Technische Hochschule **Rosenheim**

Fakultät für Informatik

Studiengang Software- und Systems-Engineering

Erkennung von Design Patterns in Quellcode durch Machine Learning

Master Thesis

von

Mehmet Aslan

Datum der Abgabe: 18.02.2024
Erstprüfer: Prof. Dr. Marcel Tilly
Zweitprüfer: Prof. Dr. Kai Höfig

Eigenständigkeitserklärung / Declaration of Originality

Hiermit bestätige ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken (dazu zählen auch Internetquellen) entnommen sind, wurden unter Angabe der Quelle kenntlich gemacht.

I declare that I have authored this thesis independently, that I have not used other than the declared sources / resources, and that I have explicitly marked all material which has been quoted either literally or by content from the used sources.

München, den 18.02.2024

Mehmet Aslan

Kurzfassung

Design Patterns oder Entwurfsmuster sind in der Software-Entwicklung gängige Lösungsansätze für wiederkehrende Probleme. Von Software-Entwicklern werden diese eingesetzt, um Probleme in der Implementierung oder in der Software-Architektur zu lösen, deren Lösungsweg bereits bekannt sind und für die jeweilige Situation abgeändert werden. Die wohl bekanntesten Design Patterns sind die 23 Entwurfsmuster nach Gamma et al. Im weiteren Verlauf der Software-Entwicklung sind die eingesetzten Entwurfsmuster aufgrund iterativer Änderungen und steigender Komplexität im Quellcode nicht mehr einfach wiederzufinden. Deswegen liegt der Fokus dieser Masterthesis auf der Etablierung eines Prozesses, welcher mit Machine Learning in der Lage ist, Design Pattern im Quellcode zu erkennen.

Das Ziel dieser Arbeit ist es, die Entwurfsmuster Singleton, Observer, Command und Adapter zu identifizieren. Im Kontext dieser Arbeit werden diese als Strukturen definiert, die in Summe von Rollen dargestellt werden. Jedes dieser Rollen übernimmt im Kontext des jeweiligen Design Patterns unterschiedliche Aufgaben und Verantwortungen. Dazu wird ein Klassifizierer trainiert, welches Rollen innerhalb dieser Entwurfsmuster durch Code-Metriken erkennt. Anhand der Klassifikation der Rollen wird im finalen Schritt ein Übereinstimmungswert berechnet, der angibt, mit welcher Zuversicht es sich hierbei um das jeweilige Entwurfsmuster handelt. Zum Schluss wird die Klassifikationsleistung der hier vorgestellten Methode anhand der Metriken *Precision*, *Recall* und *f1* evaluiert.

Schlagworte: Software-Entwicklung, Machine Learning, Design Patterns

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Motivation

1.1 Einführung in Design Patterns

Entwurfsmuster oder Design Patterns sind bewährte Lösungsansätze für wiederkehrende Probleme, die bei der Konzeption der Software-Architektur und während der Implementierung eingesetzt werden können. Dabei dienen diese Entwurfsmuster als eine Art Blaupause, die es Software-Entwicklern ermöglicht, erprobte Lösungsstrategien für häufig auftretende Probleme in der Software-Entwicklung anzuwenden. Durch den Einsatz von etablierten Entwurfsmustern können Software-Entwickler für die Software bei korrekter Anwendung unter anderem erhöhte Wartbarkeit, Wiederverwendbarkeit von Komponenten, Verständlichkeit und Skalierbarkeit ermöglichen, was widerum in qualitativ besserer Software resultiert. Dabei sollte beachtet werden, dass Design Patterns als Vorlage zu betrachten sind. Je nach Einsatzgebiet muss die Anwendung des Entwurfsmusters evaluiert und für den konkreten Fall individualisiert werden. Deshalb existiert keine universelle anwendbare Iteration eines Design Patterns, die unabhängig von Anwendungskontext eingesetzt werden kann. Dies resultiert in variierender Anwendung von Entwurfsmustern, abhängig von jeweiligem Einsatzgebiet. Im weiteren Entwicklungszyklus der Software werden durch neue oder geänderte Anforderungen bereits eingesetzte Implementierungen von Entwurfsmustern modifiziert, entfernt oder neue hinzugefügt. Währenddessen besteht die Gefahr, dass durch mangelnde Dokumentation oder andere Gründe die Entscheidungen, weshalb Entwurfsmuster so eingesetzt sind, wie es eingesetzt worden sind, verloren gehen. Dadurch besteht die Gefahr, dass angewendete Design Patterns im weiteren Verlauf der Entwicklung nicht mehr wiederzuerkennen sind. Deswegen ist die Etablierung eines Prozesses von Vorteil, der in der Lage ist, Implementierungen von Entwurfsmustern aus einem Software-System zu extrahieren und diese konkret zu benennen. Vor allem der Einsatz von Machine Learning für die Klassifizierung ist hier vorteilhaft, wodurch das Potenzial besteht, vorher nicht gesehene Implementierung von Design Patterns zu erkennen. Der Fokus dieser Arbeit besteht darin, solch einen Prozess zu etablieren, welcher für ein gegebenes Set von Quellcode-Dateien mithilfe von Machine Learning ein passendes Entwurfsmuster zuteilt.

1.2 Untersuchungsfragen

Das Ziel dieser Arbeit besteht aus der Etablierung eines Prozesses, womit durch Einsatz von Machine Learning für ein Set von Quellcode-Dateien ein Design Pattern zugeordnet wird. Hierfür dienen eine Menge an Quellcode-Dateien als Eingabe für den Prozess und durch phasenweise Transformationen und Bearbeitungen sollen ein möglichst passendes Entwurfsmuster zugeordnet werden. Um solch einen Prozess zu entwickeln, werden im Kontext dieser Arbeit folgende Fragen gestellt und beantwortet:

RQ1. Welche Design Patterns werden berücksichtigt?

RQ2. Was für ein Datensatz eignet sich für solch einen Prozess?

RQ3. Wonach wird exakt klassifiziert?

RQ4. Welche Merkmale, die aus Quellcode-Dateien extrahierbar sind, eignen sich für Klassifizierung durch Machine Learning Modelle?

RQ5. Welche Klassifizierer eignen sich?

RQ6. Wie wird eine Instanz eines Entwurfsmusters bestimmt?

1.3 Übersicht der Methodik

In diesem Abschnitt wird erläutert, wie die in dieser Arbeit vorgeschlagene Methodik aufgebaut ist. Diese ist dabei in zwei Teile aufzuteilen.

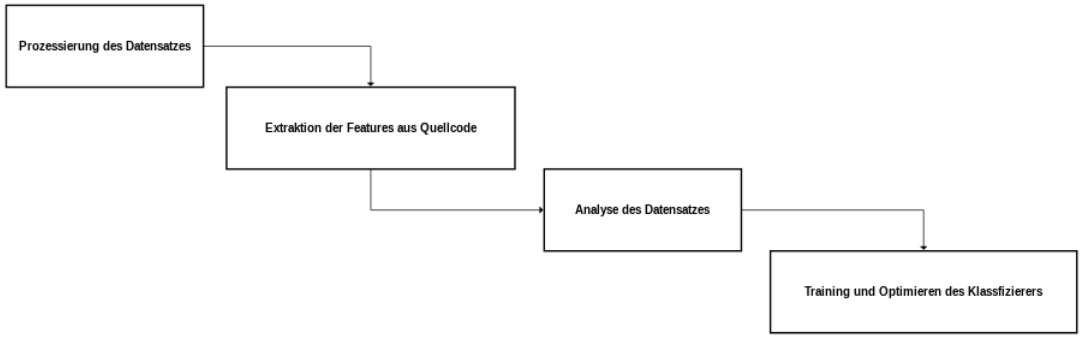

Abbildung 1.1 Prozessabbildung des Trainings des Klassifizierers

Abbildung 1.1 stellt die notwendigen Schritte dar, um einen Klassifizierer zu trainieren. Diese hat die Aufgabe, Quellcodeentitäten eine Rolle im Kontext eines Entwurfsmusters zuzuordnen, das zu klassifizieren ist. Im ersten Schritt werden die notwendigen Informationen wie zugeordnetes Design Pattern und Rolle aus dem Datensatz extrahiert und in einer CSV-Datei abgelegt. Diese CSV-Datei dient zusammen mit dem eigentlichen Quellcode als Eingabe in der Feature-Extraktion in Form von Code-Metriken. Die Einträge aus der CSV-Datei werden dabei mit den berechneten Code-Metriken erweitert. Darauffolgend wird eine explorative Datenanalyse durchgeführt, damit sich ein Überblick über die Datenpunkte geschafft wird. Anhand dieser Auswertung wird bestimmt, welche Entwurfsmuster und

Rollen zur Klassifikation verfügbar sind. Zum Schluss wird der verarbeitete Datensatz verwendet, um einen Klassifizierer zu trainieren. Der Trainingsschritt beinhaltet unter anderem die Optimierung des Modells durch Hyperparameter-Tuning und dessen Validation durch Kreuzvalidierung. Die beste Iteration des Klassifizierers dient dabei als Grundlage bei der Bestimmung eines passenden Design Patterns.

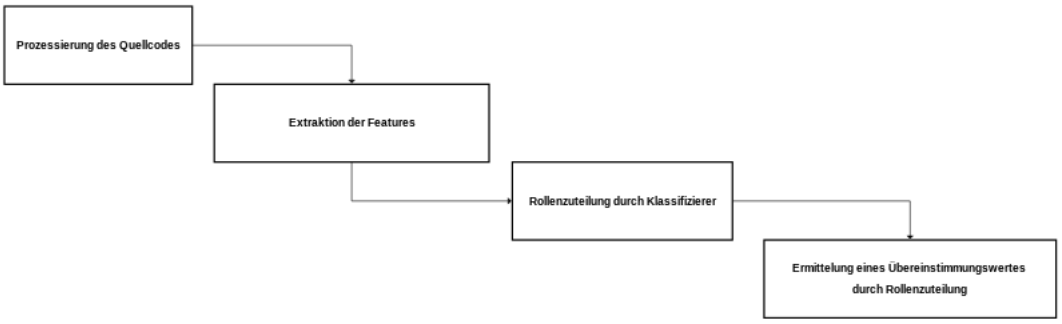

Abbildung 1.2 Prozessabbildung für Zuordnung von Entwurfsmustern

Die Abbildung 1.2 beschreibt, wie auf Basis des vorher trainierten Klassifizierers ein Übereinstimmungswert ermittelt wird, der angibt, welches Entwurfsmuster am ehesten zu den Quellcodeentitäten zuzuordnen ist. Wie im Trainingsprozess des Klassifizierers werden von den Quellcodeentitäten Code-Metriken extrahiert. Diese werden als Eingabe für den Klassifizierer verwendet. Durch die Bestimmung der Rollen wird für jedes betrachtete Design Pattern ein Übereinstimmungswert berechnet. Das Design Pattern mit dem höchsten Übereinstimmungswert ist das Entwurfsmuster, welches am ehesten zu dem Set an Quellcodeentitäten passt.

2 Literaturrecherche

2.1 Design Patterns in der Software-Entwicklung

Entwurfsmuster definieren gängige Lösungsblaupausen für häufig auftretende Probleme in Software-Entwicklung, vor allem in der Design-Phase der Architektur des Software-Systems als auch während der konkreten Implementierung. Jedoch sind diese als Schablone zu verstehen, die für den jeweiligen Einsatzfall angepasst werden müssen. Ein Werk, das das Verständnis von Design Patterns für das objektorientierte Programmieren maßgeblich geprägt, ist die von Gamma et al. verfasste Arbeit "Design Patterns: Elements of Reusable Object-Oriented Software". In diesem wird ein Katalog von 23 Entwurfsmustern definiert, die in drei Kategorien aufgeteilt sind. Dieser Katalog wird von Software-Entwicklern als "Gang of FourËntwurfsmuster bezeichnet. Gamma et al. definieren folgende Elemente für die Identifikation eines Entwurfsmusters:[Gam94, S. 3]

- **Pattern Name**: Der Name des Entwurfsmusters beschreibt in wenigen Worten, welches das zu lösende Problem, die Lösung und welche Folgen dessen Einsatz mit sich bringt. Durch die Einführung eines Bezeichners wird eine Schicht der Abstraktion hinzugefügt, welche Verständnis und Dokumentation des Design Patterns vereinfacht.
- **Problem**: Das Problem beschreibt, wo das Entwurfsmuster angewendet werden soll. Dabei kann es sich um ein konkretes Entwurfsproblem, Klassen- oder Objektstrukturen oder eine Liste von Bedingungen handeln, die zu erfüllen sind.
- **Solution**: Das Lösungselement beschreibt die Beziehungen, Verantwortlichkeiten und Zusammenarbeit der einzelnen Elemente, die die Struktur des Design Patterns definieren. Dabei werden diese Elemente in Objekte und Klassen, die die Grundbausteine der objektorientierten Programmierung repräsentieren, aufgeteilt und deren Interaktionen miteinander stellen die Verantwortlichkeiten und Beziehungen dar.
- **Consequences**: Die Folgen diskutieren, wie der Einsatz des betrachteten Entwurfsmusters sich auf das Software-System einwirkt und welche Vor- und Nachteile dadurch resultieren. Diese beeinflussen unter anderem die Zeit- und Speicherkomplexität, Erweiterbarkeit, Flexibilität und Portabilität des Software-Systems.

Im Kontext dieser Arbeit werden zu klassifizierende Strukturen, die potenziell einem Design Pattern zugeordnet werden können, als Mikroarchitekturen bezeichnet, die aus einer Menge von interagierenden Komponenten bestehen, denen je eine Rolle zugewiesen wird. Die jeweilige Rolle beschreibt, welche Funktionalität und Verantwortung diese im Kontext der Mikroarchitektur übernimmt und wie diese mit anderen Komponenten interagiert. Als Komponenten mit Rollen werden hier konkrete bzwàbstrakte Klassen oder Schnittstellen definiert, die die erforderte Rolle im Rahmen der Mikroarchitektur erfüllen.

In weiteren Verlauf dieser Sektion werden die drei erwähnten Entwurfsmusterkategorien erläutert und zu dem werden im Kontext dieser Arbeit betrachte spezifische Design Pattern genauer betrachtet.

2.2 Design Pattern Katalog

Creational Design Patterns

Die Kategorie der Creational Design Patterns oder Erzeugungsentwurfsmuster beschäftigt sich mit der Abstraktion des Prozesses der Initialisierung [Gam94, S. 81]. Entwurfsmuster dieser Kategorie fokussieren sich auf die Unabhängigkeit, wie Objekte erstellt, zusammengesetzt und repräsentiert werden. Die Entwurfsmuster dieser Kategorie mit Fokus auf Klassen nutzen den Mechanismus der Vererbung, um zu beeinflussen, wie Komponenten instantiiert werden, während dahingegen Design Patterns mit einem Fokus auf Objekten die Instanziierung auf andere Objekte delegieren. Creational Design Patterns werden dann bedeutend, wenn sich mit steigender Komplexität des Software-Systems von Vererbung distanziert wird und die Komposition aus einzelnen definierten Objekten mehr an Bedeutung gewinnt [Gam94, S. 81]. Dabei wird das Verhalten einer Komponente auf eine Menge von einzelnen kleineren Objekten delegiert und durch Zusammensetzung innerhalb der Komponente und deren Interaktion das erwünschte Verhalten erzeugt. Dadurch wird die Instanziierung von Software-Komponenten komplexer, da die Instanziierung von mehreren Objekten koordiniert werden muss. Creational Design Patterns liefern hierbei Hilfestellung, weil die exakte Komposition der konkreten Objekte, die Teil der zu instantiierenden Objektes sind und der exakte Prozess der Instanziierung im Inneren des Entwurfsmusters verborgen werden. Nach außen hin sind dahingegen nur die Schnittstellen sichtbar, die das Objekt zur Verfügung stellt, während die interne Logik die Ausführung auf andere Objekte delegiert. Im Kontext dieser Arbeit werden folgende Entwurfsmuster aus der Kategorie der Creational Design Patterns betrachtet:

Singleton

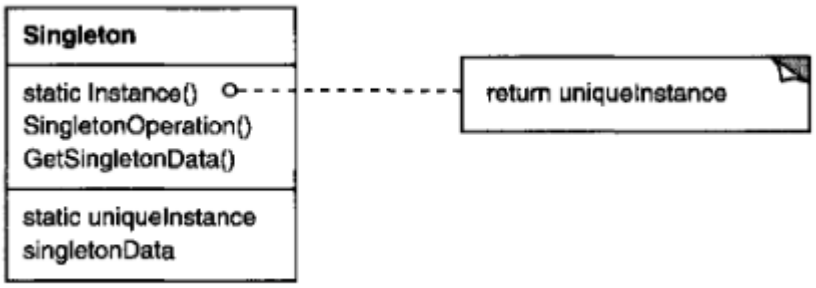

Abbildung 2.1 UML-Diagramm für Singleton

Die Abbildung 2.1 zeigt den strukturellen Aufbau eines Singletons [Gam94, S. 127]. Dabei kann dieser nach Gamma et al. wie folgt beschrieben werden:

Problem

- Exakt eine Instanz der Klasse zur selben Zeit verfügbar; Zugreifbar zu Klienten durch bekannten Zugangspunkt

- einzige Instanz erweiterbar durch Vererbung; Nutzbarkeit der Subklasse ohne Veränderung des Quellcodes auf Klientenseite

Solution

- Zugriff auf Singleton-Instanz durch Instance-Operation

Roles

Singleton

- definiert eine Instance-Operation für Zugriff auf einzige Instanz
- mögliche selbstständige Instanziierung
- Rolle einmal vorkommend

Consequences

- kontrollierter Zugriff auf Instanz
- mögliche Verfeinerung der Operationen und Repräsentation durch Vererbung
- nachträgliche Änderung der Anzahl der Instanzen

Factory Method

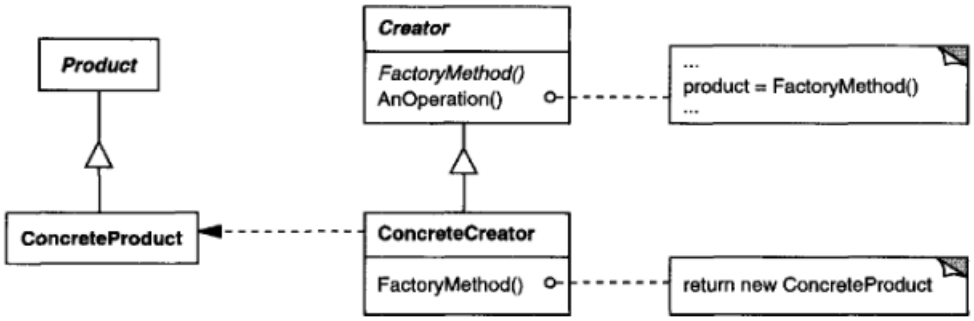

Abbildung 2.2 UML-Diagramm für Factory Method

Abbildung 2.2 beschreibt exemplarisch die Struktur einer Factory Method [Gam94, S. 108]. Diese wird nach Gamma et al. auf folgende Art und Weise beschrieben:

Problem

- Eine Klasse kann die Klasse der Objekte, die es instantiiert, im Voraus nicht erahnen
- Eine Klasse verlangt, dass ihre Unterklassen die von ihr erstellten Objekte spezifizieren
- Klassen delegieren die Instanziierung von Objekten an Helferklassen

Role

Product

- definiert die Schnittstelle der Objekte, die die Factory-Methode erzeugt

ConcreteProduct

- implementiert die Schnittstelle Product

Creator

- deklariert die Factory-Methode, die ein Objekt vom Typ Product zurückgibt
- kann auch eine Standardimplementierung der Factory-Methode definieren, die ein Standard ConcreteProduct-Objekt zurückgibt
- kann die Factory-Methode aufrufen, um ein Product-Objekt zu erzeugen

ConcreteCreator

- überschreibt die Factory-Methode, um eine Instanz eines ConcreteProduct zurückzugeben

Consequences

- Eliminierung der Bindung an applikationsspezifische Klassen; Interaktion durch Product-Schnittstelle
- Flexibilität bei der Instanziierung von Objekten.

