2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM)

Formale Definition und automatische Generierung von semantischen Metriken: Eine empirische Studie zur Fehlervorhersage

Ting Hu[1] , Ran Mo* [1], Pu Xiong[1] , Zengyang Li [1], Qiong Feng [2]
[1]Schule *für Informatik, Central China Normal University*
[2]Schule *für Informatik und Ingenieurwesen, Nanjing University of Science and Technology*
E-Mail: 826473959@qq.com, moran@mail.ccnu.edu.cn, mos365@hotmail.com, zengyangli@mail.ccnu.edu.cn,qiongfeng@njust.edu.cn

***Abstrakt* - Fehlervorhersage ist hilfreich für die Erleichterung der Fehlerbehebung und die Verbesserung der Effizienz bei der Softwareentwicklung und -wartung. In den vergangenen Jahrzehnten haben Forscher zahlreiche Studien zur Fehlervorhersage mit Hilfe von Code-Metriken vorgeschlagen. Die meisten der existierenden Studien verwenden jedoch syntaxbasierte Metriken. Es gibt nur wenige Arbeiten, die Modelle zur Fehlervorhersage mit semantischen Metriken aus dem Quellcode erstellen. In dieser Arbeit schlagen wir ein neues Modell vor, den semantischen Abhängigkeitsgraphen *(SDG)*, um semantische Beziehungen zwischen Quelltexten darzustellen. Auf der Grundlage des *SDG* definieren wir formal eine Reihe semantischer Metriken, die die semantischen Eigenschaften der Quelltexte eines Projekts reflektieren. Darüber hinaus haben wir ein Tool entwickelt, mit dem sich die Generierung der von uns vorgeschlagenen SDG-basierten Metriken automatisieren lässt. Durch unsere experimentellen Studien haben wir gezeigt, dass die SDG-basierten semantischen Metriken effektiv für die Erstellung von Fehlervorhersagemodellen sind und dass die SDG-basierten Metriken die traditionellen syntaktischen Metriken bei der Fehlervorhersage übertreffen. Darüber hinaus konnten Modelle, die die SDG-basierten Metriken verwenden, eine bessere Vorhersageleistung erzielen als zwei State-of-the-Art-Modelle, die semantische Merkmale automatisch lernen. Schließlich haben wir auch gezeigt, dass unser Ansatz in der Praxis in Bezug auf Ausführungszeit und Platzbedarf anwendbar ist.**

***Indexbegriffe* - *Semantische* Code-Metriken, Semantische Abhängigkeit, Fehlervorhersage**

I. EINFÜHRUNG

Die Fehlervorhersage ist ein aktives und wichtiges Forschungsgebiet. Genaue Vorhersagen helfen Entwicklern dabei, fehleranfällige Produkte frühzeitig zu erkennen und deren Prüfung und Test zu priorisieren, was wiederum die Effizienz und Produktivität von Entwicklungsteams verbessern könnte. In den vergangenen Jahrzehnten haben Forscher zahlreiche Ansätze und Werkzeuge zur Vorhersage fehleranfälliger Fehler vorgeschlagen.

Die Verwendung von Code-Metriken zur Erstellung von Vorhersagemodellen ist eine der am weitesten verbreiteten Richtungen auf dem Gebiet der Fehlervorhersage. Zahlreiche Studien [1], [2], [3], [4], [5], [6], [7] haben verschiedene Code-Metriken oder deren Kombinationen zur Vorhersage von fehleranfälligen Dateien genutzt. Die meisten der vorhandenen Studien verwenden jedoch nur syntaktische Codemetriken für die Vorhersage von Fehlern; es gibt nur wenige Arbeiten, die Vorhersagemodelle auf der Grundlage semantischer Codemetriken entwickeln.

Im Allgemeinen enthält ein Programm syntaktische Informationen, die sich darauf beziehen, wie ein Programm dargestellt wird. Basierend auf den syntaktischen Aspekten von Programmen wurden verschiedene syntaktische Metriken vorgeschlagen, die im Laufe der Zeit zu den am häufigsten verwendeten Code-Metriken wurden

*Korrespondierender Autor

der letzten Jahrzehnte. Zum Beispiel misst die bekannte LOC (lines of code) die Größe eines Programms; McCabe's *zyklomatische Komplexität* [8] reflektiert die Anzahl der linear unabhängigen Pfade durch den Quellcode eines Programms; die Suite von C&K-Metriken [9] besteht aus sechs Metriken, die sich auf die Messung der Eigenschaften objektorientierter Programme konzentrieren, wie z.B. die Anzahl der Abhängigkeiten, der Vererbungen und der Methoden usw.; Fan-in berechnet die Anzahl der aufrufenden Programme, Fan-out die Anzahl der aufgerufenen Programme. Alle diese syntaktischen Metriken sind formal definiert und werden häufig zur Fehlervorhersage verwendet [2], [4], [5], [10], [6].

Ein Programm enthält nicht nur syntaktische Informationen, sondern auch semantische Informationen, die die textuellen Beziehungen zwischen Programmen beschreiben. Im Vergleich zur Syntax eines Programms sind die semantischen Informationen oft impliziter und tiefer im Quellcode eines Programms verborgen. Im Gegensatz zu den syntaktischen Metriken, die gut definiert sind und für deren Extraktion und Berechnung es zahlreiche Open-Source- oder kommerzielle Tools gibt, fehlt es daher bisher an einer Reihe gut definierter semantischer Metriken, ganz zu schweigen von den Tools, die die automatische Erkennung solcher Metriken unterstützen. In der Zwischenzeit wurden semantische Informationen zwar für Code-Vorschläge verwendet [11], [12] und der Code-Vervollständigung [13], [14] gibt es nur wenige Arbeiten, die semantische Codemetriken für die Entwicklung von Fehlervorhersagemodellen gut konstruieren.

