DIPLOMARBEIT

Erkennung potentieller Feature-Interaktionen in Motorsteuergeräte-Software mittels Coupling-Metriken

ausgeführt zur Erlangung des akademischen Grades
einer Diplom-Ingenieurin unter der Leitung von

Univ.Prof. Dipl.-Ing. Dr.techn. Hermann Kaindl

am

Institut für Computertechnik (E384)
der Technischen Universität Wien

durch

Romana Wiesinger, BSc
Matr.Nr. 1027177
Arbeitergasse 47/8, 1050 Wien

Wien, am 16.4.2018 _____

Kurzfassung

Im Vergleich zu früher stellen Kraftfahrzeuge heute viel zusätzliche Funktionalität bereit, wie beispielsweise Stabilitätsprogramme und Fahrerassistenzsysteme. Es werden auch weiterhin immer mehr solcher sogenannter *Features* entwickelt, wodurch die Software in den unterschiedlichen Steuergeräten eines Kraftfahrzeugs zunehmend komplexer wird. An dieser Stelle können Probleme entstehen, wenn diese Features miteinander interagieren. Durch die steigende Anzahl an Features wird es immer schwieriger werden solche Feature-Interaktionen zu erkennen.

Ziel dieser Arbeit war es daher, Feature-Interaktionen systematisch zu erkennen. Dazu sollten geeignete Metriken zum Einsatz kommen – sogenannte Coupling-Metriken. Features in Kraftfahrzeugen können grundsätzlich auf mehreren Steuergeräten verteilt sein. Im Rahmen dieser Arbeit wurde ein Konzept zur Erkennung von Feature-Interaktionen am Beispiel der Software eines Motorsteuergeräts untersucht.

Dazu wurde zu Beginn die vorliegende Software auf vorhandene Features untersucht. Anschließend wurde eine Tool-Chain entwickelt, die das Parsing der Software durchführt und die Metriken auf die Features dieser Motorsteuergeräte-Software anwendet. Die Tool-Chain erlaubt es außerdem die Ergebnisse der Coupling-Metriken zu visualisieren. Anhand dieser Ergebnisse wird diskutiert, ob und wie mit Hilfe von Coupling-Metriken potentielle Feature-Interaktionen erkannt werden können. Erkannte Feature-Interaktionen werden analysiert. Außerdem wird untersucht, ob bzw. welche weiteren Aussagen mit Hilfe von Coupling-Metriken über Feature-Interaktionen getroffen werden können.

Abstract

Compared to the past, today's vehicles provide a great deal of additional functionality, such as stability programs and driver assistance systems. More and more *features* of this kind are being developed, making the software in the various control units of a vehicle increasingly complex. At this point, problems can arise when these features interact with each other. The growing number of features will make it increasingly difficult to detect such feature interactions.

The aim of this work was therefore to systematically detect feature interactions. For this purpose, suitable metrics should be used – so-called coupling metrics. In principle, features in vehicles can be distributed over several control units. Within the frame of this work, a concept for the detection of feature interactions was investigated using the software of an engine control unit as an example.

For this purpose, the present software was initially examined for existing features. Subsequently, a tool chain was developed that parses the software and applies the metrics to the features of this engine control unit software. Furthermore the tool chain makes it possible to visualize the results of the coupling metrics. On the basis of these results it is discussed whether and how potential feature interactions can be detected with the help of coupling metrics. Detected feature interactions are analyzed. In addition, it is examined whether or which further statements about feature interactions can be made by means of coupling metrics.

Danksagung

Diese Diplomarbeit wurde im Rahmen des Forschungsprojekts FeatureOpt am Institut für Computertechnik der Technischen Universität Wien in Kooperation mit Bosch Engineering durchgeführt.

Das Projekt FeatureOpt (No. 849928) wird vom Bundesministerium für Verkehr, Innovation und Technologie (bmvit) im Rahmen des Programms IKT der Zukunft im Zeitraum von Juni 2015 bis Mai 2018 gefördert. Weiterführende Information zu IKT der Zukunft findet sich unter `https://iktderzukunft.at/en/`.

Ich möchte mich bei Bosch Engineering und im Speziellen bei Michael Dübner und Dr. Martin Delvai bedanken, die die notwendigen Rahmenbedingungen für die Durchführung der Diplomarbeit ermöglicht haben. Besonderen Dank gilt Dr. Sven Dominka, der mich während meiner Diplomarbeit bei Bosch Engineering betreut hat. Er hat mich bei der Erarbeitung des nötigen Grundlagenwissens im Bereich der Motorsteuergeräte-Software unterstützt und stand immer mit hilfreichen Ratschlägen zur Seite. Außerdem möchte ich mich bei Univ.Prof. Dipl.-Ing. Dr.techn. Hermann Kaindl für die thematischen Ratschläge, die Betreuung und für die Korrektur dieser Diplomarbeit bedanken.

Ich bedanke mich außerdem bei allen Leuten, die mir nicht nur während dieser Diplomarbeit, sondern auch während des gesamten Studiums zur Seite gestanden haben. Besonderer Dank gilt dabei meiner Familie, die immer für mich da war und mich während des gesamten Studiums unterstützt hat. Meinem Freund Mario bin ich besonders für seine Unterstützung in stressigen Zeiten sehr dankbar.

Inhaltsverzeichnis

1 Einleitung

1.1 Umfeld und Probleme

Moderne Kraftfahrzeuge bieten zahlreiche zusätzliche Funktionalitäten zur Basisfunktionalität, wie zum Beispiel Fahrerassistenzfunktionen oder Start-Stopp-Systeme. Alle diese Funktionen stellen für den Fahrer einen Zusatznutzen dar, sei es etwa der zusätzliche Komfort oder die Effizienzsteigerung des Fahrzeugs. Durch stetige Weiterentwicklung kommen daher immer mehr Funktionalitäten – sogenannte *Features* – hinzu. Aufgrund der steigenden Anzahl solcher Funktionalitäten wird die Software in den unterschiedlichen Steuergeräten des Fahrzeugs klarerweise immer komplexer. In weiterer Folge können Probleme entstehen, wenn diese Features miteinander interagieren.

Betrachtet man beispielsweise die beiden Fahrerassistenzfunktionen Tempomat und Abstandhalteassistent, kann man erkennen, dass es hier zu einem Konflikt kommen kann. Während ein Tempomat immer versucht die vorgegebene Geschwindigkeit zu halten, versucht der Abstandhalteassistent immer einen entsprechenden Abstand zum Fahrzeug vor ihm einzuhalten. Diese unterschiedlichen Anforderungen können in gewissen Situationen zu einem Konflikt führen und müssen entsprechend erkannt und koordiniert werden. Hierbei handelt es sich um einen offensichtlichen Konflikt zweier Systeme, der jedoch durch entsprechende Koordination eindeutig aufgelöst wird.

Da die Anzahl der Features in Motorsteuergeräte-Software immer weiter wächst, wird es zunehmend schwieriger werden solche sogenannten *Feature-Interaktionen* zu erkennen. Werden diese nicht erkannt und dementsprechend nicht koordiniert, kann dies zu unerwartetem Verhalten führen, das in einem sicherheitskritischen System wie diesem fatal sein kann.

Je nach Beschaffenheit eines Features kann es konzentriert in der Software eines Steuergerätes implementiert sein oder auch verteilt in mehreren Steuergeräten im Fahrzeug. In einem Fahrzeug befindet sich heutzutage eine Vielzahl an Steuergeräten, die jeweils unterschiedliche Aufgaben erfüllen. So besitzen beispielsweise viele Komponenten in Fahrzeugen ihr eigenes Steuergerät, wie etwa die Bremsanlage, das Getriebe oder der Motor. Jedes dieser Steuergeräte ist mit seiner eigenen Software ausgestattet und steuert, regelt und überwacht primär die Funktionen der entsprechenden Komponente. Die einzelnen Steuergeräte arbeiten nicht vollständig unabhängig voneinander, sondern müssen unterschiedliche Informationen miteinander austauschen. Beispielsweise wird die aktuelle Fahrgeschwindigkeit zentral berechnet und über Bussysteme an verschiedene Steuergeräte, die diese Information benötigen, verteilt. Dementsprechend können sich auch viele Features über mehrere Steuergeräte verteilen.

Diese Arbeit beschäftigt sich mit der Erkennung von Feature-Interaktionen aus der Sicht der Motorsteuergeräte-Software. Auch im oben genannten Beispiel der Fahrerassistenzfunktionen verteilen sich die beiden Features Tempomat und Abstandshalteassistent auf unterschiedliche Steuergeräte, ein Teil davon befindet sich im Motorsteuergerät. Der Konflikt dieser beiden Features manifestiert sich aus Sicht des Motorsteuergerätes konkret in verschiedenen Drehmomentanforderungen.

1.2 Ziel und Aufgabe der Arbeit

Ziel dieser Arbeit ist nun die systematische Erkennung potentieller Feature-Interaktionen mittels geeigneter Metriken – sogenannter *Coupling-Metriken*. Konkret sollte dies untersucht werden anhand eines Beispiels der Software eines Motorsteuergeräts für einen Ottomotor.

Zu diesem Zweck sollte diese vorliegende Motorsteuergeräte-Software auf vorhandene Features untersucht werden. Außerdem sollte eine Tool-Chain entwickelt werden, die es erlaubt, Coupling-Metriken auf die Features in dieser Motorsteuergeräte-Software anzuwenden und die Ergebnisse auszuwerten. Basierend auf diesen Ergebnissen sollte untersucht werden, ob und wie stark Paare von Features miteinander interagieren. Um die Zusammenhänge zwischen den Features besser zu veranschaulichen, sollte innerhalb der Tool-Chain außerdem eine geeignete Form der Visualisierung geschaffen werden. Von großem Interesse ist an dieser Stelle auch, ob die so erkannten potentiellen Feature-Interaktionen bekannt bzw. gewollt sind. Features können oftmals nicht weitgehend unabhängig voneinander entwickelt werden. Daher wird es voraussichtlich der Fall sein, dass eine Vielzahl an Feature-Interaktionen auftreten werden, die durchaus bekannt und auch gewollt sind. Aus diesem Grund sollte schließlich diskutiert werden, ob die erkannten Feature-Interaktionen in der Motorsteuergeräte-Software bekannt bzw. gewollt sind.

Zusammenfassend sollten im Rahmen dieser Arbeit folgende Aufgaben bearbeitet werden:

- Analyse der für diese Arbeit bereitgestellten Motorsteuergeräte-Software hinsichtlich vorhandener Features
- Bereitstellung bzw. Entwicklung der beschriebenen Tool-Chain
- Anwendung der Coupling-Metriken mithilfe der Tool-Chain und Auswertung der Ergebnisse

1.3 Aufbau der Arbeit

Der Aufbau der Arbeit wird im Folgenden erläutert. Zu Beginn werden in Kapitel 2 alle grundlegenden Begriffe und Konzepte genau beschrieben und definiert. Hier wird ebenfalls eine Auswahl zur Erkennung von Feature-Interaktionen potentiell geeigneter Coupling-Metriken präsentiert. Anschließend wird in Kapitel 3 das Konzept des Lösungsansatzes dargelegt, bevor in Kapitel 4 auf die konkrete Umsetzung der Tool-Chain eingegangen wird. Hier wird im Detail beschrieben, wie die Anwendung der Coupling-Metriken bzw. wie die Auswertung und die Visualisierung ermöglicht wird. In Kapitel 5 wird zu Beginn kurz auf die untersuchten Features in der vorliegenden Motorsteuergeräte-Software eingegangen. Danach werden die Ergebnisse der ausgewählten Coupling-Metriken und die entsprechende Auswertung präsentiert. Im nächsten Kapitel werden diese Ergebnisse diskutiert (Kapitel 6). Zuletzt werden in Kapitel 7 entsprechende Schlussfolgerungen abgeleitet und es wird ein Ausblick auf eine mögliche Weiterarbeit zu diesem Thema gegeben.

2 Grundlagen

Im diesem Kapitel werden die zum Verständnis dieser Arbeit notwendigen Grundlagen erläutert. Zu Beginn wird hier ein Einblick in Motorsteuergeräte und ihre Software gegeben. Anschließend werden Features und deren Interaktionen genauer betrachtet. Die Begrifflichkeiten werden definiert und zugehörige Beispiele angeführt. Um die Coupling-Metriken in weiterer Folge definieren zu können wird hier ein kurzer Überblick über die Graphentheorie gegeben. Dabei wird außerdem eine Möglichkeit der Visualisierung von Graphen beschrieben – das sogenannten „Layout-based Clustering“. Schließlich wird der Begriff der Coupling-Metrik erklärt und die verwendeten Coupling-Metriken werden exakt definiert.

2.1 Motorsteuergerät

In diesem Abschnitt wird das Motorsteuergerät beschrieben. Zu Beginn wird seine Rolle als eingebettetes Echzeitsystem besprochen. Anschließend wird auf die Funktionsweise und den Aufbau seiner Software eingegangen.

2.1.1 Das Motorsteuergerät als eingebettetes Echtzeitsystem

Beim Motorsteuergerät handelt es sich um ein sogenanntes eingebettetes („embedded“) System. Eingebettete Systeme zeichnen sich dadurch aus, dass sie mit ihrer Umwelt interagieren. So können diese Systeme über Sensorik und Aktorik in physikalische Prozesse eingreifen. Im Gegensatz zu einem Universalcomputer („general purpose machine“) dient ein eingebettetes System einer speziellen Anwendung und ist dementsprechend auch für diese Anwendung optimiert [1], [2]. Abbildung 2.1 zeigt das Motorsteuergerät als eingebettetes System. Physikalische Größen werden von Sensoren in messbare elektrische Signale umwandelt und können nach einer Signalaufbereitung und gegebenenfalls Analog-Digital-Wandlung vom Mikroprozessorsystem eingelesen werden. Das Mikroprozessorsystem besteht dabei üblicherweise nicht nur aus einem Prozessor, sondern aus mehreren Prozessorkernen. Demnach beinhaltet das Mikroprozessorsystem auch eine entsprechende Kommunikation zwischen den Prozessoren. Anschließend können die Daten von der Software verarbeitet und entsprechende Ausgangsgrößen berechnet werden. Danach werden die Signale bei Bedarf wieder in analoge Größen umgewandelt, um schlussendlich die Aktorik anzusteuern. Das Motorsteuergerät liest beispielsweise eine Vielzahl von Temperatur- und Drucksensoren aus und steuert unterschiedliche Steller, wie z.B. die Drosselklappe, die Einspritzventile oder die Zündspule, an.

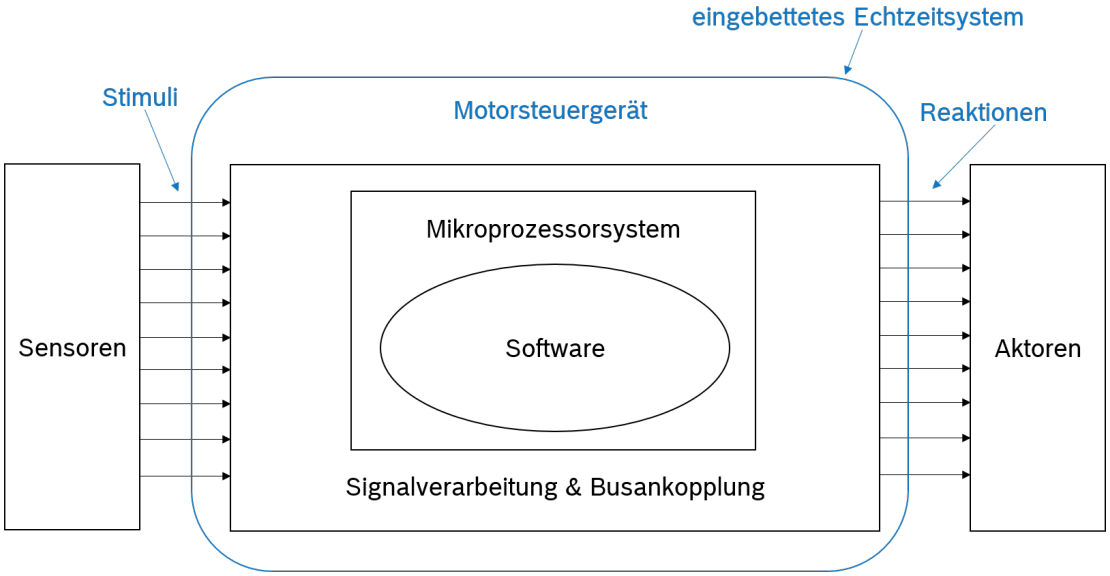

Abbildung 2.1: Das Motorsteuergerät als eingebettetes Echtzeitsystem (angelehnt an [3], [4]).

Bei eingebetteten Softwaresystemen ist die Verarbeitung der Informationen innerhalb eines gewissen Zeitraums essentiell, da sie in reale physikalische Vorgänge eingreifen und daher der exakte Zeitpunkt der Ansteuerung der Eingriffe eine große Rolle spielt. Die entsprechenden Berechnungen dieser Signale dürfen daher, abhängig vom benötigten Signal, eine gewisse Berechnungsdauer nicht überschreiten. Wie bei einem Großteil der eingebetteten Systeme handelt es sich daher auch beim Motorsteuergerät um ein Echtzeitsystem. Ein solches ist definiert als ein System, bei dem die Korrektheit der Funktion nicht nur von den produzierten Ausgabegrößen abhängt, sondern auch von der Zeit, zu welcher sie produziert wurden. Ein Echtzeitsystem muss immer innerhalb eines gewissen Zeitintervalls auf Stimuli aus der Umgebung reagieren. Der Zeitpunkt, zu dem das Ergebnis vorliegen muss, wird dabei als Deadline bezeichnet [3], [5]. Aus Abbildung 2.1 ist ersichtlich, dass bei dem Motorsteuergerät als Echtzeitsystem die eingelesenen Signale der Sensoren die Stimuli des Systems und die Ansteuerungen der Aktoren die Reaktionen darstellen. Bei Ottomotoren ist beispielsweise der exakte Zeitpunkt der Zündung sehr wichtig, da dieser über die Effizienz der Verbrennung bestimmt. Wird zum optimalen Zeitpunkt gezündet, kann ein Großteil der im Kraftstoff gespeicherten Energie in Bewegungsenergie und damit in Drehmoment an der Kurbelwelle umgewandelt werden. Verpasst man diesen Zeitpunkt allerdings und die Zündung findet nicht zum optimalen Zeitpunkt statt, wird nur ein kleinerer Teil der Energie in Bewegungsenergie und mehr in Wärmeenergie umgewandelt. Dies kann zwar durchaus auch beabsichtigt sein (wenn beispielsweise das Abgassystem aufgeheizt werden soll), jedoch sollte die Zündung auch hier immer zum angeforderten Zündzeitpunkt stattfinden. Dementsprechend muss die Motorsteuergeräte-Software die Berechnungen innerhalb der vorgegebenen Deadline abgeschlossen haben, um die Zündspule zeitgerecht ansteuern zu können.

Echtzeitsysteme können eingeteilt werden in „harte“ und „weiche“ Echtzeitsysteme. Die Klassifizierung erfolgt anhand ihrer einzuhaltenden Deadlines. Wenn das Ergebnis einer Berechnung auch nach dem Ablauf einer Deadline noch Gültigkeit hat, handelt es sich um eine weiche Deadline. Andernfalls spricht man von einer festen Deadline. Könnte das Verpassen einer festen Deadline jedoch potentiell schwerwiegende Konsequenzen nach sich ziehen, wird diese Deadline als harte Deadline

bezeichnet. Muss ein Echtzeitsystem mindestens eine harte Deadline erfüllen, handelt es sich um ein hartes Echtzeitsystem. Andernfalls spricht man von einem weichen Echtzeitsystem [5]. Beim Motorsteuergerät handelt es sich um ein hartes Echtzeitsystem. Steht beispielsweise das berechnete Ansteuersignal für die Zündspule erst nach dem angeforderten Zündzeitpunkt zur Verfügung, hat dieses keinen Nutzen mehr. Die Folge davon könnte etwa eine mechanische Beschädigung des Motors bzw. ein stotternder oder absterbender Motor sein. Ein weiteres gefährlicheres Beispiel wäre, wenn etwa der Fahrer den Fuß vom Gaspedal nimmt, um das Drehmoment zu reduzieren und hier die Deadline zur Berechnung des neuen angeforderten, reduzierten Drehmoments nicht eingehalten werden würde. In diesem Zeitraum würde sich das Fahrzeug also schneller als vom Fahrer gewünscht fortbewegen. In beiden Fällen könnte solch ein Fehlverhalten möglicherweise zu einem Unfall und im schlimmsten Fall zu einem Personenschaden führen. Aus diesem Grund spricht man bei einem Motorsteuergerät auch von einem sicherheitskritischen System.

2.1.2 Funktionsweise der Motorsteuergeräte-Software

Motorsteuergeräte-Software dient grundsätzlich der Steuerung, Regelung und Überwachung der Funktionen des Verbrennungsmotors. Wie in Unterkapitel 2.1.1 erwähnt, gehört hierzu das Einlesen diverser Sensorinformationen, eine entsprechende Datenverarbeitung und die Ansteuerung der Aktorik. Zu den wichtigsten Teilsystemen der Motorsteuergeräte-Software zählen das Kraftstoffsystem, das Luftsystem, das Zündsystem, das Abgassystem und die Momentenstruktur. Im Rahmen dieser Arbeit wird das Beispiel der Software eines Motorsteuergeräts für einen Ottomotor betrachtet. Im Folgenden wird die Bildung des Drehmoments im Ottomotor aus der Sicht der Motorsteuergeräte-Software und die Rolle ihrer einzelnen Teilsysteme dabei überblicksartig beschrieben [6], [7].

Die Aufgabe der Software ist es, ein vom Fahrer gewünschtes Fahrverhalten entsprechend umzusetzen. Zu diesem Zweck berechnet die *Momentenstruktur* das hierfür benötigte Drehmoment. Hierzu wird der Winkelsensor des Gaspedals ausgelesen, in das entsprechende geforderte Drehmoment umgerechnet und in weiterer Folge an das Kraftstoff-, Luft- und Zündsystem weitergegeben, um schlussendlich die notwendigen Aktorstellungen zu berechnen. Dabei berechnet das *Kraftstoffsystem* aus diesem geforderten Drehmoment im Wesentlichen die einzuspritzende Kraftstoffmenge und aus diesem beispielsweise die Öffnungs- und Schließzeiten der Einspritzventile oder die Aktivierung der Kraftstoffpumpe. Das *Luftsystem* ermittelt die für das Drehmoment notwendige Luftfüllung der Zylinder und steuert u.a. die Drosselklappe dementsprechend an. Das *Zündsystem* berechnet den Zündzeitpunkt bzw. den sogenannten Zündwinkel, der für das geforderte Drehmoment notwendig ist und sorgt für die notwendige Ansteuerung der Zündspulen. Im *Abgassystem* wird außerdem mittels der Lambdasonde der Restsauerstoffgehalt im Abgas gemessen, woraus das Verbrennungsluftverhältnis bestimmt und in Folge dessen geregelt werden kann. All diese Berechnungen hängen von den unterschiedlichsten Faktoren und Betriebsmodi ab.