Structural Design Patterns

Structural Design Patterns oder Strukturentwurfsmuster fokussieren sich darauf, wie einzelne Klassen und Objekte zusammengesetzt werden können, um größere Strukturen zu erzeugen [Gam94, S. 137]. Entwurfsmuster dieser Kategorie sind vorteilhaft, wenn unabhängig voneinander entwickelte Klassen oder Objekte aus verschiedenen Bibliotheken oder Frameworks miteinander interagieren müssen. Anstatt konkrete Implementierung und Schnittstellen zu nutzen, bedienen sich Structural Design Patterns der Komposition aus Objekten, um neue Funktionalitäten zur Verfügung zu stellen [Gam94, S. 137]. Die dadurch gewonnene Flexibilität ermöglicht das Ändern der Zusammensetzung des Objektes dynamisch zu der Laufzeit, welches mit statischer Komposition durch Klassen nicht möglich ist [Gam94, S. 137]. Im Kontext dieser Arbeit werden folgende Entwurfsmuster aus der Kategorie der Structural Design Patterns betrachtet:

Adapter

Abbildung 2.3 zeigt die Struktur des Adapter-Entwurfsmusters. Dabei wird ein Adapter nach Gamma et al. wie gefolgt beschrieben [Gam94, S. 141]:

.

Problem

- Eine existierende Klasse soll genutzt werden; die Schnittstelle der existierenden Klasse stimmt mit der gebrauchten nicht überein
- Recyclebare Klasse, die mit unabhängigen Klassen kooperiert

Roles

Target

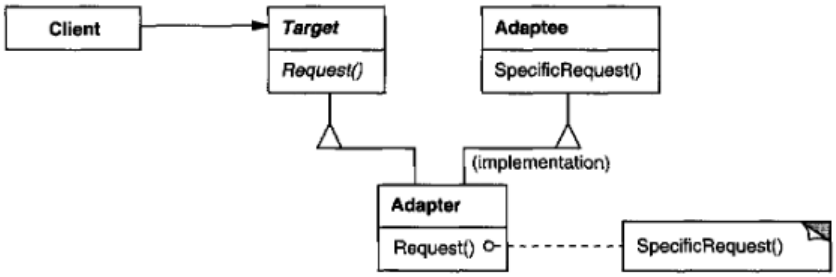

Abbildung 2.3 UML-Diagramm für Adapter

- definiert Domain-spezifische Schnittstelle für das Nutzen durch Clients
- Rolle einmal vorkommend

Client

- kollaboriert mit Objekten, die Target-Schnittstelle implementieren
- Rolle mehrfach vorkommend

Adaptee

- definiert ein bereits existierende Schnittstelle, worauf Adapter angepasst wird
- Rolle mehrfach vorkommend

Adapter

- adoptiert die Schnittstelle von Adaptee zu Target-Schnittstelle
- Rolle mehrfach vorkommend

Consequences

- Adaptierung von Adaptee zu Target-Schnittstelle durch konkrete Adapter-Klasse
- Adapter überschreibt Verhalten von Adaptee-Klasse, da Adapter Subklasse von Adaptee

Behavioral Design Patterns

Behavioral Design Patterns oder Verhaltensentwurfsmuster konzentrieren sich auf Algorithmen und der Zuweisung von Verantwortlichkeiten zwischen Objekten [Gam94, S. 221]. Dabei wird nicht nur Struktur der Entwurfsmuster betrachtet, sondern auch die Kommunikation und Interaktion der Objekte, die Teil des Entwurfsmusters sind. Charakteristisch für Design Patterns dieser Kategorie ist der Fokus auf Verknüpfung der einzelnen Teilobjekte des Entwurfsmusters, anstatt des Kontrollflusses, welcher zur Laufzeit schwer nachvollziehbar sein kann [Gam94, S. 221]. Im Kontext dieser Arbeit werden folgende Entwurfsmuster aus der Kategorie der Behavioral Design Patterns betrachtet:

Command

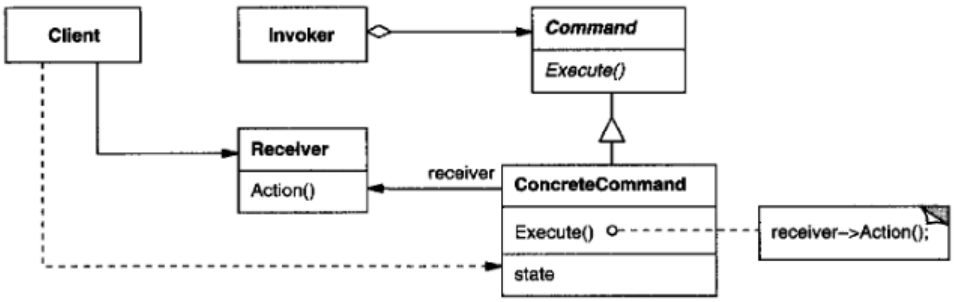

Abbildung 2.4 UML-Diagramm für Command

Abbildung 2.4 zeigt die Struktur des Command-Entwurfsmusters. Dabei wird Command nach Gamma et al. wie gefolgt beschrieben [Gam94, S. 236]:

Problem

- Kapsulierung von Aktion durch parametrisierbare Objekte
- Spezifizierung, Abwarten und Ausführung von Aktionen zu verschiedenen Zeiten;
- Option für Undo; ausgeführte Aktion wird rückgängig gemacht

Roles

Command

- deklariert eine Schnittstelle für das Ausführen von Operationen
- Rolle einmal vorkommend

ConcreteCommand

- definiert eine Bindung zwischen einem Receiver und einer Aktion
- implementiert Execute-Methode durch Ausführen der korrespondierenden Operationen auf Seite von Receiver
- Rolle mehrfach vorkommend

Invoker

- initialisiert die Ausführung eines Commands
- Rolle einmal vorkommend

Client

- instanziiert ConcreteCommand
- Rolle einmal vorkommend

Receiver

- Empfänger von Commands
- Wissen, wie empfangene Commands auszuführen sind
- Rolle einmal vorkommend

Consequences

- Entkoppelung zwischen Objekt, welches Operation initialisiert, und Objekt, das Operation durchführt
- Erweiterung von Manipulation von Verhalten von Command-Klassen durch Vererbung
- Zusammensetzung von Command-Klassen durch andere Command-Klassen
- erleichtertes Hinzufügen von neuen Command-Klassen

Observer

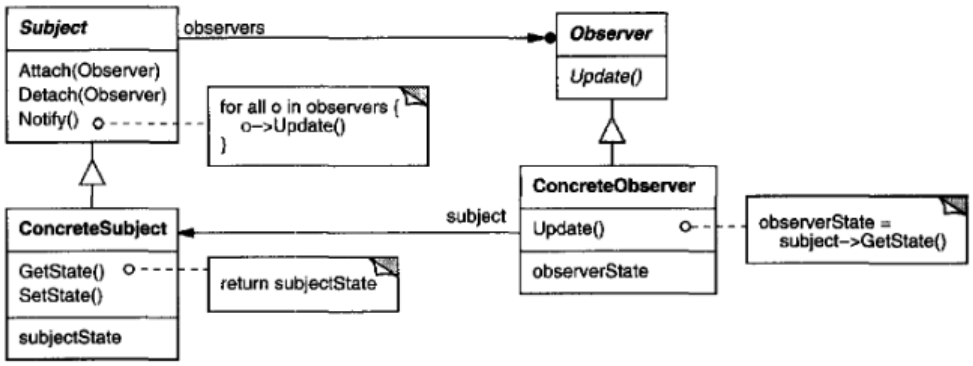

Abbildung 2.5 UML-Diagramm für Observer

Die Abbildung 2.5 zeigt den strukturellen Aufbau eines Observers [Gam94, S. 293]. Dabei kann dieser nach Gamma et al. wie folgt beschrieben werden:

Problem

- Benachrichtigung von Objekten ohne Annahmen über deren Existenz
- Änderung in einem Objekt erfordert Änderungen in abhängigen Objekten; kein Wissen wie abhängige Objekte auf Änderung reagieren

Roles

Subject

- Wissen über dessen Observer; beliebige Anzahl vom Observern an einem Subject möglich

- Schnittstelle für das Hinzufügen und Entfernen von Observern
- Rolle einmal vorkommend

Observer

- definiert Schnittstelle für das Reagieren von Änderungen am Subject seitens der abhängigen Objekte
- Rolle einmal vorkommend

ConcreteSubject

- implementiert Subject-Schnittstelle
- beinhaltet relevanten Zustand von ConcreteObserver-Objekten
- Benachrichtigung von dessen Observer-Objekten bei Änderungen am relevanten Zustand
- Rolle einmal vorkommend

ConcreteObserver

- hält Referenz zu ConcreteSubject-Objekt
- speichert selbst Zustand; konsistent zu dem von Subject
- implementiert Observer-Schnittstelle für das Synchronisieren der Zustände
- Rolle mehrfach vorkommend

Consequences :

- Abstrakte Koppelung zwischen Observer und Subject; Keine Annahmen von Subject über Observer außer Observer-Schnittstelle
- Broadcast an Observer; Benachrichtigung aller registrierten Observer
- Einspielen von unerwarteter Benachrichtigungen

2.3 Herausforderungen und Probleme bei der Erkennung von Design Patterns

In diesem Abschnitt der Arbeit werden mögliche Herausforderungen diskutiert, die beim Entwerfen eines Prozesses für die Erkennung von Entwurfsmustern auftreten können.

Variabilität der Implementierung

Design Patterns stellen in der Software-Entwicklung bewährte Lösungsmuster für bereits begegnete Herausforderungen dar. Aufgrund der abstrakten und wiederverwendbaren Natur der Entwurfsmuster, muss für diese eine konkrete Implementierung definiert werden, die von dem Einsatzfall, Kontext und anderen Faktoren wie verwendeter Programmiersprache, Bibliotheken und Erfahrungsstand des Software-Entwicklers abhängig sind. Dadurch, dass jedes Entwurfsmuster einen konzeptionellen Rahmen darstellt und jede Implementierung von nicht statischen Außenfaktoren beeinflusst wird, resultiert dies in einem breiten Spektrum an Implementierungen für ein gegebenes Entwurfsmuster. Angesichts dessen ist eine starre Definition eines Design Patterns, was als Startpunkt und Referenz für die Erkennung des jeweiligen Entwurfsmusters dienen könnte, nicht möglich. Deshalb ist eine definitive Antwort auf die Frage, ob eine betrachtete Mikroarchitektur eine Instanz eines Entwurfsmusters ist, nicht beantwortbar. Deshalb ist die Antwort von automatisierten Prozessen ein Übereinstimmungswert, der die Ähnlichkeit zu einem Design Pattern beschreibt. Um einen zufriedenstellenden Wert für diese Frage zu liefern, bedarf es eines bereiten Spektrums an Implementierungsvariationen als Referenz für die Erkennung.

Steigende Komplexität von Software-Systemen

Die steigende Komplexität von Software-Systemen stellt eine Herausforderung bei der Erkennung von Entwurfsmustern in Quellcode dar. Dies ist besonders bei langjährigen Software-Projekten der Fall, an denen über die Zeit konstante Änderungen wegen Wartung und neuen bzw. geänderten Anforderungen unterliegen. Diese Art von Software-Projekten tendiert dazu, dass mit der Zeit deren Komplexität zunimmt [Suh10, S. 7]. Bei kleineren Software-Projekten mit geringem Umfang und Komplexität sind Entwurfsmuster leichter zu erkennen und zu implementieren. Dahingegen bei langjährigen Software-Projekten steigt mit wachsender Gesamtkomplexität die Komplexität der angewendeten Entwurfsmuster in deren Quellcode, wodurch die Identifizierung dieser proportional mitsteigt. Entwurfsmuster werden durch diese Entwicklung weiter modifiziert und angepasst, womit diese von den ursprünglichen, leichter zu identifizierbaren Iterationen weiter abweichen. Deshalb beinhaltet die Erkennung von Entwurfsmustern nicht nur die momentane Iteration, sondern auch die Erfassung derer historischen Evolution und die Entwicklung dieser innerhalb der Codebasis.

Iterative Evolution des Quellcodes

Damit ein Software-System seine Anforderungen im Verlauf dessen Lebenszyklus in einer zufriedenstellenden Art und Weise erfüllen kann, muss dieses adaptieren, um diesen Anforderungen gerecht zu werden [Leh96, S. 108]. Dies hat zu Folge, dass dessen Quellcode iterativen Änderungen unterliegt. Ursprüngliche Implementierungen in der Codebasis werden analysiert und es wird überprüft, ob diese ihre Aufgaben zufriedenstellend erfüllen oder nicht. Falls nicht, werden diese so modifiziert, sodass diese Anforderungen auf eine passende Art und Weise erfüllt werden können. Implementierungen der angewandten Design Patterns als Teil des Quellcodes unterliegen ebenfalls dieser Analyse. Diese werden als Teil des Analyseprozesses genauer betrachtet und werden nach Bedarf modifiziert und angepasst. Diese Entwicklung führt zu dem im vorherigen Abschnitt diskutierten Fall, dass Entwurfsmuster von ihrer ursprünglichen leichter zu identifizierbaren Iteration weiter abweichen und die Erkennung von Entwurfsmustern nicht nur die momentane Implementierung, sondern auch die historische Entwicklung berücksichtigt werden muss. Im automatisierten Prozess der Erkennung von Entwurfsmustern kann dieser Aspekt nur bedingt berücksichtigt werden, weil das Einschließen der Historie der betrachtenden Implementierung und dessen Kontext in der Codebasis nicht pauschal und in einer allgemeinen Ansicht betrachtet werden kann.

Mangel an expliziter Dokumentation

Das Erstellen und Warten von Dokumentation für Software-Systeme ist eine Tätigkeit, die von Software-Entwicklern als wichtig eingestuft wird, jedoch wird in diese Tätigkeit relativ wenig Zeit investiert [Zhi15, S, 162]. Dies ist die Folge der Dominanz des agilen Software-Entwicklungsprozesses. In dieser wird die Verwendung von Zeit und Ressourcen für die Dokumentation eher als Verschwendung betrachtet, da diese keinen direkten Mehrwert für die Auslieferung des Software-Produktes an den Endkunden liefert [Zhi15, S. 159]. Da dies das Software-System komplett betrifft, sind Design Patterns in dessen Codebasis ebenfalls betroffen. Diese werden meist nicht direkt gekennzeichnet. Zwar kann durch Nomenklatur und Kontrollfluss indirekt Rückschlüsse auf die potenziellen Entwurfsmuster abgeleitet werden, jedoch erfordert dies konkretes Fachwissen und Erfahrung, die nicht von jedem Software-Entwickler erfüllt werden kann. Bei automatisierten Prozessen für die Erkennung von Entwurfsmustern kann diese berücksichtigt werden, sollte aber nicht als alleiniger Faktor bei dem Identifikationsprozess dienen.

Aufgrund der Variation an Implementierungsmöglichkeiten, Änderungen im Quellcode und Mangel an Dokumentation ist die manuelle Identifikation von Entwurfsmustern in Quellcode ein Problem, das einen gewissen Grad an Mitdenken erfordert. Im nächsten Abschnitt der Arbeit werden bereits entwickelte Verfahren betrachtet, die das Mitdenken bis zu einem gewissen Grad automatisieren.

2.4 Angewandte Ansätze für die Erkennung von Design Patterns

Bei der Erkennung von Design Patterns in Quellcode wurden verschiedene Verfahren entwickelt, die auf unterschiedlichen Methoden beruhen, um das gesetzte Ziel zu erreichen. Yarahmadi et al. führten in ihrer Arbeit eine Untersuchung über die Methoden, die angewandt worden sind, um Design Patterns in Quellcode zu erkennen, durch und kategorisierten diese [Yar20, S. 5805].

