



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: tt.mm.jjjj

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.

Rosenheim, den tt.mm.jjjj

Vor- und Zuname

Kurzfassung

text

Schlagworte:

Inhaltsverzeichnis

1	Motivation	1
1.1	Einführung in Design Patterns	2
1.2	Untersuchungsfragen	3
1.3	Übersicht des Prozesses	3
2	Literaturrecherche	4
2.1	Design Patterns in der Software-Entwicklung	5
2.1.1	Design Pattern Katalog	6
2.1.2	Rollenkatalog	7
2.2	Herausforderungen und Probleme bei der Erkennung von Design Patterns	8
2.2.1	Variabilität der Implementierung	8
2.2.2	Steigende Komplexität von Software-Systemen	8
2.2.3	Iterative Evolution des Quellcodes	9
2.2.4	Mangel an expliziter Dokumentation	9
2.3	Angewandte Ansätze für die Erkennung von Design Patterns	10
2.3.1	Graphen-basierende Ansätze	10
2.3.2	Maschine Learning Ansätze	11
2.3.3	Similarity-Scoring Ansätze	11
2.3.4	Diverse Ansätze	11
3	Methodologie	12
3.1	Verfügbare Datensätze	13
3.2	Extrahierte Features	14
3.3	Betrachtete Multi-Klassifizierer für Rollen	15
3.4	Training und Hyperparameter-Tuning des Klassifizierers	16
3.5	Evaluation des trainierten Models	17
3.6	Ermittlung übereinstimmendes Design Patterns	18
3.7	Diskussion der Methodologie	19
4	Implementierung	20
4.1	Präprozessierung des Datensatzes	21
4.2	Extraktion von Features aus Quellcode	22
4.3	Analyse des Datensatzes	23
4.4	Training und Hyperparameter-Tuning	24
4.5	Bestimmung des Design Patterns	25
4.6	Evaluation der Ergebnisse	26
5	Schluss	27
5.1	Fazit	28
5.2	Zukünftige Aussichten	29
A	Erstes Kapitel des Anhangs	30

Abbildungsverzeichnis

2.1 Verteilung der Kategorien der Ansätze für die Erkennung von Design Patterns in Quellcode	10
---	----

Tabellenverzeichnis

1 Motivation

1.1 Einführung in Design Patterns

Entwurfsmuster oder auf Englisch ‘Design Patterns’ sind bewährte Lösungsansätze für wiederkehrende Probleme, die bei der Konzeption der Software-Architektur oder während der Implementierung der Software eingesetzt werden kann. 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 das wiederum 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 variierenden 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 werden hinzugefügt. Währenddessen besteht die Gelegenheit, dass durch mangelnder Dokumentation oder anderer Gründe die Entscheidungen, weshalb Entwurfsmuster so eingesetzt sind wie es eingesetzt worden, verloren gehen. Dadurch besteht die Gefahr, dass angewendete Design Patterns im weiteren Verlauf derer Entwicklung nicht mehr wiederzuerkennen sind. Aus diesem Grund ist die Etablierung eines Prozesses von Vorteil, das in der Lage ist, Implementierungen von Entwurfsmustern aus einem Software-System zu extrahieren und dieses konkret benennen. Vor allem der Einsatz von Maschine Learning für die Klassifizierung ist hier vorteilhaft, wodurch das Potenzial besteht, vorher nicht gesehene Implementierung von Design Patterns zu erkennen. Durch solch einen Prozess können durch die Erkennung von eingesetzten Entwurfsmustern auf konkrete und verlorenen gegangene Design-Entscheidungen zurückgeschlossen werden, welche zukünftige Design-Entscheidungen für das Software-System beeinflussen können. Der Fokus dieser Arbeit besteht daran, solch ein Prozess zu etablieren, welches für ein gegebenes Set von Quellcode-Dateien mithilfe von Maschine Learning einem potenziellen Entwurfsmuster zuzuteilen.

1.2 Untersuchungsfragen

Das Ziel dieser Arbeit besteht aus der Etablierung eines Prozesses, womit durch Einsatz von Maschine Learning für ein Set von Quellcode-Dateien ein Design Pattern zuzuordnen. Hierfür dienen eine Menge an Quelldateien als Eingabe für den Prozess und durch phasenweiser Transformationen und Bearbeitungen sollen ein möglichst passendes aus dem in Kontext dieser Arbeit betrachteten Entwurfsmusters zugeordnet werden. Um solch ein Prozess zu entwickeln, werden in Kontext dieser Arbeit folgende Fragen beantwortet:

RQ1. Welche Design Patterns werden berücksichtigt?