Um die Lücke zwischen semantischen Metriken und Fehlerdiagnose zu schließen, schlagen wir eine Reihe von semantischen Metriken mit expliziten Definitionen und Beschreibungen vor und entwickeln ein Werkzeug zur automatischen Generierung dieser vorgeschlagenen Metriken. In dieser Arbeit extrahieren wir zunächst semantische Informationen aus Quelltexten mit Hilfe der latenten semantischen Analyse (LSA) [15] und dem TD- IDF-Maß [16]. Anhand der extrahierten semantischen Informationen schlagen wir vor, eine semantische Ähnlichkeitsmatrix und ein neuartiges Modell, den *Semantic Dependency Graph (SDG),* zu erstellen, das die semantischen Abhängigkeiten zwischen den files darstellt. Die Knoten eines *SDG* sind die Quelltexte eines Projekts, und jede Kante, $e(f_i, f_j)$, zeigt an, dass eine semantische Abhängigkeit zwischen den Texten i und j besteht. Auf der Grundlage der generierten Ähnlichkeitsmatrix und des semantischen Abhängigkeitsgraphen werden elf SDG-basierte semantische Metriken formell definiert. Unter Verwendung der vorgeschlagenen SDG-basierten semantischen Metriken als

978-1-6654-4897-0/21/$31.00 ©2021 IEEE92
DOI 10.1109/SCAM52516.2021.00020

2021 IEEE 21st International Working Conference on Source Code Analysis and Manipulation (SCAM) | 978-1-6654-4897-0/21/$31.00 ©2021 IEEE | DOI: 10.1109/SCAM52516.2021.00020

Attribute, wenden wir außerdem Klassifizierungsverfahren des maschinellen Lernens an, um Modelle zur Fehlerprognose zu erstellen.

Um die Effektivität unserer SDG-basierten semantischen Metriken bei der Vorhersage von Fehlern zu untersuchen, versuchen wir, die folgenden Forschungsfragen zu beantworten:

RQ1: Ist es möglich, auf der Grundlage der von uns vorgeschlagenen SDG-basierten semantischen Metriken wirksame Fehlerprognosemodelle zu entwickeln? **RQ2:** Übertreffen die SDG-basierten semantischen Metriken die traditionellen syntaktischen Metriken bei der Fehlervorhersage?

RQ3: Haben die SDG-basierten semantischen Metriken eine bessere Leistung bei der Fehlervorhersage im Vergleich zum Stand der Technik?

RQ4: Wie leistungsfähig ist unser Ansatz in Bezug auf die Ausführungszeit und den Platzbedarf?

Durch unsere Analysen von sieben experimentellen Studien haben wir festgestellt, dass: 1) Durch die Verwendung unserer vorgeschlagenen SDG-basierten semantischen Metriken können wir vielversprechende Fehlerprognosemodelle erstellen, bei denen 90,5% (19/21) aller abgeleiteten Modelle ein F-Maß von mehr als 0,5 haben; 2) Modelle, die unsere SDG-basierten semantischen Metriken verwenden, übertreffen diejenigen, die syntaktische Übergangsmetriken verwenden, die durchschnittlichen Leistungssteigerungen reichen von 16.3) Verglichen mit den modernsten Ansätzen, die automatisch semantische Merkmale für die Vorhersage von Fehlern erlernen, konnten die Modelle, die die von unserem SDG-basierten Ansatz generierten semantischen Metriken verwenden, eine bessere Leistung erzielen;
4) Schließlich haben wir den gesamten Prozess der SDG-basierten Metrikgenerierung verfolgt und gezeigt, dass die Ausführungszeit und der Platzbedarf unseres Ansatzes erschwinglich sind. Über alle experimentellen Studien hinweg kostet der Prozess der semantischen Metrikgenerierung 2,9-111,6 Minuten und verbraucht 106-146 MB Speicher, was darauf hindeutet, dass unser Ansatz in der Praxis anwendbar ist.

Abschließend sind wir der Meinung, dass unsere Arbeit zum Stand der Technik wie folgt beiträgt:

- In unserer Arbeit wird ein neuartiges Modell, SDG, vorgeschlagen, das die semantischen Beziehungen zwischen Quelltexten explizit darstellen kann.
- Unsere Arbeit stellt eine systematische Methodik zur automatischen Analyse der Programmsemantik, zur Erzeugung semantischer Ähnlichkeitsmetriken und semantischer Abhängigkeitsgraphen sowie zur Extraktion der semantischen Metriken vor.
- Soweit wir wissen, ist unsere Arbeit die erste Studie, die eine Reihe von semantischen Metriken vorschlägt und formal definiert und zeigt, wie diese Metriken zur Entwicklung von Fehlervorhersagemodellen genutzt werden können.

Der Rest dieses Papiers ist wie folgt gegliedert: In Abschnitt II wird der Hintergrund unserer Arbeit dargestellt. Abschnitt III zeigt verwandte Arbeiten. Abschnitt IV beschreibt die Details unseres Ansatzes zur SDG-basierten semantischen Metrikgenerierung und Fehlervorhersage. Abschnitt V präsentiert das Design unserer empirischen Studie und unsere Evaluierungsmethoden. Abschnitt VI zeigt die Auswertungsanalyse und die Ergebnisse. Abschnitt VII erörtert die Grenzen und Gefahren für die Gültigkeit unserer Arbeit, und Abschnitt VIII schließt mit einem Fazit.

II. HINTERGRUND

In diesem Abschnitt stellen wir die grundlegenden Konzepte und Techniken vor, die unserer Arbeit zugrunde liegen, einschließlich der Beschreibung des Codes

Semantik und die Konzepte im Zusammenhang mit LSA und TF-IDF.

A. Code-Semantik

Die Semantik eines Programms bezieht sich auf die Bedeutung dieses Programms, d. h. darauf, was das Programm tut. Im Allgemeinen können die semantischen Aspekte eines Programms oft durch seine lexikalischen Token wie Schlüsselwörter, Kommentare, Operatoren und Identifikatoren usw. dargestellt werden. So können wir für jedes Programm ein entsprechendes Textdokument erstellen, das die Bedeutung des Programms darstellt. Ein solches Textdokument besteht aus einer Reihe von Textbegriffen, die aus der Analyse der lexikalischen Informationen abgeleitet wurden.

B. Latente semantische Analyse

Die latente semantische Analyse (LSA), auch bekannt als latent se- mantische Indexierung (LSI), ist eine Technik zur Untersuchung semantischer Beziehungen zwischen einer Reihe von Dokumenten und den darin enthaltenen Begriffen [15]. Beim LSA-Verfahren wird zunächst eine Term-Dokument-Matrix erstellt, um das Vorkommen von Begriffen in verschiedenen Dokumenten zu beschreiben. Bei einer solchen Term-Dokument-Matrix stehen die Zeilen für Begriffe und die Spalten für Dokumente. In dieser Arbeit nutzen wir das Modell von TD-IDF (Term Frequency-Inverse Document Frequency Method) [16], um jedes Element innerhalb der Term-Dokument-Matrix zu füllen.