Das Drehmoment wird nicht immer direkt über die Gaspedalstellung angefordert, sondern kann von einer Vielzahl an Drehmomentenanforderern kommen. Dazu zählen beispielsweise das Steuergeräte des Antiblockiersystems (ABS) und des Elektronischen Stabilitätsprogramms (ESP). Diese können abhängig von der jeweiligen Fahrsituation entsprechend eingreifen, um den Fahrerwunsch aus Sicherheits- bzw. Komfortgründen aufzuheben und zu überschreiben. Außerdem wurden eine Vielzahl an Fahrerassistenzsystemen entwickelt. Beispiele wären der Tempomat oder der Abstandhalteassistent, welche ebenfalls ein Drehmoment anfordern und je nach Wunsch aktiviert werden können. Zur korrekten Funktion werden all diese angeforderten Drehmomente dem jeweiligen Betriebsmodus entsprechend in der Momentenstruktur koordiniert.

2.1.3 Aufbau der Motorsteuergeräte-Software

Wie bereits erwähnt, lässt sich die Motorsteuergeräte-Software in *Teilsysteme* gliedern. Der Aufbau der Motorsteuergeräte-Software ist dabei nicht Feature-orientiert, sondern orientiert sich am physikalischen Aufbau des Verbrennungsmotors. Dies bietet den Vorteil, dass einzelne Software-Teile besser den physikalischen Prozessen und den entsprechenden Sensoren und Aktoren zugeordnet werden können [7], [8]. Demnach existieren beispielsweise folgende grundlegende Teilsysteme:

- Kraftstoffsystem (inkl. Einspritzsystem)
- Luftsystem
- Zündsystem
- Abgassystem
- Momentenstruktur

Wie in Abbildung 2.2 dargestellt ist, können diese einzelnen Teilsysteme weiter in sogenannte Basiskomponenten, kurz *BCs* („base components“) unterteilt werden. Eine BC besteht nun wiederum aus mehreren Funktionskomponenten, auch *FCs* („function components“) genannt.

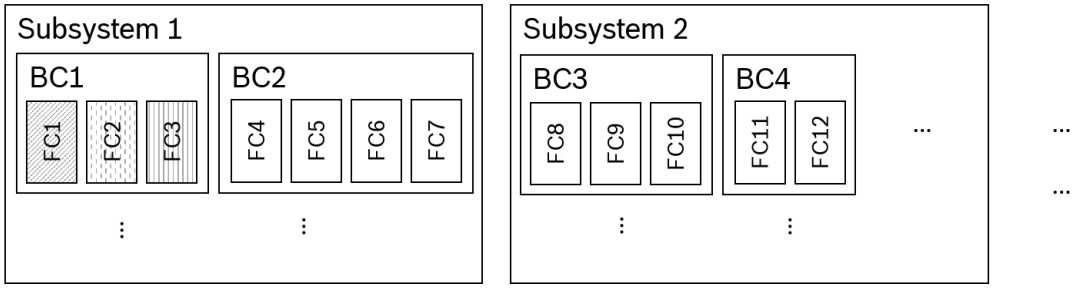

Abbildung 2.2: Aufbau der Motorsteuergeräte-Software bestehend aus ihren einzelnen Teilsystemen („Subsysteme“, wiederum in BCs und weiter in FCs gegliedert).

Da es sich bei der Motorsteuergeräte-Software um ein hartes Echtzeitsystem handelt, ist die Software in der Programmiersprache C realisiert. Gründe hierfür sind ihre Hardwarenähe und die hohe Effizienz [9]. Die Funktionalität einer FC ist jeweils in einer C-Funktion umgesetzt. Die FC stellt damit die elementarste Funktionseinheit der Motorsteuergeräte-Software dar. Die Kommunikation zwischen den FCs basiert auf der gemeinsamen Nutzung von Variablen. Dabei werden Informationen ausgetauscht, indem unterschiedliche FCs auf im Speicher liegende Variablen mittels Schreib- bzw. Lesezugriff zugreifen. Wie in Abbildung 2.3 dargestellt, lesen diese Funktionen dabei jeweils gewisse Größen vom Speicher ein und schreiben wiederum gewisse Größen in den Speicher. Diese Größen bzw. Variablen werden im Folgenden als *Eingangs-* bzw. *Ausgangsvariablen* der jeweiligen FC bezeichnet.

Der Aufruf der einzelnen Funktionen erfolgt zyklisch in einer bereits zur Designzeit festgelegten Reihenfolge (siehe Abbildung 2.4). Dabei wird je nach Dynamik der physikalischen Abläufe des angesteuerten Teilsystems der jeweiligen Funktion festgelegt, wann und wie oft diese aufgerufen werden muss. Manche Berechnungen erfordern eine Synchronisierung zu gewissen physikalischen

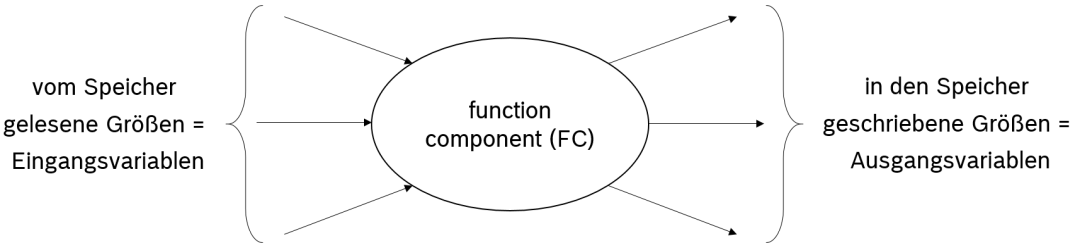

Abbildung 2.3: Die FC als elementarste Funktionseinheit mit ihren Eingangs- und Ausgangsvariablen.

Abläufen, z.B. zur Umdrehung der Kurbelwelle, und werden daher beispielsweise synchron zur Drehzahl aufgerufen. Die Periodendauer des Zyklus, in dem die Funktionen der Dynamik entsprechend aufgerufen werden, wird als Zeitraster bezeichnet.

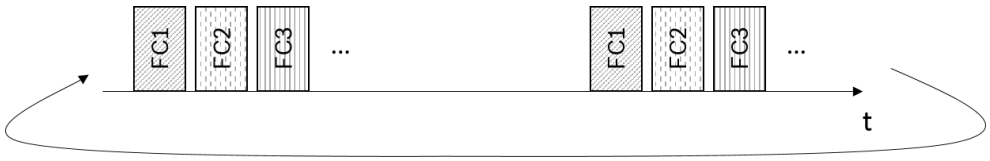

Abbildung 2.4: Zyklischer Aufruf der einzelnen Funktionen in einer festen Reihenfolge.

2.2 Features und deren Interaktionen

In diesem Abschnitt werden Features und deren Interaktionen genauer betrachtet. Wie in 1.1 bereits erwähnt, handelt es sich bei einem Feature im Groben um eine definierte Funktionalität in der Software. Der Begriff soll für diese Arbeit präzisiert werden. Daher werden im Folgenden unterschiedliche Definitionen dieses Begriffs diskutiert und eine geeignete Definition abgeleitet. Anschließend werden Beispiele für Features in der Motorsteuergeräte-Software präsentiert. Anhand der Definition des Features kann schließlich auch der Begriff der Feature-Interaktion definiert werden. Im Anschluss daran werden wiederum zugehörige Beispiele beschrieben.

2.2.1 Features

2.2.1.1 Definition

Zum Begriff des *Features* existieren unterschiedlichste Definitionen. Kang et al. [10] definieren den Begriff sehr abstrakt als einen markanten, für den Benutzer sichtbaren Aspekt, eine Eigenschaft oder Charakteristik eines Software Systems oder Systems. Eine technischere Definition eines Features liefern Apel et al. [11] mit der Beschreibung eines Features als eine Struktur, die die Struktur eines gegebenen Programms erweitert und modifiziert, um ein Stakeholder-Requirement zu erfüllen, eine Design-Entscheidung zu kapseln und eine Konfigurationsoption zu bieten. Zave [12] hingegen beschreibt den Begriff des Features kompakt als eine optionale oder inkrementelle Einheit der Funktionalität. In [13] wird ein Feature als charakteristisches oder für den Endbenutzer sichtbares Verhalten eines Software-Systems definiert.

Im Gegensatz zu anderen Definitionen soll bei der Definition eines Features im Rahmen dieser Arbeit nicht der Aspekt der Benutzer- bzw. Stakeholder-Wahrnehmbarkeit im Vordergrund stehen. Das resultiert daraus, dass viele Features nicht unmittelbar aktiv vom Benutzer verwendet werden, sondern viel mehr indirekt helfen gewisse Ziele zu erreichen. Beispielsweise wird das Feature Klopfregelung nicht vom Benutzer bewusst aktiviert, sondern wird indirekt verwendet, um Komponenten wie z.B. den Kolben vor Beschädigung zu schützen.

Der Begriff des Features wird daher im Rahmen dieser Arbeit definiert als eine charakteristische, für den Entwicklungsingenieur sichtbare Funktionalität in der Motorsteuergeräte-Software. Dabei kann zwischen *obligatorischen* und *optionalen Features* unterschieden werden. Optionale Features haben die Eigenschaft, dass sie im Rahmen der Software-Produktlinie bereits zur Designzeit durch die Konfiguration wahlweise aktiviert bzw. deaktiviert werden können. Mit ihrer Aktivierung wird die Funktionalität des Motorsteuergeräte-Software-Produkts um eine zusätzliche Funktionalität erweitert. Obligatorische Features hingegen sind zwingend für die grundlegende Funktion des Software-Systems notwendig und müssen daher in jeder Konfiguration aktiv sein.

2.2.1.2 Beispiele: Features in der Motorsteuergeräte-Software

Basierend auf der obigen Definition lassen sich in der Motorsteuergeräte-Software zahlreiche Features – sowohl obligatorische als auch optionale – finden. So ist beispielsweise die Zündungssteuerung ein obligatorisches Feature, da diese selbst Teil der Grundfunktionalität der Software und permanent aktiv ist. Die Klopfregelung hingegen, die zwar als Stellgröße einen gewissen Zündwinkelbereich für die Zündungssteuerung vorgeben kann, kann ein optionales Feature darstellen, da diese für die grundsätzliche Funktionalität des Motors nicht zwingend benötigt wird und somit auch wahlweise zur Designzeit deaktiviert werden kann. Im Folgenden werden weitere mögliche Beispiele für Features in der Motorsteuergeräte-Software angeführt.

- **Turboaufladung:** Die Leistung des Motors wird erhöht, indem der Luftmassendurchsatz erhöht wird. Dies wird erreicht, indem die Luft vor dem Eintritt in den Zylinder mittels eines Turboladers verdichtet wird [6].
- **Abgasrückführung:** Heißes Abgas wird entnommen, gekühlt und mit Frischluft vermischt wieder dem Brennraum zugeführt. Dies hat beispielsweise beim Ottomotor den Vorteil, dass die Abgastemperatur gesenkt werden kann und dadurch Bauteile wie Katalysator und Turbolader nicht beschädigt werden [6].
- **Lambdaregelung:** Damit der Dreiwegekatalysator möglichst effizient arbeiten kann, muss ein gewisses Verhältnis aus Luft und Kraftstoff („Lambda“) genau eingehalten werden. Daher wird die Gemischbildung mittels der Lambdaregelung nachgeführt [6].
- **Katalysatorheizen:** Der Zündzeitpunkt bzw. der Zündwinkel wird nach hinten verschoben, um dem Abgastrakt mehr Wärmeenergie zuzuführen, damit der Katalysator nach dem Start schnell seine benötigte Betriebstemperatur erreicht.
- **Fahrerassistenzsysteme:** Zu diesen Systemen zählen etwa der Tempomat, der Abstandhalteassistent sowie der Höchstgeschwindigkeitsregler.
- **Stabilitätsprogramm:** Das Fahrzeugverhalten wird u.a. durch Eingriffe in den Antriebsstrang bis in den physikalischen Grenzbereich hinein verbessert und bleibt daher auch in kritischen Fahrsituationen besser beherrschbar [6].

2.2.2 Feature-Interaktionen

2.2.2.1 Definition

Eine *Feature-Interaktion* (*FI*) bezeichnet im Wesentlichen die Beeinflussung eines Features durch ein anderes, wie z.B. aufgrund von Abhängigkeiten der jeweiligen Software-Teile. Auch hier existieren unterschiedliche Definitionen, die sich jedoch im Grunde in ihrer Aussage ähnlich sind. Vogelsang et al. [14] definieren Feature-Interaktionen als die Beeinflussung des Verhaltens eines Features durch den Zustand oder Daten eines anderen Features. Ähnlich wird dies von Kolesnikov et al. [15] interpretiert. In ihrer Publikation definieren sie die Feature-Interaktion als Beeinflussung des Verhaltens eines Features durch das Vorhandensein oder Nichtvorhandensein eines oder mehrerer anderer Features. Allerdings unterscheiden sie nach dem Kontext, in welchem die FI auftritt. In ihrer Arbeit bezeichnen sie eine FI, die von außen sichtbares Verhalten aufweist, als externe Feature-Interaktion. Eine interne Feature-Interaktion hingegen wird beschrieben als jene, die nur innen sichtbar ist und sich auf Code-, Datenfluss- oder Kontrollflussebene bemerkbar macht.

Im Rahmen dieser Arbeit soll die FI vorerst allgemein als für den Entwicklungsingenieur von außen sichtbare Beeinflussung eines Features durch ein anderes Feature definiert werden. Diese Definition der FI stellt eine externe FI entsprechend der Definition aus [15] dar. Diese Beeinflussung kann u.a. durch Abhängigkeiten von Software-Teilen auftreten. Solche Abhängigkeiten wiederum entsprechen den internen FI aus [15]. Im Rahmen dieser Arbeit werden ausschließlich Feature-Interaktionen zwischen Paaren von Features betrachtet.

Ein Problem durch die FI tritt grundsätzlich dann auf, wenn das Verhalten beider Features in Kombination sich nicht einfach vom separaten Verhalten beider Features ableiten lässt. Wie schon zu Beginn erwähnt wurde, ist dieser Fall in einem sicherheitskritischem System wie diesem, von dem stets deterministisches Verhalten erwartet wird, nicht akzeptabel.

2.2.2.2 Beispiele

Im Folgenden werden einige Beispiele für mögliche Feature-Interaktionen in der Motorsteuergeräte-Software angeführt.

- **ESP-/ABS-Steuergerät:** Ein Beispiel für eine Feature-Interaktion in der Motorsteuergeräte-Software wäre etwa, wenn das ESP- und das ABS-Steuergerät gleichzeitig voneinander abweichende Drehmomente vom Motorsteuergerät anfordern und dadurch ein Konflikt entsteht. Klarerweise handelt es sich hierbei um einen Konflikt, der in der Motorsteuergeräte-Software eindeutig koordiniert ist.
- **Tempomat/Abstandhalteassistent:** Dieses Beispiel wurde bereits in Kapitel 1.1 beschrieben. Hier kann ein Konflikt auftreten, wenn der Tempomat ein vom Abstandhalteassistenten abweichendes Drehmoment vom Motorsteuergerät fordert. Diese Situation könnte entstehen wenn sich das vordere Fahrzeug mit einer geringeren Geschwindigkeit als im Tempomaten eingestellt fortbewegt. Dadurch ergeben sich beim Tempomat und beim Abstandhalteassistent unterschiedliche Soll-Fahrgeschwindigkeiten und damit abweichende Anforderungen an das Drehmoment. Diese müssen in weiterer Folge koordiniert werden.

- **Automatischer Überholassistent/Abstandhalteassistent:** Ein weiteres Beispiel für Feature-Interaktionen in der Motorsteuergeräte-Software wäre etwa im Bereich der Fahrerassistenzsysteme, wenn ein Fahrzeug sowohl mit einem automatischen Überholassistent als auch mit einem Abstandhalteassistenten ausgestattet ist. Hier könnten die beiden Features, wenn beispielsweise der Überholassistent aufgrund der Verkehrslage ein Überholmanöver einleitet, interagieren. Dieser würde demzufolge eine Beschleunigung bzw. ein höheres Drehmoment beim Motorsteuergerät anfordern, um das Fahrzeug vor sich zu überholen. Währenddessen würde der Abstandhalteassistent eine Verzögerung bzw. ein geringeres Drehmoment beim Motorsteuergerät anfordern, um den Abstand zum Fahrzeug vor ihm zu halten. Auch hier muss je nach Verkehrslage eindeutig koordiniert werden [16].

Grundsätzlich existieren in der Motorsteuergeräte-Software eine Vielzahl von Feature-Interaktionen, da Features aufgrund physikalischer Abhängigkeiten oftmals nicht gänzlich unabhängig voneinander entwickelt werden können. Entsprechend lassen sich auch Abhängigkeiten in der Software wieder finden. Diese Abhängigkeiten sind den Entwicklern der Motorsteuergeräte-Software bekannt und werden u.a. durch Koordination der Interaktionen berücksichtigt. Durch die in Zukunft erwartete wachsende Anzahl an Features ist jedoch anzunehmen, dass dies zunehmend schwieriger werden wird.

2.3 Graphentheorie

In unterschiedlichsten Bereichen existieren Systeme, die sich durch Objekte und deren Beziehungen zueinander beschreiben lassen. Die Graphentheorie stellt mit dem Graphen einen Formalismus zur Verfügung, mit Hilfe dessen solche Systeme beschrieben und übersichtlich dargestellt werden können. Dabei werden die Objekte jeweils in Form von *Knoten* repräsentiert und die Beziehungen zueinander in Form von *Kanten* (siehe Abbildung 2.5). Im Folgenden werden einige Beispiele angeführt.

- **Kommunikationsnetze:** Bei einem Kommunikationsnetz stehen die einzelnen Stationen (z.B. Rechner) über ihre Datenübertragungswege (z.B. Leitungen, Funkverbindungen) miteinander in Beziehung. Daher lässt sich ein Kommunikationsnetz als Graph darstellen, wobei die einzelnen Stationen als Knoten und die Datenübertragungswege als Kanten repräsentiert werden.
- **Suchmaschinen:** Das Konzept von Suchmaschinen beruht zum Teil ebenfalls auf Graphen. Dabei wird die Hypertextstruktur des World Wide Web bestehend aus ihren einzelnen Dokumenten, die über Verweise („Links“) miteinander verbunden sind, als Graph repräsentiert. In diesem Fall stellen die Dokumente die Knoten und die Verweise die Kanten dar.
- **Soziale Netzwerke:** Soziale Netzwerke, in denen jeweils verschiedene Akteure über ihre Bekanntschaften miteinander in Verbindung stehen, lassen sich ebenfalls als Graphen darstellen. Hier können die Akteure als Knoten und die Bekanntschaften als Kanten repräsentiert werden.

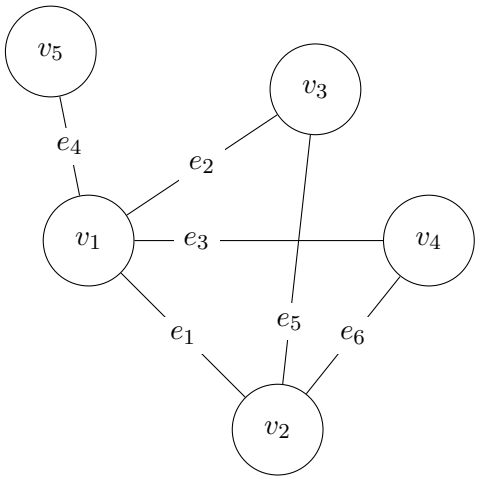

Abbildung 2.5: Einfaches Beispiel eines ungerichteten Graphen mit Knoten $V = \{v_1, ..., v_5\}$ und Kanten $E = \{e_1, ..., e_6\}$.

2.3.1 Definitionen

Der Graph in Abbildung 2.5 stellt einen *ungerichteten Graphen*

$$G = (V, E) \tag{2.1}$$

dar, wobei

$$V = \{v_1, ..., v_n\} \quad \text{mit} \quad n = |V| \tag{2.2}$$

die Menge der Knoten („vertices“, „nodes“) und

$$E = \{e_1, ..., e_m\} \quad \text{mit} \quad m = |E| \tag{2.3}$$

die Menge der Kanten („edges“) zwischen Paaren von Knoten bedeutet. Existiert eine Kante $e \in E$ zwischen den Knoten v_x und v_y

$$e = (v_x, v_y)\,, \tag{2.4}$$

so heißen die Knoten v_x und v_y *adjazent*, um deren Nachbarschaft auszudrücken. In diesem Fall werden außerdem die Kante e und der Knoten v_x (bzw. v_y) als *inzident* bezeichnet. Die Menge aller adjazenten Knoten eines Knotens v_x wird im Folgenden mit

$$V_{v_x} = \{v \in V \mid v \text{ adjazent } v_x\} \tag{2.5}$$

und die Menge seiner inzidenten Kanten mit

$$E_{v_x} = \{e \in E \mid e \text{ inzident } v_x\} \tag{2.6}$$

bezeichnet. Bei einem ungerichteten Graphen gilt

$$(v_x, v_y) = (v_y, v_x)\,. \tag{2.7}$$

Der Knotengrad eines Knoten v

$$deg\,(v_x) = |\,\{e \in E \mid e \text{ inzident } v_x\}\,| = |E_{v_x}| \tag{2.8}$$

ergibt sich aus der Anzahl seiner inzidenten Kanten. Es ist außerdem im Allgemeinen möglich, dass mehrere Kanten zwischen zwei Knoten existieren. In diesem Fall handelt es sich um Mehrfachkanten.

Um den Beziehungen zwischen den einzelnen Objekten eine Richtung zuweisen zu können gibt es neben dem ungerichteten Graphen auch *gerichtete Graphen*. Abbildung 2.6 zeigt einen solchen gerichteten Graphen $G = (V, E)$, wobei hier E die Menge der gerichteten Kanten zwischen Paaren von Knoten bedeutet. Für gerichtete gilt im Gegensatz zu ungerichteten Graphen

$$(v_x, v_y) \neq (v_y, v_x)\,. \tag{2.9}$$

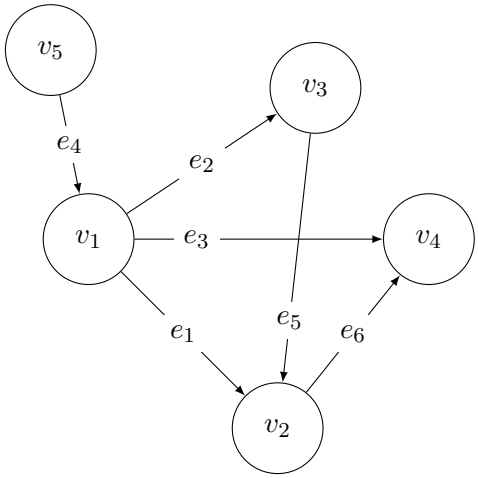

Abbildung 2.6: Einfaches Beispiel eines gerichteten Graphen mit Knoten $V = \{v_1, ..., v_5\}$ und Kanten $E = \{e_1, ..., e_6\}$.

