, the set of executed input and output parameter and the functions any observed input has on the component-state as well as the generated output. These approximated behavior models can then be used for integration testing to identify equivalence classes for the input parameters of each component in the system. In the context of this thesis, the presented approach is applicable for the reduction of the overall test space spanned by the data flow based component interaction. However, the applicability and effectiveness in reducing the test space needs to be evaluated in an additional study as it is out of scope for this thesis.

Lee *et. al.* made a comparison of failure symptoms, such as stack traces and symptom strings, as a strategy for identifying recurrences of failures[LIM94]. A stack trace being the history of procedure calls at the time the failure occurred represents the chronological sequence of data flow while a symptom string identifies the location at which the failure has been detected. The symptoms of failures are generalized into data and code-oriented symptoms. The theory of the work provided by Lee is very similar to the observation theory discussed by Zhu and He as the occurrence of a failure is an instance of a phenomenon observable for the system's dynamic behavior. Lee further discusses cluster of failure symptoms as shown in figure 3.2 which are defined along the similarity of the symptoms of multiple failures. For this he provides a definition of *complete matching*, *partially matching* and *weighted*

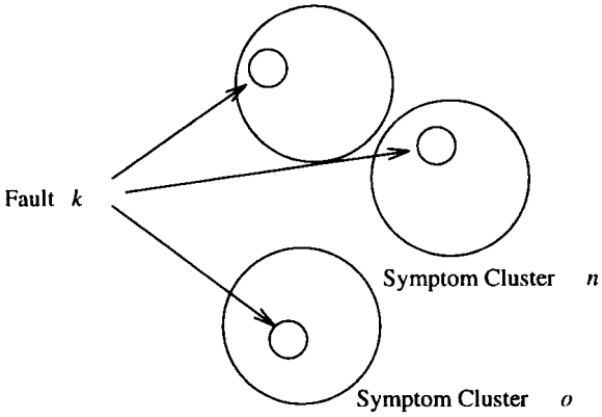



Figure 3.2: Failure symptoms and symptom cluster [LIM94]

*matching* in order to identify the strength of similarity of the symptoms of two failures when compared to each other. This definition inspired the data flow similarity definition provided in section 3.4 and is used for evaluation of the data flow based approach of test coverage analysis in this chapter.

Jin *et. al.* formalized a coupling measure which classifies the impact a certain data flow has upon interacting components[JO96][JO98]. Similar to the proposition provided in section 4.2, Jin *et. al.* states that faults typically occur at highly coupled components during integration testing. Based on that he adopts the definition of modularity along the term cohesion and coupling made by [YC79] to define a coupling model applicable for integration testing. In its essence, coupling is classified into three categories of inter-component data flow which do form an unordered aggregation of the twelve types of coupling defined by Offutt et al. [OHK93][Pag01]. To identify data flow in a given program, Jin uses the basic definition of data flow analysis from White [Whi87]. By analysing the programs *control flow graph* for definitions of variables and their usage in an either non-local way or as part of a inter-component method-invocation the overall space of data flow based component interactions is identified. Given the typification of

inter-component data flow and its identification based on the control flow graph of the system under test, Lee provides the following basic coverage criteria which are similar to generic data flow based path- and decision coverage analysis.

- **Parameter coupling:** Refers to the data flow involved in direct inter-component method invocation. Covers *scalar data coupling*, *stamp data coupling*, *scalar control coupling*, *scalar data/control coupling*, *stamp data/control coupling* and *tramp coupling*.
- **Shared data coupling:** Refers to indirect data flow involved in non-local shared parameter. Covers *non-local coupling* and *global coupling*.
- **External device coupling:** Refers to the use of shared resources. Covers *external coupling*
- **Call coupling:** Requires a test set which covers all paths involving inter-component method invocations.
- **All-coupling-defs:** Requires that for all definitions of shared variables at least one path is covered including at least one shared data use.
- **All-coupling-uses:** Requires that for all definitions of shared variables at least one path is covered to all shared data uses.
- **All-coupling-path:** Requires that for all definitions of shared variables all paths are covered to all shared data uses.

An empirical evaluation of the theory of data flow based test coverage is provided by Frankl *et. al.*[FI98]. Frankl *et. al.* presents the results of an experiment comparing the effectiveness of an all-uses data flow, which is equivalent to the *all-coupling-path* criterion, testing criterion to decision coverage (branch testing) and random testing without any adequacy criterion. Multiple large C programs were used all of which containing a fault introduced during the development process. He found that for all programs, test sets with a high level of data flow coverage were significantly more likely to detect the fault than random test sets of the same size. Furthermore he found that even for test sets with a high level of coverage, the likelihood

of detecting the fault was still limited. Even though this work does not utilize code-based coverage measures, the coverage criteria presented in this work are similar to the all-uses criteria. In addition, this work also utilizes the measure of test effectiveness to decide whether using data flow-based coverage criteria outweighs the costs.

A quality model for test criterion is given by Frankl *et. al.*[FW88] alongside different applicability properties. The applicability property for a test criterion requires that for every program there is a test set which fulfills the criterion. The author states that some test criteria such as path coverage may not satisfy the applicability property for programs which contain loops and therefore potentially infinitely many paths. Furthermore, it is stated that for many test criteria the existence of a test set which fulfills the criterion for a given program is undecidable. However, the proposed data flow test adequacy criterion satisfies the adequacy criterion by focusing only on definition use associations which are executable. Furthermore, the distinction between computational-use and predicate-use of defined variables puts emphasis on the impact a certain variable has on the execution of a program.

In summary, the provided related work provides an empirically studied foundation for the concepts of data flow classification as well as for data flow based behavior observation. However, the discussed approaches for data flow classification like the one proposed by Offutt et al.[OHK93][Pag01] require a white box perspective onto the system under test in order to be fully applicable. This is also true for the behavior observation model proposed by Zhu et. al.[ZH01]. In this section, we therefore adopt the provided concepts for behavior observation in white box software systems to define *data flow classification scheme* in section 3.3 applicable for systems containing black box components. To evaluate our proposed approach, we introduce the concept of *data flow similarity* in section 3.4. Lastly, we provide a definition of *coverage criteria* in section 3.5.

## 3.2 Research Design

During integration testing of a component based system with sufficient complexity, not the entire domain of component interactions can exhaustively be tested. To make integration testing a manageable process, testing must be guided by the system's design and therefore by the modularization and composition of components. Many systematic approaches exist for testing the interaction and interplay behavior of a component-based software system. However, most of the mentioned approaches rely on the availability of information about the implementation and code structure of each component. Because an automotive software system consists mostly of ECUs whose hardware and software development is outsourced, such information are typically not available for the process of black-box system integration testing.

In testing of large-scale software systems, testers can only observe and verify certain aspects of the system's dynamic behavior. In the previous section we presented related work which is concerned with observation theory in testing. However, an adoption of these white-box testing approaches to integration testing of black box components is needed. This includes the formalization of axioms for observability, repeatability, consistency and completeness of an observation scheme as well as the behavior model of the system and the definition of applicable coverage criteria.

In this study, we aim to close this gap of inter-component behavior observation in a distributed software system containing black-box components. As a baseline for the research question we provide the proposition that data flow is a significant indicator for component interactions in a component based system containing black-box components. The hypotheses are derived from the research question as provided below in accordance to Runeson and Höst [RH08][RHRR12]. The overall structure of the case study design is summarized in figure 3.3.

- **RQ:** How effective is inter-component based data-flow classification in capturing the system's dynamic behavior relevant for test coverage

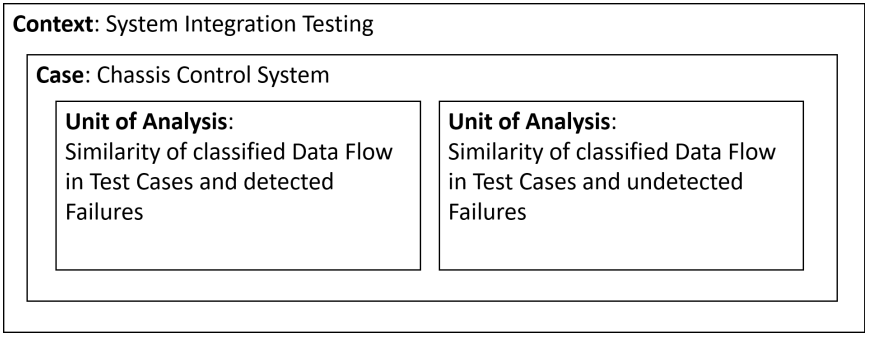



Figure 3.3: Embedded case study design for the study of inter-component base data-flow according to Runeson [RH08]

analysis and test gap identification?

By focusing on the goal of measuring test coverage and identifying potential test gaps, we further provide the proposition that aspects of the system's dynamic behavior relevant to describe the occurrence of a failure are also relevant to describe the execution of a test case. When mapped to the concept of data-flow based behavior observation, this means that inter-component based data-flow addressed during the execution of a test case will also be relevant for triggering and observing any failure found by that particular test case. As a consequence, data-flow relevant in the occurrence of a failure which has not been detected by executing a given set of test cases, is also not relevant in describing the systems behavior during test execution. This proposition stems from the assumption that in a distributed reactive real-time system, each failure as a derivation from the expected behavior can be triggered based on a defined precondition by applying a specific sequence of component interaction. Given that proposition, we will answer the research question in a two-step approach. Based on the definition of the data flow classification scheme in section 3.3 and the concept of data-flow similarity between test cases and failure reports, we first measure the similarity in data flow relevant for test cases and failure reports detected and created during their execution. In a second step we measure the similarity of test cases and

Table 3.1: Our two hypothesis pairs for measuring the effectiveness of data-flow classification for test coverage analysis

| Alternative hypothesis | | Null hypothesis | |
|---|---|---|---|
| $H_{1a}$ | There is a significant similarity in data flow required to describe the systems dynamic behavior during test case execution and the occurrence of a failure found by that particular test case. | $H_{1n}$ | There is no similarity in data flow required to describe the systems dynamic behavior during test case execution and the occurrence of a failure found by that particular test case. |
| $H_{2a}$ | There is a significant difference in data flow required to describe the systems dynamic behavior during the occurrence of failures and existing test cases if the failure have not been detected by the test cases. | $H_{2n}$ | There is no significant difference in data flow required to describe the systems dynamic behavior during the occurrence of failures and existing test cases if the failure have not been detected by the test cases. |

failure reports slipped by the test case execution. The hypotheses tested in this study are derived from the research question and accepted or rejected based on the measured data-flow similarities as part of the evaluation provided in section 3.6. The list of hypotheses is given in table 3.1.

## 3.3 Data Flow Classification Scheme

In order to provide evaluation data for the hypothesis provided in the previous section, we briefly introduce a classification scheme in this section which allows for the identification of relevant data flow in real-world test cases and failure data.

The overall inter-component communication of the chassis control system studied in this work has been extracted according to the step-wise approach described in section 5.1. It consists of 1343 distinct variables which form the

overall universe of observable information of the system. In order to describe a sequence of component interactions as part of the systems dynamic behavior in a semi-formal way, we define the term **Phenomenon**. A phenomenon consists of as set of chronologically ordered references to shared data which are required to describe a certain dynamic behavior. Such a description of the systems behavior is, similar to the definition in [ZH01], partially ordered by a binary relation x ≤ y which means that the shared data referenced in phenomenon y subsumes the information referenced in x. Furthermore, the summation of data flow referenced in a set of phenomena $p^1 + p^2 + ... + p^n$ can be expressed by $\sum_{i=1}^{n} p^i$.

For this study, a phenomenon either represents a certain test case execution or a failure occurrence recorded in a failure report. In case of a test case execution, the phenomenon is referencing those shared data, which are required to describe the systems behavior during test execution as well as the expected behavior. The phenomenon of a failure occurrence, however, contains those shared data which are required to describe the circumstances under which the fault is triggered to surface as a failure as well as the shared data required to observe the actual failure behavior as a derivation from the expected and specified behavior.

To extract the set of shared data from existing test cases and defect data, we introduce a data flow observation scheme which associates each test case or failure report with its representative set of relevant data flow. Each of these associations is further annotated according to the purpose for which that shared data is referenced. The potential purposes for which shared data can be used is designed in accordance to the general structure of a test case specification [Int08][LHL18] and described as follows:

- **Precondition:** In order to execute a test case or to trigger a failure to surface, the system under test has to be in a certain state. Such a precondition is formulated as a set of predicates on shared data for test cases and failure reports. Within commonly used test case specification templates, the preconditions are often referred to as initial states of the system under test. The use of shared data within these predicates

is captured by the observation scheme as a *precondition-use*.

- **Stimulation:** The test-input or trigger-sequence of a certain fault as a set of shared data manipulated during the execution of a test case is captured by the classification scheme as a *stimulation-use*.
- **Verification:** Shared data which is observed during the execution of a test case in order to verify the correct behavior of the system, is captured by the observation scheme as *verification-use*. For a failure occurrence, the set of verification-use references represents those shared data on which the failure behavior of the system can be observed.

Applying the observation scheme to fully automated test cases results in data flow profiles which represent an abstraction of the actual test scripts. Therefore, the data flow profile of existing test cases can automatically be generated. Steps required for applying the classification scheme to either fully automated or manual test cases are provided in section 5.2. The profile of an example test case is listed in table 3.2 showing the referenced shared data and involved ECUs. In this test case, the freewheeling protection of the electronic parking brake is verified.

**Precondition:** For the test case to execute, the vehicle has to be stationary with the electronic parking brakes engaged. In addition, the neutral-gear needs to be selected.

**Stimulation:** A vehicle speed greater zero is then stimulated during test execution to simulate a freewheeling situation.

**Verification:** The expected behavior of the chassis control system ECUs includes the setup of hydraulic brake pressure to stop the vehicle from moving. After the vehicle has reached its stationary condition, refastening of the electronic parking brake is commanded and a gear change from neutral to parking is communicated to the gearbox controller.

The data flow sequence of this test covers the interaction between the brake control system, the electronic parking brake and the gearbox controller. The data flow coverage of this test case is represented as the sum of shared

Table 3.2: Exemplary Data-Flow Profile of a functional Test Case

| **Data Flow Usage** | **Shared Data** | **Involved ECU** |
|---|---|---|
| Precondition | V_VEHICLE | Brake Control System |
| | COND_PBRK | Parking Brake |
| | GEAR_SELECT | Gear Box Controller |
| Stimulation | V_VEHICLE | Brake Control System |
| Verification | V_VEHICLE | Brake Control System |
| | HYD_BRK_TRQ | Brake Control System |
| | V_VEHICLE | Brake Control System |
| | COND_PBRK | Parking Brake |
| | REQ_GEAR_SELECT | Brake Control System |

data referenced for the respective purpose of use w.r.t. the overall flow of shared data.

In order to extract the data flow of a failure report, it has to be broken down into the minimum necessary conditions required to reproduce its occurrence. This is usually done when reproducing the failure occurrence during fault localization analysis [WGL+16] based on traces which are a mandatory part of the failure reports. Table 3.3 lists the data flow profile of an example failure which could have been detected by executing the test case shown in table 3.2. Due to an early state of maturity, the interaction between brake control system and the gearbox controller has not been implemented yet. This led to a missing gear change command after the vehicle had reached its stationary condition. Because the failure depicted in table 3.3 can be found by the test case depicted in table 3.2 these two data-flow profiles are considered associated for the purpose of this study. Comparing the set of data flow of the failure listed in table 3.3 and the data flow listed for the example test case listed in table 3.2 results in a similarity which is further detailed in section 3.4.

Table 3.3: Exemplary Data-Flow Profile of a Failure

| Data Flow Usage | Shared Data | Involved ECU |
|---|---|---|
| Precondition | V_VEHICLE | Brake Control System |
| | COND_PBRK | Parking Brake |
| Stimulation | V_VEHICLE | Brake Control System |
| Verification | REQ_GEAR_SELECT | Brake Control System |

## 3.4 Data Flow Similarity

When comparing phenomena of test cases to those created from failures, similarities in terms of shared data referenced in both phenomena can be observed. For this work, different types of similarities are introduced and used for the evaluation of the suggested approach of measuring test coverage based on the utilization of data flow provided in section 3.6.