Der Wert von TD-IDF misst, wie relevant ein Begriff für ein Dokument in einer Sammlung von Dokumenten ist. Nachdem alle TD-IDF-Werte in die Term-Dokument-Matrix eingefügt wurden, wendet LSA die *Singulärwertzerlegung (SVD)* [17] an, um eine Matrixzerlegung zum Herausfiltern von Störfaktoren durchzuführen. Auf diese Weise wird die Zeilendimension der ursprünglichen Term-Dokumenten-Matrix verringert, die Ähnlichkeitsstruktur zwischen den Spalten bleibt jedoch erhalten. Somit kann jede Spalte der Zerlegungsmatrix weiterhin ein bestimmtes Dokument repräsentieren.

Für zwei beliebige Dokumente d_1 und d_2 kann die semantische Ähnlichkeit der Dokumente d_1 und d direkt berechnet werden, indem der Kosinuswert zwischen V_1 und V_2 ermittelt wird. Jeder Ähnlichkeitswert kann in den Bereich [0, 1] normalisiert werden, und ein
Ein höherer Wert bedeutet, dass zwischen zwei Dokumenten eine größere semantische Ähnlichkeit besteht.

C. Begriffshäufigkeit - Inverse Dokumenthäufigkeit

Die Term Frequency-Inverse Document Frequency (TF-IDF) [16] misst, wie relevant ein Begriff oder ein Wort für ein Dokument in einem Korpus von Dokumenten ist. Sie wird häufig für Information Retrieval und Text Mining verwendet. In dieser Arbeit nutzen wir TF-IDF, um die Elemente einer Term-Dokument-Matrix aufzufüllen. Im Allgemeinen besteht TF-IDF aus zwei wichtigen Maßen: Term Frequency (TF) und Inverse Document Frequency (IDF). TF gibt an, wie oft ein Term in einem Dokument gefunden wurde: $tf_{t,d} = f_{t,d}$, wobei f die Häufigkeit des Terms $t_{t,d}$ ist, der in Dokument d vorkommt. Die Studie von Croft et al. [18] hat gezeigt, dass die logarithmierte Variante von TF die Leistung von TF-IDF in der Praxis effektiv verbessern kann. Darüber hinaus hat die Studie von Lan et al. [19] gezeigt, dass die logarithmische Variante von TF verwendet werden sollte, um eine Verzerrung der Ergebnisse aufgrund der hohen ursprünglichen Häufigkeit eines Terms in einem bestimmten Dokument zu vermeiden.

als Begriffshäufigkeit verwendet. Daher wird die TF oft wie folgt berechnet:

$$tf_{t,d} = 1 + log(f_{t,d}) \quad (1)$$

IDF ist der Kehrwert der Anzahl der Dokumente, in denen der Begriff im gesamten Korpus vorkommt. IDF wird wie folgt berechnet:

$$idf_{t,d} = log\left(\frac{N}{n_t}\right) \quad (2)$$

wobei N die Gesamtzahl der Dokumente im Dokumentenkorpus und n_t die Anzahl der Dokumente ist, die den Begriff t enthalten. TD-IDF wird schließlich als Produkt von TD und IDF berechnet: $tf\text{-}idf_{t,d} = tf_{t,d} \times idf_{t,d}$. Nach dieser Formel können wir feststellen, dass der TF-IDF-Wert logarithmisch mit dem Häufigkeit eines Begriffs in einem Dokument, sondern verringert sich um die Gesamtzahl der Dokumente, die einen Begriff enthalten. Durch die Kombination von TF- und IDF-Maßen kann TD-IDF einige häufige, aber nicht repräsentative Begriffe in Dokumenten herausfiltern.

III. VERBUNDENE ARBEITEN

Die Fehlervorhersage ist seit Jahrzehnten eines der aktivsten Forschungsgebiete. Die Forscher haben zahlreiche Vorhersagemodelle vorgeschlagen, die verschiedene Code-Metriken verwenden.

A. Syntaktische Metriken in der Fehlervorhersage.

Nagappan et al. [2] untersuchten verschiedene Komplexitätsmetriken und zeigten, dass diese Metriken nützlich und erfolgreich für die Vorhersage von fehlerhaften Fehlern sind. In dieser Arbeit sammelten die Autoren eine Reihe von Code-Metriken und schlugen ein Modell vor, das eine Reihe von ihnen als Prädiktoren verwendet. Sie wendeten das Modell auf fünf große Softwaresysteme an und zeigten, dass es Fehler nach der Freigabe genau vorhersagen kann. Obwohl ein bestimmter Satz von Metriken für ein bestimmtes Projekt ein guter Prädiktor sein kann, gibt es nicht einen einzigen Satz von Metriken, der für alle Projekte am besten geeignet ist. Das heißt, man müsste eine andere Menge von Metriken auswählen, um fehlerhafte Fehler in verschiedenen Arten von Projekten genau vorherzusagen.

Menzies et al. [3] haben gezeigt, dass die statischen Codemetriken als Fehlerprädiktoren verwendet werden können. Bei der Anwendung von Naive Bayes Learner zeigten sie, dass die Fehlerprädiktoren mit einer durchschnittlichen Erkennungswahrscheinlichkeit von 71 % und einer durchschnittlichen Fehlalarmrate von 25 % nützlich für die Fehlererkennung sind. Darüber hinaus wurde dargelegt, dass diese Prädiktoren als probabilistische Indikatoren und nicht als absolute Indikatoren betrachtet werden sollten.

Gyimothy et al. [1] berechneten die syntaktischen Code-Metriken aus dem Quellcode von Mozilla und nutzten dann auf der Grundlage dieser Metriken zwei statistische Methoden und zwei Techniken des maschinellen Lernens, um die Fehleranfälligkeit jeder Klasse vorherzusagen. Sie untersuchten auch, wie sich die vorhergesagte Fehleranfälligkeit und die Code-Metriken von Mozilla über sieben Versionen hinweg veränderten.