Bei gerichteten Graphen kann außerdem zwischen dem Eingangs- und dem Ausgangsknotengrad unterschieden werden. Dabei handelt es sich beim Eingangsknotengrad um die Anzahl der Kanten, die in den jeweiligen Knoten einmünden, und beim Ausgangsknotengrad um die Anzahl der Kanten, die vom Knoten ausgehen. Die Summe aller ein- und ausgehenden Kanten bei einem gerichteten Graphen wird im Rahmen dieser Arbeit als Knotengrad $deg\,(v)$ bezeichnet.

2.3.2 Layout-based Clustering

Eine Möglichkeit einen Graphen unter Berücksichtigung der Anzahl der Kanten zwischen den einzelnen Knoten darzustellen ist das *Layout-based Clustering* [17]. Im Folgenden wird beschrieben, worum es sich dabei handelt und wie ein solches Clustering Layout berechnet werden kann.

Beim Clustering allgemein wird eine Menge an Elementen (z.B. Knoten eines Graphen) anhand bestimmter Kriterien in Teilmengen zerlegt. Beispielweise können die Knoten eines Graphen entsprechend der Anzahl ihrer Kanten gruppiert werden, indem sich Knoten, die stärker verknüpft sind, im gleichen Cluster befinden. Beim Layout-based Clustering werden zusätzlich Distanzen zwischen den einzelnen Elementen berechnet und den Elementen Positionen zugeordnet. Dabei

nehmen zusammenhängende Elemente nähere Positionen und Elemente, die keinen Bezug zueinander haben, entfernte Positionen ein. Dieses Clustering Layout bietet somit den Vorteil die Struktur eines Graphen mit seinen Knoten als Elementen intuitiv darzustellen.

Clustering Layouts können mithilfe des *Force-directed Graph Drawing* berechnet werden. Dabei wird ein Energiemodell eingesetzt, welches dem Layout einen Energiewert zuweist, wobei das Layout umso besser ist, je geringer der Energiewert ist. Darauf aufbauend wird ein Algorithmus verwendet, welcher ein Layout mit minimaler Energie berechnet.

Das Layout ist eine Funktion p, die jedem Knoten in der Ebene eine Position zuweist. Jedem Layout wird mittels des Energiemodells – einer Funktion $U(p)$ – ein Energiewert zugewiesen. Dieser Energiewert ist abhängig von den einzelnen Positionen der Knoten im Layout und der Anzahl deren Kanten. Das beste Layout bildet jenes Layout, dessen Energiewert $U(p)$ ein globales Minimum der Funktion U darstellt. Ein Optimierungsalgorithmus sucht dabei nach einer guten Näherung des besten Layouts. Dieser startet mit einem Layout, in welchem die Positionen der Elemente zufällig verteilt sind. Mit jeder Iteration wird versucht, dieses Layout zu verbessern. Dabei wird die erste Ableitung der Energiefunktion gebildet („force“), um eine Richtung („directed“) und einen Abstand für die Position jedes Elements im neuen Layout zu berechnen.

Abbildung 2.7 zeigt ein Beispiel eines Graphen, für welchen ein Clustering Layout berechnet wird. Das Ergebnis ist in Abbildung 2.8 dargestellt.

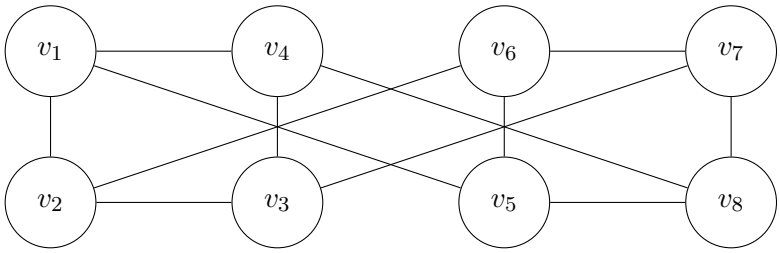

Abbildung 2.7: Beispiel eines Graphen ohne definiertem Layout.

2.4 Coupling-Metriken

Coupling-Metriken kommen im Software Engineering oft zum Einsatz, um Modularität sicherzustellen. Da Software-Systeme immer größer und komplexer werden, wird Modularität zunehmend wichtiger. Gute Modularität zeichnet sich durch hohe Kohäsion und niedrige Kopplung aus. Daher kommen entsprechende Metriken zum Einsatz, um die Eigenschaften quantitativ beurteilen zu können [18].

Im Folgenden wird der Begriff der Coupling-Metrik erörtert und es werden einige Coupling-Metriken vorgestellt und definiert.

2.4.1 Definition von Coupling

Die Struktur von Software beruht auf der *Kohäsion* innerhalb der einzelnen Software-Komponenten und der *Kopplung* der Software-Komponenten zueinander [19]. Die Kopplung und die

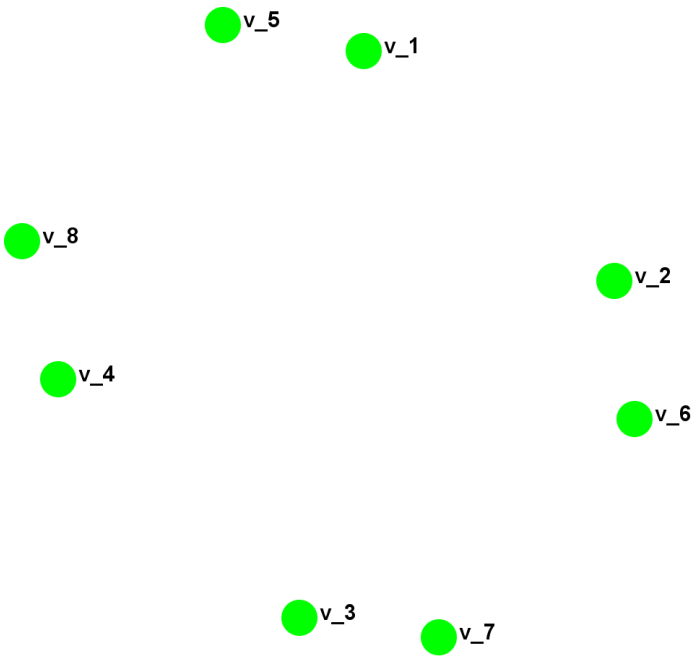

Abbildung 2.8: Clustering Layout des Graphen aus Abbildung 2.7.

Kohäsion entstehen durch Verbindungen bzw. Abhängigkeiten (z.B. Funktionsaufrufe, gemeinsame Verwendung von globalen Variablen, Message-Verbindungen) zwischen diesen Software-Komponenten (z.B. Modulen oder Klassen) [18].

Als Kohäsion („cohesion“) bezeichnet man ein Maß für den inneren Zusammenhalt einer Komponente. Die Kopplung („coupling“) definiert hingegen die Abhängigkeit zwischen einzelnen Komponenten und stellt ein Maß für die Schnittstellen zwischen den Komponenten dar [19], [20], [21].

In objektorientierter Software sind die einzelnen Komponenten die Klassen und Kopplung entsteht durch Vererbung, Assoziationen und Message Passing. Bei prozeduraler Software können beispielsweise Funktionen die einzelnen Komponenten darstellen. In diesem Fall kann Kopplung durch Funktionsaufrufe und über Speicherzugriffe, also durch das Lesen und Schreiben von Variablen im Speicher, entstehen.

2.4.2 Definition von Metriken

Metriken dienen grundsätzlich dazu, um eine Eigenschaft quantitativ zu beurteilen. Dabei stellt die Metrik eine Abbildung einer Eigenschaft auf eine Zahl dar [19]. Metriken helfen somit dabei, Eigenschaften zu quantifizieren. Mit Hilfe dieses Zahlenwertes ist es daher möglich, die gleiche Eigenschaft in unterschiedlichen Systemen zu vergleichen.

Eine Software-Metrik ist demnach ein Merkmal eines Softwaresystems, das gemessen werden kann. Einfache Beispiele für Software-Metriken sind etwa die Größe eines Produktes in Form von Lines of Code oder die zyklomatische Komplexität [3].

2.4.3 Coupling-Metriken

In Bezug auf obige Definitionen kann eine Coupling-Metrik als eine Abbildung von Kopplung auf Zahlenwerte definiert werden.

Eine klassische objektorientierte Metrik um Kopplungen zu messen ist die Metrik „Coupling between objects“ (CBO). Hier werden Klassen als gekoppelt angenommen, wenn Methoden in einer Klasse Methoden oder Attribute verwenden, die in einer anderen Klasse definiert wurden [3].

Kolesnikov et al. [15] präsentierten im Rahmen ihrer Studie über Feature-Interaktionen an Feature-orientierten Java-basierten Software-Produktlinien u.a. einige Coupling-Metriken. Im Folgenden werden zwei dieser Metriken erläutert und anhand von Graphen definiert (Abschnitte 2.4.3.1 und 2.4.3.2).

Kramer et al. [18] stellen in ihrer Arbeit über Coupling- und Cohesion-Metriken für wissensbasierte Systeme außerdem eine Coupling-Metrik für sogenannte „abstract frame systems“ bereit. Diese Metrik wird im Anschluss ebenfalls beschrieben und anhand eines Graphen definiert (Abschnitt 2.4.3.3).

2.4.3.1 Weighted Feature-Reference Graph Degree

Die Metrik „Weighted Feature-Reference Graph Degree“ ($RefW$) basiert auf einem Feature-Reference-Graphen. Der Feature-Reference-Graph ist ein Modell, welches Abhängigkeiten zwischen Features repräsentiert. Hierbei stellen die Knoten des Feature-Reference Graphen die Features und die Kanten die Referenzen bzw. die Abhängigkeiten zwischen den Features dar [15]. Kolesnikov et al. untersuchten in ihrer Studie ausschließlich Feature-orientierte Java-basierte Softwareproduktlinien. Daher werden in deren Arbeit die Referenzen durch Methodenaufrufe und Zugriffe auf Attribute und Typen dargestellt.

Bei dieser Metrik handelt es sich um eine Grad-basierte Metrik. Bei Grad-basierten Metriken wird grundsätzlich der Knotengrad des korrespondierenden Knoten, also die Anzahl seiner inzidenten Kanten, zur Berechnung herangezogen. Im Fall von $RefW$ werden jeweils alle Kanten, die zwischen zwei Features existieren, gezählt [15].

Die Metrik $RefW$ zwischen zwei Knoten v_x und v_y

$$RefW\left(v_x, v_y\right) = \left|\left\{E_{v_x} \cap E_{v_y}\right\}\right| \tag{2.10}$$

berechnet sich daher aus den jeweiligen Mengen an inzidenten Kanten zu den beiden Knoten v_x und v_y E_{v_x} und E_{v_y}. Es handelt sich bei $RefW$ daher um eine Metrik, die sich jeweils auf zwei Knoten bezieht.

Abbildung 2.9 zeigt beispielhaft einen einfachen Graphen mit drei Knoten und drei Kanten. In diesem Beispiel ergeben sich die Werte der Metrik zu $RefW\left(v_1, v_2\right) = 2$, $RefW\left(v_1, v_3\right) = 1$ und $RefW\left(v_2, v_3\right) = 0$ (vgl. Abbildung 2.10).

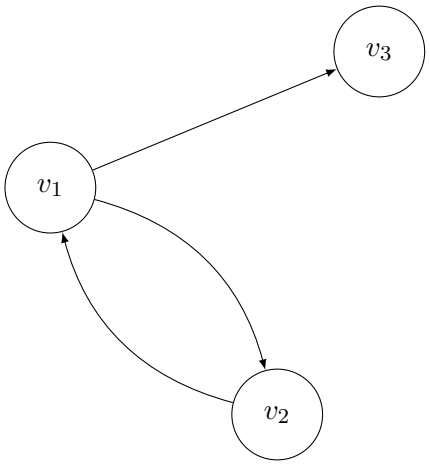

Abbildung 2.9: Einfaches Beispiel eines Graphen mit Knoten $V = \{v_1, ..., v_3\}$.

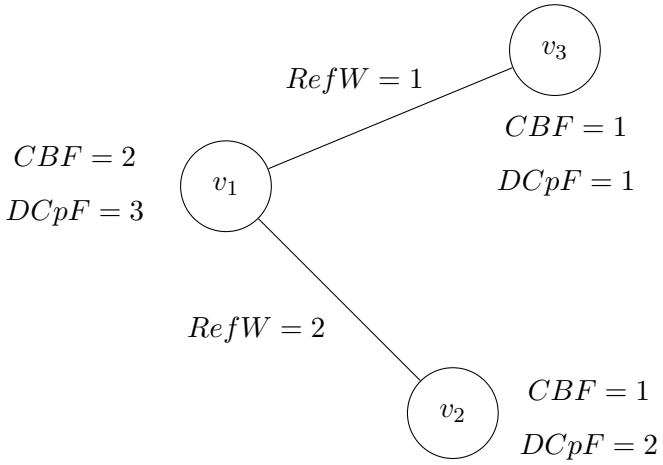

Abbildung 2.10: Ergebnisse der Metriken des Graphen aus Abbildung 2.9.

2.4.3.2 Coupling Between Features

Kolesnikov et al. definieren die Metrik „Coupling Between Features“ (CBF) als die Anzahl, zu wie vielen Features ein Feature gekoppelt ist. In ihrem Fall gelten zwei Features als gekoppelt, wenn ein Feature eine Methode des anderen Features aufruft oder auf ein Attribut eines Feature zugegriffen wird [15].

Betrachtet man die Features als Knoten und die Methoden- bzw. Attributzugriffe als Kanten in einem Graphen, lässt sich die Metrik CBF eines Knoten v_x

$$CBF\left(v_x\right) = \left|V_{v_x}\right| \tag{2.11}$$

aus der Menge an adjazenten Knoten zum Knoten v_x V_{v_x} berechnen. Im Gegensatz zu $RefW$ handelt es sich bei CBF daher um eine Metrik, die sich jeweils nur auf einen (und nicht auf zwei) Knoten bezieht.

Am Beispiel des Graphen aus Abbildung 2.9 ergeben sich die Werte der Metrik zu $CBF\left(v_1\right) = 2$, $CBF\left(v_2\right) = 1$ und $CBF\left(v_3\right) = 1$ (vgl. Abbildung 2.10).

2.4.3.3 Degree of Coupling of Frame

Kramer et al. präsentierten in [18] erste Coupling- und Cohesion-Metriken für sogenannte wissensbasierte Systeme („knowledge-based systems“). Bei diesen Systemen handelt es sich um Computer-Systeme, die Wissen repräsentieren, um beispielsweise eine bestimmte Tätigkeit ausführen zu können [22]. Anschließend definierten sie die gleichen Metriken ebenfalls für sogenannte „abstract frame systems“ (AFS). Eine dieser Metriken für AFS aus dieser Arbeit – die Coupling-Metrik „Degree of Coupling of Frame“ ($DCpF$) – wird im Folgenden beschrieben und anhand eines Graphen definiert.

Ein AFS besteht aus Elementen, Beziehungen und Frames. Ein Frame beinhaltet ein oder mehrere Elemente. Dabei können sowohl Elemente innerhalb eines Frames als auch Elemente aus unterschiedlichen Frames miteinander in Beziehung stehen. Die Metrik $DCpF$ ist definiert als die Anzahl aller Beziehungen zwischen den Elementen eines Frames und den Elementen anderer Frames.

Bildet man die Frames als Knoten und die Beziehungen zwischen den Elementen unterschiedlicher Frames als Kanten ab, kann die Metrik $DCpF$ eines Knoten v_x

$$DCpF\left(v_x\right) = \left|E_{v_x}\right| = deg\left(v_x\right) \tag{2.12}$$

anhand des Knotengrades definiert werden. Ebenso wie CBF ist $DCpF$ eine Metrik, die sich jeweils auf einen Knoten bezieht. Wie in Abbildung 2.10 ersichtlich ist, ergeben sich die Werte der Metrik $DCpF$ am Beispiel des Graphen aus Abbildung 2.9 zu $DCpF\left(v_1\right) = 3$, $DCpF\left(v_2\right) = 2$ und $DCpF\left(v_3\right) = 1$.

3 Anwendung von Coupling-Metriken auf Motorsteuergeräte-Software zur Erkennung von Feature-Interaktionen

In diesem Kapitel wird der Lösungsansatz präsentiert, mit welchem es möglich ist, Coupling-Metriken auf Features der Motorsteuergeräte-Software anzuwenden, um darin enthaltene potentielle Feature-Interaktionen zu erkennen. Dabei wird zuerst darauf eingegangen, wie Coupling-Metriken auf die vorliegende Motorsteuergeräte-Software angewendet werden können. Anschließend wird beschrieben, wie die Coupling-Metriken dazu genutzt werden können, potentielle Feature-Interaktionen zu erkennen.

3.1 Anwendung von Coupling-Metriken auf Motorsteuergeräte-Software

In diesem Abschnitt wird zunächst erläutert, wie Features in der Motorsteuergeräte-Software identifiziert werden können. Anschließend wird auf die Notwendigkeit der Berücksichtigung sogenannter Koordinatoren eingegangen. Zudem wird beschrieben, wie Kopplung in der Motorsteuergeräte-Software entsteht. Schließlich wird erklärt, wie die Struktur der Motorsteuergeräte-Software mittels eines Graphen repräsentiert werden kann.

3.1.1 Features in der Motorsteuergeräte-Software

Zu Beginn wird die für diese Arbeit bereitgestellte Motorsteuergeräte-Software auf vorhandene Features untersucht. Dieser Vorgang gestaltet sich als nicht trivial, da die Motorsteuergeräte-Software nicht Feature-orientiert aufgebaut ist. Die Software orientiert sich am physikalischen Aufbau des Verbrennungsmotors (vgl. Kapitel 2.1.3), da einzelne Software-Teile besser den physikalischen Prozessen und den entsprechenden Sensoren und Aktoren zugeordnet werden können [7], [8]. Daher ist zu erwarten, dass die Features in der Software verteilt sind und sich jedenfalls in unterschiedlichen FCs, möglicherweise sogar in verschiedenen BCs oder Subsystemen befinden. Abbildung 3.1 zeigt die mögliche Verteilung der Features in der Motorsteuergeräte-Software. Features stellen somit ein *Cross-Cutting Concern* in Bezug auf das Prinzip der *Separation of Concerns* dar, da diese die am Motoraufbau orientierte Software-Struktur „querschneidet“ [3].

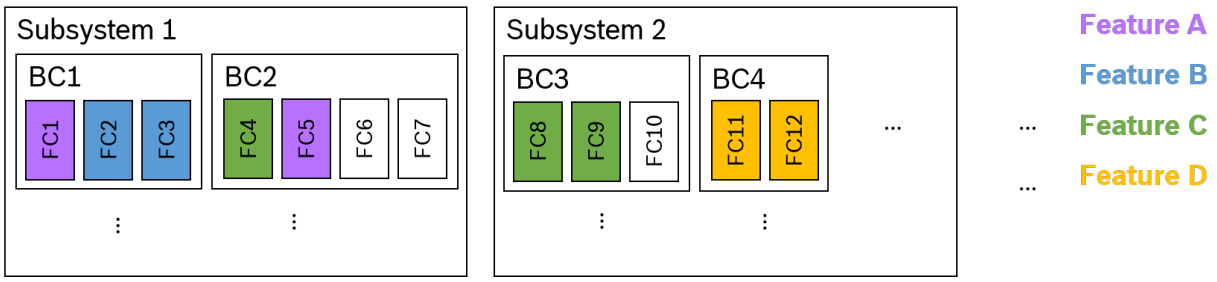

Abbildung 3.1: Mögliche Verteilung der Features in der Motorsteuergeräte-Software.

Mit Hilfe der Dokumentation wurden daher manuell Features – entsprechend der Definition aus Abschnitt 2.2.1.1 – in der vorliegenden Motorsteuergeräte-Software identifiziert. Dabei wurden sämtliche FCs der Software betrachtet und potentiell vorhandene Features ermittelt. Um die Kopplung zwischen Features bestimmen zu können, müssen die Features den entsprechenden Software-Teilen zugeordnet werden. Diese Zuordnung zwischen Features und FCs wird auf elementarster Strukturebene – also auf der Ebene von FCs – vorgenommen. Abbildung 3.2 stellt die Zuordnung zwischen Features und FCs anhand des Beispiels in Abbildung 3.1 grafisch dar.

Feature A	Feature B	Feature C	Feature D
FC1	FC2	FC4	FC11
FC5	FC3	FC8	FC12
		FC9	

Abbildung 3.2: Zuordnung von Features zu den FCs anhand des Beispiels aus Abbildung 3.1.

3.1.2 Koordinatoren

Wie in Abschnitt 2.2.2.2 anhand einiger Beispiele beschrieben, müssen Features im Falle einer Feature-Interaktion entsprechend koordiniert werden. Dies kann, je nach Aufwand dieser Koordination, direkt innerhalb einer Funktion des jeweiligen Features, oder auch in einem separaten Koordinator-Modul implementiert sein.

Ein Koordinator im Kontext dieser Arbeit kann definiert werden als ein Modul, in welchem Sollwerte anhand fester Regeln zusammengeführt („koordiniert“) werden. Abbildung 3.3 stellt einen Koordinator K dar, der einen Sollwert auf Basis von Sollwertanforderungen zweier Features $F1$ und $F2$ koordiniert. Der Koordinator liest dabei die Sollwerte zu Beginn ein. Er kann diese dann koordinieren, indem er – je nach Bedeutung dieser Sollwertanforderungen – beispielsweise Prioritäten an die einzelnen Features vergibt oder den Minimal- bzw. Maximalwert aus ihnen bildet. Als Ausgangswert entsteht ein koordinierter Sollwert, der anschließend entweder zur Weiterverarbeitung in der Software oder zur direkten Ansteuerung eines Aktors benutzt wird. Ein Beispiel aus der Praxis für einen solchen Sachverhalt sind beispielsweise unterschiedliche Drehmomentenanforderungen zweier Features, etwa die Anforderung vom Fahrpedal und die vom ESP-Steuergerät, welche über einen Momentenkoordinator zusammengeführt werden.

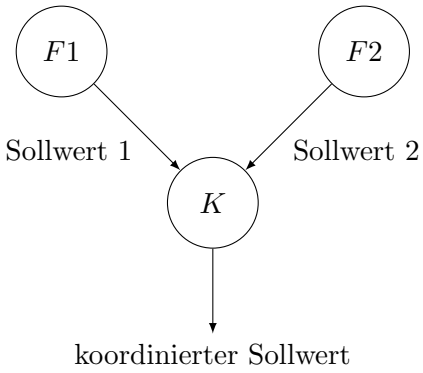

Abbildung 3.3: Koordination von zwei Sollwerten der beiden Features $F1$ und $F2$ durch einen Koordinator K.