The goal of introducing different types of similarities is to capture the *functional dependencies* as well as the *physical dependencies* existing among multiple component interactions and subsequently among multiple data flow. An example of such a functional dependency between individual component interactions can be provided based on the calculation of the vehicle speed as a shared variable. The vehicle speed is mainly calculated based on the wheel speed measured by sensors for each individual wheel. Therefore, a test-case which manipulates the shared date containing the wheel speed values subsequently stimulates the vehicle speed which may trigger a failure as a subsequent effect. In addition, an example physical dependency can also be named for the vehicle speed which, as a shared data, depends on the longitudinal acceleration as any positive or negative acceleration results in a change of the vehicle speed over time. To some degree, such subsequent effects caused by functional or physical dependencies can be compared to the number of subsequent method calls before the occurrence of a failure in code coverage analysis. Within the definition of coupling provided in section 4.3, such subsequent effects are instances of indirect coupling between two or more components of the system using additional intermediate components.

A subsequent match is therefore identified in case a certain shared data is referenced in the phenomenon of a failure but not in the phenomenon of the associated test case if there is a dependency on any of the shared data of the test case within the same purpose of use.

*Match:* A match represents a certain element of shared data to be referenced in the phenomenon of a failure as well as in the phenomenon of a test case for a given purpose. The amount of matches can be identified automatically.

*Subsequent Match:* Since in an automotive system, functional and physical dependencies between multiple shared data exist, subsequent effects of a test case execution can trigger a fault to surface as a failure. A subsequent match represents a certain shared data referenced in the phenomenon of a failure but not in the phenomenon of the associated test case. The data is identified as subsequent if there is a dependency on any of the shared data of the test case within the same executional step. Within this study, subsequent matches have been identified manually during review of non-matching data flow references. An automated identification of subsequent effects would require knowledge of the software structure for each ECU in the subsystem under test.

*No Match:* A certain element of shared data referenced in the phenomenon of a failure but not referenced in the phenomenon of the test case is identified as not matching if there is no reference to the same shared data and no functional dependency on any of the referenced shared data in the profile of the test case.

## 3.5 Coverage Criteria

The most widely used criteria include basic code-based coverage criteria, such as statement and branch coverage [RUCH99], function coverage [RUCH01][EMR02], block coverage [DRK04], modified condition/decision coverage [JHS03][FCX12], method coverage [DRK04], and statically estimated method coverage [ZZH+09][MHZ+12]. Due to the unavailability of

the code structure for the black-box integration testing process as subject of this study, traditional data-flow testing criteria as listed in [SWM+17] are not applicable. However, to address the question of when to stop testing during system integration we introduce an coverage criterion specifically designed to focus on data-flow in a distributed black-box software system. The below defined criteria utilize the information about the purpose to which a certain shared data is used in an existing test suite to measure if it is adequately covered.

- **Shared-Data-Use:** A Shared-Data-Use occurs if, for a shared data *d*, at least one test case exists which contains at least one reference to *d* for any of the purposes defined in 3.3. This criterion can be used to identify data flow which is completely untreated in a given test suite.
- **Precondition-Data-Use:** A Precondition-Data-Use occurs if for a shared data *d*, at least one test case exists which contains a reference to *d* in a precondition-predicate. This criterion can be used to analyze the variance within the precondition of a group of coherent test scenarios.
- **Stimulation-Data-Use:** A Stimulation-Data-Use occurs if, for a shared data *d*, at least one test case exists which contains a reference to *d* for the purpose of a manipulation. A manipulation of shared data represents an induced interaction of the respective sending ECU with all its receiving ECUs. As the goal of integration testing is to verify such component interactions, this criterion can be used assemble the set of interactions induced into the system by a given test suite.
- **Verification-Data-Use:** A Verification-Data-Use use occurs if, for a shared data *d*, at least one test case exists which contains a reference to *d* for the purpose of behavior verification. Using this criterion, untested data flow can be revealed.

These test adequacy criteria resemble the basic ones. Given the information about the purpose of data flow-usage in test suites as defined in section 3.3 there are more criteria to be formalized. In a future study, information about

the type of certain shared data should be used to formalize adequacy criteria, which address the combinatory within the usage of multiple shared data.

## 3.6 Evaluation

The goal of the evaluation is to show the effectiveness of data flow based behavior classification in measuring test coverage and identifying potential test gaps for black box system integration testing. In this study, the evaluation is split into two parts as also introduced in section 3.1. First, we study if the data flow used in test cases is also applied to the failures detected by that test cases. This will address our first pair of hypothesis. In addition, we study if data flow required to observe failures which have not been detected by a set of test cases is also not be covered by any of the existing test cases which cover our second pair of hypothesis. To investigate this, the approach of measuring similarity between test cases and failures in terms of data-flow observation is used as defined in section 3.4. To further highlight the usefulness of data flow classification for the practice of system integration testing in the automotive industry, we provide the step-wise identification of potential test gaps based on the data flow similarities measured in this study.

For the evaluation we studied test and defect data of a vehicle series development project covering the system integration test phase of the chassis control subsystem introduced in section 2.3. The data-flow classification of test cases and failure reports was performed by a team of test specialists also involved in analyzing failures found during testing of the chassis control system. To reduce biases within the classification, it has been ensured that the same team member does not classify test cases and its associated failure reports. The overall number of classified test cases and failure reports are listed in tables 3.4, 3.5 and 3.6 together with the data-flow coverage measured in contrast to the overall data flow of the system under test. The selection of test cases and failure reports is described as follows:

**Test Cases:** We randomly selected 90 test cases for data-flow classification.

We ensured that each test case has found at least one fault which was fixed successfully during the course of the project. In addition, for each ECU contained in the chassis control subsystem the same amount of test cases was used. The result of the data-flow classification of each test case is shown in table 3.4.

**Detected Failures:** By ensuring that each test case has found at least one fault, the amount of associated failures is assembled by the sum of failure reports created by the selected test cases during execution. As described in section 3.2, the set of associated failures represent the control group for this study as each associated failure should yield a high similarity in terms of its data flow phenomenon when compared to the test case by which it was found.

**Undetected Failures:** According to the research design of this study provided in section 3.2, faults that slipped by during the system integration-testing phase are assumed to be the result of potential test gaps within the existing test cases. Instead of being found by system integration testing efforts, these faults are either found during system testing using prototype vehicles or other consecutive testing activities. The amount of failure reports detected by other testing activities but affecting component interaction behavior relevant for system integration testing of the system under test are selected randomly. For each ECU contained in the chassis control subsystem the same amount of failure reports has been selected. As described in section 3.2, the set of undetected failures is used as the treatment group for this study as each of the undetected failures should yield a significant low similarity when comparing its data flow phenomenon to each phenomenon of the selected test cases.

Comparing the amount of referenced shared data for each purpose to the resulting data-flow coverage, reveals that shared data is referenced by multiple test cases and failure reports. In addition, it can be seen that phenomenon of failures (table 3.5 and 3.6) contain less references to shared data than phenomenon of test cases (table 3.4). This can be explained based on the process of fault localization performed after a failure has been detected during which a failure is broken down to the malfunction of a single ECU

Table 3.4: Results of Shared-Data-Use Coverage of selected Test Cases

| Name | Precondition | Stimulation | Verification |
|---|---|---|---|
| Referenced Shared Data | 435 | 282 | 359 |
| Shared-Data-Use coverage | 6,0 % (81/1343) | 5,6 % (75/1343) | 7,1 % (95/1343) |

Table 3.5: Results of Shared-Data-Use Coverage of Failures found by selected Test Cases

| Name | Precondition | Stimulation | Verification |
|---|---|---|---|
| References to Shared Data | 90 | 52 | 53 |
| Shared-Data-Use coverage | 2,4 % (32/1343) | 2,2 % (29/1343) | 2,8 % (38/1343) |

Table 3.6: Results of Shared-Data-Use Coverage of undetected Failures

| Name | Precondition | Stimulation | Verification |
|---|---|---|---|
| References to Shared Data | 100 | 60 | 63 |
| Data-Flow Coverage | 2,5 % (33/1343) | 2,8 % (37/1343) | 3,6 % (49/1343) |

that causes the problem. Consequently, test cases used for system integration testing have a focus onto the integrated system including interaction behavior of all contained ECUs while failure reports are providing details about a certain malfunction of a single ECU. It can therefore be assumed that the data-flow required to describe a certain failure is a subset of the data flow profile of the test case which leads to its finding. This is also demonstrated when the referenced shared data for the verification in table 3.3 are compared to the data-flow listed in table 3.2. Here, just one of the data flows checked by the test case has led to the detection of a failure.

### 3.6.1 Data-Flow Coverage

$H_{1a}$**:** There is a significant similarity in data-flow required to describe the systems dynamic behavior during test case execution and failure occurrence found by that particular test case.

We first evaluate if failures detected during test execution require the same data flow to be observed as the test cases which have led to them induces into the system. For this, we analyzed the similarity of the classified phenomena for test cases and their associated failure findings as listed in table 3.5. During the comparison, multiple associations between one test case and many failures or one failure and many test cases are taken into account. Those multiplicities occur if a failure is detected by multiple test cases or one test case has found a failure which, turned out to be caused by multiple faults triggered simultaneously. The absolute and relative amount of matching, subsequent matching or non-matching references to shared data are shown in table 3.4. It can be seen that the highest amount of non-matching data-flow references occur for the verification usage of the data-flow. Further, data flow referenced for the precondition and stimulation usage are subject to subsequent effects. Over all purposes of shared data usage, the majority of shared data usage could be identified as matches, the corresponding hypotheses $H_{1a}$ can therefore be accepted.

A manual review of the comparison indicates that in almost all cases of the non-matching data flow references, the failure was found by diagnosis mechanisms of one or more neighboring ECUs involved in the test case execution. Even though failures are assumed to be detectable by observing a certain data flow, test cases often detect failures by checking the error entries created by such diagnosis mechanisms implemented in each component. Further, it has been found that the subsequent effects observed for the precondition and stimulation uses are caused by functional dependencies as described in section 3.4.

Beside findings directly related to the hypothesis pairs $H_{1a}$ and $H_{1n}$, the results show the importance of functional and physical dependencies for the

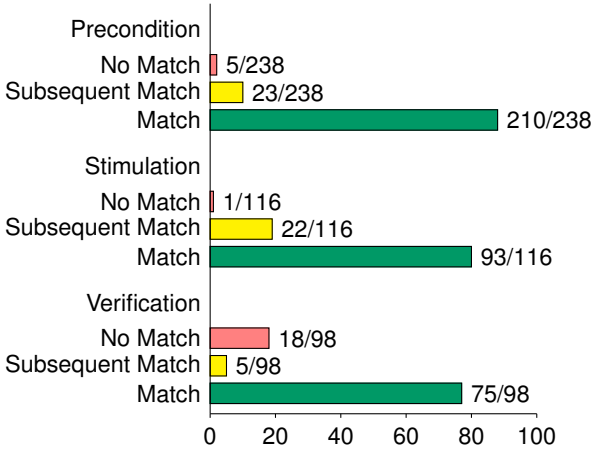



Figure 3.4: Results of Test Case Similarity to detected Failures

proposed approach of data flow based test coverage. As part of this study, these factors have been separated and controlled by classifying subsequent data flow using the introduced classification scheme. However, in order for this approach to be applicable for the practice of system integration testing, further studies are required addressing the importance of dependencies among individual data flow as defined for a system under test.

### 3.6.2 Test Gap Identification

As mentioned in section 3.2, the suggested classification scheme should help reveal test gaps in terms of component interactions not considered in existing test cases prior to test execution. To evaluate this, we analyzed the similarity of failures which have not been detected during test execution to existing test cases based on its data flow profiles. However, in contrast to comparing failures directly associated to test cases (*found-by-association*) the comparison of failures found by other test activities is not straight forward. In order to compensate for the missing association between failures and test cases, we first assigned for each failure the test case with the highest similarity to

be the counterpart for the comparison. An additional consequence of the missing causality between a test case and a failure is the fact that subsequent effects cannot be identified and the comparison metric only contains the amount of matching and non-matching data flow references.

Furthermore, the selection of the test case with the highest similarity for a given failure may result in many test cases with equal similarity. In some instances, multiple test cases show the same amount of matching data-flow pairs but different data flow is referenced within the set of matching pairs. In addition to that, some test cases could be identified which had the highest similarity for multiple failures. Consequently, the comparison shown below contains similar multiplicities as described in section 3.5.

The evaluation of the proposed approache's effectiveness in identifying test gaps is organized alongside the main hypothesis:

$H_{2a}$**:** There is no significant similarity in data flow required to describe the systems dynamic behavior during the occurrence of failures and existing test cases if the failure have not been detected by the test cases.

Figure 3.5 show the comparison of undetected failures to test cases with the highest amount of matching data flow. It shows a noticeably high amount of non-matching data flow for the verification phase of 93%. Furthermore, a high amount of non-matching data-flow also exist for the stimulation purpose. Within the precondition predicates, a moderate amount of matching data flow references is shown. The result of this comparison already points to the potential testing gap. The exceptionally high amount of non-matching data-flow references for the verification purpose reveals data flow which is unconsidered for verifying the system's behavior. In addition, the mismatches revealed for the stimulation use lists data flow unused for stimulating the system during test execution but actually triggered faults to surface as a failure. For the precondition purpose, the majority of data flow is already covered by the test cases. It can therefore be assumed that the correct data flow is already considered as a potential influence on failure occurrences. In summary there is a high amount of non-matching data flow shown in our

evaluation. Consequently, the hypothesis $H_{2a}$ can be accepted.

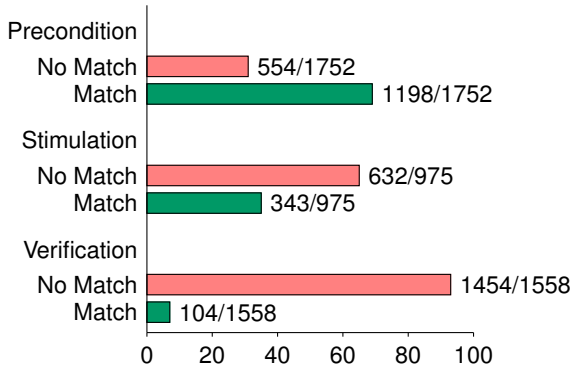



Figure 3.5: Results of Test Case Similarity to Undetected Failures

A more fine-grained perspective on potential test gaps can be extracted when successively comparing failures to test cases with high similarity for each purpose of data flow usage. Because the purpose of data-flow-usages are abstractions of the chronological order of a test case execution or a failure occurrence, the chronological order can be utilized to extract further information about the location of test gaps in terms of unconsidered component interactions. By first selecting test cases with the highest similarity within the verification-use set of data flow, we focus on those test cases with the highest potential of observing the failure. In our study, for 33 of the 52 failures we could not identify any matching data-flow within the existing test cases. Therefore, detecting those failures using the selected test cases is not possible due to the fact that data flow which is required to observe these failures is not considered for the behavior verification in any of the existing test cases. For the remaining 19 failures, the similarity is shown in figure 3.6. It shows a high amount of 91% of mismatches for the stimulation purpose which is a strong indicator that the existing test cases do not consider the right data flow for stimulation in order to trigger the failure. For the precondition predicates, a medium amount of matching and mismatching data-flow references is identified.