Die Studie von Giger et al. [5] untersuchte die Erstellung von Fehlervorhersagemodellen auf Methodenebene. Die Autoren zeigten, dass ihre Modelle, die auf syntaktischen Codemetriken und Änderungsmetriken auf Methodenebene basieren, zur genauen Vorhersage fehleranfälliger Methoden verwendet werden können. Sie zeigten auch, dass Änderungsmetriken besser abschneiden als

Quellcodemetriken zur Vorhersage von Fehlern und zeigten, dass ihre Modelle in Bezug auf unterschiedliche Verteilungen von Stichproben robust sind.

He et al. [10] sammelten eine Liste von zwanzig syntaktischen Codemetriken als Fehlerprognosemerkmale und wandten dann die Merkmalsauswahl an, um Fehlerprognosemodelle zu erstellen. Jing et al. [6] wählten zwanzig Code-Metriken aus Software-Projekten aus und lernten mehrere Wörterbücher und Sparse Representation Coefficients zur Vorhersage von Software-Fehlern.

Selby und Basili [20] haben den Zusammenhang zwischen der Abhängigkeitsstruktur und Softwarefehlern untersucht. Ostrand et al. [21] entwickelten ein negatives binomiales Regressionsmodell unter Verwendung von Informationen über die Größe und Änderung von Dateien und zeigten, dass das Modell die erwartete Anzahl von Fehlern in jeder Datei im nächsten Release eines Softwareprojekts effektiv vorhersagen kann. Lessamann et. al. [22] verwendeten 39 Code-Metriken und wendeten 22 Klassifikatoren an, um die Fehlervorhersage zu verbessern.

Wie in den oben genannten Studien gezeigt wurde, wurden zahlreiche Modelle zur Vorhersage von Fehlern unter Verwendung syntaktischer Metriken entwickelt. Unsere Studie leistet einen ergänzenden Beitrag zu diesem Bereich, indem sie eine neue Reihe semantischer Metriken vorschlägt und zeigt, wie diese Metriken für die Fehlervorhersage verwendet werden können.

B. Semantik bei der Fehlervorhersage

Auf der Grundlage der Analyse der textlichen Ähnlichkeit zwischen Fehlerberichten und Quelltexten haben Forscher auch verschiedene Ansätze zur Vorhersage fehleranfälliger Texte vorgeschlagen. Zhou et al. [23] schlugen zum Beispiel ein überarbeitetes Vektorraummodell (rVSM) vor, um die textuelle Ähnlichkeit zwischen Fehlerberichten und Quelltexten zu analysieren. In ihren Experimenten verwendeten die Autoren die gewichtete Summe der beiden Rankings, um die zugehörigen Quelltexte für einen Fehler zu finden. Li et al. [24] stellten das auf Deep Learning basierende Modell DeepFL vor. Es nutzte die textuelle Ähnlichkeit zwischen Quellcode-Methoden und fehlgeschlagenen Testinformationen, Code-Metriken und anderen Historische Maßnahmen als Attribute zur Vorhersage von Fehlern.

Ye et al. [25] schlugen ein LR+WE-Modell zur Vorhersage von Fehlern vor. Dieses Modell stellte Fehlerberichte und Quelltexte als Worthülsen dar und extrahierte die textuelle Ähnlichkeit zwischen Fehlerberichten und Texten, um sie dann als Merkmal für die Fehlervorhersage zu verwenden.

Xiao et al. [26] stellten das Modell DeepLoc vor, das ein erweitertes Faltungsneuronales Netzwerk (CNN) verwendet, um ein Vorhersagemodell zu erstellen, bei dem die Fehlerhäufigkeit, die Fehlerhäufigkeit und die textliche Ähnlichkeit zwischen Fehlerberichten und Fehlern als Merkmale berücksichtigt wurden.

Im Gegensatz zu den oben genannten Studien, die hauptsächlich die textuelle Semantik zwischen Fehlerberichten und Quelltexten zur Vorhersage von Fehlern nutzten, konzentrierte sich unsere Arbeit auf die semantische Analyse von Quelltexten, d. h. auf die semantische Ähnlichkeit zwischen Quelltexten.

Es gibt zwei frühere Studien [7], [27], die unserer Arbeit ähneln. Beide lernten automatisch semantische Metriken aus dem Quellcode, um Software-Fehler vorherzusagen. Wang et al. [7] schlugen einen DBN-basierten (Deep Belief Network) Ansatz zum automatischen Erlernen semantischer Merkmale vor. Auf der Grundlage dieser Merkmale wendeten die Autoren ADTree, NB und LR Classifiers an, um fehleranfällige Stellen vorherzusagen. Huo et al. [27] schlugen ein CAP-CNN (Convo- lutional Neural Network for Comments Augmented Programs) vor.

Modell, das auch automatisch semantische Merkmale aus dem Quellcode lernte und diese Merkmale dann direkt zur Vorhersage fehleranfälliger Stellen verwendete.

In unserer Arbeit wird jedoch eine Reihe von semantischen Metriken vorgeschlagen. Für jede semantische Metrik liefern wir eine formale Definition und eine detaillierte Beschreibung. Darüber hinaus stellen wir vor, wie solche Metriken zur Erstellung von Fehlervorhersagemodellen verwendet werden können. In den folgenden Abschnitten führen wir ausführliche Vergleiche zwischen unseren Ansätzen und diesen beiden bestehenden Modellen durch.

IV. METHODIK

Abbildung 1 zeigt einen Überblick über unseren Ansatz zur Extraktion semantischer Metriken und zur Durchführung von Fehlervorhersagen. 1) Ausgehend von den Quelltexten eines Projekts extrahieren wir zunächst Lexikoninformationen, um ein Textdokument für jeden Text zu erstellen. 2) Anhand dieser erstellten Dokumente berechnen wir dann die semantischen Ähnlichkeiten zwischen den Dokumenten, indem wir LSA mit dem TD-IDF-Maß implementieren; 3) Auf der Grundlage der semantischen Ähnlichkeiten erstellen wir eine semantische Ähnlichkeitsmatrix und einen semantischen Abhängigkeitsgraphen (SDG), aus dem wir die SDG-basierten semantischen Metriken für jede Quelle berechnen; 4) Schließlich wenden wir Algorithmen des maschinellen Lernens an, um Modelle zur Fehlerprognose zu erstellen. Wir haben die Toolkette für die Generierung von SDG-basierten semantischen Metriken in unserem GitHub-Repository [1]bereitgestellt. Als Nächstes stellen wir die Details der einzelnen Hauptverfahren innerhalb unseres Ansatzes vor.