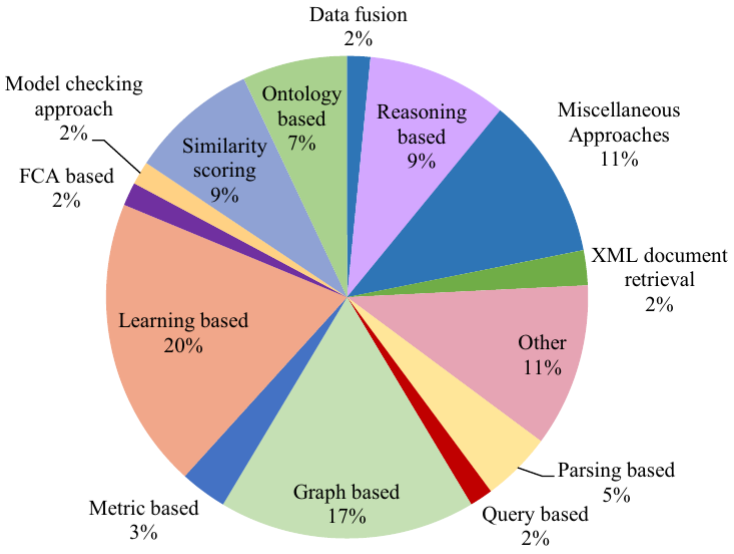

Abbildung 2.6 Verteilung der Kategorien der Ansätze für die Erkennung von Design Patterns in Quellcode

Wie aus Figur 2.6 zu entnehmen ist, wurden die entwickelten Prozesse von Yarahmadi et al. auf eine begrenzte Menge an Kategorien eingestuft. In dieser Sektion der Arbeit werden die vier größten Kategorien aus Figur 2.6 genauer erläutert und es werden exemplarische Arbeiten diskutiert, die zu der jeweilige Kategorie zugeordnet werden.

2.4.1 Graphen-basierende Ansätze

Die Methodik der Reduktion stellt in der Berechenbarkeitstheorie einen Ansatz dar, um Lösungswege für neue unbekannte Probleme zu entwickeln. Dabei wird durch einen Algorithmus das unbekannte Problem in ein bereits gelöstes Problem, dessen Lösungsweg schon vorhanden ist, umgewandelt. In Graphen-basierenden Methoden für die Erkennung von Design Pattern in Quellcode wird diese angewendet, um Quellcode in Graphen zu transformieren und diese Graphen werden als Eingabe für diverse Graphenalgorithmen verwendet.

Formell betrachtet ist ein Graph definiert als [Siu98, S. 9]:

$G = \{V(G), E(G)\}$
mit
G : der zu betrachtende Graph
$V(G)$: Nicht leere Menge von Knoten in G
$E(G)$: Menge an ungeordneten Tupeln von distinkten Elementen von V (G)

Der Quellcode selbst ist als roher Text zu betrachten, welcher verschiedene Entitäten wie Klassen, Objekte und Schnittstellen beinhaltet und definiert, wie diese miteinander interagieren. Als Graph G werden die Entitäten aus dem Quellcode als Knoten $V(G)$ und die Relationen und Interaktionen wie Vererbung oder Methodenaufrufe werden als Kanten $E(G)$ dargestellt. Hierbei stellt Unified Modeling Language (UML) eine in der Software-Entwicklung verbreitete Modellierungssprache dar und definiert verschiedene Arten von Graphen, um Software und andere Systeme zu modellieren. Die Diagramme aus der UML-Domäne werden in diesem Kontext als Graphen aufgefasst, da diese aus einer Menge aus Kanten und Konten bestehen und anhand der obigen Definition als Graphen interpretiert werden können. Eine Diagrammart aus der Domäne, welches eingesetzt wird, um objektorientierte (OO) Software-Systeme zu modellieren, sind Klassendiagramme. Klassendiagramme beschreiben, wie Klassen und deren Relation zueinander im Kontext des Paradigmas der objektorientierten Programmierung aufgefasst werden. Pradhan et al. nutzen Klassendiagramme als Eingabe für ihre entworfene Methode und generieren diese für das zu analysierende Software-System und Implementierungen von Entwurfsmustern, die als Referenz genutzt werden [Pra15, S. 2]. Diese werden als gerichtete Graphen erfasst, wobei die Klassen als Knoten und die Assoziation wie Vererbung zwischen diesen als Kanten aufgefasst werden. Zusätzlich werden die Kanten je nach Art der Assoziation unterschiedlich gewichtet [Pra15, S. 2]. Im weiteren Verlauf werden mögliche Kandidaten aus dem Graphen des Software-Systems extrahiert. Durch den Einsatz der Graphenisomorphie wird zwischen den Subgraphen des Software-Systems und den Referenzgraphen der Entwurfsmuster die normalisierten Kreuzrelation als das Maß der Übereinstimmung berechnet [Pra15, S. 3]. Graphenisomorphie beschreibt, ob zwei Graphen strukturell identisch sind, sodass jede Kante des einen Graphen einer Kante im anderen Graphen entspricht und umgekehrt [Siu98, S. 10]. Die normalisierte Kreuzrelation ist ein Maß, dessen Wertebereich zwischen 0.0 und 1.0 definiert ist. Je näher der Wert an der oberen Grenze 1.0, desto identischer sind die zwei Graphen. Zu der Evaluierung des Prozesses wurden vier Open-Source-Software-Systeme hergezogen, aus welchen fünf existierende Entwurfsmuster zu erkennen sind [Pra15, S. 6]. Dabei wurde von Pradhan et al. dokumentiert, ob ein Entwurfsmuster komplett oder partiell im Quellcode entdeckt wurde. Nach eigener Auswertung von Pradhan et al. wurden die als Referenz genommene Implementierung der Design Patterns mehrfach komplett als auch partiell in der Codebasis der zu dem Test hergezogenen Software-Systeme identifiziert [Pra15, S. 6].

In ihrer Arbeit verfolgen Dongjin et al. ebenfalls die Erkennung von Entwurfsmustern der strukturellen Kategorie in Quellcode durch den Einsatz von Graphen. In Kontext dieser werden wie im vorherigen Verfahren beschriebenen Klassendiagramme als gerichtete gewichtete Graphen eingesetzt. Entitäten wie Klassen, Objekte oder Schnittstellen werden als Knoten und die Assoziation der Entitäten wie Vererbung als gerichtete gewichtete Kanten des Graphen repräsentiert [Yu13, S. 582]. Zudem werden Referenzimplementierungen der zu identifizierenden Entwurfsmuster in gewichtete Klassendiagramme transformiert und innerhalb dieser werden Submuster definiert, die in Summe das Entwurfsmuster repräsentieren [Yu13, S. 580].

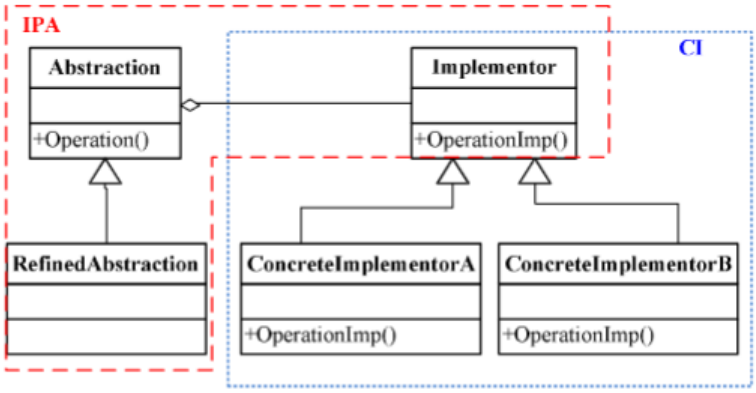

Abbildung 2.7 Referenzklassendiagramm des Bridge Patterns mit Submustern

Beispielhaft spiegelt die Abbildung 2.7 von Dongjin et al. das definierte Referenzklassendiagramm das Bridge Pattern wieder. Innerhalb dieses Klassendiagramms werden die Submuster **IPA** und **CI** festgelegt. Wie von der Abbildung 2.7 zu entnehmen ist, bestehen die Submuster aus einzelnen Entitäten und deren Assoziationen. Innerhalb der Klassendiagramme des Quellcodes, welches als Eingabe dient, wird nach solchen Submustern durch Einsatz von Graphenisomorphie gesucht [Yu13, S. 584]. Im weiteren Verlauf werden identifizierte, relevante Submuster zu einer Kandidatenstruktur verschmolzen. Dabei ist anzumerken, dass nur die Submuster, die im betrachtenden Entwurfsmuster vorkommen, in der Verschmelzung involviert sind. Als finaler Schritt werden durch Analyse des Verhaltensmusters der Kandidatenstrukturen falsch positiv identifizierte Instanzen herausgefiltert [Yu13, S. 584]. Quellcode aus vier Open-Source-Software-Systemen dient als Eingabe für dieses Verfahren [Yu13, S. 585]. Zu Evaluierung der Resultate der Methodik wird die Metrik der *Precision* verwendet. Dabei wird das Verhältnis zwischen der Anzahl der positiv falschen identifizierten Instanzen zu der Summe der Anzahl der positiv falsch und falsch positiv klassifizierten Instanzen gebildet [Yu13, S. 585]. Je höher diese ist, desto besser ist das Ergebnis einzustufen. Nach eigenständiger Evaluierung von Dongjin et al. wird für alle Software-Systeme für alle betrachten strukturellen Entwurfsmuster eine hohe Rate der *Precision* ermittelt [Yu13, S. 586].

2.4.2 Machine Learning Ansätze

Mit der ersten öffentlich zugänglichen Version des Open Source Machine Learning Frameworks TensorFlow am 9. November 2015 wurde der Zugang für praktisch angewandtes Machine Learning für Software-Entwickler erleichtert. Durch TensorFlow wird eine Abstraktionsschicht für das Entwerfen, Trainieren und den Betrieb von Machine Learning Modellen eingeführt. Der vereinfachte Einsatz resultiert mit der Anwendung von Machine Learning in Produkten und in der Forschung. Dies ist auch der Fall für die Erkennung von Design Patterns in Quellcode. Wie aus Abbildung 2.6 zu entnehmen ist, stellen nach der Untersuchung von Yarahmadi et al. Verfahren, die Machine Learning einsetzen, einen signifikanten Teil dar. Im Kontext dieser Aussage werden in diesem Abschnitt ausgewählte Verfahren erläutert, die Machine Learning als Teil des Erkennungsprozesses anwenden. Der Fokus in dieser Sektion liegt neben den angewandten Modellen für die Klassifikation auf dem Format der Eingabe, in der die Quellcode-Dateien für die Klassifikation transformiert werden.

Eine Möglichkeit, um Entitäten aus dem Quellcode wie Klassen oder Schnittstellen und deren Assoziation darzustellen, ist die Darstellung als Klassendiagramme aus der UML-Domäne. In ihrer Methodik nutzen Wang et al. UML-Klassendiagramme als Grundlage und modifizieren diese mit der Inkorporation von Farb- und Symbolcodierungen. Dieses Format benennen Wang et al. *Colored UML* [Wan22, S. 6].

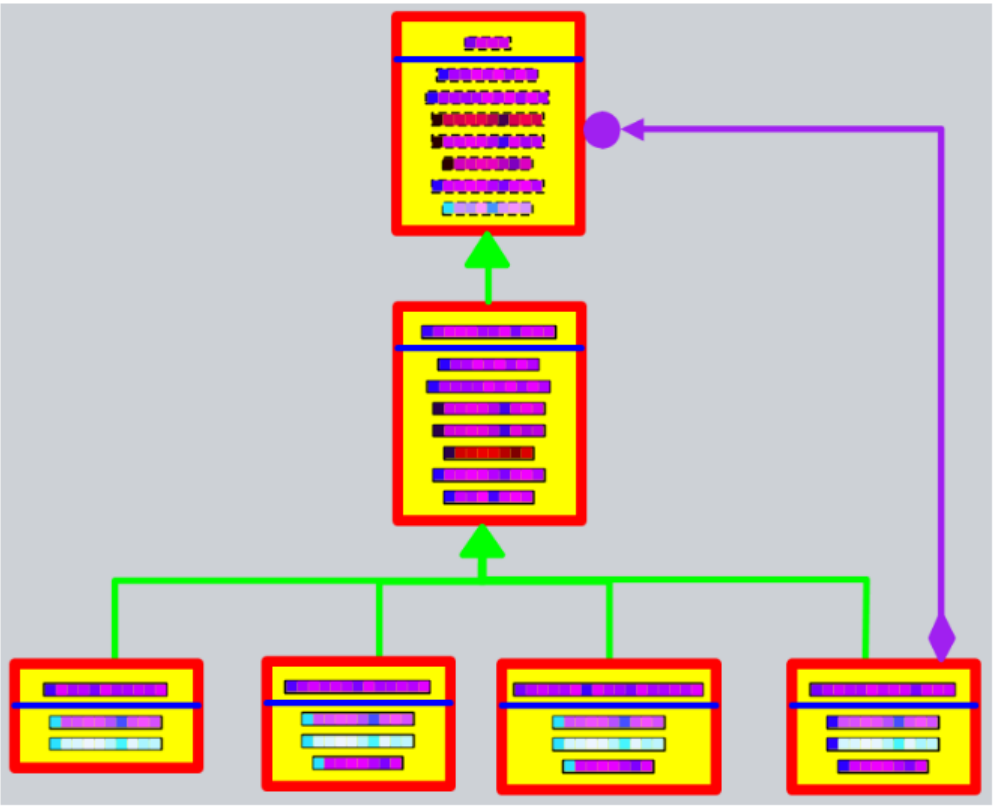

Abbildung 2.8 *Colored UMl* für eine Mikroarchitektur

Die Abbildung 2.8 zeigt von Wang et al. erstelltes Exemplar für *Colored UML* [Wan22, S. 11]. Wie aus der Abbildung 2.8 zu entnehmen ist, werden Bezeichner, Modifizierer und Assoziationen farblich und/oder symbolisch encodiert. Hierbei ist zu erwähnen, dass

die Encodierung von Klassenattributen im Kontext dieser Arbeit von Wang et al, nicht berücksichtigt wird. Die Rechtecke, die in Sektionen aufgeteilt sind, beschreiben die Klassen im Klassendiagramm. Die obere Sektion beinhaltet den Klassenbezeichner und dessen Modifizierer, während die untere Sektion die Methodennamen und deren Modifizierer beinhaltet. Jeder Bezeichner wird als Sequenz von Charakteren aufgefasst, in welchem für jeden Charakter algorithmisch eine Farbe aus dem Rot-Grün-Blau-Farbraum ermittelt wird [Wan22, S. 9, S. 10]. Modifizierer für Methoden werden als Teil der Charaktersequenz betrachtet und mitencodiert, während die Modifizierer für Klassen im Rand des Rechtecks encodiert werden, welches den Klassenbezeichner umschließt [Wan22, S. 6]. Die Kanten des Klassendiagramms, welche die Assoziationen zwischen Entitäten darstellen, werden in *Colored UML* farblich annotiert und mit symbolisch an den Enden der Kanten je nach Art der Assoziation encodiert [Wan22, S. 6].

Zunächst werden im ersten Schritt des Prozesses aus Quelldateien durch Software-Werkzeuge UML-Klassendiagramme erstellt, welche im nächsten Schritt in *Colored UML* umgewandelt werden. Auf den encodierten Quelldaten wird der Bildklassifizierer VGGNet angewendet, welches Features aus der Eingabe als numerischen Vektor mit einer Länge von 1000 extrahiert. Im finalen Schritt dient dieser Feature-Vektor als Eingabe für eine Support Vector Machine, welches die Klassifikation übernimmt, zu welchem Entwurfsmuster die Eingabe zugeordnet werden kann [Wan22, S. 13]. Das hier präsentierte Modell wird für zwölf Design Patterns trainiert [Wan22, S. 15] Für das Training des Modells und die Evaluierung der Resultate des Verfahrens werden drei Open Source Software-Systeme verwendet. Zusätzlich wird das Verfahren mit drei nicht Machine Learning-Verfahren auf *Precision* und *Recall* verglichen [Wan22, S. 20]. Nach eigener Evaluierung von Wang et. al weist das ihrerseits entwickelte Verfahren verglichen zu den anderen Methoden ähnliche oder bessere Klassifizierungsleistung auf [Wan22, S. 22].

Eine weitere Möglichkeit, um Entwurfsmuster im Quellcode zu ermitteln, ist die Extraktion von Metriken und Maßen aus dem Quellcode, die typisch für eine Instanz des betrachteten Design Pattern sind. In ihrer Arbeit fokussieren sich Uchiyama et al. auf diesen Punkt und definieren einen Katalog aus Metriken und Maßen, der als Eingabe für einen Klassifizierer verwendet werden. In dieser Arbeit definieren Uchiyama et al. Design Patterns als eine Summe von Strukturen, die im Kontext des Entwurfsmusters eine Rolle zugeordnet werden [Uch14, S. 3]. Dabei limitieren sich Uchiyama et al. auf die Erkennung von fünf Entwurfsmustern mit insgesamt 12 Rollen im Quellcode. [Uch14, S. 4]. Für diese Rollen werden Metriken und Maße ermittelt, die diese charakteristisch beschreiben. Um die Elemente des Katalogs zu bestimmen, wird in dieser Arbeit die Goal-Question-Metric Methode angewendet [Uch14, S. 4]. Hierbei werden gezielt Fragen gestellt, mit dem Ziel, die jeweilige Rolle zu bestimmen. Für die Beantwortung dieser Fragen werden Metriken bzw. Maße definiert und als Teil des Katalogs aufgenommen. Um die zwölf Rollen zu bestimmen, definieren Uchiyama et. al folgende Metriken [Uch14, S. 7]:

Tabelle 2.1 Metriken und Maße für Rollen nach Uchiyama et al.