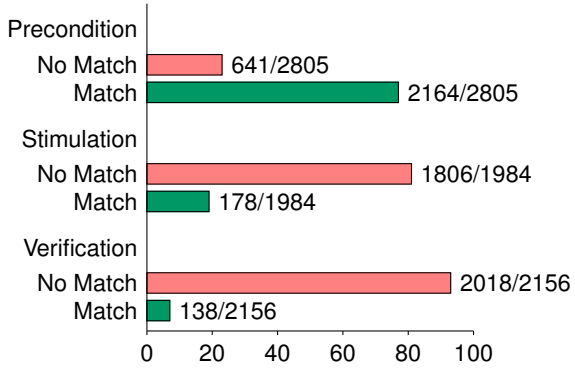



Figure 3.6: Results of Test Case Similarity to undetected Failures focusing the Usage of Data Flow for verification purpose

By selecting test cases with the highest amount of matching data flow for the stimulation purpose, we then focused on test cases which have the highest potential of triggering the failure to surface. The test case selection revealed that for 22 out of the 52 failures, no matching data flow is referenced for the stimulation purpose. The similarity of the remaining 30 test cases is shown in figure 3.7. It shows a high amount of mismatches for the precondition and verification purpose. This comparison shows that test cases which could trigger one of the failures do not perform a verification of the data flow required to observe the failure. In addition, most of these test cases may not establish the required precondition for the failures to surface.

For the last comparison, we focused on the highest similarity within the precondition predicates. This results in test cases which satisfy the conditions required for a failure to surface. The selection of test cases reveals that for 5 out of 52 failures, no matching data flow could be found in any of the existing test cases. The similarity of the remaining 47 failures is shown in figure 3.8. It shows a high amount of mismatches for the purposes of stimulation and verification. This further indicates gaps in the usage of data flow for triggering and verifying the system's dynamic behavior. In addition, the high amount of failures with matching preconditions indicate that the

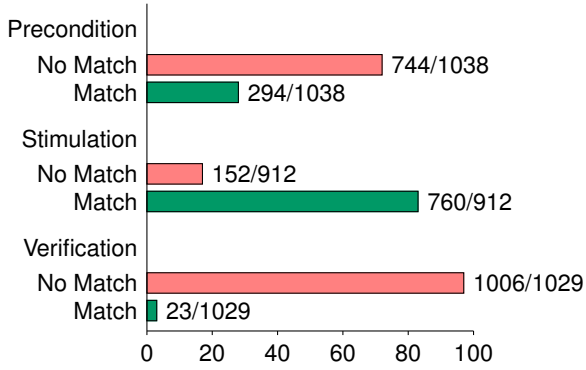



Figure 3.7: Results of Test Case Similarity to Undetected Failures Focusing the Usage of Data Flow for Stimulation Purpose

existing test cases already targets the error-prone system state.

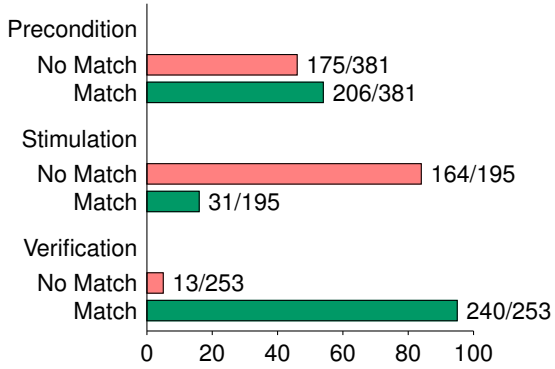



Figure 3.8: Results of Test Case Similarity to Undetected Failures Focusing on the Usage of Data Flow within Precondition Predicates

**RQ:** How effective is inter-component based data-flow classification in capturing the system's dynamic behavior relevant for test coverage analysis and test gap identification?

In summary, we analysed the data flow relevant for real world test cases

and failure reports. We are able to show high similarities in the data flow classified for test cases when compared to the data flow classified for failure reports which have been detected during test case execution. This leads to the acceptance of hypothesis $H_{1a}$. Within the second part of the evaluation we are able to show a significant amount of non-matching data flow when comparing the data flow classified for test cases to data flow classified for failures which are undetected during test case execution which leads to the acceptance of the hypothesis $H_{2a}$. Therefore, in the context of the research question for studying the effectiveness of inter component based data-flow classification in capturing the systems dynamic behavior relevant for test coverage analysis and test gap identification we consider the presented approach as effective. As part of the evaluation we are also able to showcase the identification approach of potential test gaps for the practice of system integration testing in the automotive industry.

## 3.7 Conclusion

We propose the usage of data-flow based test-coverage analysis for system integration testing. The suggested approach aims at applying well-studied code-based data-flow testing criteria and their extension to coupling-based testing to the problem of black-box integration testing of distributed software systems.

As a main contribution, we introduce a data-flow observation scheme applicable for test cases and failure reports which captures the usage of data flow for the phases for establishing preconditions, stimulating the system under test or verifying the systems dynamic behavior. The usage of the data flow of a test case is then put into perspective to the overall data flow within the system under test to resemble an overall coverage.

We evaluated the usefulness of this data flow coverage based approach using a series development project of a chassis control system introduced in section 2.3 containing several ECUs. The chassis control system contains an overall data flow of 1343 different shared data. We analyzed the approaches

ability to identify test gaps by taking 90 test cases from the system integration-testing phase and comparing its coverage of data flow to the usage of data flow in failures undetected during execution of these test cases. We found major differences in the usage of data flow between undetected failures and existing test cases. We therefore infer that the test gaps identified by the data-flow based coverage analysis cause faults to slip by the integration testing phase. We further analyzed if the data flow covered by existing test cases is also used to observe failures found during test execution. We found noticeable similarities within the usage of data flow between test cases and its associated failure reports.

However, we found a major limitation for the approach of identifying test gaps by measuring similarities of data-flow usage in test cases and failures which slipped by the integration testing phase. Functional or physical dependencies among individual shared data are not captured by simply measuring similarities of test cases and failures. In this work, such dependencies are treated as subsequent match during similarity analysis. Subsequent matches are manually identified in case a certain shared data is used in a failure but not in the associated test case within the same purpose of use.

As a second contribution, we proposed a set of basic data-flow based coverage criterion. In summary, these criteria focus on the usage of every shared data in within at least one test case (*Shared-Data-Use*) and further the usage of every shared data in all of the defined purposes of the classification scheme (*Precondition-Data-Use*, *Stimulation-Data-Use* and *Verification-Data-Use*). However, satisfying these criterion for a system-under-test with moderate complexity can become expensive in terms of time and resources when testing a system with high complexity in terms of inter-connectivity.

Overall, we find that the suggested coverage criterion is helpful in identifying untested data flow in a component based system containing black-box components. We found that abstracting the system's behavior to the usage of data flow for a certain purpose makes the data-flow classification an easy task with low additional effort for the process of test specification. However, we believe that the coverage metrics of existing test suites represent an overestimation in its ability to detect a fault. This is due to the fact that the

usage of a certain element of shared data for a particular purpose does not state that it is used in a meaningful way with a high chance of triggering or detecting failures. The data flow-based coverage measure introduced in this paper can therefore be compared to traditional code structure-based data-flow analysis in software testing. It helps to identify areas not exercised by a set of test cases or to identify redundant test cases.

## 3.8 Threats to Validity

Measures for design contamination have been implemented for the classification of data flow usage in existing test cases and failure reports.

A major threat to the *internal validity* of the provided studies is caused by the process of data collection and its potential of inducing a classification bias. It was ensured that the team who performed the classification did not know the association between a test case and the failure reports created during execution of the test case. 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.

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. In addition, the generalizability is strongly limited due to the nature of the system under test. The chassis control system represents a distributed reactive real-time software system which influences the design and implementation of test cases. Furthermore, the absence of a caller-callee paradigm within such a system has to be taken into account when applying the presented approach to more general distributed software systems.

Chapter 4

# Test Case Selection and Prioritization

Generally, faults are not evenly distributed across the structural elements of the system under test [She95]. In addition, applying the data flow based test coverage approach introduced in the previous chapter to a system with moderate complexity leads to the situation that not all data flow based component interaction can be considered when testing in a time or resource limited context like system integration testing done in the automotive industry. However, we expect that the process of system integration testing can be guided if the fault-prone component-interactions are identified successfully prior to testing. This contribution chapter therefore presents our studies in the area of component and interface coupling based test case selection and prioritization. In section 4.1 we present related work to the field of test case selection and prioritization. We then provide a brief description of the research design used for this study in section 4.2. This is followed by the selection of existing coupling measures described in section 4.3. After that, the process of data collection for the component and interface coupling as

well as for the real world failure data in section 4.4 and 4.5. The evaluation of the study is then provided in section 4.6. Lastly, the findings found in this study are provided in section 4.7 while the threats to its validity are introduced in section 4.8. The work presented in this chapter extends the following paper and puts it into the perspective of this work.

D. Hellhake et al. “Towards using coupling measures to guide black-box integration testing in component-based systems.” In: *Software Testing, Verification and Reliability* 32.4 (Mar. 2022). URL: https://doi.org/10.1002/stvr.1811

## 4.1 Related Work

Related work exists for the topic of test case selection and prioritization techniques for integration testing of component based software systems. However, only few studies have addressed the potential benefits of utilizing artifacts created during system design for the process and practice of integration testing in general. In addition, existing work in that area of research mainly focuses on the automated generation of test cases out of the system's design and structure specification. In this section, we provide existing work for the topic of test case selection and prioritization as well as for guiding integration testing based on the system design.

In a work provided by Hao *et al.*, a unified test case prioritization technique is proposed which combines the principles of total and additional test case prioritization strategies[HZZ+14][ZHZ+13]. Hao *et. al.* first states that most existing research on test case prioritization focuses on the prioritization itself and follows 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 *et al.* provide their definition of a unified test case prioritization which he evaluates using 40 C and 28 Java programs. Their results demonstrate that the fault detection probability of their proposed technique in general reside between those using purely *total* or *additional*. During the course of this chapter, we propose a coupling based approach which is heavily inspired by the work done by Hao *et. al.*.

Elbaum et.al. studied the trade-offs between fine granularity and coarse granularity prioritization techniques [EMR02]. 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 *et. al.* used 16 different approaches across these granularities in their studies. In their results, Elbaum *et. al.* 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 *et. al.* 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, their 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. The work provided by Elbaum *et. al.* provides a theoretical foundation for the studies provided in this chapter due to the fact that similar goals are being set for the evaluation. In addition, showing that the effectiveness of test case prioritization is invariant to the underlying level of granularity encourages us in following the research direction provided in section 4.2 for which we neither utilize the system's code (fine grained) nor the system's functionalities (coarse grained).

Given the definition of coupling in section 4.3 and the available set of applicable measures in section 2.4, the second part of this related work section provides an overview of existing work regarding the utilization of architectural measures for software testing. In [MDO+16], Mendes et al. conducted a systematic mapping study to investigate existing approaches to use information about the software architecture to improve the testing activities. 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 and Basili studied the ratio and strength of coupling and its relation to the error-proneness of interacting components of the system [SB91]. For this, Selby and Basili 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 2.4.1. They found that routines with the lowest coupling ratio showed significantly less errors. In addition, for those routines existing errors where less costly to fix. The goal of Selby and Basili is similar to the goals set in 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.

Rizwan *et. al.* extended the Fan-In/Fan-Out based coupling measure as also described in section 2.4.1 to a metric set called "Vovel metrics" [RNS21]. In Their work, he included aspects regarding the volume of coupling existent between two or more coupled elements. The volume of coupling is defined based on the size of each shared data element $SizeOf(M_x)$ defined for a method $(M_x)$ which consists of the volume of the method's parameters $v(M_p)$ and the volume of its return values $v(M_r)$. However, the volume of a parameter or return value is defined as the space in memory required to hold that data. In addition to that, the level of coupling is taken into account as a numerical representation of the generic coupling types defined by Yourdon *et. al.* [YC79]. They provide an evaluation of their coupling measurement approach based on multiple studies in which coupling is measured based on the systems call-graph and correlated to the failure locations. The aspects of coupling relevant for this work are described in section 4.3 and mainly focus onto quantitative aspects of coupling as also covered by the basic Fan-In/Fan-Out based coupling measurement. However, we do agree with Rizwan *et. al.* that additional aspects of coupling need to be considered in order to fully express the Impact coupling has onto the systems stability and its development.

In [SBS18], Singh et al. conducted an empirical study on different versions of open source software to analyze the benefit of various object-oriented metrics for the quality of the software. 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 and Wolf 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 [RW97]. In [HSW19], similar test criteria have been defined and evaluated against the fault distribution in a real-world software system.

As demonstrated by Mendes et. al., 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 [MDO+16]. 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.

## 4.2 Research Design

In [Xia00], Xia et. al. emphasizes the ambiguity of coupling as a concept and the resulting challenge in deriving meaningful measures. They state 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.


Figure 4.1: Association of Coupling, Complexity and Fault-Proneness [RNS20]

In this work, we study the effect of coupling mentioned by Xia *et al.* [Xia00] as a potential source of faults introduced into the system and detected during integration testing as depicted in figure 4.1. As discussed in Section 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 [RH08] and [RHRR12]. The overall structure of the case study design is summarized in figure 4.2.

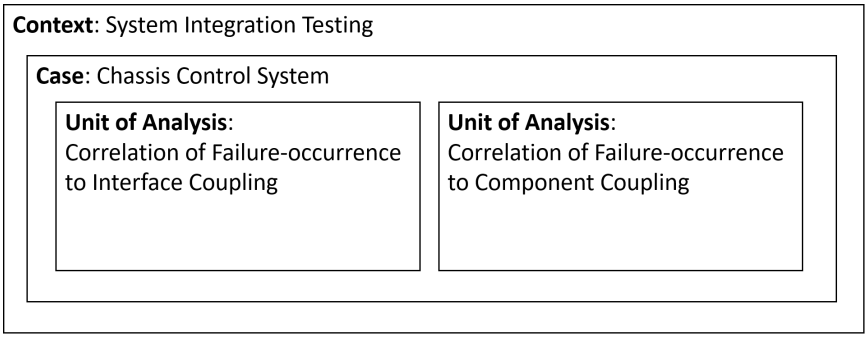



Figure 4.2: Embedded case study design for the study of component and interface coupling according to Runeson [RH08]

- **RQ2:** 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?
- **RQ3:** 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?

We further provide the proposition that high coupling of a component or interface increases the probability for this component to show fail behavior 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 overload 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 4.3, both for component coupling and interface coupling. The list of hypotheses is given in table 4.1.

Table 4.1: Our five hypothesis pairs for the correlation between measurements and failure-proneness at component level

| Alternative hypothesis | | Null hypothesis | |
|---|---|---|---|
| $H_{c1a}$ | The failure-proneness of an ECU during system integration testing is positively correlated to its $Size(m_l\|S)$ measure according to table 2.3. | $H_{c1n}$ | There is no positive correlation between an ECU's failure-proneness and its $Size(m_l\|S)$ measure according to table 2.3. |
| $H_{c2a}$ | The failure-proneness of an ECU during system integration testing is positively correlated to its $Complexity(m_k\|S)$ measure according to table 2.3. | $H_{c2n}$ | There is no positive correlation between an ECU's failure-proneness and its $Complexity(m_k\|S)$ measure according to table 2.3. |
| $H_{c3a}$ | The failure-proneness of an ECU during system integration testing is positively correlated to its $CC$ measure according to section 2.4.1. | $H_{c3n}$ | There is no positive correlation between an ECU's failure-proneness and its Component Coupling ($CC$) measure according to section 2.4.1. |
| $H_{c4a}$ | The failure-proneness of an ECU during system integration testing is positively correlated to its $CIF$ measure according to section 2.4.1. | $H_{c4n}$ | There is no positive correlation between an ECU's failure-proneness and its $CIF$ measure according to section 2.4.1. |
| $H_{c5a}$ | The failure-proneness of an ECU during system integration testing is positively correlated to its $CDC(C_k)$ measure according to section 2.4.2. | $H_{c5n}$ | There is no positive correlation between an ECU's failure-proneness and its $CDC(C_k)$ measure according to section 2.4.2. |

Table 4.2: Our four hypothesis pairs for the correlation between measurements and failure-proneness at interface level

| Alternative hypothesis | | Null hypothesis | |
|---|---|---|---|
| $H_{i1a}$ | The failure-proneness of a component interface during system integration testing is positively correlated to its $Size(n_i\|S)$ measure according to table 2.3. | $H_{i1n}$ | There is no positive correlation between a component interface's failure-proneness and its $Size(n_i\|S)$ measure according to table 2.3. |
| $H_{i2a}$ | The failure-proneness of a component interface during system integration testing is positively correlated to its $IC$ measure according to section 2.4.1. | $H_{i2n}$ | There is no positive correlation between a component interface's failure-proneness and its $IC$ measure according to section 2.4.1. |
| $H_{i3a}$ | The failure-proneness of a component interface during system integration testing is positively correlated to its $IIF$ measure according to section 2.4.1. | $H_{i3n}$ | There is no positive correlation between a component interface's failure-proneness and its $IIF$ measure according to section 2.4.1. |
| $H_{i4a}$ | The failure-proneness of a component interface during system integration testing is positively correlated to its $IDC(I_k)$ measure according to section 2.4.2. | $H_{i4n}$ | There is no positive correlation between a component interface's failure-proneness and its $IDC(I_k)$ measure according to section 2.4.2. |

## 4.3 Selection of Coupling Measures

Many studies proposed coupling and complexity metrics for component-based software systems [AMgA13], but comparatively few works exist which study the correlation of such measures to the fault-proneness of component interactions typically verified during integration testing. In this section, we provide a broad description of the term coupling alongside the set of aspects

relevant for the selection of coupling measures. For each aspect of coupling, related work is presented and the relevance for this thesis is pointed out in detail.