In der Motorsteuergeräte-Software befinden sich solche Koordinator-Module, jeweils in einer oder in mehreren FCs implementiert. Da Feature-Interaktionen offensichtlich über diese Koordinatoren passieren können, werden in weiterer Folge auch Koordinatoren in der Motorsteuergeräte-Software identifiziert. Genau wie bei der Zuordnung von Features zu FCs werden auch hier die Koordinatoren ihren entsprechenden FCs zugeordnet.

Abbildung 3.4 zeigt die mögliche Position eines Koordinators anhand des Beispiels aus Abbildung 3.1. Die Feature-Zuordnung aus Abbildung 3.2 wird dabei um die entsprechende Zuordnung des Koordinators zu den FCs in Abbildung 3.5 ergänzt.

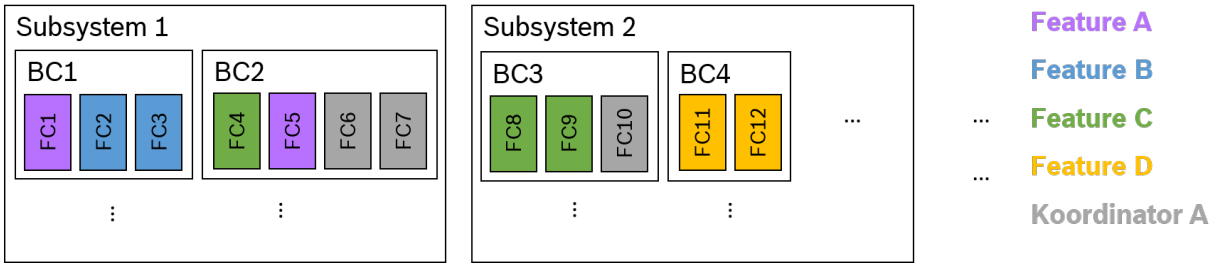

Abbildung 3.4: Mögliche Verteilung der Features und Koordinatoren in der Motorsteuergeräte-Software.

Feature A	Feature B	Feature C	Feature D	Koordinator A
FC1	FC2	FC4	FC11	FC6
FC5	FC3	FC8	FC12	FC7
		FC9		FC10

Abbildung 3.5: Zuordnung von Features und Koordinatoren zu den FCs anhand des Beispiels aus Abbildung 3.4.

3.1.3 Kopplung in der Motorsteuergeräte-Software

Im nächsten Schritt muss die Kopplung zwischen den unterschiedlichen Komponenten in der Motorsteuergeräte-Software bestimmt werden. Wie in Abschnitt 2.4.1 bereits beschrieben, entsteht Kopplung bei objektorientierter Software durch Vererbung, Assoziationen und Message Passing. Bei prozeduraler Software hingegen entsteht Kopplung durch Funktionsaufrufe und Speicherzugriffe. Im Rahmen dieser Arbeit wird die in C realisierte Software eines Motorsteuergeräts untersucht. Es handelt sich daher hierbei nicht um objektorientierte, sondern prozedurale Software.

Bei der Motorsteuergeräte-Software sind die FCs in Funktionen implementiert. Daher werden die FCs als die einzelnen Komponenten betrachtet. Kopplungen könnten dementsprechend grundsätzlich durch Funktionsaufrufe und über Speicherzugriffe, also durch das Lesen und Schreiben von Variablen im Speicher, entstehen.

Da bei der Motorsteuergeräte-Software die Funktionen in einer festen Reihenfolge in ihren jeweiligen Zeitraster aufgerufen werden (vgl. Abschnitt 2.1.3), entstehen hier keine Kopplungen durch gegenseitige Funktionsaufrufe. Jede Funktion liest jedoch gewisse Größen bzw. Variablen vom und in den Speicher – die sogenannten Eingangs- bzw. Ausgangsvariablen (vgl. Abbildung 2.3). In der vorliegenden Software entsteht die Kopplung daher ausschließlich über diese Ein- bzw. Ausgangsvariablen. Abbildung 3.6 zeigt die Kommunikation in der Motorsteuergeräte-Software über diese Variablen anhand des Beispiels aus Abbildung 3.4.

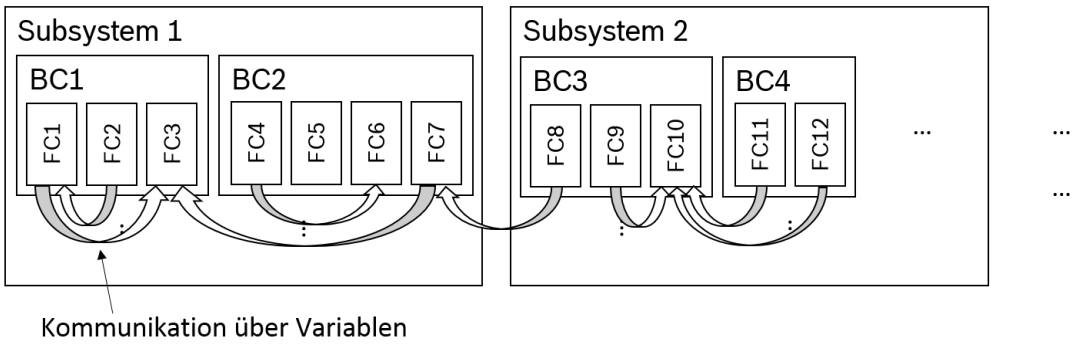

Abbildung 3.6: Kommunikation in der Motorsteuergeräte-Software über Variablen anhand des Beispiels aus Abbildung 3.4.

Wird eine Variable von einer FC gelesen, die von einer anderen geschrieben wurde, bedeutet dies, dass die lesende FC von der schreibenden abhängig ist. Dies stellt eine sogenannte Schreib-/Leseverbindung dar. In diesem Fall ist eine Ausgangsvariable einer FC gleich einer Eingangsvariable einer anderen. Solche Schreib-/Leseverbindungen stellen daher eine Abhängigkeit dar. Abhängigkeiten zwischen den untersuchten Features oder Koordinatoren entstehen in der vorliegenden Motorsteuergeräte-Software ausschließlich über diese Art von Schreib-/Leseverbindungen. Schreib-/Schreibverbindungen existieren in dieser Software zwischen den betrachteten Features nicht. Lese-/Leseverbindungen existieren, allerdings resultieren diese nicht in Feature-Interaktionen und werden daher nicht als Abhängigkeit gewertet.

Die Kopplung zwischen den Features und den Koordinatoren in der Motorsteuergeräte-Software wird durch diese Abhängigkeiten bestimmt. Je höher nun die Anzahl dieser Abhängigkeiten zwischen zwei Elementen ist, desto höher ist ihre Kopplung.

An dieser Stelle sei noch anzumerken, dass bei der Festlegung der Reihenfolge der Aufrufe der Funktionen der einzelnen FCs die Schreib- und Lesereihenfolge der einzelnen Variablen zu beachten ist. Bei der Entwicklung der Motorsteuergeräte-Software wird dies entsprechend berücksichtigt.

3.1.4 Repräsentation der Motorsteuergeräte-Software als Graph

Als Grundlage für die Berechnungen der Coupling-Metriken dient ein Graph. Daher wird die Struktur der Software in einem Graphen abgebildet. Dabei werden sowohl die Features als auch die Koordinatoren als Knoten des Graphen dargestellt. Die Kanten entstehen durch die einzelnen Abhängigkeiten. Existiert eine Schreib-/Leseverbindung, wie in Abschnitt 3.1.3 beschrieben, entsteht eine Abhängigkeit. Sind diese beiden FCs nun unterschiedlichen Features oder Koordinatoren zugeordnet, besteht diese Abhängigkeit auch zwischen den Features bzw. den Koordinatoren. Die Abhängigkeit wird daher als Kante zwischen den zwei Knoten, die die Features bzw. die Koordinatoren repräsentieren, übernommen. Da diese Abhängigkeiten durch den Schreib-/Lesezusammenhang eine Richtung besitzen, werden diese durch gerichtete Kanten dargestellt.

Auf diese Weise entstehen eine Vielzahl an Kanten im Graphen zwischen den Knoten, die die Features bzw. die Koordinatoren repräsentieren. Aufgrund der Tatsache, dass oftmals Features oder Koordinatoren etwa mehrere Variablen aus einem anderen Feature oder einem Koordinator einlesen, existieren dadurch mehrere Abhängigkeiten zwischen ihnen. Um diese Information über die Quantität der Abhängigkeiten im Graphen ausdrücken zu können, sind Mehrfachkanten erlaubt. Außerdem werden manchmal Variablen von einem Feature eingelesen, von welchem diese auch geschrieben werden. Dadurch würde eine Abhängigkeit eines Features zu sich selbst entstehen. Da im Rahmen dieser Arbeit nur die Kopplung zwischen Features relevant ist, werden solche Selbstreferenzen im Graphen nicht abgebildet.

Auf diesen Graphen können in weiterer Folge die Coupling-Metriken entsprechend ihrer Definitionen aus Abschnitt 2.4.3 angewendet werden. Außerdem kann mittels dieses Graphen die Struktur der Features visualisiert werden. Abbildung 3.7 zeigt einen solchen Graphen. Als Grundlage für dieses Beispiel dient die Verteilung der Features und Koordinatoren aus Abbildung 3.4 und Abhängigkeiten aus Abbildung 3.6.

Die beschriebenen Arbeitsschritte, beginnend mit der Bestimmung der Ein- und Ausgangsvariablen, gefolgt von der Ermittlung der Abhängigkeiten und Erstellung des Graphen bis hin zur Berechnung der Coupling-Metriken und schließlich der Erstellung der Visualisierungen, werden Tool-unterstützt durchgeführt. Dazu wurde eine eigene Tool-Chain entwickelt. Weitere Details dazu werden in Kapitel 4 erläutert.

An dieser Stelle sei noch anzumerken, dass die Repräsentation der Struktur der Motorsteuergeräte-Software in Form eines Graphen zwar die Richtung der Abhängigkeiten abbildet, diese Information von diesen Metriken allerdings nicht verwertet wird.

3.2 Erkennung von Feature-Interaktionen mittels Coupling-Metriken

In diesem Abschnitt wird zu Beginn darauf eingegangen, wie Feature-Interaktionen kategorisiert werden können. Anschließend wird beschrieben, Feature-Interaktionen welcher Kategorie mittels Coupling-Metriken potentiell erkannt werden können und welche Informationen Coupling-Metriken im Hinblick auf Feature-Interaktionen liefern können.

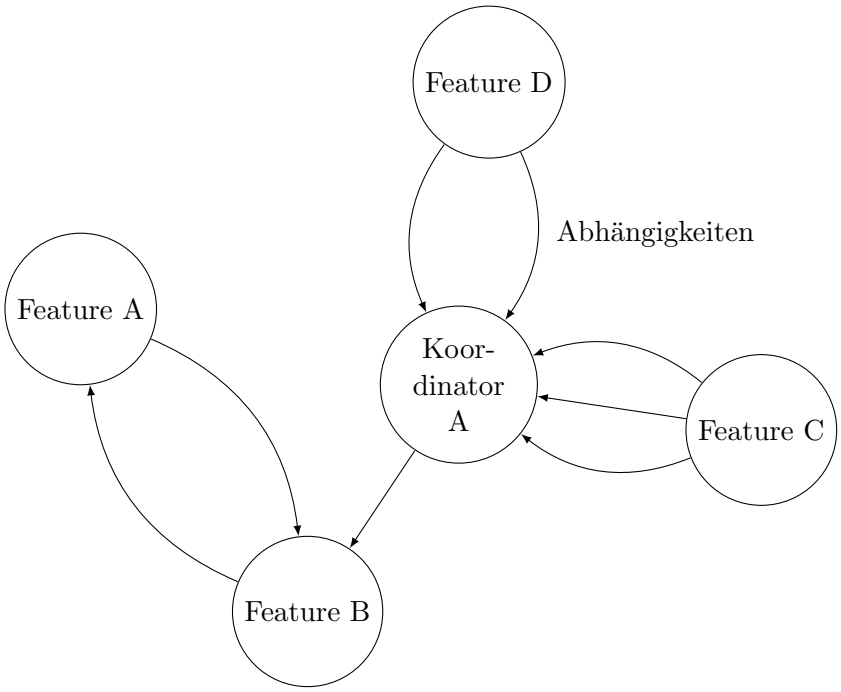

Abbildung 3.7: Gerichteter Graph mit Features bzw. Koordinatoren als Knoten und ihren Abhängigkeiten über Speicherzugriffe als Kanten anhand des Beispiels aus Abbildung 3.4 und 3.6.

Coupling-Metriken kommen zum Einsatz, um die Eigenschaft der Kopplung quantitativ beurteilen zu können. Dies hat oft den Hintergrund, dass die Modularität in der Software sichergestellt werden soll [18]. Im Gegensatz dazu werden im Rahmen dieser Arbeit Coupling-Metriken dazu verwendet, um mögliche Feature-Interaktionen zu erkennen.

Diese Arbeit beschäftigt sich nicht mit der Definition neuer Metriken, sondern es werden bereits bestehende Coupling-Metriken verwendet. Weiterführende theoretische Arbeiten zu Coupling-Metriken sind beispielsweise [18] und [23].

Kolesnikov et al. [15] präsentierten im Rahmen ihrer Studie über Feature-Interaktionen an Feature-orientierten Java-basierten Software-Produktlinien u.a. einige Coupling-Metriken. Zwei dieser Metriken werden im Rahmen dieser Arbeit verwendet. Die verwendeten Metriken wurden also bereits in objektorientierten Software-Produktlinien angewendet, werden nun hingegen in einem Echtzeit-Softwaresystem, welches in C realisiert ist, eingesetzt.

3.2.1 Kategorisierung von Feature-Interaktionen

In Abschnitt 2.2.2.1 wird die Feature-Interaktion ganz allgemein als Beeinflussung eines Features durch ein anderes Feature definiert. Anhand der Beispiele für Feature-Interaktionen (vgl. Abschnitt 2.2.2.2) kann man erkennen, dass diese oftmals entstehen, wenn unterschiedliche Features (beispielsweise der Tempomat und der Abstandhalteassistent) den Sollwert für die gleiche Größe (im Falle von Tempomat und Abstandhalteassistent das Drehmoment) vorgeben. Feature-Interaktionen können auch auf andere Art und Weise hervorgerufen werden. Im Folgenden werden vier mögliche Arten der Feature-Interaktion beschrieben (vgl. [8]).

(1) Beeinflussung der Bereitschaft eines Features: Die Bereitschaft eines Features wird durch ein anderes beeinflusst, indem das Feature aktiviert bzw. deaktiviert wird. Beispielsweise könnte das Feature Katalysatorheizen das Start/Stopp-Feature deaktivieren, wenn dieses selbst aktiv ist, um das Abkühlen des Katalysators unter seine minimal notwendige Betriebstemperatur zu vermeiden.

(2) Beeinflussung des Zustandes bzw. des Sollwerts eines Features: Der Zustand bzw. der Sollwert eines Features werden beeinflusst, indem dieses Feature Rechen- bzw. Steuergrößen von einem anderen Features einliest. So kann etwa die Klopfregelung die Zündungsregelung beeinflussen, indem erstere bei erkanntem Klopfen im Motor der Zündregelung eine Verstellung des Zündwinkels in Richtung spät durch Übergabe eines Minimalzündwinkels vorgibt.

(3) Beeinflussung des Sollwertes der gleichen Größe: Wie oben beschrieben, entsteht eine Beeinflussung zwischen zwei Features oftmals, wenn diese beiden Features den Sollwert der gleichen Größe vorgeben wollen.

(4) Beeinflussung auf physikalischer Ebene: Features können auf physikalischer Ebene durch andere Features beeinflusst werden, da das Motorsteuergerät als eingebettetes System auf physikalischer Ebene mit der Umwelt interagiert. Beispielsweise wird die Luftmasse in der Verbrennungskammer von mehreren Aktoren beeinflusst, wie der Drosselklappe, dem Turbolader oder dem Nockenwellensteller. Über diese physikalische Kopplung kann schließlich eine Interaktion zwischen Features entstehen.

Dabei wird im Rahmen dieser Arbeit die vierte Art der Feature-Interaktion – die Beeinflussung auf physikalischer Ebene – nicht berücksichtigt, da nur Kopplung in der Software, also auf Codeebene, betrachtet wird.

Sämtliche Möglichkeiten der Beeinflussung – auch jene auf physikalischer Ebene – werden bei der Entwicklung von Motorsteuergeräte-Software berücksichtigt und entsprechend koordiniert.

Feature-Interaktionen können außerdem in verschiedenen Konstellationen auftreten. Einerseits können Feature-Interaktionen entstehen, indem sich zwei Features direkt über den Austausch von Variablen beeinflussen, also im Graphen Abhängigkeiten besitzen. Andererseits kann die Konstellation auftreten, in der ein Koordinator eine Feature-Interaktion zwischen zwei Features koordiniert (vgl. Abschnitt 3.1.2). Folglich können im Allgemeinen Feature-Interaktionen anhand der beiden Konstellationen in Abbildung 3.8 und Abbildung 3.9 beschrieben werden. Abbildung 3.8 zeigt jene Konstellation, in der sich zwei Features über Abhängigkeiten direkt beeinflussen können. Abbildung 3.9 zeigt eine andere Konstellation mit zwei Features und einem Koordinator, wobei der Koordinator die Interaktion zwischen den beiden Features auflöst.

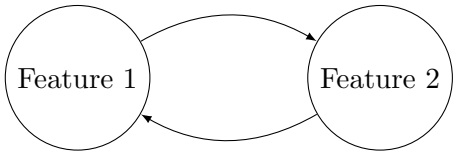

Abbildung 3.8: Konstellation, in der sich zwei Features über Abhängigkeiten direkt beeinflussen können.

In weiterer Folge können die Kategorien 1–3 diesen Konstellationen zugeordnet werden. Kategorie 1 und 2 können in der Konstellation aus Abbildung 3.8 auftreten, da sich diese beiden

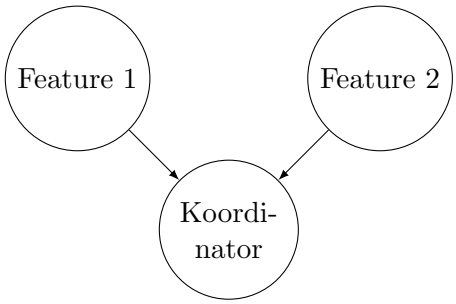

Abbildung 3.9: Konstellation, in der ein Koordinator zwei Features koordiniert und deren Interaktion auflöst.

Features sowohl hinsichtlich ihrer Bereitschaft als auch ihres Zustandes bzw. ihres Sollwerts beeinflussen können. Kategorie 3 hingegen kann in der Konstellation aus Abbildung 3.9 auftreten. Dabei können zwei Features Sollwerte der gleichen Größe beim Koordinator anfordern, der diesen Konflikt auflöst, indem er die Sollwerte entsprechend seiner Regeln koordiniert (vgl. Abschnitt 3.1.2).

3.2.2 Interpretation der Ergebnisse der Coupling-Metriken bei deren Anwendung

Liegt die Struktur der Motorsteuergeräte-Software als Graph mit den Features bzw. den Koordinatoren als Knoten und den Abhängigkeiten als Kanten vor (vgl. Abschnitt 3.1.4), können auf diesem die Coupling-Metriken angewendet werden. Dazu werden die Coupling-Metriken $RefW$, CBF und $DCpF$ entsprechend ihrer Definitionen aus Abschnitt 2.4.3 verwendet.

Die drei Metriken $RefW$, CBF und $DCpF$ unterscheiden sich dahingehend, auf wie viele Knoten sie sich jeweils beziehen. Während $RefW$ eine Metrik darstellt, die immer zwischen zwei Knoten berechnet wird, handelt es sich bei CBF und $DCpF$ um Metriken, die jeweils auf einen Knoten Bezug nehmen. Dementsprechend werden die Ergebnisse der Metriken unterschiedlich im Hinblick auf ihre Interpretation behandelt.

3.2.2.1 RefW

$RefW$ bezieht sich jeweils auf Paare von Features. Sie beschreibt daher, wie stark bestimmte Paare von Features miteinander gekoppelt sind. Folglich sollte sie darüber Auskunft geben, ob zwischen ihnen eine Interaktion besteht. Dies bezieht sich damit auf diejenigen Feature-Interaktionen, die in einer Konstellation wie in Abbildung 3.8 auftreten. Daher sollte $RefW$ Aufschluss über Interaktionen der Kategorie 1 oder 2 geben. Dabei ist zu erwarten, dass zwischen zwei Features f_1 und f_2 eine Interaktion vorhanden ist, wenn

$$RefW\,(f_1, f_2) \geq 1. \tag{3.1}$$

Diese Annahme ergibt sich empirisch. Sie entsteht beispielsweise aus der Tatsache, dass Features oft Statusvariablen anderer Features einlesen und in weiterer Folge diese eine Abhängigkeit in Form einer eingelesenen Statusvariable ausreicht, um das Feature zu deaktivieren (Feature-Interaktion der Kategorie 1).

Außerdem kann $RefW$ Auskunft darüber geben, ob und in welchem Ausmaß Abhängigkeiten zwischen einem Koordinator und einem Feature vorhanden sind. Existieren mehrere Features, die Abhängigkeiten zu einem Koordinator besitzen, besteht die Möglichkeit einer Feature-Interaktion entsprechend der Konstellation in Abbildung 3.9. Es könnte sich damit bei diesen Abhängigkeiten um Sollwertvorgaben für die gleiche Größe der beiden Features an den Koordinator handeln. Daher sollte $RefW$ einen Hinweis auf eine Interaktion der Kategorie 3 zwischen diesen Features liefern. Die Metrik wird jedoch keine Aussage über das tatsächliche Vorhandensein einer Feature-Interaktion treffen können, da allein aufgrund des Ergebnisses der Metrik nicht festgestellt werden kann, ob es sich dabei tatsächlich um Sollwerte der gleichen Größe handelt. Dies müsste in weiterer Folge manuell analysiert werden.

3.2.2.2 CBF und DCpF

CBF und $DCpF$ beziehen sich jeweils nur auf einen Knoten in Form eines Features oder eines Koordinators. Daher gibt diese Metriken an, wie stark ein Feature oder ein Koordinator generell mit anderen Features gekoppelt ist.

Dabei beschreibt der Wert von CBF, wieviele andere Knoten zu einem Knoten adjazent sind, also zu wievielen anderen Knoten grundsätzlich Verbindungen bestehen. Die exakte Anzahl, um wie viele Kanten es sich dabei insgesamt handelt, gibt der Wert von $DCpF$ an. Dabei gilt

$$DCpF\,(v_x) \geq CBF\,(v_x) \qquad \forall v_x \in V, \tag{3.2}$$

da die Anzahl an inzidenten Kanten immer mindestens genau so groß sein muss wie die Anzahl an adjazenten Knoten (vgl. Abschnitt 2.3.1).