A. Quellcode Parsen

Ausgehend vom Quellcode eines Projekts nutzen wir ein Reverse-Engineering-Tool, Understand[2] , um die Quelltexte zu analysieren. Jedes Quelltextfragment wird zunächst als abstrakter Syntaxbaum (AST) dargestellt, aus dem der Lexikonparser von Understand die lexikalischen Informationen (Identifikatoren, Kommentare usw.) generiert. Wenn wir zum Beispiel ein Codefragment wie dieses haben:

int len=10;//Länge

extrahiert der Lexikon-Parser eine Reihe von lexikalischen Token, wie in Tabelle I dargestellt.

TABELLE I: Ein Beispiel für extrahierte lexikalische Token

Text	Token	Text	Token	Text	Token
int	Schlüsselwort	10	Whitespace	a	Identifier
=	Kommentar des Betreibers		Wörtlich	;	Zeichensetzung
Länge					

B. LSA-Implementierung

Anhand der abgeleiteten lexikalischen Informationen können wir eine Sammlung von Dokumenten erstellen, $D = D_1, D_2, \ldots D_n$, wobei n die Anzahl der files ist, und jedes Dokument abgeleitet wird

für eine bestimmte Datei. Im Gegensatz zu einer Quelldatei, die Programme enthält, besteht das zugehörige Dokument aus einer Reihe von Textbegriffen, die aus den zugehörigen lexikalischen Informationen abgeleitet werden, darunter Schlüsselwörter, Identifikatoren, Kommentare und Operatoren. Bei einer Sammlung von Dokumenten implementieren wir dann LSA, um die semantische Ähnlichkeit zwischen den Dokumenten zu erfassen. Wie wir bereits in

1https://github.com/SDGSemantic/SDG Semantikcode

2https://scitools.com/

In Abschnitt II wird mit Hilfe von TD-IDF eine Term-Dokument-Matrix erstellt. Auf der Grundlage der Zersetzungsmatrix nach Anwendung der SVD wird jedes Dokument durch einen Vektor dargestellt, indem die entsprechende Spalte der Zersetzungsmatrix verwendet wird.

C. Erstellung einer semantischen Ähnlichkeitsmatrix

Gegeben eine Menge von Dokumenten $D = D_1, D_2, \ldots D_n$, und ihre zugehörigen Vektoren $V = V_1, V_2, \ldots, V_n$. Die semantische Ähnlichkeit zwischen zwei beliebigen Dokumenten kann berechnet werden, indem man die Kosinus zwischen ihren entsprechenden Vektoren. Diese Kosinus-Ähnlichkeit ist ein Maß, das die Ähnlichkeit zweier Dokumente ohne Berücksichtigung ihrer Größe angibt. Es berechnet den Kosinus des Winkels zwischen zwei Vektoren in einem mehrdimensionalen Raum. Ein kleinerer Winkel hat einen höheren Kosinuswert, was bedeutet, dass zwei Dokumente ähnlicher sind. In diesem Schritt wird eine Matrix erstellt, die die semantische Ähnlichkeit zwischen zwei beliebigen Dokumenten darstellt. Die Ähnlichkeitsmatrix ist eine quadratische Matrix (N N Matrix), die aus einer Menge von Quelltexten besteht. Jede Zelle in dieser Matrix, $cell(f_i, f_j)$, zeigt die semantische Ähnlichkeit zwischen den files i und j, d.h. die semantische Ähnlichkeit der Dokumente D_i und D_j.

D. Generierung semantischer Abhängigkeitsgraphen (SDG)

Auf der Grundlage der semantischen Ähnlichkeitsmatrix übertragen wir sie in einen semantischen Abhängigkeitsgraphen, SDG. $SDG =< V, E >$, wobei V die Menge der Quell-file und E die Menge der Kanten ist. Jede der Kanten, $e(f_i, f_j)$, zeigt an, dass die Quelle i semantisch von der Quelle j abhängt. Das Gewicht jeder Kante ist der Wert der entsprechenden semantischen Ähnlichkeit. In diesem Fall gehen wir davon aus, dass es eine Kante von A nach B gibt (d. h. A hängt semantisch von B ab), wenn die Ähnlichkeitsgewichtung zwischen A und B folgende Bedingungen erfüllt: $w(A, B) > = C$, wobei C ein Schwellenwert ist, der die Bedeutung der semantischen Ähnlichkeit angibt. Wenn ein Wert größer als der Schwellenwert ist, gehen wir davon aus, dass eine semantische Abhängigkeit besteht. In Anlehnung an das bekannte Pareto-Prinzip [28] setzen wir C auf 0,8, d. h., wenn $w(A, B) > = 0{,}8$ ist, betrachten wir die semantische Abhängigkeit von A zu B als signifikant. Um diese Einstellung zu untermauern, haben wir hundert zufällige Paare von Wörtern ausgewählt und ihre Lexikoninformationen manuell untersucht. Wir haben festgestellt, dass wir, wenn der Ähnlichkeitswert von zwei Texten größer als 0,8 ist, explizit ähnliche oder sogar identische Schlüsselwörter, Identifikatoren oder Kommentare identifizieren können.

E. SDG-basierte semantische Metriken Definition

Ausgehend von der semantischen Ähnlichkeitsmatrix und dem semantischen Abhängigkeitsgraphen, $SDG =< V, E >$, definieren wir formal elf Metriken, die die semantischen Eigenschaften jeder Rolle wie folgt darstellen:

Abhängige Anzahl (DCT): DCT_{f_i} wird berechnet als die Gesamtzahl der Elemente, die semantisch von dem Element abhängen. Ein größerer Wert von *DCT* deutet oft auf eine wesentlichere und kompliziertere Datei hin.