Constantine and Yourdon [YC79] 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. The aspect of coupling and cohesion as a measure for software quality plays a roll in the study regarding the correlation of failure occurrences to the degree of coupling of faulty components and interfaces.

One motivation for increasing cohesion while keeping coupling as low as possible is provided by Martin [MO97], 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. The potential instability of highly coupled elements of a software system is one theoretical aspect which sup-

ports the proposition of fault-proneness caused by high coupling studied in section 4.6.

Santos et al. [SRCF17] 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 analyzed 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 [AMS+18], Abdellatief discusses the generic component model depicted in figure 4.3 which is commonly used in coupling analysis. 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 component model provided by Abdellatief highlights the idea of considering coupling independently of the inner working of the coupled elements which is an important aspect for the black-box nature of the studied system as described in section 2.3.

The identification of existing coupling in a given software design based on interface specifications is further detailed by Gill [GG04], 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.

Once dependencies among multiple software modules are identified, the

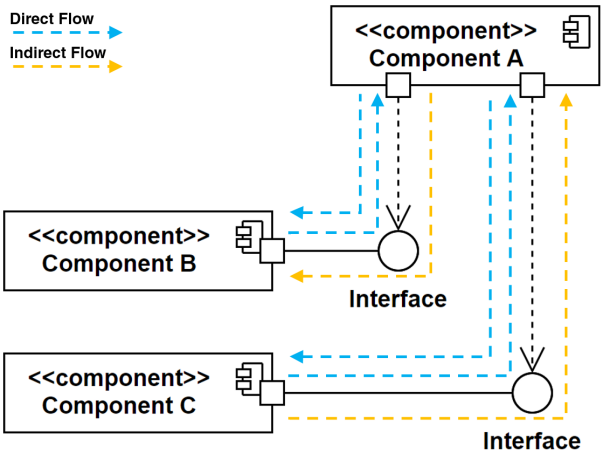



Figure 4.3: Generic Information Flow Model for component-based Systems

system design can be analyzed based on either quantitative factors like the distinct number of couplings per component, or by qualitative measures like the type of coupling between multiple components. In [OHK93], Offutt et. al. discusses principles for measuring different types of coupling, which can be used to determine the strength of the dependency between two components. For this purpose, Offutt et. al. adopted six generic types of coupling (see table 4.3), which have been introduced by Myers [Mye77] and Page-Jones [Pag01]. These types take multiple aspects of coupling into account like the type of data passed between coupled modules (data and stamp coupling), the type of communication (external and common coupling) and the degree of dynamic interference (control and content coupling). In the study provided in section 4, we focus onto quantitative factors of coupling due to the fact that qualitative factors require deeper understanding of how data flow is implemented which is typically only available in a white box system.

Besides adding more levels of coupling specific to modern programming languages, Offutt et. al. also addresses the directness of coupling as an additional aspect, which ranges from direct coupling between two modules to various levels of indirect coupling in which two modules are coupled via

Table 4.3: Generic coupling types [Mye77]

| Type | Description |
|---|---|
| Data Coupling | Two modules are data coupled if they pass data through scalar or array parameters. |
| Stamp Coupling | Two modules are stamp coupled if they pass data through a parameter that is a record. |
| Control Coupling | Two modules are control coupled if one passes a flag value that is used to control the internal logic of the other. |
| External Coupling | Two modules are external coupled if they communicate through an external medium such as a file. |
| Common Coupling | Two modules are common coupled if they refer to the same global data. |
| Content Coupling | Two modules are content coupled if they access and change each other's internal data state or procedural state. |

a chain of other modules in between[OHK93]. The directness of coupled software modules is also depicted in figure 4.3 and discussed by Gui and Scott [GS09], who extended the software abstraction model and complexity metrics defined by Chen et. al. [CWZB11] with a measure for indirect coupling. This measure has been selected for the studies in this work and a detailed description of it is provided in section 2.4.2.

In [Xia00], Xia discusses the different aspects covered by the six generic types of coupling discussed by Myers [Mye77] and the inconsistent usage of the term coupling in software design literature. Xia argues that coupling is often only considered between two modules, which would neglect frequent cases in real-world software systems, e.g. 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 4.1, 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 behavior of coupled components $PC[d]$ and the

number of modules coupled to $d$, i.e. $N[d]$. Consequently, coupling of a software module is defined as the sum over the module's out-flowing data $MC(m) = \sum_{i=1}^{n} EC[d_i]$. The definition of coupling as a composite measure plays a major role in evaluating the results of the studied coupling measures and its individual limitations in section 4.7.

$$EC[d] = C[d] * PC[d] * N[d] \tag{4.1}$$

In addition to the discussion of the concept of coupling in software design, several studies can be found about topics closely related to the topic of this paper. In [Sal06], Salman analyzes structural metrics for component-based system design based on counting of components, connectors and interface compositions. One of the main contributions of his work is a basic set of properties which a quality model for structured software design should satisfy to be a valid measurement instrument. These properties are derived from a brief discussion about complexity measures for software systems provided by Weyuker [Wey88].

Besides using software module interfaces, much research focuses on utilizing module implementations, i.e. code, to identify module coupling [IYY+05]. In addition, Sellami et. al. provide a metric for complexity in component based systems which is defined around the terms Interface, Properties, Methods and Events [AAEW18]. However, these methods rely on the internal control flow graph of the used software modules and therefore do not work for off-the-shelf or black-box modules. Because the goal of this work is to study the usefulness of coupling measures to guide black-box integration testing, code-based approaches to identify dependencies within a system design are not further considered.

In accordance to the definition of module coupling in equation 4.1, 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 behavior 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 into account. They have been selected out of existing literature and are used in section 4 for correlation analysis with a failure distribution. In section 2.4.1 and 2.4.2, conventional approaches are described to measure coupling based on the counting of software modules or the number of data flows. In section 2.4.3, an approach is described which measures coupling based on the amount of entropy created in a system by a certain data flow.

## 4.4 Data Collection for System Graph Abstraction

For this study, we selected the information theory based approach as well as the dependency analysis and data flow based approach of measuring coupling to be potential indicators of fault-prone component interactions. All of these approaches rely on a graph abstraction of the system. As Allen et. al. [AKC01] state, multiple graph abstractions can be created for a given system with each focusing on a different aspect of the system's design. In this section, we describe how the generic approaches are applied to the architecture and design documentation of the chassis control system using a fully automated proprietary tool-chain. For each step involved for the data collection we provide detailed information about which limitations and implications arise due to the nature of having black-box components contained in the system under test.

As described in section 2.1, the architectural views onto the system under test relevant for deriving a system graph abstraction are the a) *hardware architecture* describing which ECUs exist and how they are wired together using physical communication buses, b) the *logical architecture* describing which software components exists and which data flow they provide and

require based on the interfaces implemented and c) *cluster architecture* describing which software components are grouped together to form functionally coherent clusters and which cluster is partitioned onto which ECU for execution.

Consequently, the component structure of the graph abstraction used in this work is directly derived from the hardware architecture documents. According to the hardware architecture of the chassis control system, the graph abstraction of the chassis control system contains 10 modules listed in table 4.4 and named accordingly.

In [AKC01] and [All02], Allen suggests that the graph abstraction used for information theory based measurement of component coupling should model the environment of the system by using an additional module containing a single node. This node is disconnected from the system because coupling and complexity caused by relationships between the system and its environment are not relevant. The chassis control system represents a subsystem of the vehicle. Therefore, chassis control related ECUs do implement data flow to non-chassis control system related ECUs, which can be considered as part of the environment. However, in the context of this work and in particular the research question defined in the previous section, generalizing those interactions to the system's environment would diminish the understanding of a component's or software interface's context which limits the applicability of information theory based coupling measurement for the purpose of this study. In order to compensate for that, the proposed design rule of having a module and node included in the system graph abstraction representing the coupling to the system's environment is not followed for the context of this work. Instead, the system graph contains additional modules for each ECU that interacts with one ECU of the chassis control subsystem. However, for those modules only the interactions to chassis control related ECUs are considered and modelled.

Complementary to the derivation of the system graph's modules from the systems hardware architecture, the node structure is created using the logical architecture. However, to fully model each node and the connecting edges representing data flow, the cluster level architecture is further required which

also identifies which software component defined in the logical architecture corresponds to which ECU and which node corresponds to which component in the graph abstraction.

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. Within the AUTOSAR standard [AUTb], a software component is defined as the central structural element 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 [AUTa]. However, a detailed definition of the structural elements relevant to replicate the study provided in this work can be found in chapter 3.2 and 4.2 of the respective AUTOSAR Software Component Template specification. A detailed list of the number of shared data assigned as in-flow and out-flow and the total amount of software interfaces is given in table 4.4.

The incoming and outgoing communication of all software components provides the overall interaction behavior of components for this work. However, incoming and outgoing communication of a software component can be categorized as intra- or inter-ECU interactions based on the partitioning of each interconnected software component on different ECUs. 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 according to the hardware architecture specification.

In case the software architecture specification is available for each ECU contained in a vehicle subsystem, a data flow graph can be assembled which highlights all possible information flow within the system under test. However, this is often not possible in practice because the software development

Table 4.4: 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** |

of most ECUs is outsourced based on a given black-box behavior 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 for the process of system integration testing done at the vehicle manufacturer. 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 4.3, coupling analysis can directly be performed based on the inter-ECU data flow for ECUs at component level and software interfaces at interface level.

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 section 2.4.2 and 2.4.3, 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.

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 7,790.98 bits and a complexity value of 18,801.73 bits. As already mentioned, 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 2.4.3. Since it requires the module's internal interdependencies, the cohesion metric cannot be applied for both module- and system-level. In table 4.5, 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 to 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 4.4, the size of all software interfaces of the chassis control systems is shown. The majority of interfaces form groups with equal size. In information theory, the size of a node is calculated based on the node's subgraph $S_i$, which only contains edges connected 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 4.5, which shows the size and number of endpoints of each interface in a scatter

Table 4.5: Results of Module Level Complexity Measures

| Ecu | $Size(m_l \mid S)$ | $Complexity(m_k \mid S)$ |
|---|---|---|
| ECU7 | 3,553.61 | 9,802.35 |
| ECU8 | 1,756.65 | 4,828.09 |
| ECU6 | 810.69 | 1,570.60 |
| 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 |

plot.

The component model created for the dependency analysis and data-flow based measurement of coupling contains 47 components and 756 interfaces, which form 3,512 dependencies among each other. In table 4.6, 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 2.4.2, 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.

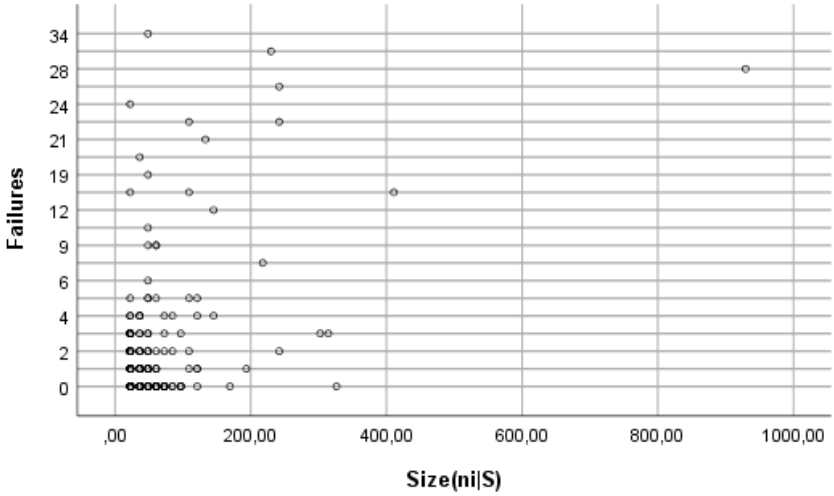



Figure 4.4: Correlation of $Size(S_i)$ of interfaces to interface level failure distribution

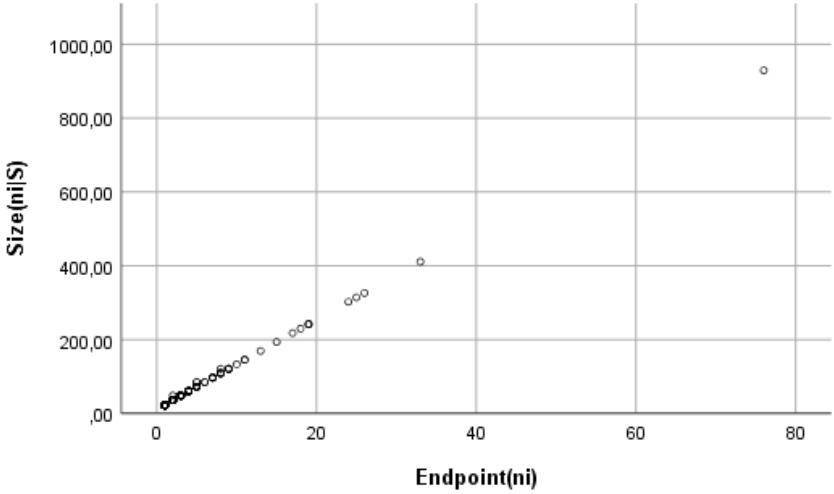



Figure 4.5: Proportion of endpoints to $Size(S_i)$ of interfaces

Table 4.6: 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 | 9,025 | 7 |
| ECU4 | 10 | 32,400 | 6 |
| ECU9 | 8 | 15,376 | 7 |
| ECU2 | 3 | 400 | 6 |

## 4.5 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 introduced in section 2.3.

In table 4.4, the number of failures detected during system integration testing are shown per ECU. In addition, figure 4.6 illustrates the set of external interfaces of each ECU similar to the system graph introduced in section 4.3. Interfaces which have shown failure behavior during integration testing are colored red. Edges indicating data flowing between nodes are not visualized due to the high amount of inter-connectivity. Regarding the selection of defects, all milestones and test-cycles of the vehicle series development project have been taken into account.

To identify the data flow involved in a detected failure, we used a defect classification scheme for black-box behavior observation, which is described in section 3.3. By using the classification scheme, a failure is broken down into the smallest number of conditions required to trigger its fault as observable failure behavior.