Abbildung 3.10 zeigt ein Diagramm, in welchem Features und Koordinatoren abhängig von CBF und $DCpF$ aufgetragen werden können. Entsprechend der Formel (3.2) existiert ein nicht zulässiger Bereich. Im übrigen zulässigen Bereich des Diagramms können drei Extremfälle auftreten, welche im Folgenden beschrieben werden.

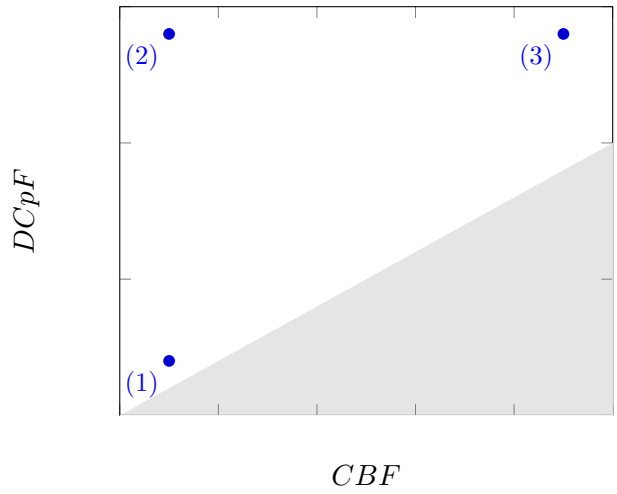

Abbildung 3.10: Diagramm für Features bzw. Koordinatoren entsprechend ihrer Werte für CBF und $DCpF$ mit drei Extremfällen (der nicht zulässige Bereich ist in grau dargestellt).

- **Fall 1:** In diesem Fall sind die Werte beider Metriken $CBF(v_x) = DCpF(v_x) = 1$. Das bedeutet, dass der Knoten v_x nur genau eine Abhängigkeit zu einem anderen Knoten aufweist, also keine hohe Kopplung aufweist. Ein korrespondierendes Feature würde daher potentiell eine einzige Feature-Interaktion zu einem anderen Feature aufweisen (vgl. Abschnitt 3.2.2.1).

- **Fall 2:** Hier ist der Wert der Metrik $CBF(v_x) = 1$ und $DCpF(v_x) \gg 1$. Dies besagt, dass der Knoten v_x nur Abhängigkeiten zu genau einem anderen Knoten besitzt, es sich aber um eine hohe Anzahl an Abhängigkeiten handelt. Die Kopplung zwischen diesen beiden Knoten ist demnach hoch, Kopplungen zu anderen Knoten hingegen existieren nicht. Feature-Interaktionen zwischen korrespondierenden Features können entsprechend auftreten.

- **Fall 3:** Im letzten Extremfall weisen beide Metriken hohe Werte auf: $CBF(v_x) \gg 1$ und $DCpF(v_x) \gg 1$. Es handelt bei diesem Knoten v_x um einen sogenannten *Hotspot*. Dieser kennzeichnet sich dadurch, dass er viele Abhängigkeiten zu vielen anderen Knoten besitzt. Handelt es sich bei diesem Hotspot um ein Feature, können Interaktionen zu den anderen Features, zu denen Abhängigkeiten vorhanden sind, bestehen. Stellt der Knoten einen Koordinator dar, kann hier wiederum allein aufgrund der Ergebnisse der Metriken keine Aussage getroffen werden. Der Koordinator könnte etwa von vielen Features die Sollwerte der gleichen Größe einlesen, womit eine Feature-Interaktion der Kategorie 3 in der Konstellation aus Abbildung 3.9 bestünde. Ob es sich dabei tatsächlich um Sollwerte der gleichen Größe handelt, kann mit Hilfe der Metriken alleine nicht festgestellt werden.

4 Entwicklung einer Tool-Chain zur Anwendung von Coupling-Metriken

In diesem Kapitel wird die konkrete Umsetzung der Tool-Chain zur Anwendung von Coupling-Metriken beschrieben. Dabei werden zu Beginn die Anforderungen an die Tool-Chain erörtert und zusammengefasst. Anschließend wird überblicksartig auf das grobe Konzept der Tool-Chain eingegangen, bevor die einzelnen Teile der Tool-Chain beschrieben werden.

4.1 Anforderungen an die Tool-Chain

Die Anforderungen an die Tool-Chain ergeben sich aus dem Lösungskonzept (vgl. Kapitel 3).

Der primäre Zweck der Tool-Chain ist es, die Anwendung und Auswertung der Coupling-Metriken auf die Features der Motorsteuergeräte-Software zu unterstützen und die Visualisierung bereitzustellen. Konkret sollte dies für die in Abschnitt 2.4.3 beschriebenen Metriken $RefW$, CBF und $DCpF$ möglich sein. Wie Abbildung 4.1 zeigt, ist es dazu im ersten Schritt notwendig, die Struktur der Motorsteuergeräte-Software statisch zu ermitteln, indem die Software auf Abhängigkeiten analysiert wird. Diese Struktur sollte dann in Form eines Graphen gespeichert werden, um später darauf zugreifen zu können. Darauf aufbauend sollte es möglich sein, die Coupling-Metriken auf die Features in dieser Struktur anzuwenden, die Ergebnisse auszuwerten sowie die Visualisierungen erstellen.

4.2 Konzept der Tool-Chain

Ausgehend von den Anforderungen an die Tool-Chain wurde ein Konzept ausgearbeitet, um diese Anforderungen zu erfüllen. Das Konzept der Tool-Chain ist in Abbildung 4.2 dargestellt.

Den Input für die Tool-Chain bilden die Motorsteuergeräte-Software selbst in Form des Maschinencodes sowie die Zuordnung von Features und Koordinatoren zu den einzelnen FCs (vgl. Abschnitte 3.1.1 und 3.1.2). Die Implementierung der Feature- und Koordinator-Zuordnung wird in Abschnitt 4.2.1 erläutert.

Wie in Kapitel 4.1 erwähnt, muss die Software zu Beginn auf ihre Abhängigkeiten hin untersucht werden. Grundsätzlich existieren viele Tools zur statischen Programmanalyse, die es ermöglichen

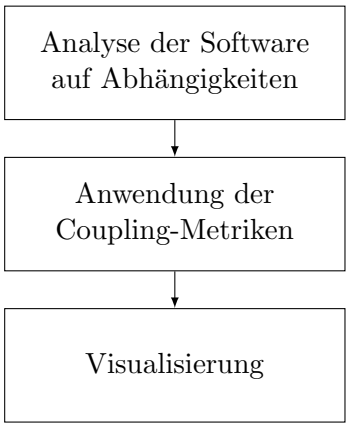

Abbildung 4.1: Aufgaben der Tool-Chain.

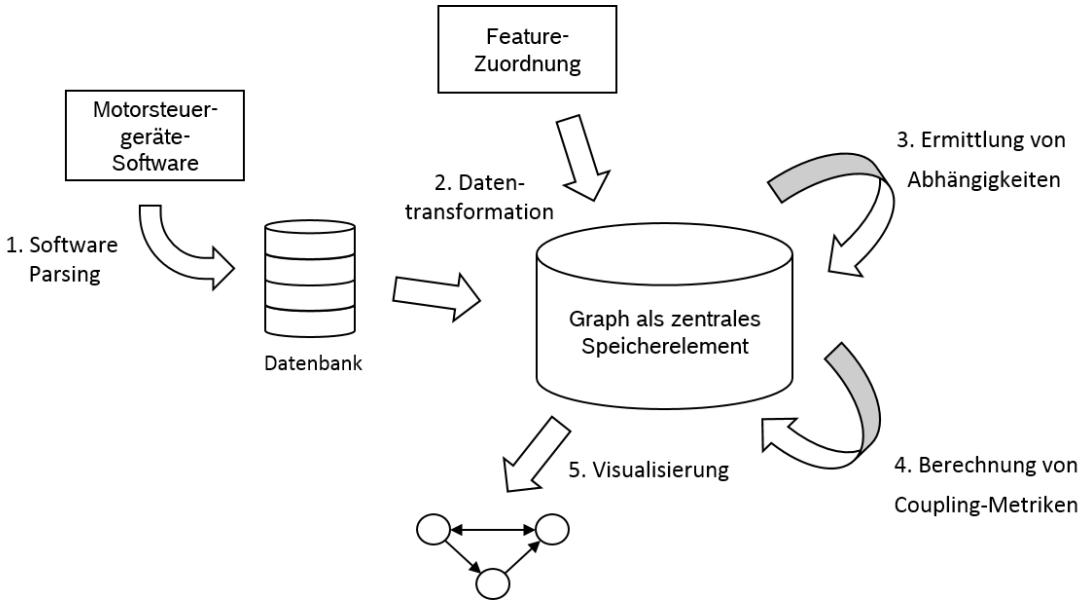

Abbildung 4.2: Konzept der Tool-Chain.

Informationen aus einem Quellcode zu extrahieren, Beispiele hierfür sind Doxygen und Fuji. Kolesnikov et al. verwendeten in ihrer Studie über Feature-Interaktionen an Feature-orientierten Software-Produktlinien beispielsweise Fuji als Werkzeug, um Informationen über die Referenzen zwischen den einzelnen Programmelementen zu sammeln [15]. Eine Übersicht über weitere Analysetools für Software-Produktlinien wird in [24] gegeben.

Im Rahmen dieser Arbeit wird jedoch für den ersten Teil des Analysierens der Software ein proprietäres Tool verwendet, welches bereits an den Aufbau der Motorsteuergeräte-Software angepasst ist. Es erlaubt das Parsing der Software und legt anschließend u.a. die Funktionen, Variablen und Variablenzugriffe in Form einer relationalen Datenbank – einer SQLlight-Datenbank – ab (siehe Abbildung 4.2). Der Schritt des Software-Parsings wird in Kapitel 4.2.2 genauer beschrieben.

Anschließend werden diese Datenbank und die Feature- bzw. Koordinator-Zuordnung mithilfe

eines selbst entwickelten Tools in einen Graphen, welcher als zentrales Speicherelement dient, transformiert. Die Implementierung des Graphen als zentrales Speicherelement wird in Abschnitt 4.2.3 genauer erläutert, die Datentransformation in Abschnitt 4.2.4.

Anschließend werden die Abhängigkeiten im Graphen ermittelt (siehe Abschnitt 4.2.5). Danach können die Coupling-Metriken schließlich anhand dieses Graphen berechnet und die Ergebnisse in eine Datei geschrieben werden. Diese Schritte werden in Abschnitt 4.2.6 näher beschrieben. Zuletzt werden drei unterschiedliche Visualisierungen generiert und gespeichert (siehe Abschnitt 4.2.7).

Die Tool-Chain wurde im Rahmen dieser Arbeit in Python selbst entwickelt und basiert zum Teil auf bereits zuvor existierenden Tools. Dabei handelt es sich neben dem bereits erwähnten Tool zum Software-Parsing auch um ein Tool, das eine der drei Arten der Visualisierungen generiert [17]. Für zusätzlich benötigte Funktionalitäten – beispielsweise die Implementierung des Graphen und das Auslesen der Datenbank – wurden außerdem diverse Python-Pakete verwendet, diese werden an den entsprechenden Stellen angeführt. Der Aufbau des Python-Programms ist in Funktionen organisiert, wobei jeder Teil der Tool-Chain (vgl. Konzept in Abbildung 4.2), sofern es sich nicht um bereits existierende Tools handelt, in einer eigenen Funktion implementiert ist.

4.2.1 Feature- und Koordinator-Zuordnung

Diese Zuordnung dient dazu, den Features und den Koordinatoren entsprechend einzelne FCs zuzuordnen (vgl. Abschnitte 3.1.1 und 3.1.2). Dabei werden sowohl die Zuordnungen der Features als auch die der Koordinatoren zu den FCs in der selben Datei gespeichert. Dazu werden neben den Namen auch die entsprechenden Typen gespeichert. Dadurch kann festgestellt werden, ob es sich bei einem Knoten um ein Feature oder einen Koordinator handelt. Außerdem wird bei der Typ-Information zwischen optionalen und obligatorischen Features unterschieden.

Die Feature-Zuordnung ist in Form einer json-Datei implementiert. Dabei handelt es sich um ein einfaches Dateiformat, welches Strukturen unterschiedlichster Art unterstützt und mittels des Python-Pakets json leicht auszulesen ist.

Der Aufbau der json-Datei wird anhand der möglichen Verteilung von Features und Koordinatoren aus Abbildung 3.4 in Listing 4.1 dargestellt. Es besteht aus einer Liste aus einzelnen Elementen, die die Knoten darstellen. Diese Elemente bestehen wiederum jeweils aus drei Attributen – dem Namen und Typen des Knoten sowie einer Liste aus FCs, die dem Knoten zugeordnet sind. Auf diese Informationen kann später mittels des Python-Pakets json im Python-Programm einfach zugegriffen werden.

4.2.2 Software Parsing

In diesem Teil werden die notwendigen Informationen aus der Motorsteuergeräte-Software extrahiert, um in weiterer Folge die Ein- und Ausgangsvariablen der einzelnen FCs ermitteln zu können. Dazu wird ein firmeninternes Tool verwendet.

Als Input dient die elf-Datei, welche den Maschinencode zur Ausführung der Motorsteuergeräte-Software auf dem Prozessorsystem enthält. Diese Datei entsteht am Ende des Build-Prozesses. Zunächst erstellt das Tool eine leichter lesbare Text-Datei aus der elf-Datei, auf welcher das

Listing 4.1: Implementierung der Feature- bzw. Koordinator-Zuordnung als json-Datei anhand des Beispiels aus Abbildung 3.4

```
[{
        "name": "FeatureA",
        "type": "Feature_o",
        "fcs": ["FC1", "FC5"]
}, {
        "name": "FeatureB",
        "type": "Feature_m",
        "fcs": ["FC2", "FC3"]
}, {
        "name": "FeatureC",
        "type": "Feature_m",
        "fcs": ["FC4", "FC8", "FC9"]
}, {
        "name": "FeatureD",
        "type": "Feature_o",
        "fcs": ["FC11", "FC12"]
}, {
        "name": "KoordinatorA",
        "type": "Coordinator",
        "fcs": ["FC6", "FC7", "FC10"]
}]
```

Parsing durchgeführt wird. Es wird ein Funktionsaufrufbaum erstellt und die gelesenen und geschriebenen Variablen werden ermittelt. Dabei wird eine SQLlight-Datenbank erstellt, in der die Daten abgelegt werden.

Die so ermittelten Daten werden in mehreren Tabellen in der Datenbank abgelegt. Dabei enthalten drei dieser Tabellen zur Ermittlung der Abhängigkeiten relevante Informationen (siehe Abbildung 4.3).

- **Variables:** Diese Tabelle enthält für jede Variable einen Eintrag, wobei für jede Variable ihr Name und ihre ID gespeichert werden. Zugriffe auf die Variablen werden hier nicht gespeichert. Diese Informationen befinden sich in der Tabelle „VariableAccess“.
- **VariableAccess:** Sie enthält für jeden Variablenzugriff einen Eintrag. Jeder Zugriff enthält seine eigene Zugriffs-ID und den Typ des Zugriffs (z.B. Lese- oder Schreibzugriff), sowie die ID der Variable, auf die zugegriffen wird. Allein diese Informationen geben noch keine Auskunft darüber, welche FC auf die Variable zugreift. Daher wird zusätzlich eine Datei-ID abgespeichert, welche in weiterer Folge Rückschlüsse auf die zugreifende Funktion zulässt. Diese ID kann mithilfe der Tabelle „Fileartefacts“ aufgelöst werden.
- **Fileartefacts:** Diese Tabelle enthält u.a. für jedes C-File einen eigenen Eintrag. Hier wird für jede Datei der Pfad inkl. des Dateinamens abgespeichert sowie seine ID. Mithilfe des Dateinamens kann direkt auf die zugehörige FC rückgeschlossen werden.

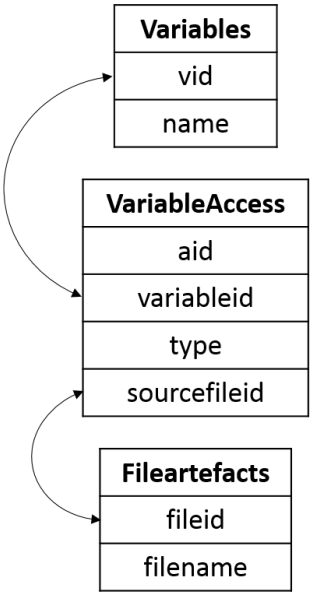

Abbildung 4.3: Zusammenhänge der Tabellen in der Datenbank.

Diese Datenbank bildet die Basis für die Ermittlung der Ein- und Ausgangsvariablen. Die Datenbank wird in Abschnitt 4.2.4 weiterverwendet, um diese zu ermitteln.

4.2.3 Graph als zentrales Speicherelement

Zur Repräsentation der Struktur der Software bietet sich ein Graph an (vgl. Kapitel 3.1.4). In eben solchen können die Features und die Koordinatoren als Knoten und die Abhängigkeiten durch Variablen als Kanten dargestellt werden. Er dient als zentrales Speicherelement, in welchem neben den Abhängigkeiten selbst diverse Attribute wie z.B. die Ergebnisse der Coupling-Metriken gespeichert werden können (siehe Tabelle 4.1). Der Graph wird mittels des Python-Pakets iGraph implementiert. Es vereinfacht die Implementierung eines Graphs in Python, indem die Klasse Graph zusammen mit nützlichen Methoden zur Verfügung gestellt wird.

4.2.4 Datentransformation

In diesem Teil werden die Datenbank und die Feature- bzw. Koordinator-Zuordnung in einen Graphen transformiert.

Es wird zunächst ein Graph erstellt, indem eine Instanz der Klasse Graph erzeugt wird. Danach wird die Feature- bzw. Koordinator-Zuordnung mithilfe des Python-Pakets json in das Python-Programm eingelesen. Dabei wird für jedes Feature und jeden Koordinator je ein Knoten im Graphen angelegt und dem Knoten werden die Attribute name, type und fcs hinzugefügt (siehe Tabelle 4.1). Der Typ des Knoten gibt an, ob es sich dabei um ein optionales, ein obligatorisches („mandatory“) Feature oder um einen Koordinator handelt.

Anschließend werden aus der Datenbank die Ein- und Ausgangsvariablen der zugehörigen FCs der Features ermittelt. Dazu muss zu Beginn die Datei-ID der entsprechenden FC aus der Tabelle Fileartefacts ermittelt werden.

In Python ist es dabei möglich mittels des Pakets sqlite3 direkt Datenbankabfragen im SQL-Format durchzuführen. Der beschriebene Vorgang ist somit mit einem SQL-Befehl durchführbar.

Anschließend werden die zwei Tabellen „Variables“ und „VariableAccess“ über das Schlüsselfeld „variableid“ bzw. „vid“ mithilfe von JOIN verknüpft und zusammengeführt. Dabei wird nach den jeweils gesuchten FCs gefiltert. Je nachdem, ob Eingangs- oder Ausgangsvariablen gesucht werden, wird an dieser Stelle der type nach read bzw. write gefiltert.

Der oben beschriebene Vorgang ist wiederum mit je einem einzigen SQL-Befehl für Ein- und Ausgangsvariablen einer FC durchführbar.

Nachdem die Ein- und Ausgangsvariablen aller FCs der Knoten ermittelt wurden, werden diese jeweils für einen Knoten in einer Liste gespeichert. Diese Liste wird in weiterer Folge dem Knoten im Graph als Attribut hinzugefügt (siehe Tabelle 4.1).

Attribut	*Datentyp*	*Beschreibung*
name	String	Name des Knotens
type	String	Typ des Knotens (Feature_o, Feature_m oder Coordinator)
fcs	Liste	Liste der FCs, die diesem Knoten zugeordnet sind
in_vars	Liste	Eingangsvariablen des Knoten
out_vars	Liste	Ausgangsvariablen des Knoten
refw	Liste	Ergebnisse der Metrik $RefW$ zwischen diesem und allen anderen Knoten
cbf	Gleitkommazahl	Ergebnis der Metrik CBF für diesen Knoten
dcpf	Gleitkommazahl	Ergebnis der Metrik $DCpF$ für diesen Knoten

Tabelle 4.1: Attribute der Knoten im Graphen.

4.2.5 Ermittlung von Abhängigkeiten

Im nächsten Teil werden die Abhängigkeiten zwischen den Knoten ermittelt. Für die Ermittlung dieser Abhängigkeiten werden die zuvor ermittelten Ein- und Ausgangsvariablen der Knoten verwendet. Abbildung 4.4 zeigt mehrere Features mit ihren Ein- und Ausgangsvariablen. Dabei stellen die eingehenden Pfeile gelesene und die ausgehenden geschriebene Variablen dar. Eine Abhängigkeit zwischen zwei Knoten tritt auf, wenn die gleiche Variable von einem Knoten gelesen und von einem anderen geschrieben wird, diese Variable also sowohl Ausgangsvariable für einen Knoten als auch Eingangsvariable für ein anderen darstellt. In Abbildung 4.4 tritt dieser Fall bei Variable 3 auf. Variable 3 wird von Feature B geschrieben – Variable 3 ist Ausgangsvariable von Feature B – und wird sowohl von Feature A als auch von Feature C gelesen – Variable 3 ist Eingangsvariable von Feature A und C. Dabei handelt es sich um zwei Schreib-/Leseverbindungen und entsprechend entstehen zwei Abhängigkeiten. Sowohl Feature A als auch Feature C sind durch diese Variable abhängig von Feature B. Dementsprechend werden im Graphen zwei gerichtete Kanten hinzugefügt – eine vom Knoten von Feature B zum Knoten von Feature A und eine zum Knoten von Feature C.

Um solche Abhängigkeiten zu finden, müssen jeweils alle Ausgangsvariablen aller Knoten betrachtet werden und mit allen Eingangsvariablen aller anderen Knoten verglichen werden. Wird eine Übereinstimmung gefunden, entspricht das einer Abhängigkeit dieser beiden Knoten. In weiterer Folge wird daher eine Kante zwischen diesen beiden Knoten im Graphen hinzugefügt.