Abkürzung	Beschreibung
NOF	Anzahl der Felder
NSF	Anzahl der statischen Felder
NOM	Anzahl der Methoden
NSM	Anzahl der statischen Methoden
NOI	Anzahl der implementierten Schnittstellen
NOAM	Anzahl abstrakter Methoden
NORM	Anzahl der überschriebenen Methoden
NOPC	Anzahl der privaten Konstruktoren in der Klasse
NOTC	Anzahl der Konstruktoren mit Objektparametern
NOOF	Anzahl der an Feldern mit Objekttypen
NCOF	Anzahl der anderer, die die Klasse/Schnittstelle als Feld referenzieren
NMGI	Anzahl der Methoden, die Instanzen generieren

Im ersten Schritt ihrer Methode extrahieren Uchiyama et al. die aus der Tabelle 2.1 definierten Metriken für jede Entität aus den Quelldateien. Diese Metriken werden als Vektor aufgefasst und diese werden als Eingabe für einen Klassifizierer verwendet, welche die Rollenzuweisung übernimmt [Uch14, S. 5]. Bei dem in dieser Methodik angewandten Klassifizierter handelt es sich um ein neuronales Netzwerk [Uch14, S. 4]. Die Ausgabe des Klassifizierers sind Werte, die angeben, mit welcher Sicherheit der Feature-Vektor zu der jeweiligen Rolle zugeordnet werden kann [Uch14, S. 5]. Anhand dieser Ausgabe werden die Rollen mit dem höchsten Konfidenzwert als Eingabe für die Bestimmung des Entwurfsmusters genommen. Unter der Berücksichtigung der Relationen der Rollen in dem Entwurfsmuster wird ein Übereinstimmungswert ermittelt, der angibt, mit welcher Wahrscheinlichkeit die Rollen zu dem jeweiligen Design Pattern passen. [Uch14, S. 6]. Je höher dieser Wert, desto höher die Konfidenz, dass es sich hierbei um das Entwurfsmuster handelt. Für das Training ihres Klassifizierers nutzen Uchiyama et al. 60 Instanzen aus klein-skalierten Software-Systemen und 158 aus drei größeren Open Source Software-Systemen. [Uch14, S. 7] Für die Evaluierung der Methode bedienen sich Uchiyama et. al den Metriken der *Precision* und *Recall* und ermitteln diese für ihr Verfahren. Dabei werden für Instanzen aus klein-skalierten Software-Systemen bessere *Precison-* und *Recall*-Werte erzielt, verglichen zu den Werten für Instanzen aus den

Open Source Software-Systemen [Uch14, S. 8]

Durch Charakterisierung der Klassen, Schnittstellen und Objekte durch Metriken und Maße sind zwar Rückschlüsse auf Struktur und Verhalten möglich, jedoch wird der lexigrafische und syntaktische Aspekt nicht mit berücksichtigt. Um dagegenzuwirken, präsentieren Nazar et. al in ihrer Arbeit ihre Methode namens DPD_F, die diese Aspekte des Quellcodes mitberücksichtigt [Naz20, S. 1]. Initial wird aus dem Quellcode durch statische Codeanalyse folgende Metriken extrahiert [Naz20, S. 5]:

Tabelle 2.2 Extrahierte Features für DPD_F

Featurebezeichner	Beschreibung
ClassName	Name der Java Klasse
ClassModifiers	Public, Protected und Private-Schlüsselwörter
ClassImplements	Binäres Features (0/1), falls eine Schnittstelle implementiert wird
ClassExtends	Binäres Features (0/1), ob Klasse von einer anderen vererbt
MethodName	Methodenname in der Klasse
MethodReturnType	Typ des Rückgabewerts einer Methode
MethodBodyLineType	Art des Ausdrucks (z.B Variablenzuweisung, Boolean-Ausdruck)
MethodNumVariables	Anzahl der Variablen/Attribute in der Klasse
MethodNumMethods	Anzahl der Methodenaufrufe in der Klasse
MethodsNumLine	Anzahl der Zeilen in Methode
MethodIncomingMethod	Anzahl an Methoden, die in eine Methode aufruft
MethodIncomingName	Name der Methoden, die von der Methode aufgerufen werden
MethodOutgoingMethod	Anzahl ausgehender Methoden
MethodOutgoingName	Name der ausgehenden Methoden

Die ersten vier in der Tabelle 2.2 aufgelisteten Features werden pro Klasse ermittelt, während die restlichen elf für jede Methode in der Klasse bestimmt werden. Jedes dieser Features wird als eine Auflistung von Schlüssel-Werte-Paaren in natürlicher Sprache formuliert. Da die Ermittelung der Metriken für jede Methodendefinition durchgeführt wird, werden Features der Klasse in der Methodenevaluierung wiederholt. Jede Evaluierung wird als Eintrag in einer Textdatei zusammengefasst. Das dabei entstehende Format wird von Nazar at al. als Syntactic and Lexical Representation (SSLR) benannt [Naz20, S. 1]. Um dieses Format in eine numerische Form zu bringen, wird SSLR als Eingabe für das Embedding-Modell Word2Vec verwendet, welches für jedes Token in der Eingabe durch Einbezug der umgebenden Tokens einen numerischen Vektor mit einer Länge von 100 determiniert, welches das jeweilige Token repräsentiert [Naz20, S. 6]. Die resultierenden Vektoren dienen als Eingabe für einen Random Forest Classifier, welches das meist passende Entwurfsmuster bestimmt [Naz20, S. 7]. Für das Training der Modelle und Evaluierung von DPD_F wird ein eigener Korpus angelegt, bestehend aus Quelldateien aus *Github Java Corpus*. Für die Evaluation und das Training werden zwölf Entwurfsmuster mit je 100 Instanzen aus dem Korpus genutzt, die durch Crowdsourcing identifiziert wurden [Naz20, S. 4]. Die Beurteilung der Resultate erfolgt durch die Bestimmung der *f1-*, *Precision-* und *Recall*-Werte wird für jedes betrachtete Design Pattern. Nach eigener Evaluation erreicht ihre Methode mit dem selbsterstellten Datensatz im Durchschnitt einen *Precision*-Wert von über 80% und einen *Recall*-Wert von 79% [Naz20, S. 8].

2.4.3 Query Ansätze

Eine weitere Möglichkeit, um Design Patterns in Quellcode zu identifizieren, ist durch das Stellen von Anfragen an das Software-System. Die Antwort auf die gestellte Anfrage ist dabei solch eine, die die in der Anfrage formulierten Bedingungen am besten befriedigt. Solch einen Ansatz verfolgen Mäder et al. in ihrer Methodik. Dabei liegt der Fokus in ihrer Methode auf Annotationen im Quellcode, die Aufschlüsse auf mögliche Entwurfsmuster geben. In ihrer Arbeit definieren Mäder et al. Annotation als Teil des Quellcodes, die keine direkte Einwirkung auf Programmsemantik haben [Mä10, S. 521] und bieten gleichzeitig zusätzliche Informationen für statische Analyse durch Software-Werkzeuge und für das Verständnis des Software-Systems an. Initial wird ein Katalog von Annotationen definiert, welche manuell von Software-Entwicklern in Quellcode des Software-Systems aufgenommen wird [Mä10, S. 521].

(1) @abstract {notification |[interface, access_to_subsystem]|state_management|list_traversal|object_identity| handling},
(2) @compose {object} from {different_objects|related_objects},
(3) @decouple {receiver} from {sender},
(4) @object {instances} {share_by_introducing} {state_handlers| intrinsic_state|central_instance},
(5) @decouple {implementation} by {dynamic variation lists},
(6) @decouple {sender} from {receiver},
(7) @provide {handlers} for {requests|expressions},
(8) @ instantiation {eager|lazy|replaceable}.

Abbildung 2.9 Ausschnitt an Annoation von Guhlam et al.

Abbildung 2.9 zeigt einen Ausschnitt des Annotationskatalogs, welcher im Kontext der Arbeit definiert wird. Dabei sind diese Annotationen so bestimmt, dass diese Informationen über die Struktur und Verhalten des Software-Artefakts widerspiegeln. Der annotierte Quellcode wird im weiteren Verlauf durch das Software-Werkzeug Enterprise Architekt prozessiert. Dieses Werkzeug generiert ein Modell des Quellcodes und inkludiert die Beziehungen zwischen Software-Artefakten [Mä10, S. 521]. Informationen über Aggregation, Delegation und Freundschaften sind nicht Teil dieses Modells [Mä10, S. 521]. Um die Existenz von Entwurfsmustern zu erkennen, wird ein Katalog an Regeln definiert, die verschiedene Aspekte von Design Patterns durch Mitberücksichtigung der Annotationen widerspiegelt [Mä10, S. 523]. Diese Regeln werden in SQL-Anfragen und reguläre Ausdrücke transformiert und in Sets von Regeln aufgeteilt, die auf die Existenz des zugewiesenen Design Patterns überprüfen [Mä10, S. 523]. Zu der Evaluierung ihrer Methode vergleichen Mäder et al. ihre Methode mit zwei anderen Verfahren. Dabei wird verglichen, welche Entwurfsmuster diese erkennen [Mä10, S. 525]. Konkrete Metriken sind zu dem Zeitpunkt des Verfassens ihrer Arbeit nicht verfügbar [Mä10, S. 525].

Anstatt die Modellierung an ein professionelles Software-Werkzeug zu delegieren, entwerfen Stencel et al. ein eigenes Metamodell, wonach Entitäten aus dem Quellcode und ihre Relationen zueinander modelliert werden [Ste08, S. 27]. Dabei besteht das Metamodell aus Kernelementen wie types, attributes operations und instances und strukturellen und verhaltensbezogenen Relationen zwischen diesen Kernelementen. Sowohl die Kernelemente als auch die Relationen in diesem Metamodell sind möglichst ähnlich nach dem aus der objektorientierter Programmierung bekannten Verständnis definiert [Ste08, S. 27]. Jedes Design Pattern, das in dieser Arbeit betrachtet wird, wird von Stencel et al. in ihrem Metamodell encodiert [Ste08, S.28 - 29]. Diese dienen als Referenz für die Erkennung des jeweiligen Entwurfsmusters im Quellcode und werden als Grundlage für SQL-Anfragen genutzt [Ste08, S. 29]. In der Implementierung der Methode wird durch Strukturanalyse, Analyse des Datenflusses und Analyse der Methodenaufrufe Artefakte in das Metamodell transformiert und in einer relationalen Datenbank abgelegt. Durch die SQL-Abfragen wird in der Datenbank nach Instanzen abgefragt, die die von der Referenzdefinition bereitgestellten Bedingungen erfüllen. Zu Evaluierung vergleichen Stencel et al. ihre Methode mit zwei anderen und überprüfen anhand von Software-Systemen, welche Entwurfsmuster erkannt werden [Ste08, S. 30]. Als Resultat erkennt das hier erläuterte Verfahren mehr Instanzen an als die anderen Methoden [Ste08, S. 29].

2.4.4 Diverse Ansätze

In ihrer Arbeit präsentieren Kramer et al. eine der frühesten Verfahren, um Entwurfsmuster in Quellcode zu erkennen. Mit der Absicht, die Wartbarkeit von Software-Systemen zu verbessern, nutzen die Kramer et al. die Programmiersprache PROLOG, um Design Patterns als einzelne Regeln zu repräsentieren [Kra01, S. 2]. Diese dienen als Referenz im weiteren Verlauf der Methode. Initial wird der zu verarbeitende Quellcode durch ein Software-Werkzeug in ein Object-Modelling Technique Diagramm transformiert und im nächsten Schritt wird das erstellte Diagramm in PROLOG-Code übersetzt. Die als Referenz genutzten Definitionen der Entwurfsmuster dienen als Anfrage, die in dem transformierten Quellcode nach Instanzen des jeweiligen Design Patterns suchen [Kra01, S. 2]. Zu Evaluierung werden von Kramer et al. die Definitionen für die Design Patterns Adapter, Bridge, Decorater und Proxy angelegt und die Methode an vier C++-Bibliotheken angewendet. Nach eigener Evaluierung von Kramer et al. erzielt die Methode einen *Precision*-Wert zwischen 14% und 50% [Kra01, S. 7].

Anstatt ein Verfahren von Grund aus neu zu entwerfen, bedienen sich Binun et al. in ihrer Methode auf bereits etablierte Software-Werkzeuge und Methoden. Sie erläutern, dass unterschiedliche Verfahren die einzelnen betrachteten Entwurfsmuster unterschiedlich präzise erkennen [Bin09, S.15 - 17]. In ihrer Methode wird die Ausgabe von fünf Verfahren zu einem finalen Wert zusammengefasst, der angibt, mit welcher Wahrscheinlichkeit es sich um das determinierte Design Pattern handelt. Falls die einzelnen Methoden oder Werkzeuge die Eingabe unterschiedlich klassifizieren, wird versucht, durch die Identifikation einzelner Rollen im Quellcode das Entwurfsmuster zu rekonstruieren und dadurch die Eingabe einem Entwurfsmuster zuzuordnen [Bin09, S. 15 - 16]. Für die Evaluierung bedienen sie sich zwölf Software-Systeme als Eingabe für ihre Methode. In diesen wird nach Instanzen von fünf Entwurfsmustern gesucht [Bin09, S. 9, S. 23]. Nach eigener Evaluierung von Binun et al. werden drei von fünf Entwurfsmustern mit einem *Precision*-Wert von über 70% erkannt, während für die restlichen zwei mehrheitlich von den aggregierten Methoden nicht erkannt werden [Bin09, S. 22].

2.5 Machine Learning

Der Fokus dieser Arbeit ist die Ermittlung von Design Patterns in Quellcode durch die Anwendung von Machine Learning. Dieser Teil der Arbeit ist für die Erläuterung von verwendeten Metriken, angewandten Techniken und ausgewählten Klassifizierern gewidmet.

2.5.1 Metriken für Evaluierung von Modellen

Um die Leistung eines Klassifizierers zu ermitteln, bedarf es einer Menge an Metriken, womit beurteilt werden kann, ob der Klassifizierer wie erwartet performt. Im Kontext dieser Aussage werden in der Domäne des Machine Learnings bekannte Metriken aus dem Bereich der Statistik angewendet. In diesem Abschnitt wird eine Auswahl von solchen Metriken erläutert, die in der Literaturrecherche als auch in den Sektionen der Methodik und Implementierung erwähnt werden. Für die Ermittelung der Metriken werden folgende Größen definiert:

TP (True Positive): Anzahl echt positiver Klassifizierungen

TN (True Negative): Anzahl echt negativer Klassifizierungen

FP (False Postive): Anzahl falsch positiv Klassifizierungen

FN (False Negative): Anzahl falsch negativ Klassifizierungen

Anhand dieser Größen werden folgende Metriken kakuliert [Eri21, S. 3]:

Accuracy (Genauigkeit): Dieser Wert gibt an, wie das Verhältnis zwischen den klassifizierten Beobachtungen zur Gesamtzahl der Beobachtungen ist.

$$Accuracy = \frac{\text{Anzahl der korrekten Vorhersagen}}{\text{Gesamtzahl der Vorhersagen}} = \frac{TP+TN}{TP+TN+FP+FN}$$

Precision (Präzision): Dieser Wert gibt das Verhältnis an, wie die korrekt positiv klassifizierten Beobachtungen zur Gesamtzahl der als positiv klassifizierten Beobachtungen stehen.

$$Precision = \frac{TP}{TP+FP}$$

Recall (Sensitivität): : Dies ist das Verhältnis der positiv klassifizierten Beobachtungen zur Gesamtzahl der tatsächlichen positiven Beobachtungen.

$$Recall = \frac{TP}{TP+FN}$$

f1-Score: Der f1-Score ist das harmonische Mittel von Präzision und Recall und gibt ein besseres Maß für die unbalancierten Klassen als die Genauigkeit allein.

$$f1\text{-}Score = 2 * \frac{Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

2.5.2 Validierung und Optimierung von Modellen

Neben dem Zusammentragen von relevanten Datenpunkten für die Erhebung eines geeigneten Datensatzes ist die Validierung und Optimierung des Machine Learning Modells eines der relevanten Schritte im gesamten Prozess, damit das Modell dessen Aufgabe möglichst zufriedenstellend erfüllen kann. In diesem Teil der Arbeit werden Techniken bzw. Methoden erläutert, die für die Validierung oder Optimierung angewendet werden können.

Kreuzvalidierung von Modellen

Nach dem Trainieren des Modells für einen Datensatz ist es von Interesse, wie das Modell mit neuen unbekannten Datenpunkten umgeht und diese Leistung anhand einer Metrik zu messen. Dies stellt den Kernpunkt der Validierungsphase dar. Ein naiver Ansatz ist es, die gleichen Datenpunkte zu verwenden, worauf das Modell in der Trainingsphase trainiert wurde. Dabei ist jeder Datenpunkt des Datensatzes dem Modell bereits bekannt, wodurch es nicht auf wirklich unbekannte Datenpunkte validiert wird. Eine Möglichkeit dem entgegenzukommen ist das Aufteilen des Datensatzes in Trainings- und Validationsdatensatz, wobei meist dem Trainingsdatensatz die größere Menge an Datenpunkten zugesprochen wird. Jedoch besteht hier der Nachteil, dass in der Trainingsphase die Datenpunkte in dem Validationsdatensatz wegfallen. Um diesen Nachteil zu negieren, kann die Technik der Cross Validation oder Kreuzvalidierung angewendet werden.

Hierbei wird zunächst eine natürliche Zahl n bestimmt, die angibt, in wie viele gleich große Teile der Datensatz aufgeteilt wird. Die Trainings- und Validierungsphase wird hierbei zu einem Schritt zusammengefasst und das Modell wird iterativ n-Mal trainiert und anhand der vorgegebenen Metrik evaluiert [Van16, S. 387].

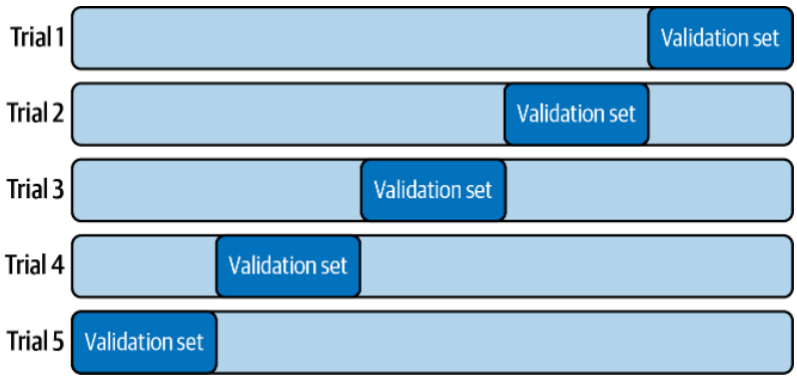

Abbildung 2.10 Kreuzvalidierung mit $n = 5$

Wie aus Abbildung 2.10 zu entnehmen ist, wird in der i-ten Iteration das $k - i$-te Sektion des Datensatzes für die Validierung und die restlichen für das Training des Modells verwendet. Das Resultat dieser Operation ist die Iteration des Modells mit der besten Evaluation und eine Liste von Werten der definierten Metrik, die die Leistung des Modells in der jeweiligen Iteration angeben.