**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

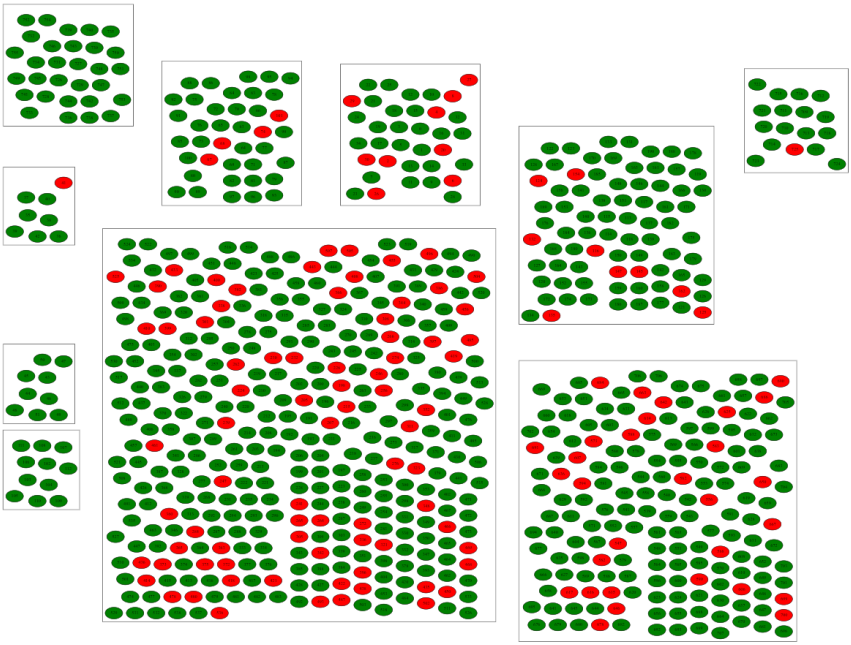


Figure 4.6: Illustration of the Defect Distribution of the Chassis Control System

flowing among participating modules. The data flow is classified as a set of predicates. In the context of this study, the data-flow required to describe the precondition phase is not relevant. **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 behavior of the system under test can be observed on one or more data flows as a derivation from the specified behavior. These data flows are classified as verification-use.

Because only software interfaces which include out-going interactions can show failure behavior, the correlation study is only applied to those interfaces, even if the set of measures described in section 2.4.2, 2.4.1 and 2.4.3 are applied to the complete set of software interfaces. In table 4.4, 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.

## 4.6 Evaluation

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. Out of general guidelines for interpreting the results of the Pearson correlation analysis we consider the strength of the correlation as small for $r$-values up to 0.3 while $r$-values up to 0.5 are considered as medium strength. $r$-Values above 0.5 are considered as strongly associated. The same rule also applies to the results of the Spearman correlation analysis.

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 [BMD+09]. To control for this, we use a p-value adjustment method introduced by Benjamini and Hochberg [BH95]. Alongside its definition, all tests in table 4.7, 4.8, 4.9, 4.10 have been ranked according to its p-value 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.

**RQ2:** 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?

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 table 4.7 and 4.8 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 4.1 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 failures found at a certain component of the system can potentially be explained by its coupling.

**RQ3:** 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?

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. Connecting failures found to the interfaces which are affected by that failure is done based on the data flow relevant to describe the systems dynamic behavior during occurence of the failure. This is described in more detail in chapter 3.3. 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 rejected. In table 4.9

Table 4.7: Results of component level Pearson

| Measure | $r_p$ | $r_p^2$ | $P_p$ | $Rank_p$ |
|---|---|---|---|---|
| $CDC$ | 0.898 | 0.806 | 0.000425 | 1 |
| $Complexity(m_k\|S)$ | 0.853 | 0.728 | 0.002 | 2 |
| $Size(m_k\|S)$ | 0.846 | 0.716 | 0.002 | 2 |
| $CC$ | 0.722 | 0.521 | 0.018 | 3 |
| $CIF$ | 0.55 | 0.303 | 0.094 | 4 |

and 4.10, 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 its coupling to the rest of the system.

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. One potential explanation for this is given by the proposition that ECUs implementing a highly complex software also implement more complex interfaces on average. An example is given when comparing the amount of interfaces implemented by each ECU in table 4.4 to the overall complexity of each ECU shown in table 4.5. According to that, the most complex ECU7 is implementing 348 interfaces implementing an overall complexity of $Complexity(m_k|S)$=9802.35 therefore holding an average complexity per interface of 28.4. In contrast to that, ECU2 is implementing the least complexity of $Complexity(m_k|S)$=120.56 while implementing 8 interfaces resulting in an average interface complexity of 18.4. However, this proposition needs to be evaluated in a more systematic way in a future work.

Table 4.8: 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 4.9: Results of interface level Pearson correlation

| Measure | $r_p$ | $r_p^2$ | $P_p$ | $Rank_p$ |
|---|---|---|---|---|
| $IDC$ | 0.548 | 0.300 | 1.99E-21 | 1 |
| $Size(n_i\|S)$ | 0.476 | 0.227 | 7.84E-16 | |
| $IC$ | 0.367 | 0.135 | 1.50E-09 | 3 |
| $IIF$ | 0.201 | 0.040 | 0.001 | 4 |

Table 4.10: 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 |
| $Size(n_i\|S)$ | 0.253 | 0.064 | 0.00004 | 2 |
| $IC$ | 0.243 | 0.059 | 0.0009 | 3 |
| $IIF$ | 0.043 | 0.002 | 0.497 | 4 |

## 4.7 Conclusion

The data flow based coupling measures tested in this study were generally weakly correlated for interface coupling.

The interface and component dependency coefficient measure showed the best performance for both interface and component coupling. According to the definition given in equation 2.5, 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 utilized 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, $Size(mk|S)$ and $Complexity(mk|S)$ achieved the strongest correlation. Similar to the dependency coefficient, the measures for $Size(mk|S)$ and $Complexity(mk|S)$ mainly reflect the number of coupled components or interfaces. This is also demonstrated in figure 4.5. Unlike the dependency coefficient measure which solely reflects the number of coupled components or interfaces, the measures for $Size(mk|S)$ and $Complexity(mk|S)$ account for redundancy and patterns in a system design [AK99].

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 inter-connectivity does not conform to Martin's definition of component instability [MO97] 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 4.2 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 table 4.7 and 4.8 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.

## 4.8 Threats to Validity

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

### 4.8.1 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 contol 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.

### 4.8.2 Internal Validity

A major threat to the validity of the provided experiments is the manual data flow classification of failure-reports used for the correlation analysis. As described in section 4.5, a classification scheme has been used which was introduced and evaluated in a work prior to the presented study. Even though, the classification scheme's effectiveness of describing the relevant data flow of a certain failure-report has been studied in a dedicated evaluation, 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 analyze in this study.

When applied to the chassis control system introduced in section 4.2, measures for interface coupling and component coupling showed limited applicability. The definition of interface coupling in equation 2.1 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 out-flow 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 2.2, 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 providing sensor-data) 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 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 behavior in terms of a deviation from specified behavior but also defects caused by incorrect or incomplete specification and other types of defects [NT08]. Additionally, 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.

### 4.8.3 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 2.1. 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 2.4.3 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.

### 4.8.4 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.

Chapter 5

# Coupling based System Integration Testing

This contribution chapter presents our proposal for a Coupling based System Integration Testing process as it has been established in the company at which the studies provided in this work have been conducted. During the course of this chapter we combine the approach of data flow based test coverage analysis presented in chapter 3 together with our proposed approach for test case prioritization in chapter 4 to formulate a coupling based integration testing process. For this, we first describe how the system design specification and documentation can be used to derive the formal specification of the system's structure and component interaction behavior in a fully automated way. In addition, we describe the manual steps required to classify the component interactions exercised in existing test cases and the component interactions required to describe the systems behavior during failure occurrence. Lastly, we combine these practical method descriptions to a generic process using an activity flow chart alongside a detailed description. The work presented in this chapter is unique to this thesis.

## 5.1 Software and Hardware Architecture Model

As discussed in section 2.2, related work exists for system integration testing which highlights the need for formal specification regarding the systems structure as well as for the systems dynamic behavior. In this section, our approach to address this need for formal specification is presented which a) derives the relevant structural information about the system from its architecture documentation and b) extracts the component interaction related information from the component communication specification. In order to support our approach, the collection of both types of information is explained alongside a combined entity relationship model which resembles a broader abstraction of the architecture views introduced in section 2.1.

During development, a modern car as a highly distributed responsive real-time system is specified using several architectural models. The most relevant architectural model for the purpose of system integration testing is the *logical architecture* representing the software's component structure as explained in more detail in section 4.4. In addition, the *cluster architecture* is used as an intermediate step to identify groups of functionally coherent software components which are then partitioned onto different ECUs using the *hardware architecture* model. To derive information about the system's specified dynamic behavior, inter-component communication specification is used which is part of the *logical architecture* for the software component based communication. However, the inter-ECU level communication is specified as part of the *hardware architecture* specification.

In figure 5.1, the entity relationship model for the hardware architecture is shown which has also been used for the data collection phase of the studies conducted during this work and presented in section 3 and 4. The most crucial element of the hardware architecture is the *ECU* which executes multiple software components. The first part describing an ECUs interconnection to two or more other ECUs is given by the *ECUInterfaces*. In practice, these interfaces are created for individual atomic functionalities implemented by the ecu in order to group multiple shared data in a coherent way. However, to fully capture the interconnection between two or

more ECUs, data flow is described using Protocol Data Units (*PDU*). PDUs group multiple *DataElements* defined by the software architecture and are either sent or received by an ECUInterface, which is classified as *Provided* or *Required*. The last element captured by the hardware architecture model is the communication *Bus*. This element is taken into account in order to incorporate special cases like gateway-functionalities in which a single ECU is connected to multiple communication buses in order to receive and forward PDUs without providing the carried data-elements to the executed software components. To support the understandability of the described hardware architecture model, an example can be provided using the Inertial Measurement Unit (IMU) which is a sensor type of *ECU* commonly found in a vehicle. Such IMU typically implements multiple *interfaces* in order to *provide* its different sensor-data (e.g. acceleration, angular rate, inclination) on a certain communication bus by grouping individual data elements into *PDU*s (e.g. values for X, Y and Z-axis).

The entity relationship model for the software architecture is shown in figure 5.1 as well and is very similar to the hardware architecture model. Its central structural element is the *Software Component* which is partitioned onto at least one ECU. A software component implements multiple *Software Interfaces* as the grouping element to describe software level data flow while a software interface sends or receives data elements, which are denoted as *Provided* or *Required*. In case of inter-ECU based data flow, data elements are further assigned to PDUs which are then communicated on a communication bus. To also support the understandability of the described software architecture model, the example of the IMU can be continued with the calculation of the vehicles movement in space. A *Software Component* executed by a certain ECU and implementing such a functionality would *require* sensor-data like the measured acceleration or the angular rate as an input in order to calculate the vehicles velocity or its position in space. These calculated data elements are then *provided* on software level using a separate software interface.

During the case studies presented in section 3 and 4, data-collection for the hardware- and software architecture model has been performed using

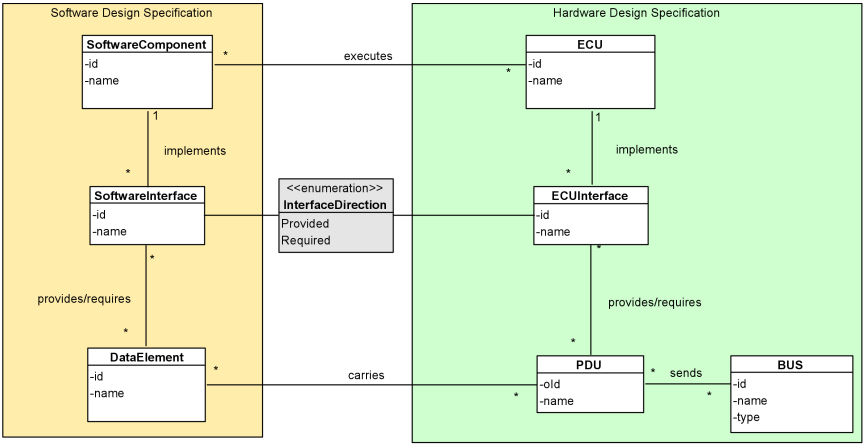



Figure 5.1: UML Class Diagram for Software- and Hardware Architecture

existing and machine-readable system description formats. The hardware architecture definition of the studied system described in section 2.3 has been extracted from the existing Field Bus Exchange Format (FIBEX). FIBEX is an XML-based standardised format widely used in the automotive industry to describe the network structure and data flow of a vehicle. In figure 5.2, the data structure of the FIBEX format is shown. Beside the relevant elements, other aspects of a vehicle's network like the communication timing or the functional architecture are treated by the FIBEX standard as well. The software architecture elements are extracted from the *System Description Specification* which is a widely adopted XML-based format defined by the AUTOSAR standard for software [AUTb]. However, the system description is typically not available for off-the-shelf ECUs. As mentioned in section 2.1, such ECUs are considered as black-box and only inter-ECU based communication has been taken into account for the studies provided in this work.

In summary, given the presented data model for the software and hardware structure of an automotive subsystem, the system's overall structure can directly be collected based on specification formats commonly available

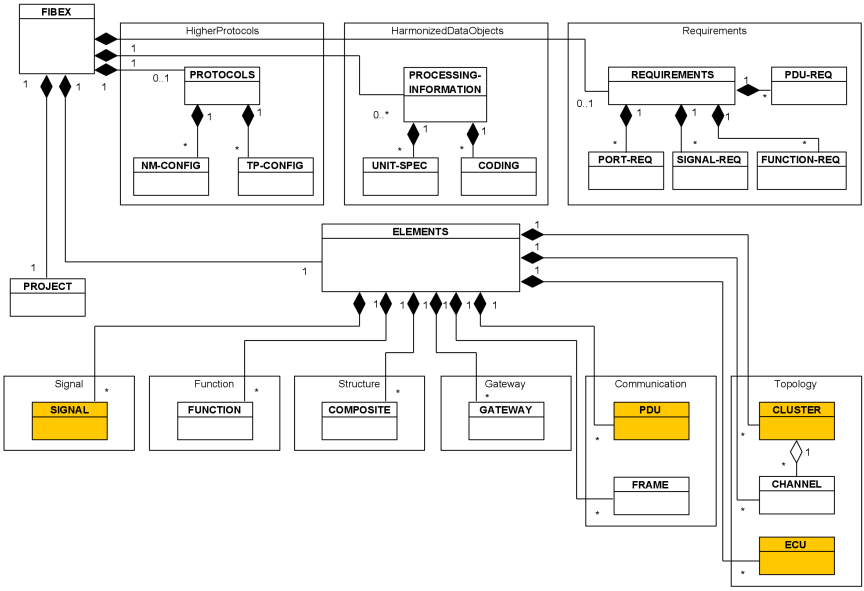



Figure 5.2: FIBEX Class Diagram [Gro]

early in development. Once the data has been collected, a system graph abstraction can be derived as described in section 2.4 and measures for component coupling and interface coupling can be calculated as described in section 3.6. The resulting coupling annotated system graph can then be used for test coverage analysis and test selection and prioritization which is explained in more detail in section 5.

## 5.2 Test Data Classification Model

One of the results of our study about data flow based test coverage measurement shown in section 3 is the identification of the benefits of test data classification with respect to the system design. Being able to identify which elements of the system's structure are exercised during test execution or failure occurrence is a mandatory precondition for systematic test gap iden-

tification in integration testing. In this section, our approach for performing test data classification in practice is presented and explained alongside an entity relationship model which shows the relevant elements in figure 5.3. Furthermore, for each element a detailed description is provided on how it has been used in the studies conducted in this work.

The central element for test design classification is a *Phenomenon*. As discussed in section 3.3, the execution of a test case as well as the occurrence of a failure can, from the viewpoint of the theory for observing a systems dynamic behavior, be seen as a sequence of component interactions. On the one hand, a test case exercises a certain scenario by executing a control sequence and verifying the correct response behavior of the system given a predefined starting point. On the other hand, the occurrence of a failure describes the behavior of the system in a very similar way by highlighting the required control sequence and starting point necessary to trigger the failure which is then captured as a derivation of the expected system behavior. Within the provided entity relationship model, this similarity is implemented by using the *Phenomenon* element as a generalization for the *FailureReport* and *TestCase* element.