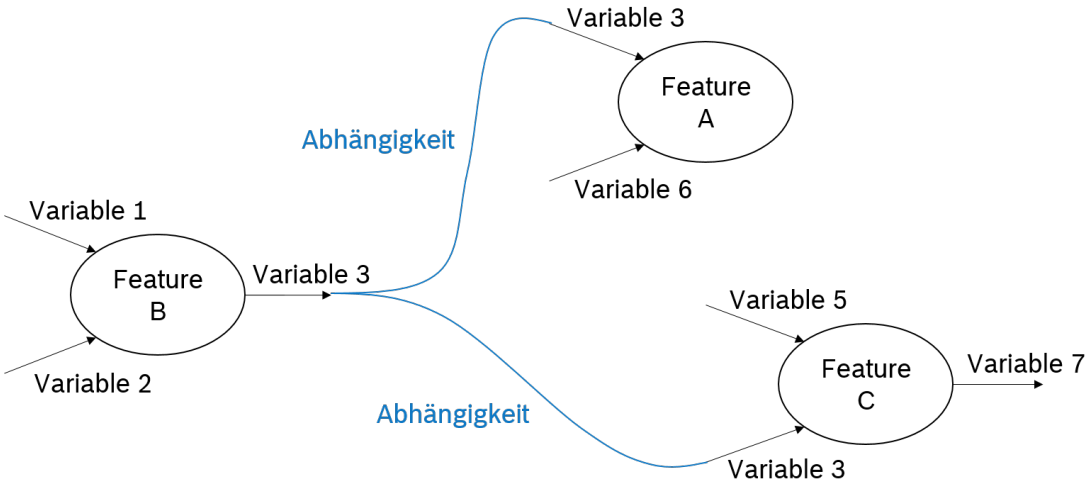

Abbildung 4.4: Ermittlung von Abhängigkeiten.

4.2.6 Berechnung von Coupling-Metriken

In diesem Teil werden die Coupling-Metriken anhand des Graphen berechnet. Basis für diese Berechnung sind die zuvor bereitgestellten Abhängigkeiten zwischen den Knoten. Je nach Art der Metrik wird die Anzahl der Abhängigkeiten herangezogen, um die jeweilige Metrik zu berechnen. Im Folgenden werden die Berechnungen der verwendeten Coupling-Metriken $RefW$, CBF und $DCpF$ beschrieben. Danach werden die Ergebnisse der Metriken ebenfalls als Attribute des Graphen im Programm gespeichert (siehe Tabelle 4.1). Außerdem werden die Ergebnisse der beiden Metriken jeweils in eine csv-Datei geschrieben, um die Daten anschließend auswerten zu können.

4.2.6.1 RefW

Diese Metrik bezieht sich jeweils auf Paare von Knoten. Entsprechend ihrer Definition aus Abschnitt 2.4.3.1 muss berechnet werden, wie viele Referenzen exakt zwischen zwei Knoten existieren.

Dabei wird über alle Kombinationen von Knoten iteriert. Es werden alle Kanten zwischen diesen beiden Knoten und schließlich deren Anzahl ermittelt. Die Metrik wird in Form einer Matrix abgespeichert, wobei die Spalten und die Zeilen jeweils die Knoten darstellen. Für diesen Zweck wurde das Python-Paket numpy verwendet.

4.2.6.2 CBF

Diese Metrik bezieht sich jeweils nur auf einzelne Knoten. Entsprechend ihrer Definition aus Abschnitt 2.4.3.2 muss berechnet werden, zu wie vielen anderen Knoten ein Knoten Abhängigkeiten besitzt. Dabei ist unerheblich, wie viele Referenzen zu einem der anderen Knoten genau vorhanden sind.

Wie bei der Berechnung der Metrik $RefW$ beschrieben wird auch in diesem Fall über alle Kombinationen an Knoten iteriert. Existiert zwischen einem Paar von Knoten nun mindestens eine Referenz, wird CBF für diesen Knoten inkrementiert.

4.2.6.3 DCpF

Diese Metrik bezieht sich ebenfalls jeweils nur auf einen Knoten. Entsprechend ihrer Definition aus Abschnitt 2.4.3.3 muss die genaue Anzahl der Abhängigkeiten, die der Knoten insgesamt besitzt, berechnet werden.

Für diese Berechnung wird die iGraph-Methode zur Berechnung des Knotengrades verwendet.

4.2.7 Visualisierung

Im letzten Teil wird die Visualisierung durchgeführt. Die ersten beiden Arten der Visualisierung wurden im Rahmen dieser Arbeit selbst in Python implementiert. Die dritte Art ist mittels eines externen Tools realisiert [17].

4.2.7.1 Visualisierung der Abhängigkeiten

Die erste Art der Visualisierung stellt den Graphen mit seinen Knoten und sämtlichen Kanten grafisch dar (siehe Abbildung 2.9). Dabei repräsentieren die Knoten entsprechend die Features und die Koordinatoren. Jede Kante stellt genau eine Abhängigkeit dar, also eine Variable, die in einem Knoten gelesen und im anderen geschrieben wird. Hierfür wird die plot Methode der Klasse Graph aus dem iGraph-Paket verwendet. Die einzelnen Typen von Knoten werden farblich unterschiedlich dargestellt. Optionale Features werden blau dargestellt, obligatorische Features hellblau und Koordinatoren grau. Diese Art der Visualisierung soll einen Überblick über die Menge an Referenzen zwischen den Knoten vermitteln. Mit ihrer Hilfe kann man auf den ersten Blick erkennen, welche Knoten stark an andere gekoppelt sind und welche eine sehr schwache Kopplung aufweisen. Bei einer sehr hohen Anzahl an Referenzen kann diese Art der Visualisierung jedoch sehr unübersichtlich werden und es kann auf den ersten Blick keine quantitative Aussage über die Kopplungen gemacht werden.

4.2.7.2 Visualisierung der Ergebnisse der Coupling-Metriken

Daher beschäftigt sich die zweite Art der Visualisierung mit der grafischen Darstellung der Ergebnisse der Coupling-Metriken (siehe Abbildung 2.10). Hier werden diese Ergebnisse ebenfalls in einem Graphen dargestellt, jedoch nicht mit sämtlichen Abhängigkeiten als Kanten, da diese schnell unübersichtlich werden. Es wird daher eine Visualisierung generiert mit einem Graphen mit reduzierter Kantenanzahl. Es wird jeweils nur eine Kante zwischen zwei Features angezeigt, wenn mindestens eine Referenz zwischen den beiden vorhanden ist. Alle weiteren Kanten werden nicht angezeigt. Stattdessen werden die jeweiligen Ergebnisse der Metriken eingeblendet. An welcher Stelle im Graphen eine sinnvolle Darstellung möglich ist, ist abhängig davon, worauf sich die Metrik bezieht. CBF und $DCpF$ beziehen sich jeweils auf nur eine einzelne Knoten, wohingegen sich $RefW$ auf Paare von Knoten bezieht. Daher werden die Ergebnisse von CBF und $DCpF$ beim jeweiligen Knoten platziert. Die Ergebnisse von $RefW$ werden hingegen an der Kante zwischen den beiden Features platziert, sofern mindestens eine Referenz vorhanden ist. Andernfalls werden die Ergebnisse nicht angezeigt, da diese null ergeben. Mit Hilfe dieser Visualisierung können die Ergebnisse der Metriken den einzelnen Features und Koordinatoren optisch einfach zugeordnet werden und dies erleichtert somit die Auswertung.

4.2.7.3 Visualisierung mittels Layout-based Clustering

Mit diesen beiden Visualisierungen ist die Aufgabenstellung erfüllt. Es wurde eine geeignete Visualisierung geschaffen, indem es sowohl möglich ist die Abhängigkeiten selbst als auch die Ergebnisse der Coupling-Metriken darzustellen. Zusätzlich wurde als Erweiterung noch eine weitere Visualisierungsvariante implementiert (siehe Abbildung 2.8). Es handelt sich dabei um das Layout-based Clustering, welches bereits in Abschnitt 2.3.2 beschrieben wurde. Damit wird die Struktur der Software, die von der Kopplung bestimmt wird, intuitiv dargestellt. Es wird bei einer hohen Kopplung zwischen Features sofort die räumliche Nähe zwischen ihnen sichtbar. Hotspots, gänzlich entkoppelte Features und Blöcke von gekoppelten Features bzw. Koordinatoren („Cluster“) sind so optisch leicht ersichtlich. Um das Clustering Layout zu berechnen wird das Tool aus [17] verwendet. Es wurde im Rahmen dieser Studie schon zur Untersuchung von Kohäsion in Feature-orientierten Software-Produktlinien verwendet. Sämtliche Produktlinien waren jedoch Java-basiert, wohingegen das Tool im Rahmen dieser Arbeit zur Untersuchung von Kopplung in einem Echtzeit-Softwaresystem, welches in C implementiert ist, verwendet wird.

Beim Clustering Layout werden Knoten, die eine hohe Kopplung aufweisen, näher dargestellt als jene, die eine niedrige Kopplung aufweisen. Damit wird bei einer hohen Kopplung zwischen Features sofort die räumliche Nähe zwischen ihnen sichtbar. Außerdem ist der Flächeninhalt der Kreise, durch welche die Knoten dargestellt werden, proportional zur Anzahl der inzidenten Kanten.

Als Input für dieses Tool dient ein Graph im rsf-Format [17]. Diese Eingabedatei wird im Python-Programm erstellt. Die Implementierung eines Graphen im rsf-Format wird anhand des Beispiels aus Abbildung 3.7 in Listing 4.2 gezeigt. Jede Zeile in der Datei stellt eine Kante oder Mehrfachkanten zwischen zwei Knoten dar. Dabei gibt das erste Schlüsselwort Auskunft über den Namen des Graphen, in welchem die Kante(n) hinzugefügt werden sollen. Die einzelnen Schlüsselwörter sind durch Leerzeichen voneinander getrennt. Das zweite Schlüsselwort gibt den Knoten an, von welchem die Kante ausgeht. Der Knoten, an welchem die Kante eingeht, wird durch das dritte Schlüsselwort festgelegt. Dabei werden die im Graphen vorkommenden Knoten implizit durch diese zeilenweise Instanzierung der Kanten erstellt. Zuletzt gibt eine Zahl an, um wie viele Kanten es sich zwischen diesen beiden Knoten handelt.

Listing 4.2: Implementierung eines Graphen im rsf-Format

```
Graph FeatureA FeatureB 1.0
Graph FeatureB FeatureA 1.0
Graph KoordinatorA FeatureB 1.0
Graph FeatureC KoordinatorA 3.0
Graph FeatureD KoordinatorA 2.0
```

5 Ergebnisse und Auswertung

In diesem Kapitel werden die Ergebnisse dieser Arbeit präsentiert. Dabei werden zu Beginn die untersuchten Features und Koordinatoren in der Motorsteuergeräte-Software vorgestellt. Danach wird die Struktur der Software betrachtet, welche sich aus der Darstellung mittels des Graphen mit seinen Abhängigkeiten ergibt. Anschließend werden die Ergebnisse der Coupling-Metriken, die auf die Motorsteuergeräte-Software angewendet wurden, dargelegt und diskutiert. Zuletzt wird das Clustering Layout präsentiert und analysiert.

5.1 Untersuchte Features und Koordinatoren

Zu Beginn wurde die vorliegende Motorsteuergeräte-Software auf Features und Koordinatoren untersucht (vgl. Abschnitte 3.1.1 und 3.1.2). Dabei wurde eine Menge an Features und Koordinatoren festgelegt, die im Rahmen dieser Arbeit untersucht wurde.

Um Features und Koordinatoren in der Software zu identifizieren, wurde die Dokumentation dieser Software verwendet. Sämtliche FCs wurden auf potentiell enthaltene Features und Koordinatoren untersucht. Anschließend wurden die Features und Koordinatoren den einzelnen FCs zugeordnet. Die Software ist nicht Feature-orientiert aufgebaut, sondern sie orientiert sich am physikalischen Aufbau des Motors, da so einzelne Software-Teile besser den physikalischen Prozessen und den entsprechenden Sensoren und Aktoren zugeordnet werden können [7], [8]. Aus diesem Grund sind Features über die Software verteilt (vgl. Abschnitt 3.1.1) und es war oftmals schwierig Features exakt abzugrenzen. Bei der Auswahl wurde darauf geachtet, Features zu selektieren, die potentiell miteinander in Interaktion stehen. Außerdem wurden Koordinatoren identifiziert, die potentielle Interaktionen zwischen diesen Features auflösen.

Tabelle 5.1 listet die untersuchten Features und Koordinatoren auf und erläutert um welchen Typ es sich handelt (optionales bzw. obligatorisches Feature oder Koordinator). Insgesamt wurden 10 Features und 4 Koordinatoren betrachtet, wobei von den 10 Features 6 optional und 4 obligatorisch sind.

Bei den obligatorischen Features handelt es sich um Funktionen, die für die grundlegende Funktion der Motorsteuerung notwendig sind. Optionale Features hingegen können im Rahmen der Software-Produktlinie bereits zur Designzeit durch die Konfiguration wahlweise deaktiviert werden, da diese nicht zwingend für die Funktionalität des Motors benötigt werden. Da es sich bei der untersuchten Software um proprietäre Software eines Motorsteuergeräts handelt, können die

Name	*Typ*
Feature_1	optionales Feature
Feature_2	optionales Feature
Feature_3	optionales Feature
Feature_4	optionales Feature
Feature_5	optionales Feature
Feature_6	optionales Feature
Feature_7	obligatorisches Feature
Feature_8	obligatorisches Feature
Feature_9	obligatorisches Feature
Feature_10	obligatorisches Feature
Coordinator_1	Koordinator
Coordinator_2	Koordinator
Coordinator_3	Koordinator
Coordinator_4	Koordinator

Tabelle 5.1: Untersuchte Features und Koordinatoren in der Motorsteuergeräte-Software mit den zugehörigen Typen (optionales, obligatorisches Feature oder Koordinator).

einzelnen Features und Koordinatoren hier nicht beschrieben werden. Allerdings existieren beispielsweise wesentliche Zusammenhänge zwischen einigen obligatorischen Features und einigen Koordinatoren, die im Folgenden exemplarisch beschrieben werden.

Bei den drei Kombinationen aus je einem obligatorischen Feature und einem Koordinator

- Feature_8 und Coordinator_2,
- Feature_9 und Coordinator_3 bzw.
- Feature_10 und Coordinator_4

handelt es sich bei dem obligatorischen Feature jeweils um eine Steuerung bzw. Regelung für je eine bestimmte Größe. Der entsprechende Koordinator in der Kombination ist dabei für die Koordinierung von Sollwerten für diese Größe zuständig.

5.2 Resultierende Abhängigkeiten

Anschließend wurde die zuvor erstellte Zuordnung der Features und Koordinatoren zu den FCs und die Motorsteuergeräte-Software selbst dazu verwendet, um mit Hilfe der in Kapitel 4 entwickelten Tool-Chain einen Graphen zu generieren. Dabei werden mittels Software-Parsing die Zugriffe der FCs auf die Variablen ermittelt. Durch die Feature- bzw. Koordinator-Zuordnung können diese Zugriffe den Features und den Koordinatoren zugeordnet werden. Tabelle 5.2 zeigt die Anzahl an gelesenen und geschriebenen Variablen der einzelnen Features und Koordinatoren. Außerdem wird aufgelistet, wie viele Abhängigkeiten die jeweiligen Knoten insgesamt besitzen. Dabei werden sowohl bei den gelesenen bzw. geschriebenen Variablen als auch bei den Abhängigkeiten selbst nur jene hinzugezählt, die entsprechend zu einer Abhängigkeit innerhalb

der ausgewählten Menge an Features und Koordinatoren führen – d.h. es muss eine Schreib-/Leseverbindung innerhalb der ausgewählten Menge an Features und Koordinatoren bestehen. Andere Variablen werden nicht berücksichtigt. Insgesamt ergaben sich im betrachteten System daher 101 Abhängigkeiten. Die Summe der einzelnen Abhängigkeiten der letzten Spalte der Tabelle 5.2 ergibt $202 = 2 \cdot 101$, da hier die Summe die Abhängigkeiten aus beiden Richtungen betrachtet ergibt.

Name	*gelesene Variablen*	*geschriebene Variablen*	*Abhängigkeiten*
Feature_1	8	7	15
Feature_2	4	1	5
Feature_3	6	18	24
Feature_4	4	6	10
Feature_5	11	2	13
Feature_6	5	2	7
Feature_7	2	9	11
Feature_8	15	7	22
Feature_9	9	7	16
Feature_10	6	6	12
Coordinator_1	18	22	40
Coordinator_2	3	5	8
Coordinator_3	7	6	13
Coordinator_4	3	3	6

Tabelle 5.2: Anzahl der gelesenen und geschriebenen Variablen sowie gesamte Anzahl der Abhängigkeiten der jeweiligen Knoten der einzelnen Features und Koordinatoren.

Anschließend wird ein Graph mit den Features und Koordinatoren als Knoten und diesen Abhängigkeiten als Kanten dargestellt (vgl. Abschnitt 4.2.7.1). Abbildung 5.1 zeigt die Visualisierung dieses Graphen.

Dabei zeigt sich, dass zwischen einigen Features bzw. Koordinatoren durchaus eine hohe Anzahl an Abhängigkeiten besteht. Im Folgenden werden einige dieser Situationen diskutiert.

- **Coordinator_1 und Feature_1:** Sie weisen offensichtlich eine hohe Anzahl gegenseitiger Abhängigkeiten auf. Der Koordinator liest einige Variablen von Feature_1 ein. Hierbei könnte es sich potentiell um Sollwerte handeln, die der Koordinator koordiniert.
- **Coordinator_1 und Feature_8:** Auch zwischen diesen beiden Knoten bestehen viele Abhängigkeiten, wobei der Koordinator ebenfalls einige Variablen von Feature_8 liest. Auch in diesem Fall könnten diese Sollwerte zur Koordination darstellen.
- **Coordinator_1 und Feature_3:** Feature_3 ist das dritte Feature, zu welchem Koordinator_1 viele Abhängigkeiten besitzt. Wiederum könnten die vom Koordinator eingelesenen Variablen Sollwerte sein.
- **Coordinator_1:** Dieser Koordinator erscheint durch seine vielen Abhängigkeiten besonders zentral. Er besitzt zu mehreren Features viele Abhängigkeiten. Dieser stellt somit einen potentiellen Hotspot dar.

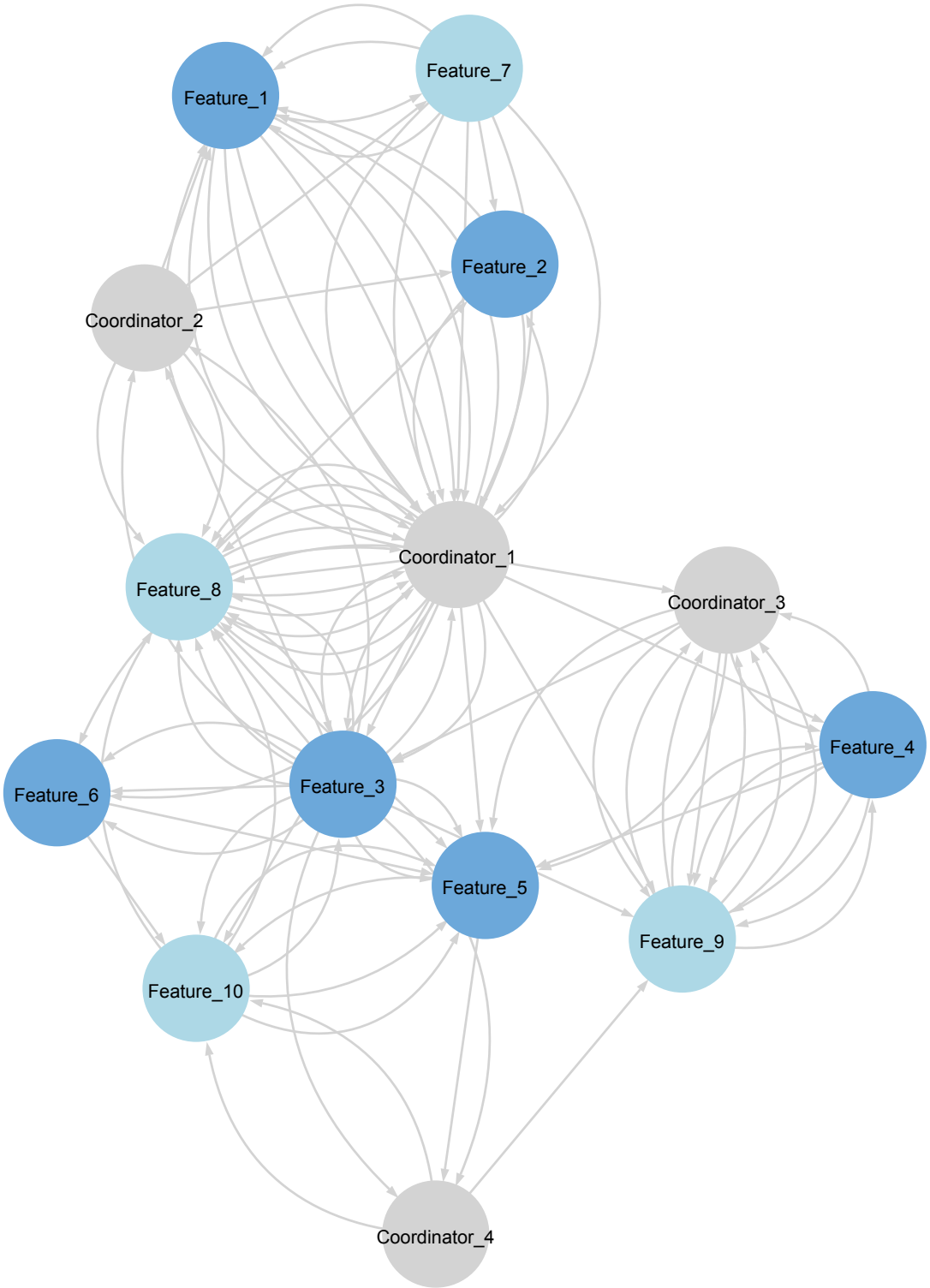

Abbildung 5.1: Visualisierung des Graphen bestehend aus Knoten, welche Features und Koordinatoren in der Motorsteuergeräte-Software repräsentieren, und Kanten, die Abhängigkeiten zwischen ihnen darstellen.

- **Coordinator_3 und Feature_9:** Auch dieses Feature und dieser Koordinator weisen eine besonders hohe Anzahl an gegenseitigen Abhängigkeiten auf. Wie bereits in Abschnitt 5.1 beschrieben, handelt es sich dabei um eine spezielle Kombination aus Feature und Koordinator. Dieses Feature implementiert die Regelung einer Aktorgröße, dessen Sollwerte von diesem Koordinator koordiniert werden.
- **Feature_4 und Feature_9:** Wie im vorigen Punkt bereits beschrieben, stellt Feature_9 die Regelung für eine Aktorgröße dar. Die hohe Anzahl an Abhängigkeiten zu Feature_4 lässt sich u.a. damit erklären, dass dieses Feature eine Funktionalität beinhaltet, die den Wertebereich dieser Größe einschränken kann.

In den folgenden Abschnitten 5.3 und 5.4 werden die obigen Situationen weiter analysiert.