Optimieren von Modellen durch Hyperparameter-Tuning

Machine Learning Modelle können mithilfe von Parametern konfiguriert werden. Dabei wird zwischen zwei Arten von Parametern entschieden: Modellparameter und Hyperparameter. Modellparameter werden während der Trainingsphase automatisiert, direkt durch den Trainingsdatensatz bestimmt und sind nicht von Außen zu manipulieren. Dahingegen bestimmen Hyperparameter unter anderem, wie sich das Modell während der Klassifikation verhält und beeinflussen dessen Architektur. Jedes Modell definiert hierbei selbst, welche Hyperparameter verfügbar sind. Hyperparameter werden während der Instanziierung des Modells definiert. Die Ermittelung der bestmöglichen Kombinationen an Hyperparametern-Werten ist hilfreich, falls die Leistung des Modells nicht den Erwartungen entspricht und der verwendete Datensatz nicht weiter argumentiert werden kann. Um die Leistung des Modells zu evaluieren, wird wie in der Kreuzvalidierung eine Metrik bestimmt, die Leistung der einzelnen Iterationen numerisch bewertet. Das Ziel der Hyperparameter-Optimierung besteht hier darin, eine Kombination an Hyperparametern-Werten zu bestimmen, wodurch die Metrik maximiert bzw. minimiert wird. Dies kann entweder manuell durchgeführt werden oder es werden Methoden eingesetzt, die dies automatisiert erledigen. Im Folgenden wird dazu eine Auswahl an automatisierten Strategien aufgelistet und erläutert:

Random Search/Grid Search: In diesen Methoden werden zunächst Wertebereiche für die jeweiligen Hyperparameter bestimmt. In der Regel wird hierbei eine Menge an diskreten Werten definiert, die der jeweilige Hyperparameter annehmen kann. Jede Permutation der Hyperparameter-Konfiguration wird als eine Zelle in einem Gitter erfasst. Jede besuchte Zelle dient dabei als Konfiguration für eine neue Instanz des Modells, welches im weiteren Verlauf zusätzlich durch Einsatz von Kreuzvalidierung trainiert und evaluiert wird [Lia19, S. 3]. Das Ergebnis der Evaluation wird vermerkt. Bei Random Search wird zufällig bestimmt, welche Zellen besucht werden, während bei Grid Search alle Zellen besucht werden. Das Ergebnis ist die Konfiguration des Modells mit der besten Evaluierung.

Bayessche Optimierung: Bei Grid bwz. Random Search wird für jede Permutation der Hyperparameter-Werte eine Instanz des Modells trainiert und evaluiert. Unter Umständen kann der Trainingsprozess je nach Modell und Datensatz rechenintensiv sein, wodurch es lange dauern, bis diese Methoden Resultate liefern. In solch einem Fall kann das Verfahren der Bayesschen Optimierung angewendet, um die Dauer der Evaluierung zu verkürzen [Sno12, S. 2 - 3]. Im Kern wird das Machine Learning Modell als Black Box Funktion interpretiert, welches als Eingabe eine Permutation an Hyperparameter-Werten akzeptiert und der Wert der Zielmetrik für die jeweilige Permutation dient als Rückgabewert der Black Box Funktion. Dadurch, dass die Optimierung der Black Box Funktion durch Ableitungsverfahren nicht möglich ist, wird dieses durch eine Ersatzfunktion substituiert, welches die Black Box Funktion approximiert. Die Ersatzfunktion ist ein probabilistisches Modell, welches genutzt wird, um eine Wahrscheinlichkeitsverteilung über die möglichen Werte der Zielmetrik für verschiedene Kombinationen an Hyperparameter-Werten zu erstellen. Die Bayessche Optimierung beginnt typischerweise mit einer zufälligen Auswahl von Hyperparametern, um einige Datenpunkte zu generieren. Diese werden verwendet, um das probabilistische Modell zu initialisieren. Anschließend wird iterativ die Akquisitionsfunktion angewendet, um neue und vielversprechende Hyperparameter-Kombinationen zu identifizieren und zu testen. Diese Funktion bestimmt, welche Permutation an Hyperparameter-Werten auf Basis

des jetzigen Standes des probabilistischen Modells als Nächstes bewertet werden soll. Sie balanciert die Exploration unbekannter Bereiche des Hyperparameterraums mit der Exploitation von Bereichen, die voraussichtlich zu besseren Ergebnissen führen. Nach jedem Schritt wird das Modell mit den neuen Ergebnissen aktualisiert, was zu einer kontinuierlichen Verbesserung der Schätzungen und Entscheidungen führt. Am Ende der Methode ist die Kombination an Hyperparameter-Werten bekannt, die für die Zielmetrik das globale Minimum bzw. Maximum als Rückgabewert zurückgibt.

Tree-structured Parzen Estimators (TPE): Bei TPEs handelt es sich um einen speziellen Anwendungsfall der Bayesschen Optimierung [Ber11, S. 4 - 5]. Anstatt eine Wahrscheinlichkeitsverteilung für die Kombination an Hyperparameter-Werten zu verwenden, wird diese in zwei aufgeteilt: eine für Kombinationen, die zu besseren Werten für die Zielmetrik führen ("gute Verteilung") und eine für solche, die zu schlechteren Ergebnissen führen (ßchlechte Verteilung"). Wie bei der normalen Bayesschen Optimierung wird eine Aquisefunktion eingesetzt, um zu bestimmen, welche Kombination an Hyperparameter-Werten als Nächstes evaluiert werden soll. Hierbei basiert die Akquisitionsfunktion bei TPE auf dem Verhältnis der Dichten dieser beiden Verteilungen. Durch den Einsatz von Parzen Estimatoren werden für beide Wahrscheinlichkeitsverteilungen Dichtefunktionen approximiert, wodurch die Dichten in Akquisitionsfunktion bestimmt werden können. Sie wählt die nächste zu evaluierende Hyperparameter-Kombination, indem sie Bereiche bevorzugt, in denen das Verhältnis der Dichte der "guten"Verteilung zur Dichte der ßchlechten"Verteilung hoch ist. Wie in der normalen Bayesschen Optimierung wird das probabilistische Modell iterativ mit neuen Werten aktualisiert, um ein globales Minimum bzw. Maximum der Black Box Funktion zu ermitteln.

2.5.3 Betrachtete Klassifizierer

Für die Bestimmung eines Design Patterns wird eine Menge an Rollen bestimmt, die die Submuster innerhalb des jeweiligen Entwurfsmusters erfüllen. Die Bestimmung der Rollen für einzelne Submuster fällt in dem Bereich des Machine Learnings in den Bereich der Klassifikation. Im Sinne dieser werden in diesem Abschnitt der Arbeit eine Auswahl von Klassifizierern betrachtet und erläutert, die im Kontext dieser Arbeit in Anbetracht gezogen werden.

Support Vector Machines

Support Vector Machine oder SVM sind Machine Learning Modelle, die sowohl für Regression als auch für Klassifikation einsetzbar sind. Die Datenpunkte werden in einem n-dimensionalen Raum abgebildet, wobei n die Anzahl der Features der Datenpunkte beschreibt [Van16, S. 435]. Bei der Klassifizierung trennen SVMs den Raum der Datenpunkte in verschiedene Zonen. Jede Zone entspricht einem Label, zu welchem die Datenpunkte in der jeweiligen Zone zugeordnet werden. Die Grenzen der einzelnen Zonen werden durch Hyperplanes beschrieben. Das Ziel ist, dieses Hyperplanes so zu bestimmen, sodass die Datenpunkte möglichst distinkt einer Zone zugeordnet werden.

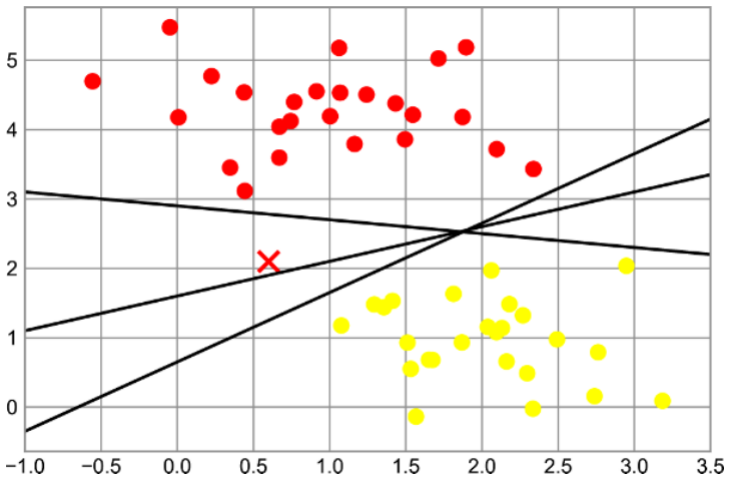

Abbildung 2.11 Grafische Darstellung einer SVM im zwei-dimensionalen Raum

Abbildung 2.11 zeigt grafisch, wie eine SVM funktioniert [Van16, S. 437]. Die Punkte sind die Datenpunkte des Datensatzes und die Farben der Punkte die Klasse, in der die Punkte eingeordnet werden. Die Linien beschreiben die möglichen Hyperplanes, womit die Grenzen zwischen den Klassen determiniert werden. Das Kreuz zeigt einen neuen Datenpunkt, der zu klassifizieren ist. Mit diesem neuen Datenpunkt stellt sich die Frage, welche der drei möglichen Hyperplanes zu wählen ist, um die Datenpunkte möglichst distinktiv einer Klasse zuzuordnen.

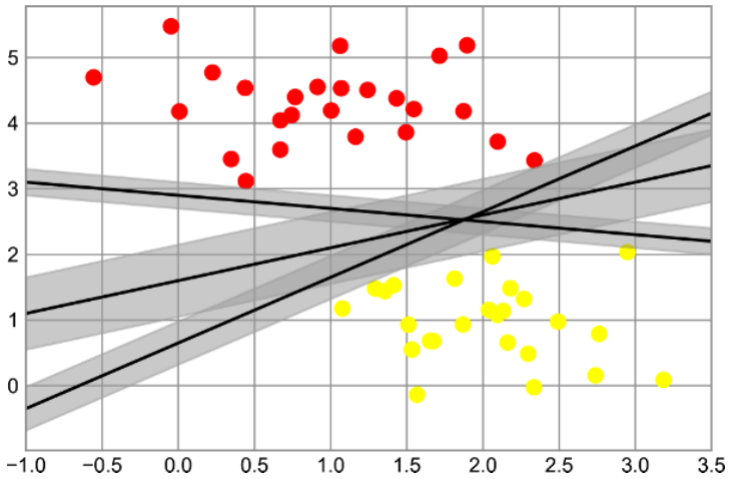

Abbildung 2.12 Hyperplanes mit Margen

Wie in Abbildung 2.12 zu sehen ist, wird für jede Hyperplane die Marge zwischen der Hyperplane und den nächsten Datenpunkten bestimmt [Van16, S. 438]. Die Hyperplane mit der größten Marge, also die mit der größten Distanz zwischen sich und den nächsten Datenpunkten, wird als finale Hyperplane für die Grenzbildung zwischen den Klassen gewählt. Im Falle der Abbildung 2.12 ist es die mittlere von den drei Kandidaten. Mit jedem neuen Datenpunkt wird während Trainings die ideale Hyperplane neu bestimmt. Jedoch stellt sich die Frage, wie man mit Datensätzen umgeht, die nicht durch lineare Hyperplanes in Klassen aufgeteilt werden.

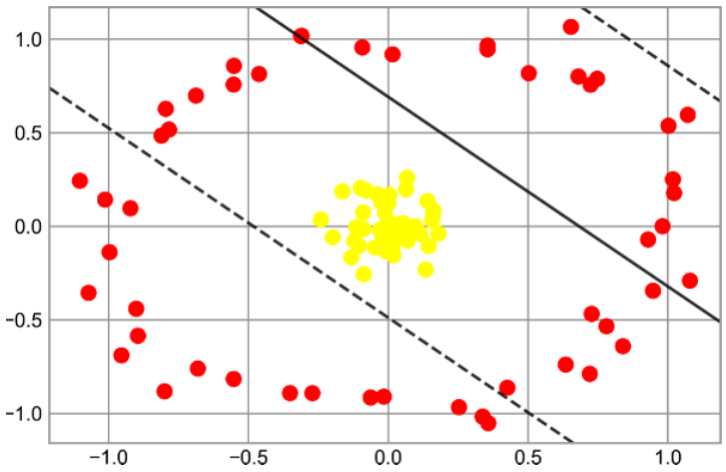

Abbildung 2.13 Nicht mögliche Einteilung durch lineare Hyperplanes

Abbildung 2.13 zeigt einen Datensatz, dessen Datenpunkte nicht distinktiv durch lineare Hyperplanes zugeordnet werden können [Van16, S. 441]. Um trotzdem lineare Hyerplanes zu bilden, wird der bei SVMs Kernels verwendet. Bei dem Kernel handelt es sich um eine Funktion, die den Datensatz in einen Raum mit einer höheren Dimension projektiert. Durch die Projektion in einem Raum mit einer höheren Dimension ist es möglich, lineare Hyperplanes zu bestimmen. Diese Funktion werden als Kernel bezeichnet und werden bei der Instanziierung des Modells als Hyperparameter mitgegeben.

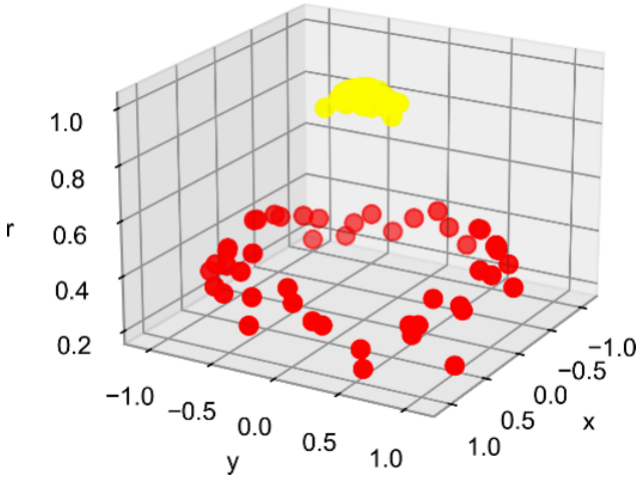

Abbildung 2.14 Transformierter Datensatz aus Abbildung 2.13 durch RBF

Abbildung 2.14 zeigt wie durch Anwendung des Radial-Basis-Function Kernels, das den ursprünglich zwei-dimensionalen Raum auf einem drei-dimensionales projektiert [Van16, S. 442]. Durch die Transformation kann nun eine Hyperplane als Ebene bestimmt werden, wodurch die Klassifikation ermöglicht wird.

k-Nearest Neighbor Classifier

k-Nearest Neighbors Classifier oder KNN kann wie die SVM für Regressions- als auch für Klassifikationsprobleme eingesetzt werden. Dabei wird bei KNN unter der Annahme agiert, dass ähnliche Datenpunkte in der Nähe zueinander sind [Gru15, S. 216]. Der Hyperparameter k definiert die Anzahl der nächsten Nachbarn, die verwendet werden, um den neu eingefügten Datenpunkten eine Klasse zuzuordnen. Um die k nächsten benachbarten Datenpunkte zu identifizieren, ist ein Maß erforderlich, welches die Distanz zwischen diesen beschreibt. Dieses wird als Hyperparameter bei der Instanziierung des Modells diesem übergeben. Folgende Größen können als Metrik für die Distanz verwendet werden [AA19, S. 6]:

Euklidische Distanz ist die Länge des Liniensegmentes zwischen den beiden Punkten im euklidischen Raum.
$d_{euclidean}(P,Q) = \sqrt{(p_1 - q_1)^2 + (p_2 + q_2)^2 + \ldots + (p_n - q_n)^2}$

mit

$P = (p_1, p_2, \ldots, p_n)$
$Q = (q_1, q_2, \ldots, q_n)$

Manhattan Distanz ist die Summe der absoluten Differenz zwischen den Komponenten der Punkte.
$d_{manhatten}(P,Q) = |p_1 - q_1| + |p_2 - q_2| + \ldots + |p_n - q_n|$

mit

$P = (p_1, p_2, \ldots, p_n)$
$Q = (q_1, q_2, \ldots, q_n)$

Minkowski Distanz ist eine allgemeinere Metrik, um die Distanz zwischen zwei Punkten im n-dimensionalen Raum ermittelt.
$d_{minkowski}(P,Q) = (|p_1 - q_1|^p + \ldots + |p_n - q_n|^p)^{\frac{1}{p}}$

mit

$P = (p_1, p_2, \ldots, p_n)$
$Q = (q_1, q_2, \ldots, q_n)$
$p =$ Typ der zu kalkulierenden Distanz ($p = 1$ –> Manhatten Distanz, $p = 2$ –> Euklidische Dista

Die Klassifizierung des neuen Datenpunktes erfolgt im Kontext der k nächsten benachbarten Datenpunkte. Die Klasse, welche am häufigsten in den k benachbarten Punkten vorkommt, wird dem neuen Datenpunkt zugewiesen.