The next crucial element for test design classification is the *DataFlow* as the connecting element between a Phenomenon and the system software design specification in case of a certain *DataElement*. Given the fact that the overall universe of component interaction in a distributed component based system containing black box components is defined by the inter component related data flow, a phenomenon describes the systems dynamic behavior as a sequence of inter-component data flow. Within such a phenomenon, each data flow is assigned to one of the following three purposes: *Precondition*, *Stimulation* and *Verification*. As introduced in section 3.3, the annotation for *Precondition* is used to describe the required starting point for the system while the *Stimulation* annotated data flow describes the control sequence applied to the system. The *Verification* however is either used to verify the correct behavior of the system as the result of applying the control sequence or to describe the derivation from the expected behavior. In practice, these annotations are used for test coverage analysis and in particular for test

gap identification. By comparing the data flow classified for failures not found during integration testing to the existing test cases used for integration testing missing test cases can be identified in a systematic way. If for example data flow for the *Verification*-purpose is referenced by failures slipped-by the integration testing phase which is not used by existing test cases, there is a potential lack in state-space coverage of the systems precondition-states. In addition, if data flow for the *Stimulation* is referenced by undetected failures but not used in existing test cases, there is a potential lack of test scenarios defined for the already considered precondition states of the system. And finally, if data flow for the *Verification*-purpose is missing in existing test cases, not the complete system's behavior as a result of the applied control-sequence is considered for verification.

The final element relevant for the studies provided in this work is the *TestExecution*. In its essence, the test execution captures the history of test data in a chronological way. However, in addition this element also serves as the connecting element between an executed test case and the *FailureReport* created based on a failed test result. In practice, this element is used to perform the analysis described in section 3 and in particular the above mentioned test gap identification. Due to the fact that a *FailureReport* does not necessarily require a test execution as the context of its occurrence, test reports created during other test activities then integration testing can be considered as well. The test gap identification can therefore be performed in a fully automated way by conduction the data flow similarity analysis introduced in section 3.4 to failure reports with no root test case execution and the overall existing test cases.

In practice, the data collection for the described elements is not a straightforward task. In practice, adding the activities for test data classification to an existing test process has an impact on not only the process of system integration testing. Beside classifying existing test cases for system integration testing and its failure findings, failures found during other test activities need to be classified as well. In this work, data flow classification of test cases and failure reports was performed by a team of test specialists also involved in analyzing failures found during system integration testing. This

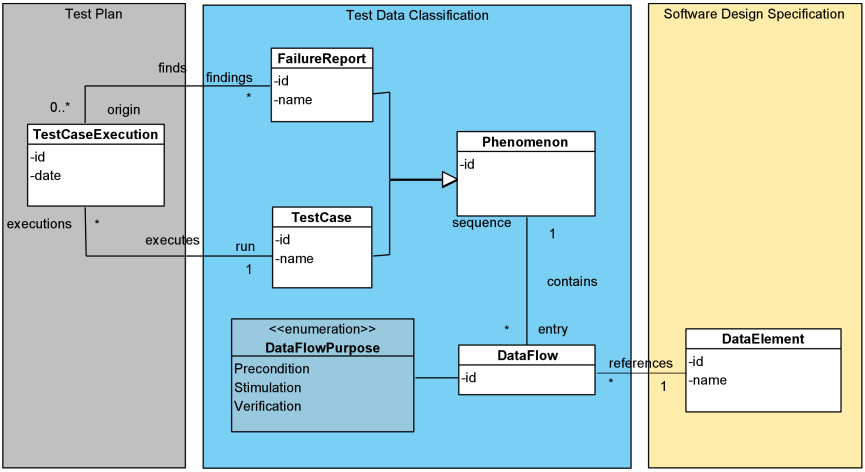



Figure 5.3: Entity Relationship Model for Test Design Classification

ensures a time effective classification with minimal error rate due to the fact that the team is familiar with the components contained in the system. However, it also includes the risk for potential biases introduced into the classification as described in section 3.6. However, based on the test data classification done in section 3.6, we assume that including the activities required for classifications into existing activities would result in an time and cost effective way of data collection. For example, test case classification can be included at the end of a test cases implementation while the classification of a failure report should be done after fault localization. In both cases, the classification would be conducted in a situation where all relevant information are at hand and the additional effort for the practice of system integration would, in its essence, be an additional step of documentation.

## 5.3 Coupling based System Integration Testing Process

Our proposal for the steps required for the practice of system integration testing in the automotive industry involves the development process of the

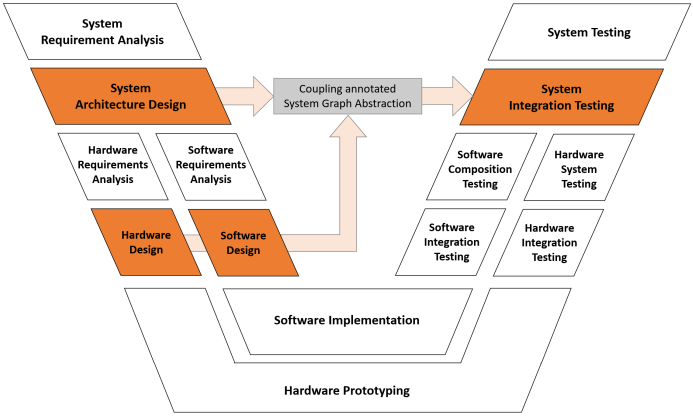



Figure 5.4: Collaborating activities of the V-Model involved in the proposed approach of Coupling-based System Integration Testing

*system architecture and design* as well as the *system integration testing*. In the context of the test data required for conducting data flow based test coverage and test prioritization explained in the previous section, this section describes the data collection in practice. In figure 5.4, the activities addressed for coupling based system integration testing of the V-Model are highlighted as a continuation of the V-Model based development process described in section 2.1. Within this section, these activities are explained in detail alongside a activity flow diagram shown in figure 5.5.

The most important activity for Coupling based System Integration Testing is the *System Architecture Design* as well as the *Software Design* due to the fact that these provide a formal description of the system's 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 and can be converted to a system graph in a straight forward way as described in section 4.4. Even though the graph abstraction used in the studies provided in this work has been generated out of a database export as described in section 5.1

without manual steps involved, the system design tools where proprietary to the company at which the studies have 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, the coupling measures are used to measure 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* in order to create a *Coupling annotated System Graph Abstraction*. This annotated graph abstraction is then provided to the system integration testing process as its contained coupling values are indicating the probability of each element of containing faults according to the correlation measured in section 4.6.

The process of *System Integration Testing* starts with a *Data Flow Analysis* of the already existing test cases, if any, in order to identify the data flow already used and covered for a) describe the system's state for a test cases precondition, b) describe the stimulation applied to the system during execution and c) verification of the systems behavior. A brief description of how component and interface interactions are identified in test cases is provided in section 5.2. The data flow analysis of these existing test cases results in the *Data Flow annotated System Integration Test Suites* which is then used in combination 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 5.1 the overall space of observable data flow based system behavior can become very large in a distributed system with moderate complexity. Accomplishing a full test coverage is in practice often not feasible. In order 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 the failure occurrence for the system under test is used according to the results shown in section 4.7. As a result, component behavior 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 parallel to the activities of System Integration Testing, the *Test Gap Identification* is conducted. The input required for these activities is a pre-selected set of failures related to component interaction faults but found by other testing activities then System Integration Testing. Such a set of failure reports is a strong hint for potentially existing test gaps as described in section 3.6. As part of the activity for *Data Flow Analysis*, this set of failure reports is classified according to the data flow required to describe the failure behavior. The resulting data flow annotated failure reports are then used for a fully automated comparison against the existing data flow annotated system integration test cases in order to *Identify uncovered Component-interaction* which are then used to further guide the specification and implementation of additional test cases.

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.

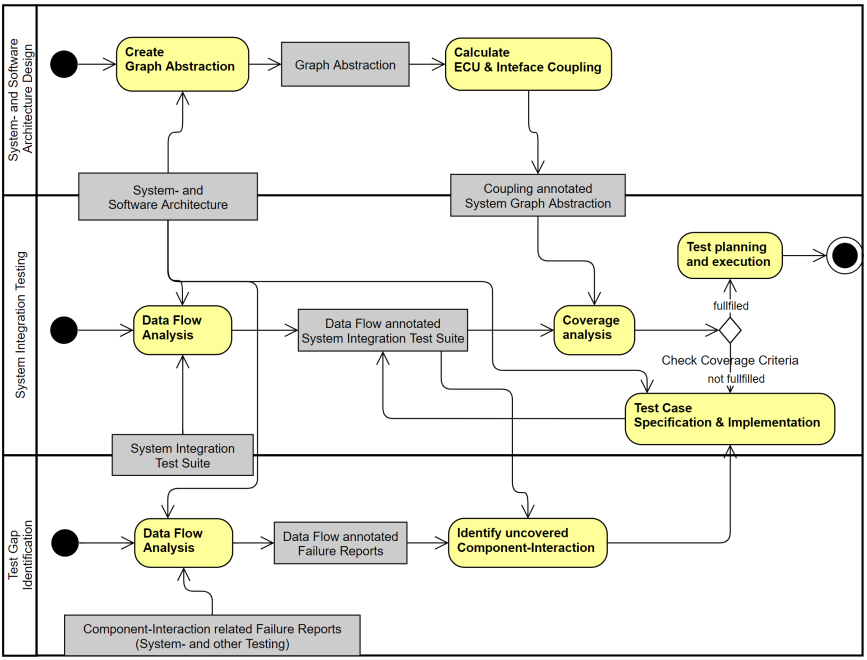



Figure 5.5: Proposed activities for Coupling-based System Integration Testing

Chapter 6

# Discussion and Conclusion

This final chapter first summarizes the contributions of this thesis. For our two main research areas of this thesis, we first discuss the implications of these contributions from a holistic perspective and also provide their limitations in detail. This discussion then leads to a section on possible future research to address presented short-comings or to extend our work. Lastly, we close with some words on the perceived importance of this research field for the practice in the context of the automotive industry.

## 6.1 Summary of Contributions

In chapter 3 and 4 we presented our two main contribution in the area of test coverage analysis for black-box integration testing as well as the selection and prioritization of test case execution based on coupling measures used as an indicator for failure proneness. A visual presentation of the contributions provided in this work is shown in figure 6.1. Before discussing their impact onto practice and research, we briefly summarize it here.

**C1: We provide an approach for test coverage analysis for black-box integration testing based on data flow.**

By first introducing a data flow classification scheme for test cases and failures, we then collected the usage of individual shared data of test cases and failures found during execution as well as failures slipped by the integration testing phase of a real world series development project. By showing that the failures which slipped by the integration test phase do have significantly lower similarity to the executed test cases compared to the failures found during execution we demonstrated the effectiveness of data flow coverage analysis for black-box integration testing. Given that, we introduced basic coverage criteria **(C1.1)** specifically designed to measure the utilization of

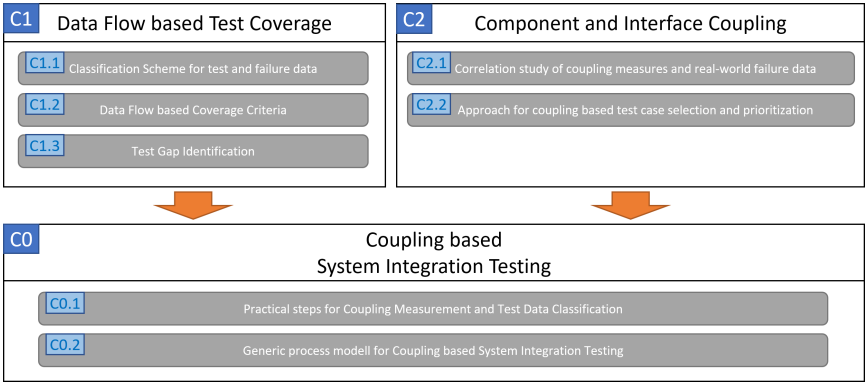



Figure 6.1: Contributions C0-C2 in relation to research process

component interaction based on data flow. Lastly, we demonstrated how the data flow classification of existing test cases and failures slipped by the integration testing phase can be used to identify potential test gaps in terms of unconsidered component interactions **(C1.2)**.

**C2: We provide an empirical evaluation of the usefulness of several coupling measures as defined in literature as an indicator for failure prone component interaction.**

Based on a brief presentation of different approaches of measuring coupling we selected three approaches according to their applicability for component based systems containing black box components: *Information Theory based Coupling*, *Dependency Analysis based Coupling* and *Data-Flow based Coupling*. As part of a correlation study **(C2.1)**, we applied these measures to the interface- and component level architecture of a real world system. The failure distribution over interfaces and components of the system has been collected based on the data-flow involved during occurrence of the failure. The correlation showed that dependency based coupling measures showed the best correlation while the information theory based measures also showed good correlation. Based on that, we lastly defined how the correlation of coupling to the occurrence of failures can be used to select and prioritize test cases during system integration testing **C2.2** based on the strength of coupling of the component interactions covered by a certain test case.

These two contributions lead to our central contribution provided in this dissertation, namely the conceptualization of a **Coupling based Test Methodology for Black-Box Integration Testing (C0)**. For this, we are the first to combine the current industry state of practice for integration testing of distributed systems containing black-box components as well as the scientific state of measuring component and interface coupling to form a systematic methodology for integration testing. In chapter 5, we show how a coupling measurement of the system under test can be derived from the specification documents commonly available in a vehicle series development

in a fully automatic way **C0.1**. In addition, we provide a definition of a generic process model **C0.2** which describes which activities of the development process of a modern vehicle is involved in the proposed coupling based system integration testing approach. For the practice of integration testing, we provide:

- a classification scheme to identify shared data used in existing test cases designed for black-box integration testing as well as in failure reports.
- a basic set of data-flow based coverage criteria for integration testing of black-box components.
- an approach for identifying test gaps based on failures slipped-by the system integration test phase
- an empirical correlation study of coupling measures defined in literature and applicable for black-box component integration testing to real world failure data.
- an approach for coupling based test case prioritization for system integration testing
- a generic process model for Coupling based System Integration Testing

Furthermore we assess the effort and limitations of these approaches during its application to a real world development project alongside its corresponding studies.

## 6.2 Discussion and Limitations

The presented design contributions (C0–C2) have several implications for research and practice, but also significant limitations.

Our **Data Flow Classification Scheme (C1.1)** can in practice be used in order to identify used data flow in existing test cases and failure data. In our empirical evaluation of the classification scheme we showed that it is

directly applicable for test cases used for system integration testing as well as for failure reports describing component interaction related derivations from the expected system behavior. In addition, we also evaluated the effectiveness of the classification scheme to capture the systems dynamic behavior during test case execution or failure occurrence in the context of the architectural and structural specification of the system under test. However, the proposed classification scheme is affected by two major limitations. First, when applying the classification scheme only the usage of data flow is classified. Other aspects, e.g., the exact values applied to the data flow during precondition, stimulation or verification phase of a test execution are not considered. The second noteworthy limitation of the classification scheme is the fact that it requires manual effort to be applied in practice. Even though data flow addressed in a fully automated test case can be extracted out of the formal test script, data flow required to describe the conditions and stimulation required to trigger a failure to surface and which data flow to check in order to observe the relevant failure behavior needs to be done in a manual classification. However, at the company's department performing system integration testing of the chassis control system studied in this work the classification scheme has been established in order to constantly track which component interactions are considered during testing and which are relevant in component interaction related failures.

The **Data Flow based Coverage Criteria (C1.2)** are heavily inspired by code-based data flow coverage criteria defined in literature. The impact onto the practical use of these criteria is therefore straight forward. However, the coverage criteria as described in section 3.5 resemble the most basic ones which only target coverage of the overall data flow defined for the system under test for the different phases of a test case: *Precondition*, *Verification* and *Verification*. Additional work and studies have to be conducted in order to define more elaborate coverage criteria which provide practical use in terms of test exit criteria. However, at the department for system integration testing at which the studies provided in this work have been conducted these coverage criteria are not considered for practical use. This is caused by the

fact that test exit criteria are established during which the test coverage to be achieved is mainly derived from a cost-benefit analysis.