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

RQ3. Wonach wird exakt klassifiziert?

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

RQ5. Welche Klassifizierer eignen sich?

RQ6. Wie ist das Endresultat zu beurteilen?

1.3 Übersicht des Prozesses

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, ist das von Gamma et al. verfasste Werk "Design Patterns: Elements of Reusable Object-Oriented Software". In diesem wird ein Katalog von 23 Entwurfsmustern definiert, welche in drei Kategorien aufgeteilt. Dieser Katalog wird von Software-Entwicklern als "Gang of Four"-Entwurfsmuster 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, welches das 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 darstellen, 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 bzw. abstrakte Klassen oder Schnittstellen definiert, die die erforderliche Rolle im Rahmen der Mikroarchitektur erfüllen.

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

2.1.1 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 Instantiierung auf andere Objekte delegieren. Creational Design Patterns werden dann bedeutend, wenn mit steigender Komplexität des Software-Systems sich von Vererbung distanziert wird und die Komposition aus einzelnen definierten Objekt mehr an Bedeutung gewinnt[Gam94, S. 81]. Dabei wird das Verhalten einer Komponente auf eine Menge von einzelnen kleinere Objekten delegiert und durch Zusammensetzung innerhalb der Komponente und deren Interaktion das erwünschte Verhalten erzeugt. Dadurch wird die Instantiierung von Software-Komponenten komplexer, da die Instantiierung von mehreren Objekten koordiniert werden muss. Creational Design Patterns liefern hierbei Hilfestellung, weil die exakte Komposition der konkreten Objekte, die Teil der zu instantiierenden Komponente sind, und der exakte Prozess der Instantiierung im Inneren des Entwurfsmusters verborgen werden. Nach außen hin sind dahingegen nur die Schnittstellen sichtbar, die die Komponente zur Verfügung stellt, während dessen interne Logik die Ausführung auf andere Objekte delegiert. Im Kontext dieser Arbeit werden folgende Entwurfsmuster aus der Kategorie der Creational Design Patterns betrachtet:

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:

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:

2.1.2 Rollenkatalog

2.2 Herausforderungen und Probleme bei der Erkennung von Design Patterns

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

2.2.1 Variabilität der Implementierung

Design Patterns stellen in der Software-Entwicklung bewährte Lösungsmuster für bereits begegnete Herausforderung 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. Dadurch, dass jedes Entwurfsmuster einen konzeptionellen Rahmen darstellt und jede Implementierung von nicht statischer Außenfaktoren beeinflusst wird, resultiert dies in einem breiten Spektrum an Implementierungen für ein gegebenes Entwurfsmuster. Aus diesem Grund ist eine Definition einer starren 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, nicht beantwortbar, weshalb die Antwort von automatisierten Prozessen von Design Patterns eher mit einem Wert besteht, welches die Ähnlichkeit zu einem Design Pattern beschreibt. Um einen zufriedenstellenden Wert für diese Frage zu liefern, bedarf es eines breiten Spektrums an Implementierungsvariationen als Referenz für die Erkennung.

2.2.2 Steigende Komplexität von Software-Systemen

Die steigende Komplexität von Software-Systemen stellt eine erhebliche Herausforderung bei der Erkennung von Entwurfsmustern in Entwurfsmustern 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 tendieren 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 der ursprünglichen leichter zu identifizierbaren Iterationen weiter abweichen. Deshalb beinhaltet die Erkennung von Entwurfsmustern nicht nur auf die momentane Iteration, sondern auch die Erfassung derer historischen Evolution und die Entwicklung dieser innerhalb der Codebasis.

2.2.3 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 erwartete Art und Weise erfüllt werden können. Implementierungen von 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 wie das 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 berücksichtigen werden muss. In automatisierten Prozess der Erkennung von Entwurfsmustern kann dieser Aspekt nur bedingt berücksichtigt, weil das Einschließen der Historie der betrachtenden Implementierung und dessen Kontext in der Codebasis nicht pauschal und in einer allgemeinen Ansicht betrachtet werden kann.

2.2.4 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 in diese Tätigkeit relativ wenig Zeit investiert wird [Zhi15, S. 162]. Dies ist die Folge der Dominanz des agilen Software-Entwicklungsprozesses, in dieser die Verwendung von Zeit und Ressourcen für die Dokumentation eher als Verschwendung betrachtet wird, da diese keinen direkten Mehrwert für die Auslieferung des Software-Produktes an den Endkunden liefert [Zhi15, S. 159]. Das 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 Erfahrungen, 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 Prozess dar, in der 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 und dieses als Teil des Prozesses mitincludieren.