Random Forest Classifier

Der Random Forest Classifier ist ein ensemble-basiertes Lernverfahren im Bereich des maschinellen Lernens, das für Regressions- und Klassifizierungsaufgaben eingesetzt wird. Es kombiniert die Vorhersagen mehrerer Entscheidungsbäume, um zu einer endgültigen Entscheidung zu kommen [Van16, S. 456], wobei es sich durch hohe Genauigkeit und die Fähigkeit auszeichnet, Overfitting zu vermeiden [Van16, S. 455]. Overfitting beschreibt dabei das Phänomen, bei dem ein Modell zu stark an die spezifischen Details und das Rauschen des Trainingsdatensatzes angepasst ist, sodass das Erkennen zugrundeliegender Muster nicht erfasst wird. Das grundlegende Element in einem Random Forest Classifier stellen die Decision Trees oder Entscheidungsbäume dar, die in Summe den "Wald"des Random Forest Classifiers bilden. Diese werden während des Trainings erzeugt. Ein Entscheidungsbaum ist dabei ein Modell, das Entscheidungen und deren mögliche Konsequenzen, einschließlich Zufallsereignissen, Kosten und Nutzen, in Form eines Baumdiagramms darstellt. Ein Entscheidungsbaum besteht aus Knoten, die Testfragen oder -kriterien repräsentieren, und Blättern, die die Endentscheidungen oder Ausgänge darstellen. Die Auswahl eines Pfades von der Wurzel bis zu einem Blatt repräsentiert eine Reihe von Entscheidungen, die zu einem bestimmten Ergebnis führen. Die Ergebnisse der einzelnen Entscheidungsbäume werden zu einem Gesamtergebnis durch Bagging aggregiert [Van16, S. 456]. Jeder Entscheidungsbaum im Random Forest Classifier wird aus einer zufälligen Auswahl von Trainingsdaten mit Zurücklegen gebildet. Dieser Prozess führt zu unterschiedlichen Bäumen, die unterschiedliche Aspekte der Daten erfassen.

3 Methodologie

3.1 Verfügbare Datensätze

Um ein Machine Learning Modell zu trainieren, benötigt es einen Datensatz mit passenden gekennzeichneten Klassen. In der Erkennung von Entwurfsmustern existiert zu dem Zeitpunkt der Verfassung dieser Arbeit keine standardmäßiger oder weit akzeptierter Datensatz. Jedoch existieren Versuche, solch einen Datensatz zu etablieren. Im Folgenden wird eine Auswahl an möglichen existierenden Datensätzen aufgelistet [Alh16, S. 104 - 105]:

Pattern-like Micro-Architecture Repository (P-MArt): Dieser Datensatz ist der einzige der hier aufgelisteten, welcher Peer-Reviews unterzogen wurde. Zwar bestehen die Autoren nicht darauf, dass alle möglichen Instanzen von Entwurfsmustern in den Software-Systemen identifiziert wurden, haben aber Zuversicht, dass die Mehrheit der Instanzen durch mehrfache Analysen durch mehrere Teams erkannt wurde.

DEsign pattern Evaluation BEnchmark Environment (DEEBEE): Dieser Datensatz beinhaltet Resultate aus automatischer und einer manuellen Analyse von fünf Software-Systemen auf Entwurfsmustern.

Design Pattern Benchmark (DPD): DPD beinhaltet die Ergebnisse von Software-Systemen durch mehrere Software-Werkzeuge. Zudem sind die Daten des Datensatzes durch eine Webseite verfügbar, in der nach Instanzen von Entwurfsmustern in Software-Systemen gesucht werden kann. Zusätzlich können auf dieser Webseite Ergebnisse evaluiert und durch öffentlich zugängliche Abstimmung einer Instanz das passende Design Pattern zugeordnet werden.

PattErn Repository and Components Extracted from OpeN Source software (PERCERONS): PERCERONS ist das derzeit von dem Umfang her der größte verfügbare Datensatz mit über 537 analysierten Software-Systemen. Die Ergebnisse wurden durch das Kombinieren der Resultate zweier Software-ErgebnisWerkezeuge ermittelt.

Software Engineering Research Center benchmark (SERC): SERC wurde konstruiert, um als Vergleichsmaßstab für ein Software-Werkzeug zu dienen, welches von den gleichen Autoren stammte. Dabei werden die Ergebnisse von mehreren anderen Werkzeugen mit den Ergebnissen ihres eigenes kombiniert.

Die Problematik bei der Ermittelung eines geeigneten Datensatzes besteht darin, dass der Umfang der analysierten Software-Systeme sich in Grenzen hält und meist eher als Maßstab für die Leistung der selbst entwickelten Methode verwendet wird, wodurch Bias zu der eignen Methode nicht ausgeschlossen werden kann. Zudem sind die hier aufgeführten Datensätze bis auf einige Ausnahmen durch die Kombination von mehreren Software-Werkzeugen entstanden. Durch solch ein Verfahren ist zwar die Analyse von mehreren Software-Systemen in einer relativ kurzen Zeitspanne möglich, jedoch wäre eine manuelle Verifizierung der Ergebnisse aufgrund von hohen Zeit- und Leistungsaufwands schwer umsetzbar. Dahingegen wäre das ideale Vorgehen, das solch ein Datensatz durch manuelles Identifizieren von Entwurfsmustern in Quellcode durch die Analyse von mehreren Software-Systemen durch ein Experten-Team erstellt wird. Jedoch ist solch ein Verfahren zeitaufwendig und erfordert ein tiefes Verständnis des jeweiligen Software-Systems, welches durch mangelnde oder obsolete Dokumentation eine Hürde darstellt. Die nächst bestmögliche Alternative dazu ist die Option des Crowdsourcings wie in DPD. Das Problem ist hier, dass die Qualifikation der Personen, die Instanzen von Entwurfsmustern identifizieren und über die korrekte

Zuweisung abstimmen, nicht garantiert sind. Zudem erfordert dies ein Publikum mit der Bereitschaft, dies zu tun, welches auch nicht garantiert werden kann. Bei dem im Kontext dieser Arbeit vorgestellten Verfahren für die Ermittelung von Entwurfsmustern wird meist ein Datensatz im kleinen Umfang erstellt. Auf Anfrage. um Zugang zu einigen Datensätzen zu erhalten, gibt es zu dem Zeitpunkt zur Verfassung dieser Arbeit keine Antwort seitens der Autoren. Zudem sind die Webseiten, worüber auf die Datensätze zugegriffen werden kann, nicht mehr verfügbar und der Zugang über eine Archivierungsdienstleistung wie der Wayback Maschine war ebenfalls nicht möglich. Aus den hier aufgelisteten Gründen wird im Zuge dieser Arbeit der P-MArt Datensatz verwendet und somit die Untersuchungsfrage RQ2 zu beantworten. Dieser ist zum Zeitpunkt des Verfassens dieser Arbeit zugänglich und ist der Einzige, der Peer-Reviews unterzogen wurde. In der Sektion der Implementierung wird dieser genauer betrachtet.

3.2 Extrahierte Features

Im Kontext dieser Arbeit werden Design Patterns als eine Summe von Substrukturen definiert, zu diesem jeweils eine Rolle zugeordnet wird. Im Sinne dieser ist eine Menge an Erkennungsmerkmalen oder Features notwendig, welche Rückschlüsse auf die jeweilige Rolle ermöglicht. Dabei müssen sich die Aspekte für Struktur, Verhalten und Relation zu anderen Komponenten in ausgewählten Features widerspiegeln. Die in einem vorherigen Abschnitt erläuterten Machine Learning Verfahren nutzen verschiedene Arten von Features, um diese Aspekte zu encodieren. Im Kern läuft es darauf hinaus, die Erkennungsmerkmale in eine numerische Repräsentation zu transformieren, sodass ein Klassifizierer mit diesen umgehen kann. Deswegen werden in dieser Arbeit die Merkmale der jeweiligen Instanzen der Rollen als numerische Vektoren dargestellt, die die Aspekte der Struktur, Verhalten und Relation als Metriken encodieren.

Die Entscheidung, Metriken als Features für Rollen zu verwenden, bietet mehrere Vorteile. Zu einem ermöglicht der Einsatz von Metriken, Eigenschaften der Implementierung der jeweiligen Rollen in einer abstrakten Form darzustellen. Dadurch kann die Identifikation der Rollen unabhängig von Programmiersprache und Projektstruktur erfolgen. Die Ermittelung der jeweiligen Metriken erfolgt über statische Codeanalyse, die durch Parallelisierung in der Lage ist, einen großen Umfang an Quelldateien in relativ kurzer Zeit zu bearbeiten. Der wohl größte Verteil, die Metriken erbringen, ist Transparenz und Reproduzierbarkeit. Bevor ein Wert zu einer Metrik ermittelt werden kann, muss zunächst definiert werden, was die jeweilige Metrik aussagt und wie diese ermittelt wird. Durch eine automatisierte und strikte Befolgung der Definition der Berechnung kann durch statische Codeanalyse in mehreren Durchläufen der gleiche Wert für die Metrik im Kontext der Rolleninstanz bestimmt werden. Dadurch entsteht ebenfalls eine Transparenz, wie die Werte ermittelt werden. Bei den hier erläutern Verfahren, die die Features der Entitäten aus dem Quellcode grafisch oder in natürlicher Sprache encodiert haben, müssen diese zusätzlich in eine numerische Form gebracht werden. Dies erfolgt meist durch das Inkludieren eines zusätzlichen Modells. Zu einem erfordert dies zusätzliches Training und Validierung des encodierenden Modells, welches je nach Modell und Umfang des Datensatzes auch noch Zeit und Rechenleistung in Anspruch nimmt. Dazu ist die Leistung des Klassifizierers abhängig von der Qualität der Resultate des encodierenden Modells. Mögliche Leistungsdefizite des Encodierens propagieren zu der Klassifizierungskapazität des gesamten Verfahrens. Zwar ist die Ermittlung von Metriken davon nicht ausgeschlossen, jedoch sind Rückschlüsse möglich, wie diese Werte zustande gekommen sind. Dahingegen sind die Modelle, die die Features transformieren, als Black Box zu betrachten. Die Werte innerhalb des Modells, die die Transformation der Eingabedaten beeinflussen, sind für das Menschenauge größtenteils nur schwer verstehbar und bieten keinen Rückschluss, wie diese berechnet worden sind. Zusammenfassend ermöglichen Metriken als Feature eine Unabhängigkeit von Programmiersprache und Projektstruktur, Skalierbarkeit, Reproduzierbarkeit und Transparenz, die bei dem weiteren Verfeinern der hier vorgestellten Methodik wünschenswert sind.

Um die Kennmerkmale hier zu ermitteln, erfordert dies ein Katalog an Metriken, die angegeben, wie diese ermittelt werden und was diese aussagen. Hierbei werden Metriken verwendet, die die Aspekte der Struktur, des Verhaltens und der Relation widerspiegeln. Dazu werden Metriken verwendet, die bereits in anderen Arbeiten verwendet wurden, die die Struktur repräsentieren und Metriken von Chidamber et al., die Einblicke in Verhalten und Relation bieten [Chi94]:

Tabelle 3.1 ausgewählte Metriken als Features

Kürzel	Langform	Beschreibung	Datentyp
COA	COUNT_OF_ABSTRACT_METHODS	Anzahl der abstrakten Methoden in einer Klasse	int
COF	COUNT_OF_FIELDS	Anzahl der Felder in einer Klasse	int
COM	COUNT_OF_METHODS	Anzahl der Methoden in einer Klasse	int
COF	COUNT_OF_OBJECT_FIELDS	Anzahl der Felder in einer Klasse mit Objekttyp	int
COPC	COUNT_OF_PRIVATE_CONSTRUCTORS	Anzahl der privaten Konstruktoren	int
COPF	COUNT_OF_PRIVATE_FIELDS	Anzahl der privaten Felder	int
COSF	COUNT_OF_STATIC_FIELDS	Anzahl an statischen Feldern	int
CBO	COUPLING_BETWEEN_OBJECTS	Anzahl der Abhängigkeiten in der Klasse nach Chidamer et al.	int
DIT	DEPTH_OF_INHERITANCE	Anzahl der Elterklassen einer Klasse nach Chidamer et al.	int
RPC	RESPONSE_FOR_A_CLASS	Anzahl der Methoden nach Chidamer et al, die direkt durch Klassenmethoden aufgerufen werden	int
WMC	WEIGHTED_METHOD_CLASS	Zyklomatische Komplexität nach McCabe [McC76]	float

Der in Tabelle 3.1 definierte Katalog an Metriken dient für die Beantwortung der Untersuchungsfrage RQ4 und die Identifikation an Rollen im Kontext des jeweiligen Design Patterns der Untersuchungsfrage RQ3.

3.3 Modellauswahl, Training und Validierung

Nach dem Bestimmen eines passenden Datensatzes und zu extrahierenden Features wird ein Klassifizierer trainiert, welcher mit einem Wahrscheinlichkeitswert angibt, welche Rolle am ehesten der Quellcodeentität zugeordnet werden kann. Dazu wird in dieser Sektion erläutert, welche Modelle betrachtet werden und wie diese trainiert und evaluiert werden können.

Auswahl der Klassifizierer

Im Zuge dieser Arbeit werden in der Sektion 2.5.3 erläuterten Klassifizierer als die bevorzugten Modelle hergenommen. Diese Auswahl der Modelle ist auf unterschiedliche Gründe zurückzuführen. Klassifizierer können je nach Art und Weise, wie sie innerlich funktionieren, in Kategorien untergliedert werden. Dabei dienen die Modelle aus Sektion 2.5.3 als Repräsentanten ihrer Kategorie, sodass ein Spektrum von unterschiedlich funktionierenden Modellen getestet werden kann. Ohne das Testen des jeweiligen Models ist dessen Leistung für das in dieser Arbeit genutzten Datensatzes nicht vorherzusagen. Zudem können Implementierungen der jeweiligen Modelle in verschiedenen Programmiersprachen für verschiedene Plattformen vorgefunden werden. Dies ermöglicht die Nutzung in Programmiersprachen wie Python oder R, welche für Machine Learning bevorzugt werden. Das Software-Framework, welches die Implementierungen für die Modelle zur Verfügung stellt und im Kontext dieser Arbeit angewendez wird, wird im weiteren Verlauf der Arbeit genauer erläutert. Der größte Vorteil dieser Klassifizierer ist deren Simplizität und damit resultierende Effizienz in Zeit- und Speicherkomplexität. Da die Klassifizierer im weiteren Verlauf der Methode durch den Einsatz von Hyperparameter-Tuning iterativ optimiert werden, ist dies von besonderer Bedeutung. Deshalb werden Modelle mit komplexerer Architektur aufgrund des Mangels verfügbarer Rechenressourcen zu Zeitpunkt des Verfassens der Arbeit nicht weiter betrachtet. Die Auswahl der Klassifizierer aus Sektion 2.5.3 beantwortet die Untersuchungsfrage RQ5

Training der Modelle

In dieser Arbeit wird das Trainieren der Modelle mit dem Hyperparameter-Tuning aus Sektion 2.5.2 gekoppelt. Zuerst wird der Datensatz in ein Trainings- und Validationsdatensatz aufgeteilt. Danach wird für jedes Modell ein Suchraum für die Hyperparameter-Werte, Anzahl der Trainingsiterationen und die Metrik bestimmt, wonach die Leistung des Klassifizierers beurteilt wird. In jeder Iteration wird zufällig eines der hier vorgestellten Klassifizierer mit einer vorher bestimmten Hyperparameter-Konfiguration instanziiert, mit dem Trainingsdatensatz trainiert und durch Einsatz des Validationsdatensatzes der Wert der Leistungsmetrik bestimmt. Die Hyperparameter-Konfiguration der Instanz mit der besten Leistung aus allen Iterationen wird im weiteren Verlauf der Arbeit verwendet.

Validierung des Modells

Dadruch, dass der Datensatz für das Training aufgeteilt wird, wird die beste Hyperparameter-Konfiguration eines der hier vorgestellten Klassifizierer durch Kreuzvalidierung zusätzlich validiert. Dabei wird der gesamte verfügbare Validationsdatensatz verwendet. Die Evaluierung der Leistung der Klassifikationskapazität erfolgt mit der gleichen Metrik wie in der Trainingsphase. Es bestünde die Option, dass man die Kreuzvalidierung direkt in die Trainingsphase integriert. Jedoch ist dabei zu beachten, dass die Kreuzvalidierung selbst

iterativ agiert und dies in jeder Iteration des Hyperparameter-Tunings durchgeführt wird. Dies erhöht die Laufzeit der Trainingsphase signifikant.

3.4 Ermittelung des übereinstimmenden Entwurfsmusters

In der vorherigen Sektion wird erläutert, wie anhand eines Machine Learning Modells den Quellcodeentitäten eine Rolle aus einem Entwurfsmuster zugeteilt wird. Das Ergebnis der Klassifikation besteht aus einem Kollektiv an Rollen, die in einem der betrachteten Design Pattern vorkommen. Diese Kollektion an Rollen allein reicht nicht aus, um eine Aussage über die Zugehörigkeit eines Design Patterns zu tätigen. Daher wird in diesem Abschnitt der Arbeit ein Algorithmus in Form von Pseudocode vorgestellt, womit ein Wert berechnet werden kann, welcher angibt, zu welchem Design Pattern die Kollektion an ermittelten Rollen am ehesten passt. Dieser Abschnitt wird verwendet, um die Untersuchungsfrage RQ6 zu beantworten. Dabei werden folgende Schritte getätigt:

1. Extrahiere Metriken für Quellcode nach Tabelle 3.1
2. Wende Klassifizierer auf Metriken an für Vorhersagen über Rollen
3. Definiere jedes klassifizierbare Design Pattern (Singleton, Command, Observer und Adapter) als Menge von Rollen nach Gamma et al. Bestimme, ob Rollen öfters aufkommen dürfen.
4. Iteriere über Design Pattern und vergleiche Rollen mit Beachtung folgender Fälle:
   a) Initialisiere eine Hash Map für die Frequenz der aufkommenden vorhergesagten Rollen
   b) Iteriere über vorhergesagte Rollen
      i. Falls die vorhergesagte Rolle nicht in Referenzentwurfsmuster vorhanden: Überspringe Rolle.
      ii. Vorhergesagte Rolle in Referenzentwurfsmuster; Rolle kam davor nicht vor; Rolle darf exakt ein mal vorkommen: Erhöhe Frequenz um 1.
      iii. Vorhergesagte Rolle in Referenzentwurfsmuster; Rolle kam davor vor; Rolle darf exakt ein mal vorkommen: Überspringe Rolle.
      iv. Vorhergesagte Rolle in Referenzentwurfsmuster; Rolle darf mehrmals vorkommen: Erhöhe Frequenz um 1.
   c) Addiere Frequenzen zu einer Summe und teile durch die Anzahl der vorhergesagten Rollen als Übereinstimmungswert
5. Design Pattern mit höchstem Übereinstimmungswert als ehester Kandidat

4 Implementierung

4.1 Präprozessierung des Datensatzes

Wie in Sektion 3.1 erläutert, wird in dieser Arbeit der Datensatz P-MArt verwendet. P-MArt ist ein Katalog von Design Pattern mit Instanzen aus mehreren Software-Systemen. Dabei wird bei jeder Instanz eines Design Pattern die aufkommenden Software-Entitäten wie Klassen oder Schnittstellen mit Rollen nach Gamma et al. annotiert.