Our approach for **Test Gap Identification (C1.3)** has been demonstrated as applicable for the practice of system integration testing. By using the approach, data flow base component interactions not considered in existing test cases for integration testing are identified. Furthermore, for each uncovered data flow information about the purpose to which it needs to be considered in additional test case specification can be derived with minimal effort. The noteworthy limitations of the approach are direct consequences of the limitations named for the classification scheme. Accordingly, the proposed test gap identification only highlights data flow which is unused in existing test cases. Data flow which can be used in a more meaningful way is not pointed out. At the company at which the study related to test gap identifications has been conducted the approach has been established for practice as it is presented in section 4.6.

Our **Correlation Study for Coupling Measures and Failure Occurrences (C2.1)** mainly provides the evaluation for the proposed approach of test case selection and prioritization during system integration testing. 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 are therefore convinced that the number of coupled elements is an indicator for fault-proneness and can be used to guide test case prioritization during system integration testing. However, in addition to that, the correlation study has also been used for the identification of future research directions. One of the key findings which could be derived from the study is the fact that existing coupling measures are not directly applicable to reactive real-time systems like the studied chassis control system as they depend to a certain degree on the programming paradigm (e.g. object oriented, caller/-

callee communication) used for the implementation of the system under test. In addition to that, the study also showcased only moderate correlation of coupling measures to the failure prone component interactions which highlights that other aspects are potentially causing component interaction related failures other than coupling.

The proposed approach for **Coupling based Test Case Selection and Prioritization (C2.2)** combines the data flow based coverage analysis and the insights derived from the correlation study in order to allow for a selection of test cases with the highest projected fault detection probability during system integration testing. Our results provide the empirical foundation for potential approaches to guide test case selection by utilizing coupling measures. Within the context of this work, the approach has only been compared to the experience based test selection and prioritisation established at the department for system integration testing at which the provided studies have been conducted. As also named in section 6.3, a comparison of the presented approach to other approaches for test case selection and prioritization applicable for integration testing of black box components has been left out for future work. However, for the practice of system integration testing the proposed approach has been established as an addition to the experience based test selection in order to collect data about its effectiveness.

## 6.3 Future Work

Based on our contributions, a wide variety of follow-up research is possible which address either the limitations discussed in the previous section or the enhancement of the presented approaches for its use in practice. In this section, we split our proposal for future research directions into two sections. First, we address potential future work regarding the use of coupling as an indicator for error prone component interactions in reactive real-time systems. Second, we discuss research directions regarding the data flow

based test coverage measurement.

For the goal of defining a systematic test methodology and process model for system integration testing of automotive subsystems, there is a clear need for future research in the direction of coupling and its definition. In our studies, we only consider the existence of data flow between two or more components or interfaces as a coupling indicator. Therefore, our studies in that direction only cover the quantitative aspect of coupling while qualitative aspects as described in 4.3 are left out. As a consequence, coupling has been measured based on the assumption that each individual data flow based component interaction has an equally likelihood of showing failure behavior during system integration testing. However, we highly agree with Xia's definition of coupling [Xia00] as a composite measure combining the *number of dependencies*, the *complexity of shared information* as well as the *impact on dynamic behavior*. Accordingly, future work should investigate additional qualitative aspects of coupling addressing the complexity and impact as defined in literature. In addition, aspects unique to the programming paradigm used for automotive subsystems like the frequency of cyclic component interactions and the differentiation of cyclic and event triggered data flow should be considered for future research.

However, beside taking additional details of data flow based coupling measurement into account another major research direction can be identified based on the architectural view provided in section 2.1. In this thesis, we studied coupling between individual ECUs (*Component Coupling*) and external interfaces implemented by these ECUs (*Interface Coupling*). However, the architectural views commonly available in the automotive industry also provide a functional architecture in which functions are defined on *usage level* and broken down into sub-functions. Each individual sub-functions provide and consume shared data elements. In a follow-up study, coupling measures can be applied to such a functional architecture to derive the *Function Coupling* of the system under test in order to evaluate its usefulness as an fault prediction model or to select and prioritize test case for execution.

In summary, we are convinced that additional work in the mentioned di-

rections would lead to an coupling measurement with improved correlation to the failure locations and occurrence rates. Furthermore, this research direction will result in the theoretical foundation required to derive a comprehensive *Coupling based Fault Prediction Model* for system integration testing. In addition, research in the mentioned directions would also provide arguments to the question if and to which degree the definition of coupling is dependent to the programming paradigm implemented by the system under test.

New insights about additional aspects relevant for measuring coupling or additional architectural views used to guide system integration testing would also be relevant for the topic of data flow based test coverage analysis. Similar to the measurement of coupling as studied in this work, the test coverage analysis also only takes the use of existing data flow in a given test case or failure report into account in combination with the purpose of the use. Therefore, a coverage analysis done based on the classification scheme proposed in section 3.3 results in an overestimation of actual test coverage which is described in more detail in section 3.6. One potential aspect to be considered in future work regarding data flow based test coverage analysis is the coverage of the value spaces for each relevant data element. The extension of the classification approach should address existing approaches for code-based data flow coverage. By using the definition of data flow similarity as provided in section 3.4, real world test data can be identified and used for evaluation which show high similarity to failures in terms of data flow utilization but have not lead to their detection. Given additional aspects relevant for data flow based coverage analysis, the data flow based coverage criteria could be revisited in a future work as well. Here the extension of the criteria to valuable test exit criteria for practice and the evaluation is the most important line for future work.

Another major research direction can be identified for the practical use of the data flow based test coverage analysis approach. As described in section 5.2, manual work is required in order to conduct the coverage analysis and ultimately perform the test gap identification as it is presented in this

work. Consequently, the automated identification of data flow covered by an existing test case or relevant in a certain failure report represents a valuable research direction. In addition, the generation of data flow sequences for potential test case implementation or the generation of test cases based on the system's architecture definition would be of high value for the practice of system integration testing.

In summary, the mentioned future research directions for data flow based test coverage analysis would increase the accuracy of the actual coverage archived by existing test cases which would subsequently also improve the test gap identification. Furthermore, addressing the major drawback of requiring manual work provides an important line for future work.

## 6.4 Conclusion

The task for system integration testing of automotive subsystems is extensive. On system level, multiple independently developed of-the-shelf ECUs are integrated to implement a coherent set of software functions. In this thesis, we have highlighted some of the relevant aspects of system integration testing for the context of the automotive industry. We have studied the correlation between several coupling measures proposed in scientific literature and the number of observed failures and failure locations at two architectural levels: the component level, which shows the inter-connectivity of ECUs as components of the chassis control system, and the software interface level, which represents the external software interfaces implemented by each ECU. In addition, we investigated the problem of test coverage analysis applicable for integration testing of a component based system containing black box components. Inspired by code-based component behavior observation we defined a data flow centric model of the overall component interactions which can automatically be derived from the system system- and architecture specification. In addition to a classification scheme for test and failure data, we derived an approach of coverage analysis based on component interactions. However, given the fact that the amount of software based

functions in a modern vehicle is growing exponentially, an increasing amount of the effort for software development is done by different development teams external to the car manufacturer. Consequently, this leads to an increasing need for systematic methodologies and processes for integrating and testing these black box software and hardware components in a time and cost constrained context. For this, we believe that the work provided in this thesis provides a unique contribution to this field and holds value for both researchers and practitioners. However, we want to highlight that much more work is required in this field as modern vehicles represent highly distributed and complex software systems.

# Bibliography


[AAEW18] S. Ali, M. Abdellatief, M. Elfaki, A. Wahaballa. “Complexity Metrics for Component-based Software Systems: Developer Perspective.” In: *Indian Journal of Science and Technology* 32 (Sept. 2018) (cit. on p. 91).

[AK99] E. B. Allen, T. M. Khoshgoftaar. “Measuring coupling and cohesion: an information-theory approach.” In: *Proceedings Sixth International Software Metrics Symposium (Cat. No.PR00403)*. Nov. 1999, pp. 119–127 (cit. on pp. 43, 105).

[AKC01] E. B. Allen, T. M. Khoshgoftaar, Y. Chen. “Measuring coupling and cohesion of software modules: an information-theory approach.” In: *Proceedings Seventh International Software Metrics Symposium*. Apr. 2001, pp. 124–134 (cit. on pp. 43, 92, 93).

[All02] E. B. Allen. “Measuring graph abstractions of software: an information-theory approach.” In: *Proceedings Eighth IEEE Symposium on Software Metrics*. June 2002, pp. 182–193 (cit. on pp. 43–48, 93).

[AMgA13] M. Abdellatief, A. B. Md Sultan, A. a. abdul ghani, M. A. Jabar. “A mapping study to investigate component-based software system metrics.” In: *Journal of Systems and Software* 86 (Mar. 2013), pp. 587–603 (cit. on p. 86).

[AMS+18] M. Abdellatief, A. B. Md Sultan, M. Sultan, A. a. abdul ghani, A. Abd Ghani, M. A. Jabar. “Component-based Software System Dependency Metrics based on Component Information Flow Measurements.” In: *The Sixth International Conference on Software Engineering Advances*. May 2018 (cit. on pp. 41, 88).

[AO00] A. Abdurazik, J. Offutt. “Using UML Collaboration Diagrams for Static Checking and Test Generation.” In: *≪UML≫ 2000 — The Unified Modeling Language*. Ed. by G. Goos, J. Hartmanis, J. van Leeuwen, A. Evans, S. Kent, B. Selic. Vol. 1939. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, pp. 383–395 (cit. on p. 50).

[ASR09] M. Anan, H. Saiedian, J. Ryoo. “An architecture-centric software maintainability assessment using information theory.” In: *Journal of Software Maintenance* 21 (2009), pp. 1–18 (cit. on p. 43).

[AUTa] AUTOSAR. *Application Interfaces User Guide*. `https://www.autosar.org/fileadmin/user_upload/standards/classic/4-2/AUTOSAR_EXP_AIUserGuide.pdf`. URL: `http://www.autosar.org` (cit. on pp. 36, 94).

[AUTb] AUTOSAR. *Software Component Template*. `https://www.autosar.org/fileadmin/user_upload/standards/classic/4-1/AUTOSAR_TPS_SoftwareComponentTemplate.pdf`. URL: `http://www.autosar.org` (cit. on pp. 94, 114).

[BDW97] L. C. Briand, J. W. Daly, J. Wust. “A unified framework for cohesion measurement in object-oriented systems.” In: *Proceedings Fourth International Software Metrics Symposium*. Nov. 1997, pp. 43–53 (cit. on p. 44).

[BDW99] L. C. Briand, J. W. Daly, J. K. Wust. “A unified framework for coupling measurement in object-oriented systems.” In: *IEEE Transactions on Software Engineering* 25.1 (Jan. 1999), pp. 91–121 (cit. on p. 44).

[BFL08] L. Bocchi, J. L. Fiadeiro, A. Lopes. “Service-Oriented Modelling of Automotive Systems.” In: *2008 32nd Annual IEEE International Computer Software and Applications Conference*. 2008, pp. 1059–1064 (cit. on p. 30).

[BH95] Y. Benjamini, Y. Hochberg. “Controlling the False Discovery Rate: A Practical and Powerful Approach to Multiple Testing.” In: *Journal of the Royal Statistical Society: Series B (Methodological)* 57.1 (Jan. 1995), pp. 289–300. URL: `https://doi.org/10.1111/j.2517-6161.1995.tb02031.x` (cit. on p. 101).

[BMB96] L. C. Briand, S. Morasca, V. R. Basili. “Property-based software engineering measurement.” In: *IEEE Transactions on Software Engineering* 22.1 (1996), pp. 68–86 (cit. on p. 17).

[BMD+09] R. E. Blakesley, S. Mazumdar, M. A. Dew, P. R. Houck, G. Tang, C. F. Reynolds, M. A. Butters. “Comparisons of methods for multiple hypothesis testing in neuropsychological research.” In: *Neuropsychology* 23.2 (2009), pp. 255–264. URL: `https://doi.org/10.1037/a0012850` (cit. on p. 101).

[Bro03] M. Broy. “Automotive software engineering.” In: June 2003, pp. 719–720 (cit. on pp. 29, 30).

[CWZB11] J. Chen, H. Wang, Y. Zhou, S. Bruda. “Complexity Metrics for Component-based Software Systems.” In: *International Journal of Digital Content Technology and its Applications* 5 (Mar. 2011), pp. 235–244 (cit. on p. 90).

[DAM+21] B. Du, S. Azimi, A. Moramarco, D. Sabena, F. Parisi, L. Sterpone. “An Automated Continuous Integration Multitest Platform for Automotive Systems.” In: *IEEE Systems Journal* PP (May 2021), pp. 1–12 (cit. on p. 35).

[DRK04] H. Do, G. Rothermel, A. Kinneer. “Empirical studies of test case prioritization in a JUnit testing environment.” In: *15th International Symposium on Software Reliability Engineering*. 2004, pp. 113–124 (cit. on p. 63).

[EJA14] A. Elsafi, D. N. A. Jawawi, A. Abdelmaboud. “Inferring approximated models for integration testing of component-based software.” In: *2014 8th. Malaysian Software Engineering Conference (MySEC)*. 2014, pp. 67–71 (cit. on p. 52).

[EMR02] S. Elbaum, A. G. Malishevsky, G. Rothermel. “Test case prioritization: a family of empirical studies.” In: *IEEE Transactions on Software Engineering* 28.2 (2002), pp. 159–182 (cit. on pp. 63, 79).

[FCX12] C. Fang, Z. Chen, B. Xu. “Comparing logic coverage criteria on test case prioritization.” In: *Science China Information Sciences* 55 (Dec. 2012) (cit. on p. 63).

[FI98] P. G. Frankl, O. Iakounenko. “Further empirical studies of test effectiveness.” In: *ACM SIGSOFT Software Engineering Notes* 23.6 (1998), pp. 153–162 (cit. on p. 54).

[FW88] P. G. Frankl, E. J. Weyuker. “An applicable family of data flow testing criteria.” In: *IEEE Transactions on Software Engineering* 14.10 (1988), pp. 1483–1498 (cit. on pp. 50, 55).

[GB08] N. S. Gill, Balkishan. “Dependency and Interaction Oriented Complexity Metrics of Component-based Systems.” In: *ACM Special Interest Group on Software Engineering* 33.2 (Mar. 2008), 3:1–3:5. URL: `http://doi.acm.org/10.1145/1350802.1350810` (cit. on p. 42).

[GG04] N. S. Gill, P. S. Grover. “Few Important Considerations for Deriving Interface Complexity Metric for Component-based Systems.” In: *ACM Special Interest Group on Software Engineering* 29.2 (Mar. 2004), pp. 4–4. URL: `http://doi.acm.org/10.1145/979743.979758` (cit. on p. 88).

[GKJ16] G. L. Gopu, K. V. Kavitha, J. Joy. “Service Oriented Architecture based connectivity of automotive ECUs.” In: *2016 International Conference on Circuit, Power and Computing Technologies (ICCPCT)*. 2016, pp. 1–4 (cit. on p. 30).

[Gra07] R. M. Gray. *Entropy and information theory. 2nd ed*. Springer US, Jan. 2007 (cit. on p. 45).

[GRM+18] M. Golagha, A. M. Raisuddin, L. Mittag, D. Hellhake, A. Pretschner. “Aletheia: A Failure Diagnosis Toolchain.” In: *2018 IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*. 2018, pp. 13–16 (cit. on p. 25).

[Gro] V. C. Group. *FIBEX und AUTOSAR – die Unterschiede und Gemeinsamkeiten im Detail*. `https://www.elektronikpraxis.vogel.de/fibex-und-autosar-die-unterschiede-und-gemeinsamkeiten-im-detail-a-226651`. Accessed: 2021-08-23 (cit. on p. 115).

[GS09] G. Gui, P. Scott. “Measuring Software Component Reusability by Coupling and Cohesion Metrics.” In: *Journal of Computers* 4 (Sept. 2009) (cit. on p. 90).