5.3 Ergebnisse und Auswertung der Coupling-Metriken

Danach wurden ebenfalls innerhalb der in Kapitel 4 entwickelten Toolchain die Coupling-Metriken für die einzelnen Knoten berechnet. Anschließend wurden diese Ergebnisse, wie in Abschnitt 4.2.7.2 beschrieben, auch mittels eines Graphen visualisiert (siehe Abbildung 5.2). Die Ergebnisse der Coupling-Metriken in tabellarischer Form befinden sich im Anhang A. Entsprechend dem in Abschnitt 3.2 beschriebenen Konzept werden diese schließlich ausgewertet. Dabei können drei unterschiedliche Arten von Feature-Interaktionen auftreten. Diese können zwei verschiedenen Konstellationen zugeordnet werden. Im Folgenden werden die Ergebnisse der einzelnen Metriken hinsichtlich Feature-Interaktionen diskutiert.

5.3.1 RefW

Wie in Abschnitt 3.2.2.1 beschrieben, bezieht sich die Metrik $RefW$ auf Paare von Knoten und sollte daher Aufschluss über Feature-Interaktionen der Kategorie 1 und 2 geben können. Dabei ergibt sich die empirische Annahme, dass eine potentielle Feature-Interaktion zwischen Features f_1 und f_2 besteht, wenn

$$RefW\,(f_1, f_2) \geq 1. \tag{5.1}$$

Das bedeutet, dass bereits eine potentielle Feature-Interaktion entsteht, sobald mindestens eine einzige Abhängigkeit zwischen zwei Features besteht. Daher wurden sämtliche Abhängigkeiten dahingehend überprüft, ob sie eine Feature-Interaktion im Sinne der Definition aus Abschnitt 2.2.2.1 darstellen. Dabei wird die Feature-Interaktion als Beeinflussung eines Features durch ein anderes definiert. In Bezug auf Abhängigkeiten durch Schreib-/Leseverbindungen (vgl. Abschnitt 3.1.3) bedeutet dies eine Beeinflussung des die Variable lesenden Features durch das schreibende Feature. Es wurde also evaluiert, ob es sich bei den einzelnen Abhängigkeiten tatsächlich um Beeinflussungen der jeweiligen lesenden Features handelt. Hierbei stellte sich heraus, dass diese Beeinflussung in jedem der Fälle gegeben ist. An dieser Stelle sei hinzuzufügen, dass es sich dabei durchwegs um gewollte Interaktionen der Kategorie 1 oder 2 handelt.

Entsprechend können Feature-Interaktionen zwischen Paaren von Features f_1 und f_2, die Gleichung (5.1) genügen, auftreten. Es handelt sich daher um alle Paare von Features, die im Graphen in Abbildung 5.2 durch eine Kante verbunden sind. Ein Beispiel für Features mit vielen Abhängigkeiten (Features 4 und 9) wurde bereits in Abschnitt 5.2 diskutiert.

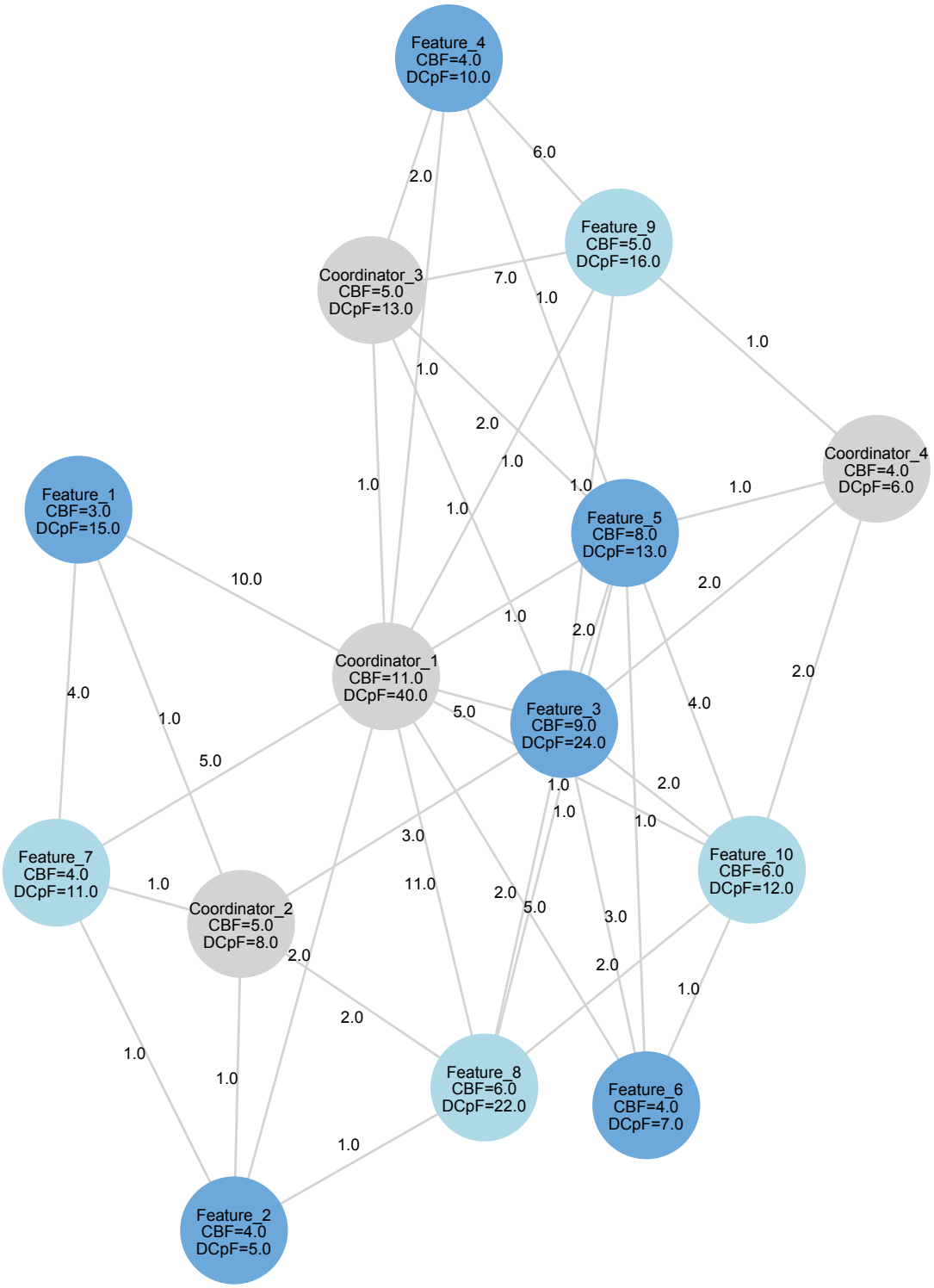

Abbildung 5.2: Visualisierung eines Graphen mit den Ergebnissen der Metriken $RefW$, CBF und $DCpF$.

Mit Hilfe von Gleichung (5.1) konnte also festgestellt werden, ob Feature-Interaktionen vorhanden sind. Es stellt sich in weiterer Folge die Frage, ob der tatsächliche Wert der Metrik weitere Schlüsse zulässt, wenn Gleichung (5.1) erfüllt ist. Es wurde daher analysiert, ob der Wert von $RefW$ eine Aussage über den Grad der Beeinflussung durch die Feature-Interaktion treffen kann. Der Grad der Beeinflussung bedeutet an dieser Stelle, in welchem Ausmaß ein Feature ein anderes beeinflusst. Beispielsweise kann ein Feature einerseits ein anderes dahingehend beeinflussen, dass eine geringfügige Änderung des Sollwertes einer Größe auftritt (Feature-Interaktion der Kategorie 2). Andererseits kann durch eine Feature-Interaktion auch der Sollwert eines Features maßgeblich verändert werden (ebenfalls Feature-Interaktion der Kategorie 2) oder es kann etwa ein Features gänzlich deaktiviert werden (Feature-Interaktion der Kategorie 1).

Es stellt sich allerdings heraus, dass sich mit Hilfe der Ergebnisse der Metrik $RefW$ nicht auf den Grad der Beeinflussung schließen lässt. Dies lässt sich mit der Tatsache begründen, dass nicht alle Abhängigkeiten den gleichen Grad an Beeinflussung besitzen. Beispielsweise kann eine einzelne Abhängigkeit etwa in Form einer Statusvariable bereits dazu führen, dass ein anderes Feature deaktiviert wird. Oder es kann zum Beispiel eine einzelne Abhängigkeit existieren, bei der die eingelesene Variable Daten enthält, die die weiteren Berechnungen innerhalb des Features wesentlich beeinflussen. Es können aber auch viele Abhängigkeiten zwischen Features bestehen, die insgesamt zu keinem hohen Ausmaß an Beeinflussung führen. Dies ist beispielsweise der Fall, wenn von einem Feature Variablen eingelesen werden, die in weiterer Folge die Berechnungen allesamt nur geringfügig beeinflussen. Daher kann aus der Anzahl dieser Abhängigkeiten nicht auf den Grad der Beeinflussung zwischen Features geschlossen werden.

Allerdings drückt der konkrete Wert von $RefW$ den Grad der Kopplung zwischen zwei Features aus. Es können also mit Hilfe von $RefW$ Schlüsse gezogen werden, in welchem Ausmaß zwei Features verzahnt sind. Im Falle einer Weiterentwicklung an einem der Features müssen entsprechend sämtliche Abhängigkeiten analysiert werden, um hier die Entstehung einer ungewollten Interaktion auszuschließen. Somit stellt der Wert der Metrik ein Maß für die Verzahnung der Features dar. Es ist anzunehmen, dass der Arbeitsaufwand, der für eine mögliche Weiterentwicklung zusätzlich benötigt wird, proportional zum Ausmaß der Verzahnung ist.

In obigem Text wurden Aussagen der Ergebnisse der Metrik $RefW$ in Bezug auf direkte Interaktionen zwischen Features (Feature-Interaktionen der Kategorie 1 und 2) diskutiert. $RefW$ kann allerdings auch Auskunft darüber geben, ob Abhängigkeiten zwischen einem Feature und einem Koordinator vorhanden sind. Bestehen nun Abhängigkeiten zweier Features mit dem selben Koordinator, liefert dies einen Hinweis auf eine mögliche Feature-Interaktion der Kategorie 3. Ob es sich jedoch tatsächlich um solch eine Interaktion handelt, kann nicht ausschließlich mit Hilfe von Metriken beantwortet werden. Es muss in diesem Fall manuell analysiert werden, ob es sich bei den Variablen um Sollwerte der gleichen Größe handelt, die der Koordinator einliest, um diese Entscheidung zu treffen. Einige Beispiele für Abhängigkeiten zwischen Features und Koordinatoren wurden in Abschnitt 5.2 beschrieben. Im Folgenden werden diese weiter analysiert.

- **Coordinator_1 und Feature_1, Feature_3 und Feature_8:** Hierbei handelt es sich tatsächlich um Features, die Sollwerte für eine Größe erzeugen, die in weiterer Folge vom Koordinator koordiniert werden. Aus diesem Grund bestehen Feature-Interaktionen der Kategorie 3 zwischen den Feature_1, Feature_3 und Feature_8.
- **Coordinator_3 und Feature_9:** Wie in Abschnitt 5.1 bereits beschrieben, handelt es sich hierbei um die Regelung und Koordination einer Aktorgröße. Feature_9 liefert Coordinator_3

also den Sollwert einer Größe, die von diesem koordiniert wird. Daher bestehen Feature-Interaktionen zwischen Feature_9 und sämtlichen Features, die dem Koordinator ebenfalls einen Sollwert der selben Größe liefern, wobei diese durch den Koordinator aufgelöst werden.

In der Struktur der Abhängigkeiten in Abbildung 5.1 findet sich die Konstellation für Feature-Interaktionen der Kategorie 3 (vgl. Abbildung 3.9) auch in einigen weiteren Fällen wieder, wo Koordinatoren von unterschiedlichen Features Variablen einlesen. Bei einigen davon handelt es sich tatsächlich ebenfalls um eine Feature-Interaktion der Kategorie 3, also um Anforderungen von Sollwerten für die gleiche Größe, die entsprechend koordiniert werden.

5.3.2 CBF und DCpF

Wie in Abschnitt 3.2.2.2 beschrieben, beziehen sich die beiden Metriken CBF und $DCpF$ je auf ein Feature oder auf einen Koordinator und sollten daher Auskunft darüber geben können, ob und wie stark ein Feature oder ein Koordinator generell mit anderen Features gekoppelt ist. Um diesen Zusammenhang zu diskutieren, wird, wie in oben genanntem Abschnitt beschrieben, ein Diagramm verwendet, in welchem Features und Koordinatoren abhängig von ihren Werten für CBF und $DCpF$ aufgetragen werden können (vgl. Abbildung 3.10). Die Ergebnisse der Metriken der untersuchten Features und Koordinatoren sind in Abbildung 5.3 dargestellt. Dabei stellt der graue Bereich den nicht zulässigen Bereich dar. Im Folgenden werden diese Ergebnisse diskutiert.

- **Coordinator_1:** Wie schon in Abschnitt 5.2 erkannt wurde, weist dieser Koordinator eine sehr hohe Anzahl an Abhängigkeiten auf und scheint sehr zentral zu sein und damit ist er ein potentieller Hotspot. Diese Vermutung lässt sich anhand der Ergebnisse der Metriken CBF und $DCpF$ und mit Hilfe von Abbildung 5.3 bestätigen. Sowohl die Werte für CBF als auch für $DCpF$ sind sehr hoch und damit stellt Coordinator_1 einen Hotspot dar. Aus diesem Grund können Interaktionen zwischen Features, zu denen Abhängigkeiten vorhanden sind, bestehen. Wie in Abschnitt 5.3.1 bereits anhand einiger Beispiele beschrieben, ist dies tatsächlich der Fall. Durch den Koordinator werden diese Interaktionen zwischen den Features jedoch aufgelöst.
- **Feature_3:** Hierbei handelt es sich um ein Feature, welches ebenfalls hohe Werte bei beiden Metriken aufweist. In Abbildung 5.1 kann man erkennen, dass dieses Feature sehr zentral ist und Abhängigkeiten zu vielen Features aufweist. Eine starke Kopplung zu Coordinator_1 wurde bereits in Abschnitt 5.2 erkannt und in Abschnitt 5.3.1 analysiert. Dieses Feature stellt ebenfalls einen Hotspot dar. Die hohe Anzahl an Abhängigkeiten zu unterschiedlichen Knoten lässt sich wie folgt erklären. Es handelt sich dabei um ein Feature, das in einem bestimmten Betriebsmodus für einen beschränkten Zeitraum aktiv ist. In dieser Zeit muss es wesentlich in andere Teilsysteme eingreifen, um seine Aufgabe zu erfüllen. Während der restlichen Betriebsdauer ist dieses Feature nicht aktiv und greift daher auch nicht mehr in andere Funktionen ein.

Die restlichen Features weisen im Verhältnis zu Coordinator_1 und Feature_3 geringere Werte für CBF und $DCpF$ auf. Es zeigt sich, dass außerdem kein Feature oder Koordinator existiert, der besonders viele Abhängigkeiten zu einem oder nur zu wenigen anderen Knoten aufweist (vgl. Fall 2 aus Abschnitt 3.2.2.2). Zudem kommen auch keine Features oder Koordinatoren vor, welche nur wenige Abhängigkeiten zu wenigen anderen Knoten besitzt (vgl. Fall 1 aus Abschnitt 3.2.2.2).

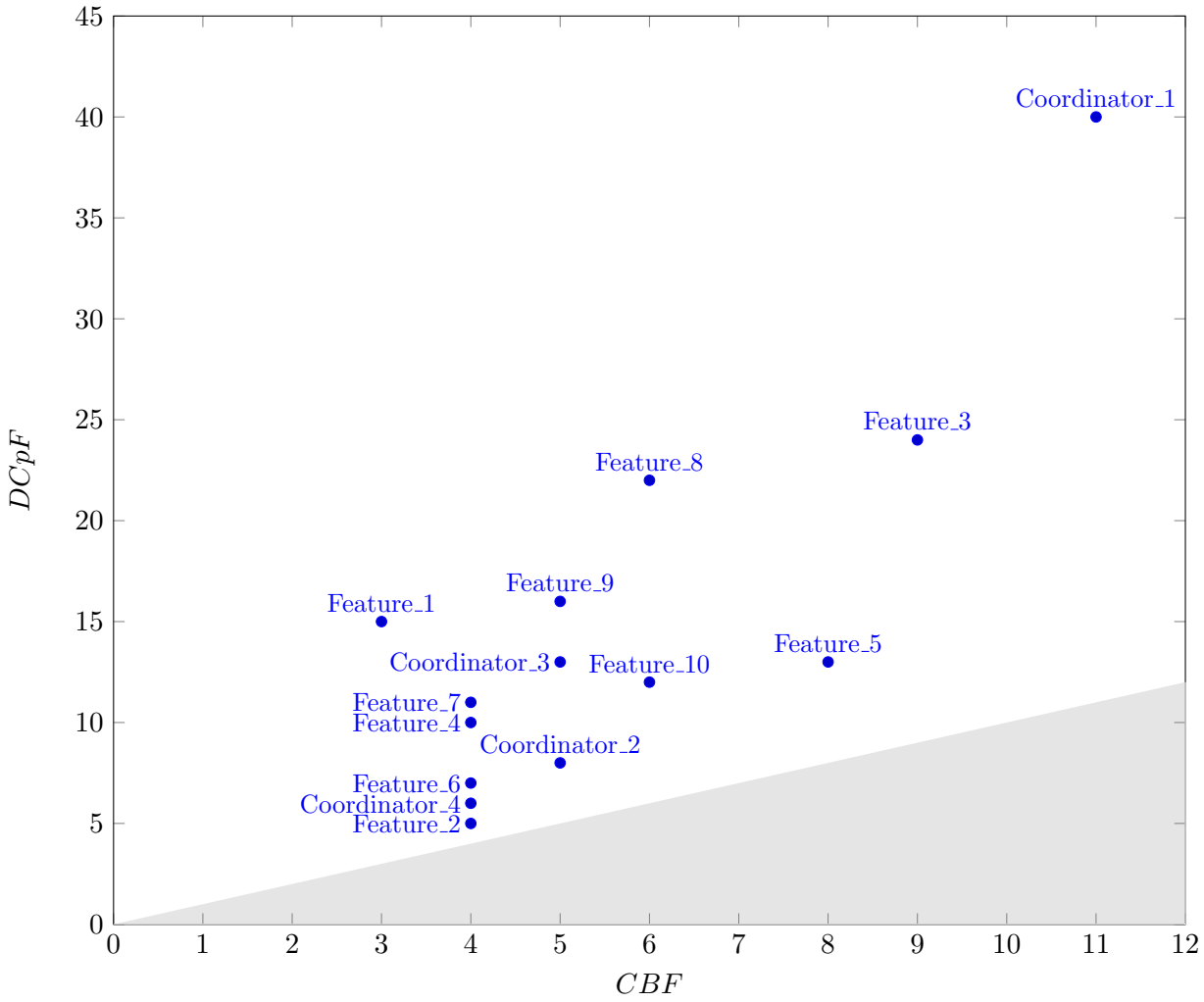

Abbildung 5.3: Resultierende Werte der Metriken CBF und $DCpF$ aller Features und Koordinatoren (nicht zulässiger Bereich entsprechend Gleichung (3.2) in grau dargestellt).

Stellt ein Feature oder ein Koordinator einen Hotspot dar, hat dies wesentliche Auswirkungen. Sind etwa Weiterentwicklungen an einem Hotspot oder eine Integration eines Hotspots in ein bestehendes Softwaresystem notwendig, ist sehr wahrscheinlich Expertenwissen aus unterschiedlichen Bereichen der Motorsteuergeräte-Software erforderlich. Der Aufwand der Weiterentwicklung kann daher entsprechend hoch eingestuft werden. Entsteht an dieser Stelle ein Fehler in der Software, kann dies starke Auswirkungen auf andere Teile der Software haben, da ein Hotspot viele Schnittstellen zu anderen Features aufweist.

CBF und $DCpF$ messen wie $RefW$ ebenfalls den Grad der Kopplung. Mit Hilfe der beiden Metriken können also Schlüsse gezogen werden, in welchem Ausmaß das jeweilige Feature oder der Koordinator mit anderen Features verzahnt ist. Im Falle einer Weiterentwicklung oder einer Integration des Features in die Software lassen diese Metriken ebenfalls Rückschlüsse über den entsprechenden Arbeitsaufwand zu.

5.4 Analyse des Clustering Layouts

Zuletzt werden die Ergebnisse anhand der Visualisierung mittels Layout-based Clustering, welches in Abschnitt 2.3.2 beschrieben wurde, diskutiert. Der resultierende Graph der untersuchten Features und Koordinatoren wird in Abbildung 5.4 dargestellt.

In dieser Darstellung lassen sich intuitiv Cluster von Features bzw. Koordinatoren erkennen. Diese werden im Folgenden analysiert.

(1) Feature_4, Feature_9, Coordinator_3: Wie in Abschnitt 5.1 bereits beschrieben, handelt es sich bei Feature_9 und Coordinator_3 um eine Kombination aus Regelung und Koordination der selben Größe. Auch die hohe Anzahl an Abhängigkeiten zwischen Feature_4 und Feature_9 sowie zwischen Coordinator_3 und Feature_9 wurde bereits in Abschnitt 5.2 analysiert. Es ist daher plausibel, dass sich diese Knoten im selben Cluster befinden. In Abbildung 5.4 lässt sich außerdem erkennen, dass dieser Cluster eine hohe Distanz zu den anderen Clustern aufweist. Dies lässt sich damit erklären, dass alle drei Knoten Teil des gleichen Subsystems der Motorsteuergeräte-Software sind und nicht in starkem Zusammenhang mit den anderen Funktionalitäten stehen.

(2) Feature_5, Feature_10, Coordinator_4: Bei Feature_10 und Coordinator_4 handelt es sich ebenfalls um die Steuerung bzw. Regelung und den Koordinator für die selbe Größe. Die Nähe beider Knoten zu Feature_5 ist aufgrund ihrer durchaus in Beziehung stehenden Funktionalitäten plausibel. Daher ist es naheliegend, dass sich diese Knoten im selben Cluster befinden. Feature_5 steht außerdem in Zusammenhang mit Funktionalitäten aus Cluster 1, daher erscheint es auch logisch, dass Feature_5 die näheste Position zu Cluster 1 einnimmt.

(3) Feature_3, Feature_6: Diese beiden Features interagieren zwar geringfügig miteinander, aber ihre Funktionalitäten stehen in keinem starken Zusammenhang. Die Position ihres gemeinsamen Clusters zentral zwischen anderen Clustern erscheint allerdings plausibel. Es handelt sich dabei um Features, die beide stark mit anderen Features bzw. Koordinatoren aus anderen Clustern interagieren, da es ihre Aufgabe notwendig macht, zum Teil in mehrere Subsysteme eingreifen. Da sie dementsprechend stark mit den gleichen Features bzw. Koordinatoren in Verbindungen stehen, ist es naheliegend, dass sich beide im gleichen Cluster befinden.