Tabelle 4.1 Software-Systeme in P-MArt

Projekt	Beschreibung	Programmiersprache
QuickUML 2001	An Anfänger gerichtetes Werkzeug für die Erstellung UML-Diagrammen	Java
Lexi v0.1.1 alpha	einfaches Textverarbeitungssystem	Java
JRefactory v2.6.24	Software-Werkzeug für das Refactoring für Java	Java
Netbeans v1.0.x	integrierte Entwicklungsumgebung für Software	Java
JUnit v3.7	Test-Framework für Java-Projekte	Java
JHotDraw v5.1	Framework für das Erstellen von Editoren	Java
MapperXML v1.9.7	Presentation-Framework	Java
Nutch v0.4	Open-Source-Suchmaschine für das Web	Java
PMD v1.8	Werkzeug für statische Codeanalyse	Java
Software architecture design patterns in Java	Kollektion von Implementierung von Design Patterns nach Kuchana [Kuc04]	Java
DrJava v20020619	integrierte Entwicklungsumgebung für Java	Java
DrJava v20020703	integrierte Entwicklungsumgebung für Java	Java
DrJava v20020804	integrierte Entwicklungsumgebung für Java	Java
DrJava v20030203	integrierte Entwicklungsumgebung für Java	Java

Tabelle 4.1 listet die Software-Systeme auf, die in P-MArt betrachtet werden. Dabei sind alle Software-Systeme in der Programmiersprache Java verfasst und unterliegen einer Open-Source-Lizenz, wodurch der Quellcode frei zugänglich ist. Zudem waren nicht alle Software-Systeme zugänglich. Die verschiedenen Versionen der Entwicklungsumgebung DrJava waren archiviert über die Webplattform SourceForge verfügbar. Das Werk SSoftware architecture design patterns in Java"von Kuchana ist ein Fachbuch für den Einsatz von Design Pattern in Java. In P-MArt werden dessen Software-Beispiele inkludiert, die im Kontext des Werkes zur Erläuterung dienen und durch die Webplattform des Verlags zugänglich sind. Die im Werk erwähnte URL für die Beispiele war zu dem Zeitpunkt der Verfassung der Arbeit nicht aufrufbar. Der Quellcode für die restlichen Software-Systeme ist als Archiv über die Webseite der Autoren vom P-MArt zugänglich.

P-MArt selbst wird in der Form einer Extensible Markup Language (XML) Datei verfügbar gestellt. Für einfachere Verarbeitung des Datensatzes werden die notwendigen Informationen aus der XML-Datei extrahiert und in einer Comma-Separated Values (CSV) Datei abgelegt.

Tabelle 4.2 Spalten der prozessierten CSV-Datei für P-MArt

Spalte	Beschreibung
project	Name des Software-Systems
micro_architecture	Identifikator der Mikroarchitektur in P-MArt
design_pattern	Zugeordnetes Entwurfsmuster nach Gamma et al.
role	Rolle der Entität innerhalb des Design Patterns nach Gamma et al.
role_kind	Art der Entität (*Class* für Klasse oder *AbstractClass* für abstrakte Klassen)
entity	qualifizierender Name der Entität innerhalb des Software-Systems

Die Tabelle 4.2 zeigt die Spalten der resultierenden CSV-Datei. Für die Erstellung dieser Datei wird die Skriptsprache Python und dessen Standardbibliothek verwendet. Neben den in Tabelle 4.2 aufgeführten Informationen beinhaltet die beschreibende XML-Datei von P-MArt zusätzliche Details wie Kommentare der Autoren über einzelne Mikroarchitekturen. Diese werden in der weiteren Verarbeitung der Daten nicht berücksichtigt. Die resultierende CSV-Datei dient zusammen mit dem Quellcode der in P-MArt analysierten Software-Systeme als Eingabe für die Extraktion der Features.

4.2 Extraktion von Features aus Quellcode

Die extrahierte CSV-Datei aus Abschnitt 4.1 und der Quellcode der betrachteten Software-Systeme in Tabelle 4.1 dient als Eingabe für die Extraktion der Features, die die jeweiligen Rollen in Entwurfsmustern charakterisieren. Für die Berechnung der in Tabelle 3.1 bedarf es einer statischen Codeanalyse. Deshalb wird für diese Arbeit eigens ein Software-Werkzeug featureextractor auf Basis der Programmiersprache Java entwickelt. Dabei werden für die Extraktion der Features folgende Schritte befolgt:

1. **Laden des Quellcodes**: D die Software-Systeme in P-MArt alle in Java verfasst sind, wird die Bibliothek javaparser verwendet. Diese Software-Bibliothek ist in der Lage, gegebenen Java-Quellcode in ein Abstract Syntax Trees (AST) zu transformieren. Ein AST ist eine baumartige Struktur, die Quellcode unabhängig von Programmiersprache repräsentiert und ermöglicht das Durchführen von Operationen wie Datenmanipulation oder Anfragen am Quellcode. Die resultierenden ASTs durch javaparser nutzen die Projektdefinition als Wurzel.

2. **Suche nach Klassen und Schnittstellen**: Nach dem Erstellen der ASTs wird eine Suche nach allen Klassen oder Schnittstellen innerhalb aller Software-Systeme durchgeführt. Die gefundenen Definitionen werden für späteren Zugriff in einer Java Hash Map abgelegt. Dabei dient eine Kombination aus Projekt- und Entitätsname als Schlüssel.

3. **Extraktion der Metriken nach Chidamer et al.**: Für die Ermittelung der Metriken nach Chidamer et al. wird die externe Software-Bibliothek ckmetrics [Ani15] verwendet. Dabei werden alle nach Chidamer et al. definierten Metriken in Tabelle 3.1 durch diese berechnet, nachdem die Bibliothek den Quellcode der Software-Systeme prozessiert hat. Die Ergebnisse werden für spätere Zugriff in einer Java Hash Map abgelegt.

4. **Einlesen von P-MArt**: Die in der Sektion 3.1 prozessierte CSV wird zeilenweise eingelesen. Jede Zeile in dieser CSV-Datei beschreibt eine Quellcodeentität aus eines der Software-Systeme. Falls die jeweilige Entität in der Schritt 2 ermittelten Hash Map vorgefunden wird, wird die AST der Entität weiterverarbeitet. Ansonsten wird diese verworfen und alle Features wird der Wert 0 zugewiesen.

5. **Ermittelung der restlichen Features**: Die Berechnung der restlichen Features aus Tabelle 3.1 erfolgt durch den Einsatz selbst entwickelter Extraktoren. Jede dieser Extraktor ermittelt jeweils eine Metrik und nimmt eine AST einer Klasse- oder Schnittstelle als Eingabeparameter entgegen. Der Rückgabewert ist der berechnete Wert der Metrik.

6. **Aggregation der Features**: Nach dem die AST von den Feature Extraktoren verarbeitet wurde, werden die Metriken von ckmetrics zusammen mit diesen zu einem Feature-Vektor kombiniert. Die eingelesenen Daten aus Schritt 3 werden mit dem Feature-Vektor erweitert und in einer neuen CSV-Datei abgespeichert.

4.3 Analyse des Datensatzes

In der Sektion 4.2 wird der Datensatz verarbeitet und mit den in Tabelle 3.1 erweitert. In diesem Abschnitt wird das Resultat der Sektion 4.2 analysiert. Für den weiteren Verlauf der hier vorgestellten Methode wird die Skriptsprache Python in Kombination mit Jupyter-Notebooks benutzt. Für die Analyse der Daten wird die Python-Bibliothek pandas verwendet. Für die Visualisierung durch verschiedene Arten von Diagrammen wird die Python-Bibliothek plotly eingesetzt.

Tabelle 4.3 Verteilung der Entwurfsmuster in P-MArt

Entwurfsmuster	Anzahl
Singleton	15
Adapter	9
Command	8
Observer	6
Strategy	6
Iterator	6
Factory Method	5
Template Method	5
Composite	5
Proxy	4
Memento	4
Builder	4
Visitor	4
State	3
Abstract Factory	3
Decorator	2
Facade	1
Bridge	1
Null Object	1
Prototype	1
FactoryMethod	1
Mediator	1

Die Tabelle 4.3 zeigt die Verteilung der Instanzen der Design Pattern an. Dabei ist abzulesen, dass die Singleton-, Adapter-, Command- und Observer-Entwurfsmuster die am häufigsten vertretenen Entwurfsmuster sind, während Instanzen der Design Patterns wie Decorator oder Bridge nur einmal vorkommen.

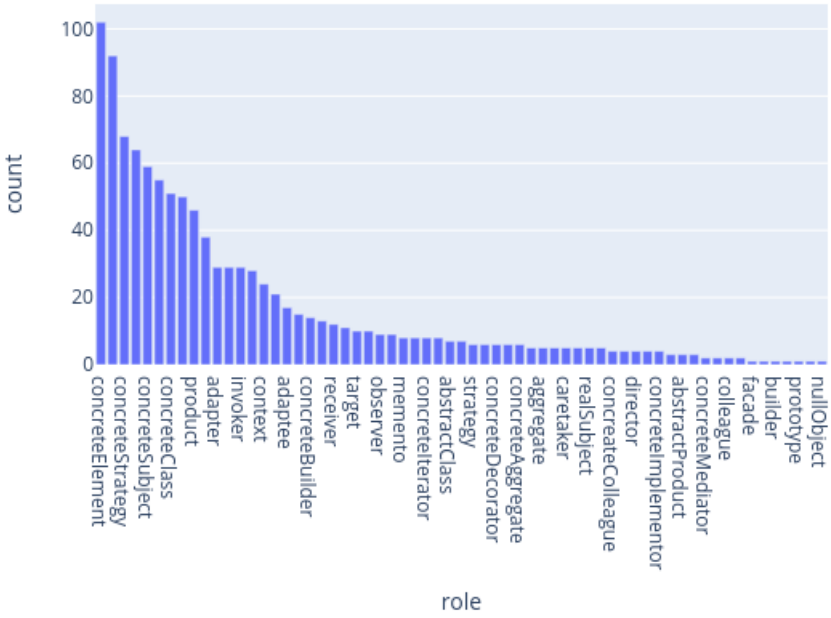

Abbildung 4.1 Rollenverteilung in P-MArt

Abbildung 4.1 ermöglicht einen Überblick über die Rollenverteilung. Dabei ist zu vermerken, dass die Verteilung der Rollen und die Anzahl der Instanzen der zugehörigen Design Pattern unbalanciert sind. Beispielsweise ist die Rolle *concreteelement* des Visitor-Entwurfsmusters am häufigsten vorzufinden, obwohl in P-MArt vier Instanzen zu diesem Design Pattern vermerkt sind. Um ein möglichst aussagekräftiges Ergebnis zu erhalten, muss die Menge der zu identifizierenden Entwurfsmuster eingeschränkt werden. Ansonsten ist bei zu vielen identifizierbaren Klassen mit wenigen Datenpunkten eine unzufriedenstellende Leistung der Klassifizierer zu erwarten. Deshalb werden im weiteren Verlauf Instanzen der Entwurfsmuster Singleton, Observer, Command und Adapter weiter betrachtet. Dies beantwortet die Untersuchungsfrage RQ1.

Tabelle 4.4 Aufteilung des Datensatzes für Training und Validierung

Entwurfsmuster	Rolle	Anzahl
Singleton	singleton	15
Adapter	adaptee	17
	adapter	29
	client	13
	target	10
Command	client	13
	command	6
	concreteCommand	50
	invoker	29
	receiver	12
Observer	concreteObserver	28
	concreteSubject	59
	observer	9
	subject	5
Insgesamt		295

Tabelle 4.4 zeigt den Datensatz an, der im weiteren Verlauf verwendet wird. Auf vier Design Pattern aufgeteilt, beträgt der Umfang des Datensatzes 295 Datenpunkte auf vierzehn Rollen aufgeteilt.

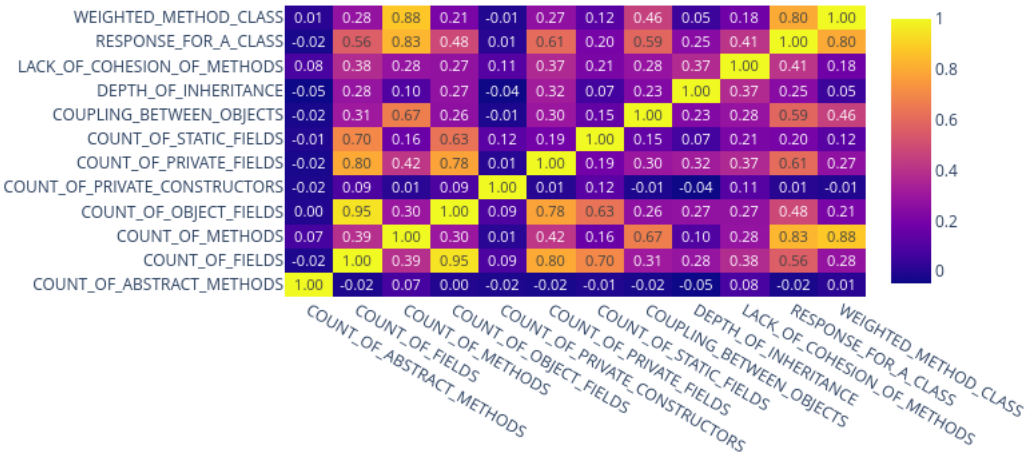

Abbildung 4.2 Koeffizienten-Heatmap der Features

Abbildung 4.2 beschreibt die Koeffizienten der Korrelation der einzelnen Features zueinander. Der Koeffizient der Korrelation wird paarweise für jede Metrik ermittelt. Dabei kann der Koeffizient wie gefolgt interpretiert werden:

- Zwischen -1.0 und 0: Dies wird als eine negative Korrelation bezeichnet. Je größer der eine Wert, desto kleiner wird der andere. Je kleiner der Koeffizient, desto größer der Einfluss.
- Gleich 0: Keine Korrelation ist vorzufinden.
- Zwischen 0 und 1.0: Ein Koeffizient in diesem Wertebereich wird als positive Korrelation benannt. Je größer der eine Wert, desto größer wird der andere. Je größer der Koeffizient, desto größer der Einfluss.

Bei der Datenanalyse von Features wird eine Koeffizienten-Heatmap dazu verwendet, um gegebenenfalls Komponenten aus dem Feature-Vektor zu entfernen, um dessen Dimensionalität zu reduzieren. Die Dimensionalität beschreibt in diesem Kontext die Anzahl der Komponenten im Feature-Vektor. Je höher die Dimensionalität, desto mehr Möglichkeiten müssen berücksichtigt werden, um im Falle der Klassifikation eine möglichst passende Klasse zuzuordnen. Je kleiner die Dimensionalität, desto präziser kann klassifiziert werden. Features mit einem besonders geringen oder hohen Koeffizient der Korrelation zu einer anderen encodieren ähnliche oder gleiche Informationen über die Entität, weshalb diese unter Umständen nicht mehr als Eingabe für Klassifizierer berücksichtigt werden. Im Falle der Heatmap aus Abbildung 4.2 sind besonders hohe Koeffizienten der Korrelation vorzufinden. Jedoch werden unter Berücksichtigung des Domainwissens alle Features für die Klassifikation verwendet.

4.4 Training und Validation der Klassifizierer

Die in Sektion 2.5.3 beschriebenen Klassifizierer, Random Forest Classifier, Support Vector Machine (SVM) und k-Nearest-Classifier (KNN) werden verwendet. Die Implementierung dieser Klassifizierer und andere Funktionalitäten werden durch die Machine Learning Bibliothek scikit-learn [Ped11] zur Verfügung gestellt. Der Prozess der Optimierung und das Training der Klassifizierer wird durch die Bibliothek hyperopt-sklearn [Kom19] realisiert. Dabei kann der Trainingsprozess in folgende Schritte unterteilt werden:

1. **Aufteilen des Datensatzes**: Für das Training wird der Datensatz aus Sektion 4.3 in einen Trainings- und Validationsdatensatz aufgeteilt. Dabei werden 70 % der Datenpunkte dem Trainingsdatensatz und die restlichen 30 % dem Validationsdatensatz zugewiesen. Die Rollen, die als Python String dargestellt werden, werden durch einen sklearn LabelEncoder ein kategorischer Integer-Wert zugewiesen.

2. **Initialisierung der Klassifizierer**: Die Instanzen der Klassifizierer werden mit den Standardwerten für die Hyperparameter initialisiert, die von hyperopt-sklearn zur Verfügung gestellt werden. Zusätzlich wird für jedes Modell einen Zahlenwert zwischen 0.0 und 1.0 definiert. Dieses gibt an, mit welcher Wahrscheinlichkeit das jeweilige Modell in einer Trainingsiteration ausgewählt wird. Falls notwendig, wird bei der Instanziierung der Klassifizierer ein zusätzlicher Schritt für die Skalierung der Werte der Datenpunkte durchgeführt. Dies ist der Fall bei SVM oder KNN, da bei unskalierten Datenwerten die Klassifikationsleistung von SVM und KNN negativ beeinträchtigt werden kann. Dabei wird der durch sklearn zur Verfügung gestellte StandardScaler verwendet, der Features standardisiert, indem dieser den Mittelwert entfernt und die Datenpunkte auf Einheitsvarianz skaliert. Für den Random Forrest Classifier wird die Skalierung der Datenpunkte ebenfalls angewandt, obwohl dessen Leistung davon nicht negativ beeinträchtigt wird.