[HBSW22] D. Hellhake, J. Bogner, T. Schmid, S. Wagner. “Towards using coupling measures to guide black-box integration testing in component-based systems.” In: *Software Testing, Verification and Reliability* 32.4 (Mar. 2022). URL: https://doi.org/10.1002/stvr.1811 (cit. on pp. 24, 78).

[HK81] S. Henry, D. Kafura. “Software Structure Metrics Based on Information Flow.” In: *IEEE Transactions on Software Engineering* SE-7.5 (Sept. 1981), pp. 510–518 (cit. on p. 41).

[HS91] M. J. Harrold, M. L. Soffa. “Selecting and using data for integration testing.” In: *IEEE Software* 8.2 (1991), pp. 58–65 (cit. on p. 50).

[HSS+13] L. Heidrich, B. Shyrokau, D. Savitski, V. Ivanov, K. Augsburg, D. Wang. “Hardware-In-The-Loop Test Rig for Integrated Vehicle Control Systems.” In: vol. 7. Sept. 2013, pp. 683–688 (cit. on pp. 33, 34).

[HSW19] D. Hellhake, T. Schmid, S. Wagner. “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)*. Apr. 2019, pp. 420–429 (cit. on pp. 24, 50, 82).

[HW18] D. Hellhake, S. Wagner. *Kommunikationsfluss-orientiertes Testen von Softwarefunktionen im Steuergeräteverbund*. Tech. rep. 7. FACHKONFERENZ AUTOTEST, 2018. URL: https://fkfs-veranstaltungen.de/fileadmin/4_AutoTest/pdf/AutoTest_2018/ (cit. on pp. 24, 50).

[HZZ+14] D. Hao, L. Zhang, L. Zhang, G. Rothermel, H. Mei. “A Unified Test Case Prioritization Approach.” In: *ACM Trans. Softw. Eng. Methodol.* 24.2 (Dec. 2014). URL: https://doi.org/10.1145/2685614 (cit. on p. 79).

[Int08] International Standard. *829-2008 - IEEE Standard for Software and System Test Documentation*. Mar. 2008. URL: https://standards.ieee.org/standard/829-2008.html (cit. on p. 59).

[Int11] International Standard. *ISO 26262: Road Vehicles - Functional Safety*. Nov. 2011. URL: http://www.iso.org (cit. on pp. 21, 32, 35).

[Int13] International Standard. *ISO/IEC/IEEE 29119: Software and systems engineering - Software testing*. Sept. 2013. URL: http://www.iso.org (cit. on p. 32).

[IS06] D. Ince, M. Shepperd. "An empirical and theoretical analysis of an information flow based design metric." In: *Proceedings of the European Conference on Software Engineering*. Jan. 2006, pp. 86–99 (cit. on p. 41).

[IYY+05] K. Inoue, R. Yokomori, T. Yamamoto, M. Matsushita, S. Kusumoto. "Ranking significance of software components based on use relations." In: *IEEE Transactions on Software Engineering* 31.3 (Mar. 2005), pp. 213–225 (cit. on p. 91).

[JHS03] J. Jones, M. Harrold, I. Society. "Test-Suite Reduction and Prioritization for Modified Condition/Decision Coverage." In: *IEEE Transactions on Software Engineering* 29 (June 2003) (cit. on p. 63).

[JO96] Z. Jin, A. J. Offutt. "Coupling-based integration testing." In: *Proceedings of ICECCS '96: 2nd IEEE International Conference on Engineering of Complex Computer Systems (held jointly with 6th CSESAW and 4th IEEE RTAW)*. IEEE Comput. Soc. Press, 1996, pp. 10–17 (cit. on p. 53).

[JO98] Z. Jin, A. J. Offutt. "Coupling-based criteria for integration testing." In: *Software Testing, Verification and Reliability* 8.3 (1998), pp. 133–154 (cit. on p. 53).

[KE] S. Kandl, M. Elshuber. *A Formal Approach to System Integration Testing*. URL: %5Curl%7Bhttp://arxiv.org/pdf/1404.6743v1%7D (cit. on p. 50).

[KH81] D. Kafura, S. Henry. "Software Quality Metrics Based on Interconnectivity." In: *Journal of Systems and Software* 2.2 (June 1981), pp. 121–131. URL: http://dx.doi.org/10.1016/0164-1212(81)90032-7 (cit. on p. 41).

[KS08] L. Kharb, R. Singh. "Complexity Metrics for Component-oriented Software Systems." In: *ACM Special Interest Group on Software Engineering* 33.2 (Mar. 2008), 4:1–4:3. URL: http://doi.acm.org/10.1145/1350802.1350811 (cit. on pp. 41, 42).

[KU11] U. Kumari, S. Upadhyaya. “An Interface Complexity Measure for Component-based Software Systems.” In: *International Journal of Computer Applications* 36 (Jan. 2011) (cit. on p. 41).

[LHL18] C. Lee, Y. Huang, I. Lan. “Hardware-in-the-Loop Test Case Specification for Verification of Software Safety Requirements in the Context of ISO 26262.” In: *2018 International Conference of Electrical and Electronic Technologies for Automotive*. 2018, pp. 1–6 (cit. on p. 59).

[Li03] B. Li. “Managing Dependencies in Component-Based Systems Based on Matrix Model.” In: *Proc. Of Net.Object.Days 2003*. 2003, pp. 22–25 (cit. on p. 42).

[LIM94] I. Lee, R. K. Iyer, A. Mehta. “Identifying software problems using symptoms.” In: *Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing*. IEEE Comput. Soc. Press, 1994, pp. 320–329 (cit. on pp. 52, 53).

[MCT08] L. Mei, W. Chan, T. Tse. “Data Flow Testing of Service-Oriented Workflow Applications.” In: May 2008, pp. 371–380 (cit. on p. 50).

[MDO+16] N. Mendes, D. Dias, L. Oliveira, C. Lana, E. Nakagawa, J. Maldonado. “Exploring together Software Architecture and Software Testing: A Systematic Mapping.” In: *International Conference of the Chilean Computer Science Society*. Vol. 35. Oct. 2016 (cit. on pp. 80, 82).

[MHZ+12] H. Mei, D. Hao, L. Zhang, L. Zhang, J. Zhou, G. Rothermel. “A Static Approach to Prioritizing JUnit Test Cases.” In: *IEEE Transactions on Software Engineering* 38.6 (2012), pp. 1258–1275 (cit. on p. 63).

[MO97] R. Martin, October. *OO Design Quality Metrics*. 1997 (cit. on pp. 87, 105).

[MS12] M. Meitner, F. Saglietti. “Software Reliability Testing Covering Subsystem Interactions.” In: *Measurement, Modelling, and Evaluation of Computing Systems and Dependability and Fault Tolerance*. Ed. by J. B. Schmitt. Vol. 7201. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 46–60 (cit. on p. 50).

[Mye77] G. J. Myers. “Reliable Software Through Composite Design.” In: *Journal of the American Society for Information Science* 28 (1977), pp. 303–303 (cit. on pp. 89, 90).

[NH07] L. Narasimhan, B. Hendradjaya. “Some theoretical considerations for a suite of metrics for the integration of software components.” In: *Information Sciences* 177 (Feb. 2007), pp. 844–864 (cit. on pp. 41, 42).

[NT08] K. Naik, P. Tripathy. “System Integration Testing.” In: *Software Testing and Quality Assurance*. John Wiley & Sons, Ltd, 2008. Chap. 7, pp. 158–191. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9780470382844.ch7. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470382844.ch7 (cit. on pp. 18, 32, 33, 108).

[OHK93] A. J. Offutt, M. J. Harrold, P. Kolte. “A Software Metric System for Module Coupling.” In: *Journal of Systems and Software* 20.3 (Mar. 1993), pp. 295–308. URL: http://dx.doi.org/10.1016/0164-1212(93)90072-6 (cit. on pp. 53, 55, 89, 90).

[Pag01] M. Page-Jones. *The Practical Guide to Structured Systems Design: 2nd Edition*. USA: Yourdon Press, 2001 (cit. on pp. 53, 55, 89).

[RH08] P. Runeson, M. Höst. “Guidelines for conducting and reporting case study research in software engineering.” In: *Empirical Software Engineering* 14.2 (Dec. 2008), p. 131. URL: https://doi.org/10.1007/s10664-008-9102-8 (cit. on pp. 56, 57, 83).

[RHRR12] P. Runeson, M. Host, A. Rainer, B. Regnell. *Case Study Research in Software Engineering: Guidelines and Examples*. 1st. Wiley Publishing, 2012 (cit. on pp. 56, 83).

[RNS20] M. Rizwan, A. Nadeem, M. A. Sindhu. “Theoretical Evaluation of Software Coupling Metrics.” In: *2020 17th International Bhurban Conference on Applied Sciences and Technology (IBCAST)*. 2020, pp. 413–421 (cit. on pp. 40, 83).

[RNS21] M. Rizwan, A. Nadeem, M. Sindhu. “Vovel metrics-novel coupling metrics for improved software fault prediction.” In: *PeerJ Computer Science* 7 (June 2021) (cit. on p. 81).

[RUCH01] G. Rothermel, R. Untch, C. Chu, M. Harrold. “Prioritizing test cases for regression testing.” In: *IEEE Transactions on Software Engineering* 27.10 (2001), pp. 929–948 (cit. on p. 63).

[RUCH99] G. Rothermel, R. Untch, C. Chu, M. Harrold. “Test case prioritization: an empirical study.” In: *Proceedings IEEE International Conference on Software Maintenance - 1999 (ICSM'99). 'Software Maintenance for Business Change' (Cat. No.99CB36360)*. 1999, pp. 179–188 (cit. on p. 63).

[RW97] D. Richardson, A. Wolf. “Software Testing at the Architectural Level.” In: *International Software Architecture Workshop, Proceedings, ISAW* (Mar. 1997) (cit. on p. 82).

[Sal06] N. Salman. “Complexity Metrics AS Predictors of Maintainability and Integrability of Software components.” In: *Journal of Arts and Sciences* (Jan. 2006) (cit. on p. 91).

[SB91] R. W. Selby, V. R. Basili. “Analyzing error-prone system structure.” In: *IEEE Transactions on Software Engineering* 17.2 (1991), pp. 141–152 (cit. on p. 80).

[SBH15] J. Schroeder, C. Berger, T. Herpel. “Challenges from integration testing using interconnected hardware-in-the-loop test rigs at an automotive OEM – an industrial experience report.” In: *2015 First International Workshop on Automotive Software Architecture (WASA)*. 2015, pp. 39–42 (cit. on p. 33).

[SBS18] A. Singh, R. Bhatia, A. Singhrova. “Object Oriented Coupling based Test Case Prioritization.” In: *International Journal of Computer Sciences and Engineering* 6 (Sept. 2018), pp. 747–754 (cit. on p. 81).

[SCK10] S. P. Shashank, P. Chakka, D. V. Kumar. “A systematic literature survey of integration testing in component-based software engineering.” In: *2010 International Conference on Computer and Communication Technology (ICCCT)*. IEEE, 2010, pp. 562–568 (cit. on pp. 19, 37, 50).

[SG14] M. Shahbaz, R. Groz. “Analysis and testing of black-box component-based systems by inferring partial models.” In: *Software Testing, Verification and Reliability* 24 (June 2014) (cit. on p. 52).

[SGK09] A. Sharma, P. Grover, R. Kumar. “Dependency analysis for component-based software systems.” In: *ACM SIGSOFT Software Engineering Notes* 34 (July 2009), pp. 1–6 (cit. on p. 42).

[She95] S. Sherer. “Software fault prediction.” In: *J. Syst. Softw.* 29 (1995), pp. 97–105 (cit. on p. 77).

[SKB11] S. Sengupta, A. Kanjilal, S. Bhattacharya. “Measuring complexity of component based architecture: a graph based approach.” In: *ACM SIGSOFT Software Engineering Notes* 36 (Jan. 2011), pp. 1–10 (cit. on p. 42).

[SN13] J. Sobotka, J. Novák. “Automation of automotive integration testing process.” In: *2013 IEEE 7th International Conference on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS)*. Vol. 01. 2013, pp. 349–352 (cit. on p. 37).

[Spi95] A. Spillner. “Test criteria and coverage measures for software integration testing.” In: *Software Quality Journal* 4.4 (1995), pp. 275–286 (cit. on p. 50).

[SRCF17] D. Santos, A. Resende, E. de Castro Lima, A. Freire. “Software Instability Analysis Based on Afferent and Efferent Coupling Measures.” In: *Journal of Software* 12 (Jan. 2017), pp. 19–34 (cit. on p. 88).

[SWM+17] T. Su, K. Wu, W. Miao, G. Pu, J. He, Y. Chen, Z. Su. “A Survey on Data-Flow Testing.” In: *ACM Computing Surveys* 50.1 (2017), pp. 1–35 (cit. on p. 64).

[UY93] H. Ural, B. Yang. “Modeling Software for Accurate Data Flow Representation.” In: *Proceedings of the 15th International Conference on Software Engineering*. ICSE ’93. Baltimore, Maryland, USA: IEEE Computer Society Press, 1993, pp. 277–286. URL: `http://dl.acm.org/citation.cfm?id=257572.257633` (cit. on p. 41).

[WCO03] Y. Wu, M.-H. Chen, J. Offutt. “UML-Based Integration Testing for Component-Based Software.” In: vol. 2580. Feb. 2003, pp. 251–260 (cit. on p. 50).

[Wey88] E. J. Weyuker. “Evaluating software complexity measures.” In: *IEEE Transactions on Software Engineering* 14.9 (Sept. 1988), pp. 1357–1365 (cit. on p. 91).

[WGL+16] W. E. Wong, R. Gao, Y. Li, R. Abreu, F. Wotawa. “A Survey on Software Fault Localization.” In: *IEEE Transactions on Software Engineering* 42.8 (2016), pp. 707–740 (cit. on p. 61).

[Whi87] L. J. White. “Software Testing and Verification.” In: *Advances in Computers*. Ed. by M. C. Yovits. Vol. 26. Advances in Computers. Elsevier, 1987, pp. 335–391. URL: `http://www.sciencedirect.com/science/article/pii/S0065245808600108` (cit. on p. 53).

[WZM11] M. Wagner, D. Zöbel, A. Meroth. “An adaptive Software and Systems Architecture for Driver Assistance Systems based on service orientation.” In: *International Journal of Machine Learning and Computing* 1 (Jan. 2011), pp. 359–366 (cit. on p. 30).

[Xia00] F. Xia. “On the concept of coupling, its modeling and measurement.” In: *Journal of Systems and Software* 50 (2000), pp. 75–84 (cit. on pp. 82, 83, 90, 130).

[XLKR00] Xia Cai, M. R. Lyu, Kam-Fai Wong, Roy Ko. “Component-based software engineering: technologies, development frameworks, and quality assurance schemes.” In: *Proceedings Seventh Asia-Pacific Software Engeering Conference. APSEC 2000*. 2000, pp. 372–379 (cit. on p. 17).

[YC79] E. Yourdon, L. L. Constantine. *Structured Design: Fundamentals of a Discipline of Computer Program and Systems Design*. 1st. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1979 (cit. on pp. 53, 81, 87).

[ZH01] H. Zhu, X. He. “An observational theory of integration testing for component-based software development.” In: *25th Annual International Computer Software and Applications Conference. COMPSAC 2001*. IEEE, 2001, pp. 363–368 (cit. on pp. 51, 55, 59).

[ZHZ+13] L. Zhang, D. Hao, L. Zhang, G. Rothermel, H. Mei. “Bridging the gap between the total and additional test-case prioritization strategies.” In: *2013 35th International Conference on Software Engineering (ICSE)*. 2013, pp. 192–201 (cit. on p. 79).

[ZZH+09] L. Zhang, J. Zhou, D. Hao, L. Zhang, H. Mei. “Prioritizing JUnit test cases in absence of coverage information.” In: *2009 IEEE International Conference on Software Maintenance*. 2009, pp. 19–28 (cit. on p. 63).

# List of Figures

# List of Tables