Sum of Dependent Weight (SDW): File i's SDW is calculated as the sum of the semantic weight of the edges formed by file i and the files that semantically depend on it. $SDW_{f_i} = w(f_i, f_j)$, where $i \nsim j$, and

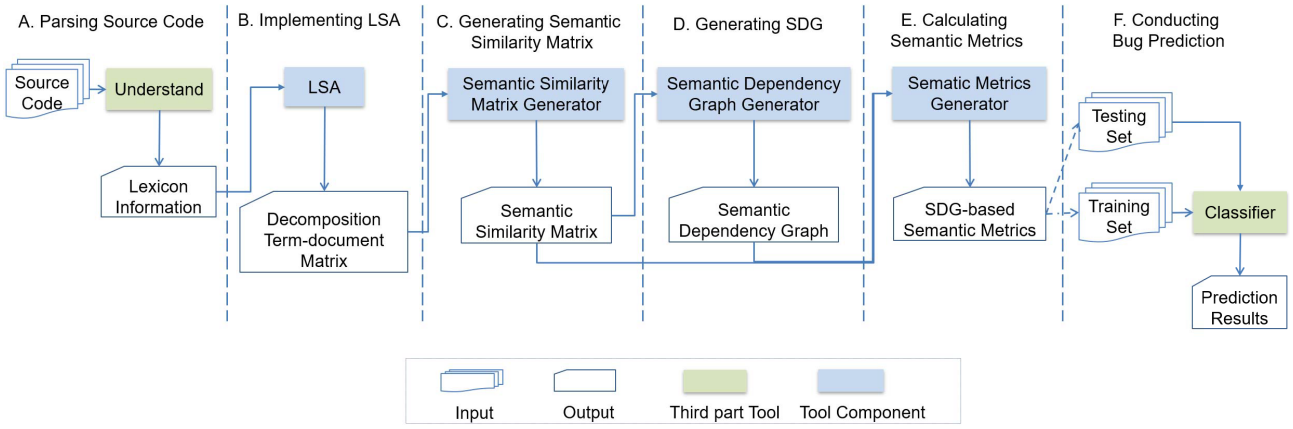

Abb. 1: Überblick über unsere SDG-basierte semantische Metrik-Extraktion und Fehlervorhersage

$j = [1, 2, 3, ..., m]$, m ist die Gesamtzahl der files, die semantisch von der $Datei_i$ abhängen.

Sum of Semantic Similarity (SSS): File i's SSS is calculated as the sum of the semantic similarities between file i and other files in the same project. $SSS_{f_i} = w(f_i, f_k)$, where $i \neq k$, and $k = [1, 2, 3, ..., n]$, n ist die Gesamtzahl der F l e e s in einem Projekt. Im Gegensatz zu *SDW*, das die Gewichtung jeder semantischen Abhängigkeit von fi l e i gemäß einer SDG summiert, SSS berücksichtigt alle semantischen Ähnlichkeiten in Bezug auf die R o l l e i zu berücksichtigen.

Durchschnitt der Abhängigkeitsgewichtung (ADW): Datei i's ADW stellt den Durchschnitt der semantischen Gewichtung der Kanten, die durch das B l a t t i und die B l ä t t e r gebildet werden, die semantisch davon abhängen: $ADW_{f_i} = \frac{SDW_{f_i}}{DC_{f_i}}$.

Durchschnitt der semantischen Ähnlichkeit (AS): Die AS von Datei i ist der Durchschnitt der semantischen Ähnlichkeiten zwischen D a t e i i und anderen D a t e i e n desselben Projekts: $AS_{f_i} = \frac{SSS_{f_i}}{N}$, wobei N die Anzahl der files ist, die semantisch ähnlich sind, d. h. der semantische Ähnlichkeitswert ist größer als 0.

Längster Abhängigkeitspfad (Longest Dependency Path, LDP): LDP_{f_i} wird berechnet als die Anzahl der fi l e s, die am längsten Pfad des fi l e i im semantischen Abhängigkeitsgraphen beteiligt sind. Ein größerer LDP bedeutet, dass mehr files transitiv verbunden sind, was die Komplexität erhöhen kann, da Änderungen an einer file auf andere files auf demselben Pfad übertragen werden können.

Die oben genannten sechs semantischen Metriken werden auf der Grundlage der direkten Abhängigkeiten zwischen den Wörtern berechnet. Als Nächstes definieren wir eine Reihe von semantischen Metriken, indem wir die Transitivität der Abhängigkeiten berücksichtigen. Ausgehend von der Rolle i wird der BFS-Algorithmus angewendet, um den semantischen Abhängigkeitsgraphen zu durchlaufen. Auf diese Weise können wir eine Gruppe von Flees erhalten, die direkt oder indirekt von der R o l l e i abhängt. Wir haben eine solche Rollengruppe als *Impact Space* bezeichnet. Für jedes file würde ein bestimmter Wirkungsraum erzeugt. Aus dem Impact Space eines Flees berechnen wir die anderen fünf semantischen Metriken wie folgt:

Impact Space File Count (ISF): ISF_{f_i} ist die Gesamtzahl der D a t e i e n, die am Impact Space von D a t e i i beteiligt sind. Diese misst die transitive Wirkung einer R o l l e auf das gesamte Projekt in Bezug auf die Semantik.

Summe der Dateigewichte (SFW): SFW_{f_i} ist die Summe aus Ähnlichkeitsgewichte zwischen fi l e i und die anderen fi l e s in seinem Wirkungsbereich.

Durchschnittliches Gewicht der Dateien (AFW): AFW_{f_i} ist der Durchschnitt der Ähnlichkeitsgewichte zwischen file i und die anderen files in seinem Wirkungsraum: $AFW_{f_i} = \frac{SFW_{f_i}}{ISF_{f_i}}$.

Impact Space Gewicht (ISW): Jeder Impact Space kann einen Untergraphen des semantischen Abhängigkeitsgraphen G bilden. ISW_{f_i} ist die Summe der Gewichte aller Kanten im Untergraphen (d. h. dem Impact Space).

Durchschnittliches Gewicht des Wirkungsraums (AWIS): $AWIS_{f_i}$ ist das durchschnittliche Gewicht des Auswirkungsraumes von file i: $AWIS_{f_i} = {}^{ISWfi}$, wobei n die Anzahl der Kanten im Wirkungsraum ist
Raum.

Ausgehend vom Quellcode eines Projekts kann unser Tool automatisch die entsprechende semantische Ähnlichkeitsmatrix und den semantischen Abhängigkeitsgraphen generieren und dann die semantischen Metriken für jede Quelle extrahieren und berechnen.