3. **Starten des Trainings**: Wie bereits erwähnt, wird das Training und Optimieren der Modelle durch hyperopt-sklearn automatisiert durchgeführt. Die Instanzen der Klassifizierer werden in einer Instanz der Klasse HyperoptEstimator gebündelt, welche als Abstraktionsschicht das Training und das Optimieren der Hyperparameter kapselt. Bei dessen Instanziierung wird ebenfalls bestimmt, wie viele Iterationen die Modelle für das Training insgesamt durchlaufen müssen, nach welcher Metrik die Iterationen evaluiert und nach welchem Algorithmus die Permutationen der Hyperparameter-Werte bestimmt werden. Für das Training werden 20 Iterationen hergenommen und die Evaluation jeder Iteration erfolgt durch Ermittelung des *f1*-Wertes. Für die Ermittelung der nächsten Konfiguration an Hyperparameter-Werten wird Tree of Parzen Estimators verwendet.

4. **Kreuzvalidierung der besten Iteration**: Mit einem k-Wert von 5 wird die Kreuzvalidierung mit dem Validationsdatensatz und dem *f1*-Wert als Evaluierungsmetrik an dem optimierten Model durchgeführt.

4.5 Bestimmung des Design Patterns

Algorithm 1 Funktion zur Bewertung von Rollenübereinstimmungen

Require: *role_encoder*: Der LabelEncoder für Rollen;
Require: $roles\_df$: Pandas DataFrame mit Rollendaten
Require: *model*: Trainiertes Modell zur Vorhersage von Rollen
Require: *dps*: Hash Map mit Rollendefinitionen pro Entwurfsmuster

function MATCH_PATTERN(*role_encoder*, $roles\_df$, *model*, *dps*)
  $X \leftarrow$ GET_FEATURE_COLUMNS($roles\_df$)
  $y \leftarrow model.predict(X)$
  $roles \leftarrow role\_encoder.inverse\_transform(y.ravel())$
  scores $\leftarrow$ leere Hash Map
  $predicted\_role\_freq \leftarrow$ leere Hash Map
  for all *role* in *roles* **do**
    $predicted\_role\_freq[role] \leftarrow predicted\_role\_freq[role] + 1$
  end for
  for all dp, dp_roles in *dps* **do**
    $matched\_roles \leftarrow$ leere Hash Map
    $dp\_roles\_dict \leftarrow$ Konvertiere dp_roles zu Hash Map
    for all *role* in $predicted\_role\_freq$ **do**
      if *role* in dp_roles_dict **then**
        $dp\_role \leftarrow dp\_roles\_dict[role]$
        if ($dp\_role.mutliple\_occurences$ oder $predicted\_role\_freq[role] = 1$) und $matched\_roles[role] < predicted\_role\_freq[role]$ **then**
          $matched\_roles[role] \leftarrow matched\_roles[role] + 1$
        end if
      end if
    end for
    $total\_possible\_matches \leftarrow$ Summe der möglichen Übereinstimmungen
    score $\leftarrow$ Berechne Übereinstimmungswert
    $scores[dp] \leftarrow score$
  end for
  return *scores*
end function

Der Pseudocode 1 beschreibt die konkrete Implementierung das in der Sektion 3.4 erläuterten Algorithmus. Dabei wird der trainierte Klassifizierer initialisiert, die Metriken aus 3.1 für die zu klassifizierenden Entitäten aus dem Quellcode ermittelt und als Eingabeparameter für die vorgestellte Funktion verwendet. Die Berechnung des Übereinstimmungswertes erfolgt, indem zunächst die Gesamtzahl möglicher Übereinstimmungen bestimmt wird, die sowohl einmalige als auch mehrfache Vorkommen von Rollen berücksichtigt. Der Übereinstimmungswert für jedes Entwurfsmuster wird als das Verhältnis der Anzahl abgeglichener Rollen zur Gesamtzahl möglicher Übereinstimmungen berechnet. Dieser Wert gibt an, wie gut die vorhergesagten Rollen mit den für das Design Pattern definierten Rollen übereinstimmen. Der höchste Übereinstimmungswert wird als bestmögliche Antwort betrachtet.

4.6 Evaluation der Ergebnisse

Hier wird die Leistungskapazität des Klassifizierers und die des Algorithmus beurteilt, der anhand der identifizierten Rollen der Mikroarchitektur ein Design Pattern zuweist. Für die Evaluation werden zufällig vier Instanzen für jedes Design Patterns aus dem Validationsdatensatz ausgewählt. Diese repräsentieren neue Datenpunkte, worauf der Klassifizierer nicht trainiert wurde. Den Entitäten aus den Validierungsinstanzen wird durch den Klassifizierer Rollen zugeordnet. Die Rolle ist die, die nach dem Klassifizierer am ehesten für die ermittelten Metriken passt.

Tabelle 4.5 Ergebnisse der Evaluation

Entwurfsmuster	Precision	Recall	f1
Adapter	0.67	0.5	0.57
Command	0.12	0.25	0.17
Observer	0.5	0.5	0.5
Singleton	0.0	0.0	0.0

Die Tabelle 4.5 listet die Werte der Metriken, nachdem die in dieser Arbeit vorgestellte Methode beurteilt wird. Je höher der Wert der Metriken aus Tabelle 4.5, desto besser ist, die Klassifizierungsleistung des Verfahrens zu beurteilen. Dabei werden die Instanzen des Singletion-Entwurfsmusters nicht erkannt. Diese werden in diesem Verfahren als Instanzen des Observer-Entwurfsmusters zugeordnet, da den Singleton-Instanzen der Rolle ädapteräus dem Adapter-Design-Pattern zugewiesen wird. Die wäre damit zu erklären, dass für die ßingletonRolle aus dem Singleton-Entwurfsmuster und die ädapterRolle ähnliche Metriken ermittelt werden.

Tabelle 4.6 Metriken für Rollen aus Validationsdatensatz

Rolle	Precision	Recall	f1
adaptee	0.02	0.33	0.04
adapter	0.00	0.00	0.00
client	0.00	0.00	0.00
command	0.00	0.00	0.00
concreteCommand	0.43	0.93	0.59
concreteObserver	0.00	0.00	0.00
concreteSubject	0.00	0.00	0.00
invoker	0.10	0.25	0.14
observer	0.00	0.00	0.00
receiver	0.00	0.00	0.00
singleton	0.00	0.00	0.00
subject	0.00	0.00	0.00
target	0.00	0.00	0.00

Die Tabelle 4.6 zeigt die Metriken für die einzelnen Rollen. Dabei wird der gesamte Validationsdatensatz verwendet. Als Klassfizier wird ein Random Forest Classifier verwendet, welches im vorherigen Schritt des Trainings die beste Leistung erbrachte. Wie aus den Werten abzulesen ist, ist die Leistung des Klassifizierers nicht zufriedenstellend. Die meisten Rollen werden nicht korrekt erkannt. Das kann verschiedene Gründe haben. Zu einem können

sich die Werte der Features der einzelnen Quellcodeentitäten in Aspekten von Umfang und Komplexität aufgrund der Varianz an Implementierungsmöglichkeiten unterscheiden. Entitäten mit gleicher Rolle besitzen stark voneinander abweichende Metriken. Somit entstehen Outlier-Werte, welche die Klassifikation negativ beeinflussen. Zusätzlich muss der Fall berücksichtigt werden, dass die Features der Rollen entweder zu ähnlich zueinander sind oder der Umfang des Datensatzes nicht ausreichend ist. Zusammenfassend ist die Leistung des Klassifizierers als mangelhaft einzustufen und weist zusätzlichen Arbeitsbedarf auf.

5 Schluss

5.1 Fazit

Zusammenfassend ist das Ergebnis dieser Arbeit als negativ zu beurteilen. Zwar ist der Algorithmus für das Zuweisen der Entwurfsmuster in der Lage, teilweise das korrekte Design Pattern zuzuordnen, jedoch ist dessen Aussagekraft von der Leistung des Klassifizierers abhängig. Wie in der Sektion der Evaluation erläutert, ist diese als mangelhaft zu beurteilen. Die Problematik besteht darin, dass für das Trainieren von Machine Learning Modellen für Klassifikationsaufgaben keine definitiven Prozesse existieren. Dies gilt vor allem für die Bestimmung der Features, die durch den verfügbaren Datensatz und dessen Qualität und Umfang limitiert sind. Wie in Sektion 4.3 zu sehen ist, ist das Verhältnis zwischen Datenpunkten und Design Pattern von dem Umfang der einzelnen Implementierungen abhängig. Dadurch, dass die Design Patterns so unbalanciert im Datensatz repräsentiert sind, kann es vorkommen, dass einige Design-Pattern-Rollen überrepräsentiert, während andere unterrepräsentiert sind. Zudem ist die Konfiguration der Hyperparameter ebenfalls eine weitere Methode, um die Klassifikationsleistung zu verbessern. Zwar stellen automatisiertes Hyperparameter-Tuning und der Einsatz von Kreuzvalidierung eine Option dar, um die Leistung des Klassifizierers zu optimieren und zu validieren, jedoch ist das beste Ergebnis dadurch nicht garantiert. Es besteht immer Wahrscheinlichkeit, dass die bestmögliche Konfiguration an Hyperparameter-Werten außerhalb des Suchraumes liegt.

Insgesamt gesehen ist es davon auszugehen, dass verschiedene Aspekte und Optionen existieren, um die hier vorgeschlagene Methode weiter zu verbessern. Entweder kann unter anderem ein anderer Katalog an Metriken als Features genutzt werden oder es werden andere Klassifizierer verwendet.

5.2 Zukünftige Aussichten

Die fortschrittlichen Entwicklungen im Bereich des maschinellen Lernens (ML) und speziell der Large Language Models (LLMs) eröffnen neue Perspektiven für die automatisierte Erkennung von Design Patterns in Quellcode. In diesem Feld wurden zum Zeitpunkt des Verfassens der Arbeit Fortschritte getätigt, da LLMs das Potenzial geben, die Effizienz und Genauigkeit bei der Identifizierung von Design Patterns erheblich zu verbessern. Im Folgenden werden die zukünftigen Aussichten dieser Technologie beleuchtet.

1. **Erweiterte Erkennungskapazitäten**: Mit der zunehmenden Verfeinerung von LLMs ist zu erwarten, dass ihre Fähigkeit, komplexe Muster und Abstraktionen im Quellcode zu erkennen, deutlich zunimmt. Diese Modelle können aus einer umfangreichen Datenmenge lernen und somit eine breite Palette von Design Patterns identifizieren, die in verschiedenen Programmiersprachen und -stilen zum Einsatz kommen. Die Flexibilität von LLMs ermöglicht es ihnen, auch seltene oder weniger dokumentierte Patterns zu erkennen, die herkömmliche Methoden möglicherweise übersehen.

2. **Verbesserung der Präzision und Reduzierung von Fehlalarmen**: Durch das Training mit großen Datensätzen können LLMs nicht nur eine Vielzahl von Patterns erkennen, sondern auch den Kontext, in dem diese Patterns verwendet werden, besser verstehen. Dies führt zu einer höheren Präzision bei der Erkennung und einer signifikanten Reduzierung von Fehlalarmen. Die Fähigkeit, den Kontext zu berücksichtigen, ist besonders wichtig, da viele Design Patterns nur in bestimmten Situationen angemessen sind. LLMs können feine Unterschiede im Code erfassen, die darauf hinweisen, ob ein bestimmtes Pattern tatsächlich beabsichtigt ist oder nicht.

3. **Automatisierte Verbesserungsvorschläge und Refactoring**: Zukünftige Entwicklungen könnten LLMs befähigen, nicht nur existierende Patterns zu erkennen, sondern auch Verbesserungsvorschläge zu machen. Basierend auf der erkannten Implementierung eines Design Patterns könnten diese Modelle Empfehlungen für ein effizienteres oder klareres Pattern geben, das in den aktuellen Code eingeführt werden kann. Ferner ist das Potenzial für automatisiertes Refactoring beträchtlich, wobei LLMs Vorschläge für Code-Umstrukturierungen machen können, um Design Principles wie SOLID besser einzuhalten.

4. **Interaktive Entwicklungsumgebungen**: Die Integration von LLMs in Entwicklungsumgebungen und IDEs (Integrated Development Environments) könnte zu einer interaktiveren und unterstützenden Codierungserfahrung führen. Entwickler könnten in Echtzeit Feedback zu den von ihnen verwendeten Design Patterns erhalten, einschließlich Hinweisen zur Anwendung und möglichen Optimierungen. Diese Art der direkten Integration fördert ein tieferes Verständnis für Design Patterns und unterstützt Entwickler dabei, best practices effektiver in ihren Code zu integrieren.

Literaturverzeichnis

[AA19] H. A. Abu Alfeilat, A. B. Hassanat, O. Lasassmeh, A. S. Tarawneh, M. B. Alhasanat, H. S. Eyal Salman und V. S. Prasath. Effects of Distance Measure Choice on K-Nearest Neighbor Classifier Performance: A Review. *Big Data*, 7(4):221–248, 2019. PMID: 31411491.

[Alh16] S. Alhusain. *Intelligent Data-Driven Reverse Engineering of Software Design Patterns*. Dissertation, 03 2016.

[Ani15] M. Aniche. Java code metrics calculator (CK). `https://github.com/mauricioaniche/ck/`, 2015.

[Ber11] J. Bergstra, R. Bardenet, Y. Bengio und B. Kégl. Algorithms for hyper-parameter optimization. *Advances in neural information processing systems*, 24, 2011.

[Bin09] A. Binun und G. Kniesel-Wünsche. Witnessing Patterns: A Data Fusion Approach to Design Pattern Detection. 05 2009.

[Chi94] S. R. Chidamber und C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on software engineering*, 20(6):476–493, 1994.

[Eri21] B. J. Erickson und F. Kitamura. Magician's Corner: 9. Performance Metrics for Machine Learning Models. *Radiology: Artificial Intelligence*, 3(3):e200126, 2021.

[Gam94] E. Gamma, R. Helm, R. Johnson und J. M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1 Aufl., 1994.

[Gru15] J. Grus. *Data Science from Scratch: First Principles with Python*. O'Reilly Media, Inc., 1st Aufl., 2015.

[Kom19] B. Komer, J. Bergstra und C. Eliasmith. *Hyperopt-Sklearn*, S. 97–111. Springer International Publishing, Cham, 2019.

[Kra01] C. Kramer, C. Gmbh, L. Prechelt und F. Informatik. Design Recovery by Automated Search for Structural Design Patterns in Object-Oriented Software. 04 2001.

[Kuc04] P. Kuchana. *Software Architecture Design Patterns in Java*. Auerbach Publications, USA, 2004.

[Leh96] M. M. Lehman. Laws of software evolution revisited. In C. Montangero, Hg., *Software Process Technology*, S. 108–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 1996.

[Lia19] P. Liashchynskyi und P. Liashchynskyi. Grid Search, Random Search, Genetic Algorithm: A Big Comparison for NAS. *CoRR*, abs/1912.06059, 2019.

[McC76] T. J. McCabe. A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320, 1976.

[Mä10] G. R. I. P. P. Mäder. Design pattern recovery based on annotations. *Advances in Engineering Software (Thomson Reuters) 2010-apr vol. 41 iss. 4*, 41, apr 2010.

[Naz20] N. Nazar und A. Aleti. Feature-Based Software Design Pattern Detection. *CoRR*, abs/2012.01708, 2020.

[Ped11] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot und E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[Pra15] P. Pradhan, A. K. Dwivedi und S. K. Rath. Detection of design pattern using Graph Isomorphism and Normalized Cross Correlation. In *2015 Eighth International Conference on Contemporary Computing (IC3)*, S. 208–213. 2015.

[Siu98] M.-K. Siu. Introduction to graph theory (4th edition), by Robin J. Wilson. Pp. 171. £14.99. 1996. ISBN : 0-582-24993-7 (Longman). *The Mathematical Gazette*, 82:343 – 344, 1998.

[Sno12] J. Snoek, H. Larochelle und R. P. Adams. Practical Bayesian Optimization of Machine Learning Algorithms, 2012.

[Ste08] K. Stencel und P. Wegrzynowicz. Detection of Diverse Design Pattern Variants. S. 25–32. 01 2008.

[Suh10] I. Suh, Steve D.; Neamtiu. [IEEE 2010 21st Australian Software Engineering Conference - Aukland, New Zealand (2010.04.6-2010.04.9)] 2010 21st Australian Software Engineering Conference - Studying Software Evolution for Taming Software Complexity. 2010.

[Uch14] S. Uchiyama, A. Kubo, H. Washizaki und Y. Fukazawa. Detecting Design Patterns in Object-Oriented Program Source Code by Using Metrics and Machine Learning. *Journal of Software Engineering and Applications*, 7, 01 2014.

[Van16] J. VanderPlas. *Python Data Science Handbook: Essential Tools for Working with Data*. O'Reilly Media, Inc., 1st Aufl., 2016.

[Wan22] L. Wang, T. Song, H.-N. Song und S. Zhang. Research on Design Pattern Detection Method Based on UML Model with Extended Image Information and Deep Learning. *Applied Sciences*, 12(17), 2022.

[Yar20] H. Yarahmadi und S. M. H. Hasheminejad. Design pattern detection approaches: A systematic review of the literature. *Artificial Intelligence Review*, 53:5789–5846, 2020.

[Yu13] D. Yu, Y. Zhang, J. Ge und W. Wu. From Sub-patterns to Patterns: An Approach to the Detection of Structural Design Pattern Instances by Subgraph Mining and Merging. In *2013 IEEE 37th Annual Computer Software and Applications Conference*, S. 579–588. 2013.

[Zhi15] J. Zhi, V. Garousi-Yusifoğlu, B. Sun, G. Garousi, S. Shahnewaz und G. Ruhe. Cost, benefits and quality of software development documentation: A systematic mapping. *Journal of Systems and Software*, 99:175–198, 2015.