2.3 Angewandte Ansätze für die Erkennung von Design Patterns

Bei der Erkennung von Design Patterns in Quellcode wurden verschiedene Verfahren entwickelt werden, 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, um Design Patterns in Quellcode zu erkennen, und kategorisierten diese [Yar20, S. 5805].

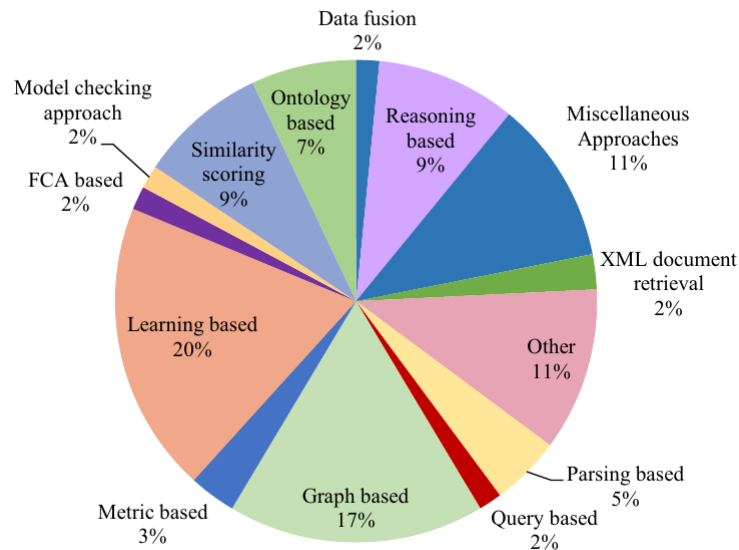


Abbildung 2.1 Verteilung der Kategorien der Ansätze für die Erkennung von Design Patterns in Quellcode

Wie aus Figur 2.1 zu entnehmen ist, lassen sich die entwickelten Prozesse auf eine begrenzte Menge an Kategorien eingestuft werden. In dieser Sektion der Arbeit werden die vier größten Kategorien aus Figur 2.1 genauer erläutert und es werden exemplarisch Arbeiten diskutiert, die in die jeweilige Kategorie zugeordnet werden können.

2.3.1 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 bereits 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, welches 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 dar, 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 Knoten 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 OO-Programmierung aufgefasst werden. Pradhan et al. nutzen Klassendiagramme als Eingabe für ihre entworfene Methode und generieren diese für das zu analysierende Software-System und Implementierung 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 darstellen. Zusätzlich werden die Kanten je nach Art der Assoziation unterschiedlich gewichtet [Pra15, S. 2]. Im weiteren Verlauf werden mögliche Menge Kandidaten aus dem Graphen des Software-Systems als dessen Subgraphen durch die Eigenschaft der Graphenisomorphie extrahiert und anhand der normalisierten Kreuzrelation der Maß der Übereinstimmung bestimmt [Pra15, S. 3]. Graphenisomorphie beschreibt, ob zwei Graphen die strukturell identisch sind, sodass jede Kante des einen Graphen einer Kante im anderen Graphen entspricht und umgekehrt [Siu98, S. 10]. Die normalisierten Kreuzrelation ist ein Maß, dessen Wertebereich zwischen 0.0 und 1.0 definiert ist. Je näher der Wert an 1.0, desto identischer sind die zwei Graphen.

2.3.2 Maschine Learning Ansätze

2.3.3 Similarity-Scoring Ansätze

2.3.4 Diverse Ansätze

3 Methodologie

3.1 Verfügbare Datensätze

3.2 Extrahierte Features

3.3 Betrachtete Multi-Klassifizierer für Rollen

3.4 Training und Hyperparameter-Tuning des Klassifizierers

3.5 Evaluation des trainierten Models

3.6 Ermittlung übereinstimmendes Design Patterns

3.7 Diskussion der Methodologie

4 Implementierung

4.1 Präprozessierung des Datensatzes

4.2 Extraktion von Features aus Quellcode

4.3 Analyse des Datensatzes

4.4 Training und Hyperparameter-Tuning

4.5 Bestimmung des Design Patterns

4.6 Evaluation der Ergebnisse

5 Schluss

5.1 Fazit

5.2 Zukünftige Aussichten

A Erstes Kapitel des Anhangs

Wenn Sie keinen Anhang benötigen, dann bitte einfach rausnehmen.

Literaturverzeichnis

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