F. Entwicklung eines Fehlervorhersagemodells

Wir erstellen Modelle zur Vorhersage von fehleranfälligen Dateien, indem wir drei überwachte Algorithmen des maschinellen Lernens anwenden: Alternierender Entscheidungsbaum, Naive Bayes und logistische Regression, die in verschiedenen Modellen zur Vorhersage von Fehlern weit verbreitet sind. Im Folgenden werden alle in unseren Experimenten verwendeten maschinellen Lernverfahren kurz beschrieben.

- Der ***alternierende Entscheidungsbaum (ADTree)*** ist ein Klassifizierer, der aus Entscheidungsknoten und Vorhersageknoten besteht. Jeder Entscheidungsknoten enthält eine Prädikatsbedingung, und jeder Vorhersageknoten hat eine einzelne Zahl. Um eine Instanz zu klassifizieren, berücksichtigt der ADTree, dass alle Entscheidungsknoten wahr sind und summiert alle Vorhersageknoten, die durchlaufen wurden.
- ***Naive Bayes (NB)*** ist eines der einfachen probabilistischen Klassifizierungsverfahren. Der Naive Bayes-Algorithmus nutzt die Bayes-Theo- rem, um die bedingte Wahrscheinlichkeit aller Klassen aus den Trainingsdaten zu berechnen, und geht davon aus, dass die Attribute unabhängig sind. Naive Bayes hat sich in verschiedenen Studien als ein sehr effektiver Klassifizierungsalgorithmus erwiesen.
- Die ***logistische Regression (LR)*** ist eine Erweiterung des linearen Regressionsmodells. Die logistische Regression überträgt ihre Ausgabe in einen Wahrscheinlichkeitswert auf der Grundlage der logistischen Sigmoidfunktion. Auf der Grundlage der Wahrscheinlichkeitswerte kann die logistische Regression die Wahrscheinlichkeiten für Klassifizierungsprobleme modellieren, indem zwei mögliche Ergebnisse zugewiesen werden.

Für jede Vorhersagestudie erstellen wir zunächst die SDG-basierten semantischen Metriken und verwenden die Trainingsmenge, um die Vorhersagemodelle zu trainieren, und wenden dann die trainierten Modelle auf die Testmenge an, um ihre Leistung bei der Fehlervorhersage zu validieren.

V. BEWERTUNG

A. Forschungsfragen

In unserer empirischen Studie untersuchen wir die folgenden Forschungsfragen:

RQ1: *Ist es möglich, auf der Grundlage der von uns vorgeschlagenen, auf den SDGs basierenden semantischen Metriken wirksame Fehlervorhersagemodelle zu erstellen?* In dieser Frage untersuchen wir die Leistung der entwickelten Vorhersagemodelle unter Verwendung unserer vorgeschlagenen semantischen Metriken.

RQ2: *Übertreffen die SDG-basierten semantischen Metriken die traditionellen syntaktischen Metriken bei der Fehlervorhersage?* Syntaktische Metriken sind in der Fehlervorhersage weit verbreitet. In dieser Frage versuchen wir zu verstehen, ob unsere SDG-basierten semantischen Metriken bessere Prädiktoren sind als die traditionellen syntaktischen Metriken zur Erstellung von Fehlervorhersagemodellen.

RQ3: *Haben die SDG-basierten semantischen Metriken eine bessere Leistung bei der Vorhersage von Fehlern im Vergleich zum Stand der Technik?* Auf der Grundlage von zwei State-of-the-Art-Studien, die automatisch semantische Merkmale aus dem Quellcode lernen, untersuchen wir weiter, ob unsere SDG-basierte Metrik verwendet werden kann, um bessere Modelle zur Fehlervorhersage zu erstellen.

RQ4: *Wie ist die Leistung unseres Ansatzes in Bezug auf Ausführungszeit und Platzbedarf?* Schließlich stellen wir dar, ob unser Ansatz in der Praxis anwendbar ist, indem wir seine Ausführungszeit und seinen Platzbedarf untersuchen.

B. Themen

In dieser Arbeit konstruieren wir unsere Themen aus den in [29] [3]aufgelisteten PROMISE-Projekten, die ein offener Datensatz sind und über

3https://github.com/opensciences/opensciences.github.io/tree/master/repo/defect

wurde in vielen früheren Studien zur Vorhersage von Fehlern verwendet [10], [30], [7], [27]. Um einen fairen und konsistenten Vergleich zu gewährleisten, wählen wir fünf Projekte aus, die in früheren Studien verwendet wurden und mit denen wir vergleichen, darunter: Log4j[4]- ein Java-basiertes Logging-Dienstprogramm; Lucene [5]- eine Softwarebibliothek für Suchmaschinen; Synapse [6]- ein leichtgewichtiger Enterprise Service Bus; Xalan [7]- eine Softwarebibliothek zur Transformation von XML; und Xerces [8]- eine Bibliothek zur Bearbeitung von XML-Dokumenten.

Tabelle II zeigt die grundlegenden Fakten unserer Vorhersagedatensätze. Für jedes Experiment verwenden wir eine frühere Version als Trainingsdatensatz und die spätere Version als Testdatensatz. Die Spalte "# files" gibt die Anzahl der files in einem Datensatz an. Am Beispiel der ersten Zeile wird das Experiment mit der Version 1.0 des Projekts Log4j durchgeführt, um das Vorhersagemodell zu trainieren, und mit der Version 1.1, um es zu testen. Version 1.0 enthält 119 Quelltexte, und Version 1.1 enthält 104 Quelltexte.

In dieser Arbeit geht es darum, Modelle zur Vorhersage von Fehlern zu trainieren und zu validieren, um Dateien als fehleranfällig oder nicht fehleranfällig zu klassifizieren. Da die überwachten Algorithmen für die Erstellung von Vorhersagemodellen verwendet werden, müssen wir zunächst jede Quelle im Datensatz als *fehleranfällig* oder *nicht fehleranfällig* kennzeichnen. In Anlehnung an die früheren Studien von [29], [7] wird eine Datei als *fehleranfällig* identifiziert, wenn sie in der Revisionshistorie der Testversion mindestens einmal wegen Fehlern geändert wurde. Andernfalls wird diese Datei als *nicht fehleranfällig eingestuft*. Um zu untersuchen, ob eine Datei wegen eines Fehlers geändert wurde, müssen wir die Revisionshistorie und die Fehlerberichte eines Projekts durchsuchen, indem wir die IDs der Fehlertickets in den Commits abgleichen. Weitere Einzelheiten über die Erstellung des Datensatzes finden sich in [29].