(4) Coordinator_1, Coordinator_2, Feature_1, Feature_2, Feature_7, Feature_8: Bei den Knoten in diesem Cluster handelt es sich durchwegs um Features und Koordinatoren, die im Wesentlichen in der Momentenstruktur zu finden sind und daher ist es höchst plausibel, dass diese sich im gleichen Cluster befinden. Betrachtet man nun diesen Cluster genauer, kann man die Nähe zwischen zwei Paaren von Knoten erkennen.

Coordinator_2, Feature_8: Hierbei handelt es sich wiederum um eine Kombination aus Regelung und Koordinator. Dementsprechend ist die räumliche Nähe naheliegend.

Feature_1, Feature_7: Diese beiden Features haben sehr viel gemeinsam, da sie beide eine wesentliche Rolle in der Drehmomentberechnung spielen. Daher erscheint auch deren Nähe plausibel.

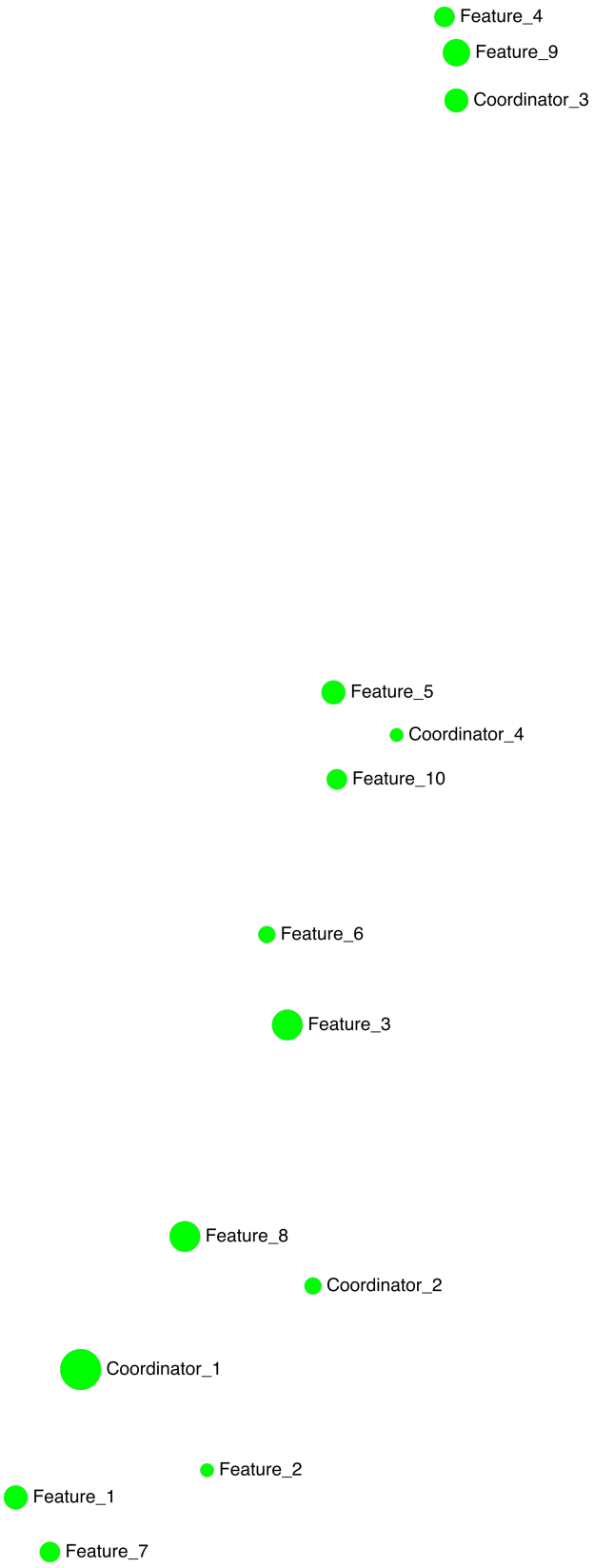

Abbildung 5.4: Visualisierung Layout-based Clustering [17].

6 Diskussion

Eine Diskussion der einzelnen Ergebnisse wurde bereits in Kapitel 5 durchgeführt. In diesem Kapitel werden die wesentlichen Aspekte der Ergebnisse herausgearbeitet und bewertet. Außerdem wird ein Vergleich der Ergebnisse mit der ursprünglichen Aufgabenstellung vorgenommen.

Das Ziel dieser Arbeit war es, potentielle Feature-Interaktionen in der für diese Arbeit bereitgestellten Motorsteuergeräte-Software mittels Coupling-Metriken zu erkennen. Dazu wurden im ersten Schritt Features und Koordinatoren in dieser Software identifiziert und den FCs zugeordnet. Wie in Abschnitt 3.1.1 bereits beschrieben, gestaltet sich dieser Vorgang nicht trivial, da die Motorsteuergeräte-Software nicht Feature-orientiert aufgebaut ist, sondern sich am physikalischen Aufbau des Motors orientiert. Dies bietet den Vorteil, dass einzelne Software-Teile besser den physikalischen Prozessen und den entsprechenden Sensoren und Aktoren zugeordnet werden können [7], [8]. Es mussten daher alle FCs in dieser Software auf Features und Koordinatoren untersucht werden. Aufgrund dieses Aufbaus ist es oftmals nicht einfach, eine eindeutige Zuordnung zu treffen bzw. klare Grenzen für die Features zu ziehen. Ein wesentlicher und wichtiger Teil dieser Arbeit war es daher, Features und Koordinatoren zu identifizieren und den FCs sorgfältig zuzuordnen.

Anschließend wurde aus diesen Features und Koordinatoren ein Graph erstellt und Abhängigkeiten wurden ermittelt. Dabei wurden eine Vielzahl an Abhängigkeiten zwischen vielen Features und Koordinatoren festgestellt. Handelte es sich dabei um Abhängigkeiten zwischen zwei Features, stellten diese eine Feature-Interaktion dar. Trat diese Abhängigkeit zwischen einem Feature und einem Koordinator auf, kann dies auf eine mögliche Feature-Interaktion hinweisen. Klarerweise deutet auch das Vorhandensein eines Koordinators selbst auf eine durch ihn aufgelöste Feature-Interaktion hin.

Schließlich wurden in diesem Graphen mit den dort repräsentierten Abhängigkeiten die Coupling-Metriken berechnet. Diese sollten es ermöglichen, potentielle Feature-Interaktionen zu erkennen. Tatsächlich ist dies mit den betrachteten Features in der vorliegenden Motorsteuergeräte-Software mit Hilfe der Metrik $RefW$ möglich. Weist diese Metrik einen Wert von mindestens eins auf, existiert also mindestens eine Referenz zwischen zwei Features, gibt es eine Interaktion zwischen beiden Features. Auf dieser Basis lassen sich viele Situationen in der Motorsteuergeräte-Software diskutieren. Außerdem kann die Metrik $RefW$ einen Hinweis auf Interaktionen von Features über einen Koordinator geben. Existiert ein positiver Wert der Metrik $RefW$ zwischen Features und dem selben Koordinator, kann auch hier eine Feature-Interaktion auftreten. Ob dies tatsächlich der Fall ist, muss manuell überprüft werden. Feature-Interaktionen dieser Art lassen sich im

Allgemeinen nicht durch diese Art von Coupling-Metriken ermitteln, da diese nichts über die Bedeutung der Variablen aussagen.

Außerdem ist es möglich mittels der Ergebnisse der Metrik $RefW$ auf die Verzahnung von Features und Koordinatoren zu schließen. Die Verzahnung zwischen Features oder zwischen Features und Koordinatoren kann einen Hinweis auf den Arbeitsaufwand geben, der aufgrund von Abhängigkeiten zu anderen Features zusätzlich notwendig ist, wenn beispielsweise eine Weiterentwicklung oder eine Integration eines Features in ein bestehendes Softwaresystem umzusetzen ist.

Allerdings ist es nicht möglich, mittels der Ergebnisse der Coupling-Metriken auf den Grad der Beeinflussung einer Feature-Interaktion zu schließen.

Die Metriken CBF und $DCpF$ erlauben es auf sogenannte Hotspots zu schließen. Handelt es sich bei einem Feature oder bei einem Koordinator um einen Hotspot, erfordert dieser im Falle einer Weiterentwicklung oder Integration in ein bestehendes System einen hohen Analyseaufwand, da mit hoher Wahrscheinlichkeit Expertenwissen aus unterschiedlichen Teilen der Software benötigt wird.

Außerdem wurde eine Tool-Chain entwickelt, mit der es möglich ist Coupling-Metriken zu berechnen. Damit wurde die Grundlage für Untersuchungen von Abhängigkeiten und Feature-Interaktionen in der Motorsteuergeräte-Software gelegt. Mittels der Tool-Chain ist es möglich sämtliche Abhängigkeiten darzustellen. Im Falle der Notwendigkeit einer Analyse bestimmter Teile der Software oder auch der gesamten Software kann diese Tool-Chain daher außerdem ein hilfreiches Werkzeug bei der Analyse von Abhängigkeiten darstellen, mit welchem die Struktur der Software auf unterschiedliche Arten visualisiert werden kann.

Für die Analyse einzelner Interaktionen zwischen Features stellt sich die Visualisierung der Abhängigkeiten, wie in Abbildung 5.1 dargestellt, als geeignet heraus. In dieser Darstellung können einzelne Abhängigkeiten genau nachverfolgt werden. Für die globale Analyse der gesamten Struktur der Software zeigt sich die Visualisierung mittels Layout-based Clustering als hilfreich, da hier die globalen Zusammenhänge dargestellt werden. So lässt sich mit Hilfe dieser intuitiven Darstellung die Struktur der Software einfach analysieren und erklären.

Es existiert in der untersuchten Motorsteuergeräte-Software eine hohe Anzahl an Feature-Interaktionen. Dies war zu erwarten und lässt sich damit erklären, dass tatsächlich viele Funktionalitäten nicht unabhängig voneinander sind. Anwendungsbedingt müssen in der Motorsteuergeräte-Software viele Informationen ausgetauscht werden und dadurch entsteht eine hohe Anzahl an Abhängigkeiten durch gemeinsame Variablen. Die Feature-Interaktionen entstehen somit bewusst und werden entsprechend koordiniert. Die Koordination einer Feature-Interaktion kann einerseits durch ein separates Koordinator-Modul erfolgen, welches im Rahmen dieser Arbeit als eigener Knoten dargestellt wird, oder die Koordination ist Teil des Features selbst und somit integriert in jenes.

Zudem war es Ziel dieser Arbeit zu ermitteln, ob es sich bei den erkannten Feature-Interaktionen in der vorliegenden Motorsteuergeräte-Software um bekannte bzw. gewollte Feature-Interaktionen handelt. Dies wurde überprüft und konnte in sämtlichen Fällen bestätigt werden.

7 Fazit und Ausblick

In diesem Kapitel wird schließlich ein Fazit gezogen und ein Ausblick auf mögliche Weiterarbeit gegeben.

Im Rahmen dieser Arbeit wurden Coupling-Metriken dazu verwendet, um potentielle Feature-Interaktionen zu erkennen. Dies wurde anhand einer realen Motorsteuergeräte-Software durchgeführt. Dabei handelt es sich um ein eingebettetes Echtzeit-Softwaresystem, welches in der hardwarenahen Sprache C realisiert ist.

Mittels der Coupling-Metrik $RefW$ ist es möglich, bestimmte Arten von potentiellen Feature-Interaktionen in der Motorsteuergeräte-Software zu erkennen. Bei den Arten von Feature-Interaktionen, die erkannt werden können, handelt es sich um direkte Interaktionen zwischen zwei Features. Bei Feature-Interaktionen über Koordinatoren lässt die Metrik nur Hinweise auf selbige zu, welche manuell auf eine tatsächlich vorliegende Feature-Interaktion überprüft werden müssen.

Die Ergebnisse aller untersuchten Coupling-Metriken, $RefW$, CBF und $DCpF$, können außerdem dazu genutzt werden, um den Grad der Kopplung zu quantifizieren und damit eine grobe Aussage über den Arbeitsaufwand im Falle einer Weiterentwicklung oder einer Integration eines Features in ein bestehendes Softwaresystem zu geben.

Bei sämtlichen erkannten Feature-Interaktionen in der vorliegenden Motorsteuergeräte-Software handelt es sich um bekannte bzw. gewollte Feature-Interaktionen.

Mit der im Rahmen dieser Arbeit entwickelten Tool-Chain wurde die Grundlage dafür gelegt, um Untersuchungen von Abhängigkeiten und Feature-Interaktionen in der Motorsteuergeräte-Software effizient zu ermöglichen.

Mittels der Ergebnisse dieser Arbeit in Form von Metriken und Visualisierungen kann eine Diskussion über die Struktur der Motorsteuergeräte-Software und deren Zusammenhänge geführt werden.

Diese Erkenntnisse beschränken sich jedoch nur auf die betrachtete Motorsteuergeräte-Software. Es können keine allgemeingültigen Aussagen über die generelle Eignung von Coupling-Metriken zur Erkennung von Feature-Interaktionen in Softwaresystemen getroffen werden.

Wie in diesem Kapitel bereits beschrieben, ist es mittels der Metrik $RefW$ zwar möglich direkte Interaktionen zwischen Features, aber nicht Feature-Interaktionen, die von Koordinatoren aufgelöst werden, zu erkennen. Das Vorhandensein eines Koordinators deutet klarerweise darauf hin, dass hier eine Feature-Interaktion koordiniert wird. Hier muss jedoch aktuell für jede Situation

manuell überprüft werden, ob es sich tatsächlich um Sollwerte der selben Größe handelt, um eine Aussage über das Vorhandensein einer Feature-Interaktion zu treffen. Um auch die Erkennung dieser Art von Feature-Interaktionen zu automatisieren, wird folgendes Konzept vorgeschlagen. Variablen, die bei dieser Art von Feature-Interaktion eine Rolle spielen – Sollwerte – könnten kategorisiert werden, indem jene Variablen, die Sollwerte der selben Größe darstellen, der selben Kategorie zugeordnet werden. Damit wäre eine automatisierte Erkennung möglich, ob es sich bei Abhängigkeiten mit dem selben Koordinator auch um Sollwerte der selben Größe handelt und damit wäre eine automatisierte Erkennung von Feature-Interaktionen in dieser Konstellation möglich.

Im Rahmen dieser Arbeit werden potentielle Interaktionen zwischen Features im Sinne einer Beeinflussung eines Features durch ein anderes untersucht. Wie in Abschnitt 3.1.4 beschrieben, werden in dem erstellten Graphen zwar die Richtungen der Abhängigkeiten abgebildet, jedoch wird diese Information von den Metriken nicht genutzt. Demtentsprechend geben die Metriken bei den jeweiligen Abhängigkeiten auch keine Auskunft darüber, von welchem der beiden Features die Beeinflussung ausgeht. Um an dieser Stelle das Ausmaß des Einflusses einzelner Features oder Koordinatoren zu ermitteln, wird vorgeschlagen ebenfalls Coupling-Metriken zu untersuchen, die diese Richtungsinformation verwenden.

A Ergebnisse der Coupling-Metriken

In Tabellen A.1, A.2 und A.3 sind alle Werte der Coupling-Metriken für die untersuchten Knoten in Form von Features und Koordinatoren aufgelistet.

	Feature_1	Feature_2	Feature_3	Feature_4	Feature_5	Feature_6	Feature_7	Feature_8	Feature_9	Feature_10	Coordinator_1	Coordinator_2	Coordinator_3	Coordinator_4
Feature_1	-	0	0	0	0	0	4	0	0	0	10	1	0	0
Feature_2	0	-	0	0	0	0	1	1	0	0	2	1	0	0
Feature_3	0	0	-	0	2	3	0	5	1	2	5	3	1	2
Feature_4	0	0	0	-	1	0	0	0	6	0	1	0	2	0
Feature_5	0	0	2	1	-	1	0	1	0	4	1	0	2	1
Feature_6	0	0	3	0	1	-	0	0	0	1	2	0	0	0
Feature_7	4	1	0	0	0	0	-	0	0	0	5	1	0	0
Feature_8	0	1	5	0	1	0	0	-	0	2	11	2	0	0
Feature_9	0	0	1	6	0	0	0	0	-	0	1	0	7	1
Feature_10	0	0	2	0	4	1	0	2	0	-	1	0	0	2
Coordinator_1	10	2	5	1	1	2	5	11	1	1	-	0	1	0
Coordinator_2	1	1	3	0	0	0	1	2	0	0	0	-	0	0
Coordinator_3	0	0	1	2	2	0	0	0	7	0	1	0	-	0
Coordinator_4	0	0	2	0	1	0	0	0	1	2	0	0	0	-

Tabelle A.1: Ergebnisse der Coupling-Metrik $RefW$ der untersuchten Features und Koordinatoren.

Knoten v	$CBF\,(v)$
Feature_1	3
Feature_2	4
Feature_3	9
Feature_4	4
Feature_5	8
Feature_6	4
Feature_7	4
Feature_8	6
Feature_9	5
Feature_10	6
Coordinator_1	11
Coordinator_2	5
Coordinator_3	5
Coordinator_4	4

Tabelle A.2: Ergebnisse der Coupling-Metrik CBF der untersuchten Features und Koordinatoren (Knoten).

Knoten v	$DCpF\,(v)$
Feature_1	15
Feature_2	5
Feature_3	24
Feature_4	10
Feature_5	13
Feature_6	7
Feature_7	11
Feature_8	22
Feature_9	16
Feature_10	12
Coordinator_1	40
Coordinator_2	8
Coordinator_3	13
Coordinator_4	6

Tabelle A.3: Ergebnisse der Coupling-Metrik $DCpF$ der untersuchten Features und Koordinatoren (Knoten).

Abbildungsverzeichnis

Tabellenverzeichnis

Literaturverzeichnis

[1] Edward A. Lee. Embedded software. In *Advances in Computers*, page 2002. Academic Press, 2002.

[2] Steve Heath. *Embedded Systems Design.* Elsevier Science, second edition, December 2002.

[3] Ian Sommerville. *Software engineering.* Pearson international edition. Pearson, Boston, Mass. [u.a.], 9., internat. edition, 2011.

[4] Konrad Reif. *Automobilelektronik: Eine Einführung für Ingenieure.* Springer Fachmedien Wiesbaden, Wiesbaden, 2014.

[5] Hermann Kopetz. *Real-time systems : design principles for distributed embedded applications.* Real-time systems series. Springer, New York, second edition, 2011.

[6] Karl-Heinz Dietsche and Konrad Reif. *Kraftfahrtechnisches Taschenbuch.* Springer Vieweg, Wiesbaden, 28., revised and extended edition, 2014.

[7] Konrad Reif. *Ottomotor-Management: Steuerung, Regelung und Überwachung.* Springer-Verlag, 2015.

[8] Sven Dominka, Dominik Ertl, Michael Dübner, Romana Wiesinger, and Hermann Kaindl. Taming and optimizing feature interaction in software-intensive automotive systems. In *2018 IEEE 1st International Conference on Industrial Cyber-Physical Systems (in Druck)*. IEEE, May 2018.

[9] Tobias Häberlein. *Hardwarenahe C-Programmierung.* Vieweg+Teubner, Wiesbaden, 2011.

[10] Kyo Kang, Sholom Cohen, James Hess, William Novak, and A. S. Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Carnegie-Mellon University, Pittsburgh Pa., Software Engineering Institute, 1990.

[11] Sven Apel, Christian Lengauer, Bernhard Möller, and Christian Kästner. An algebra for features and feature composition. In *Proceedings of the International Conference on Algebraic Methodology and Software Technology (AMAST), volume 5140 of Lecture Notes in Computer Science*, pages 36–50. Springer-Verlag, 2008.

[12] Pamela Zave. An experiment in feature engineering. In Annabelle McIver and Carroll Morgan, editors, *Programming Methodology*, pages 353–377. Springer New York, New York, NY, 2003.

[13] Sven Apel, Don Batory, Christian Kästner, and Gunter Saake. *Feature-Oriented Software Product Lines: Concepts and Implementation.* Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[14] Andreas Vogelsang and Steffen Fuhrmann. Why feature dependencies challenge the requirements engineering of automotive systems: An empirical study. In *2013 21st IEEE International Requirements Engineering Conference (RE)*, pages 267–272. IEEE, July 2013.

[15] Sergiy Kolesnikov, Judith Roth, and Sven Apel. On the relation between internal and external

feature interactions in feature-oriented product lines: A case study. In *Proceedings of the 6th International Workshop on Feature-Oriented Software Development*, FOSD '14, pages 1–8, New York, NY, USA, 2014. ACM.

[16] Dominik Ertl, Sven Dominka, and Hermann Kaindl. Using a mediator to handle undesired feature interaction of automated driving. In *2013 IEEE International Conference on Systems, Man, and Cybernetics*, pages 4555–4560. IEEE, October 2013.

[17] Sven Apel and Dirk Beyer. Feature cohesion in software product lines: an exploratory study. In *Software Engineering (ICSE), 2011 33rd International Conference on*, pages 421–430. IEEE, 2011.

[18] Stefan Kramer and Hermann Kaindl. Coupling and cohesion metrics for knowledge-based systems using frames and rules. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 13(3):332–358, 2004.

[19] Helmut Balzert. *Lehrbuch der Softwaretechnik: Basiskonzepte und Requirements Engineering.* Spektrum Akademischer Verlag, Heidelberg, third edition, 2009.

[20] Alexander Schatten, Markus Demolsky, Dietmar Winkler, Stefan Biffl, Erik Gostischa-Franta, and Thomas Östreicher. *Best Practice Software-Engineering: Eine praxiserprobte Zusammenstellung von komponentenorientierten Konzepten, Methoden und Werkzeugen.* Spektrum Akademischer Verlag, Heidelberg, 2010.

[21] Pierre Bourque and Richard E. Fairley. *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0.* IEEE Computer Society Press, Los Alamitos, CA, USA, 3rd edition, 2014.

[22] Mark Stefik. *Introduction to Knowledge Systems.* Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1995.

[23] Lionel C. Briand, John W. Daly, and Jürgen Wüst. A unified framework for coupling measurement in object-oriented systems. *IEEE Transactions on software Engineering*, 25(1):91–121, 1999.

[24] Jens Meinicke, Thomas Thüm, Reimar Schröter, Fabian Benduhn, and Gunter Saake. An overview on analysis tools for software product lines. In *Proceedings of the 18th International Software Product Line Conference: Companion Volume for Workshops, Demonstrations and Tools-Volume 2*, pages 94–101. ACM, 2014.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass die vorliegende Arbeit gemäß dem Code of Conduct, insbesondere ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel, angefertigt wurde. Die aus anderen Quellen direkt oder indirekt übernommenen Daten und Konzepte sind unter Angabe der Quelle gekennzeichnet. Die Arbeit wurde bisher weder im In- noch im Ausland in gleicher oder in ähnlicher Form in anderen Prüfungsverfahren vorgelegt.

Wien, April 2018

Romana Wiesinger