TABELLE II: Datensatz der einzelnen experimentellen Studien

Projekt	Trainingssatz		Testsatz	
	Freigabe	#Akten	Freigabe	#Akten
Log4j	1.0	119	1.1	104
Lucene	2.0	186	2.2	238
Lucene	2.2	238	2.4	334
Synapse	1.0	157	1.1	222
Synapse	1.1	222	1.2	256
Xalan	2.4	676	2.5	762
Xerces	1.2	439	1.3	452

VI. ERGEBNISSE

RQ1: Ist es möglich, auf der Grundlage der von uns vorgeschlagenen SDG-basierten semantischen Metriken wirksame Fehlerprognosemodelle zu entwickeln? Um diese Frage zu beantworten, nutzen wir zunächst drei weit verbreitete Algorithmen des maschinellen Lernens, ADTree, Naive Bayes und logische Regression, um entsprechende Vorhersagemodelle zu entwickeln. Dann wenden wir die abgeleiteten Modelle auf sieben Sätze von experimentellen Studien an und untersuchen ihre Leistung. Für jedes Experiment enthält der Datensatz zwei aufeinanderfolgende Versionen eines Projekts, wobei die vorherige Version als Trainingsset und die zweite als Trainingsset verwendet wird.

4http://logging.apache.org/ 5https://lucene.apache.org/
6https://synapse.apache.org/ 7https://xalan.apache.org/
8https://xerces.apache.org/

letztere Version wird als Testsatz festgelegt. In dieser Arbeit verwenden wir das *F-Maß*, um die Leistung der einzelnen Vorhersagemodelle zu bewerten. Das *F-Maß wurde* in früheren Studien zur Fehlervorhersage häufig verwendet [10], [31], [7], [27]. Es berücksichtigt sowohl die Präzision als auch den Rückruf, um zu beurteilen, wie genau und wie robust eine Klassifizierung ist [32], [33].

Wir haben die experimentellen Ergebnisse in Tabelle III zusammengefasst. Am Beispiel von *Log4j 1.0 1.1* können wir feststellen, dass bei Verwendung unserer SDG-basierten semantischen Metriken die auf ADTree-, NB- und LR-Klassifikatoren basierenden Vorhersagemodelle einen F-Maß-Wert von 0,62, 0,56 bzw. 0,65 erreichen konnten. Gemäß früheren Studien [34], [35] deutet ein F-Maß-Wert größer oder gleich 0,5 auf eine vielversprechende Vorhersage hin. Dies bedeutet, dass alle drei abgeleiteten Vorhersagemodelle unter Verwendung der SDG-basierten semantischen Metriken effektiv fehleranfällige `Fehler` vorhersagen können.

Wenn wir alle Experimente zusammen betrachten, können wir feststellen, dass die meisten Modelle, die auf den SDG-basierten semantischen Metriken basieren, eine vielversprechende Vorhersageleistung erzielen: 90,5 % (19/21) der Vorhersagen erhalten einen *F-Wert* von über 0,5. Das bedeutet, dass die Verwendung unserer semantischen Metriken zur Erstellung eines Fehlervorhersagemodells vielversprechend ist. Wie in der letzten Zeile gezeigt, liegen die durchschnittlichen F-Maß-Werte bei 0,593, 0,633 und 0,731 im Vergleich zu den Modellen, die auf ADTree, NB und LR basieren. Daher glauben wir, dass das von unseren SDG-basierten semantischen Metriken abgeleitete Vorhersagemodell effektiv fehleranfällige `Artikel` vorhersagen kann.

TABELLE III: Leistung der einzelnen Vorhersagemodelle

Projekt	Datensätze	DT	NB	LR
Log4j	1.0→1.1	0.620	0.560	0.650
Lucene	2.0→2.2	0.582	0.803	0.847
Lucene	2.2→2.4	0.545	0.743	0.756
Synapse	1.0→1.1	0.616	0.551	0.616
Synapse	1.1→1.2	0.583	0.546	0.530
Xalan	2.4→2.5	0.444	0.497	0.938
Xerces	1.2→1.3	0.761	0.731	0.777
Durchschnitt		0.593	0.633	0.731

Antwort auf Frage 1: Mit Hilfe der von uns vorgeschlagenen semantischen Metriken konnten wir vielversprechende Modelle zur Vorhersage von Fehlern erstellen, indem wir Entscheidungsbaum-, Naive-Bayes- und logistische Regressionsklassifizierer einsetzten.

RQ2: Übertreffen die SDG-basierten semantischen Metriken die syntaktischen Metriken bei der Fehlervorhersage?

Syntaktische Metriken werden häufig für die Vorhersage von Fehlern verwendet. Um die Nützlichkeit unserer SDG-basierten semantischen Metriken für die Fehlervorhersage weiter zu validieren, vergleichen wir die SDG-basierten Metriken mit traditionellen syntaktischen Metriken. In diesem Fall nutzen wir direkt die zwanzig traditionellen syntaktischen Metriken, die in [10] für diesen Vergleich aufgelistet sind.

Tabelle IV zeigt die Vergleichsergebnisse von 21 Experimenten zwischen Vorhersagemodellen, die die SDG-basierten semantischen Metriken und die ausgewählten syntaktischen Metriken verwenden. Der höhere Wert des F-Maßes eines jeden Vergleichs ist hervorgehoben.

Am Beispiel von *Lucene 2.2 2.4*, wobei die Version 2.2 als Trainingsmenge und die Version 2.4 als Testmenge verwendet wird, können wir Folgendes feststellen: 1) Die F-Maß-Werte der SDG-basierten Fehlerprognosemodelle liegen bei 0,582, 0,803 und 0,847 in Bezug auf ADTree-, NB- und LR-Klassifikatoren; 2) Die F-Maß-Werte der syntaktischen Fehlerprognosemodelle liegen bei 0,502, 0,500 bzw. 0,598. Dies deutet darauf hin, dass unsere SDG-basierten semantischen Metriken die untersuchten syntaktischen Metriken in diesem Experiment übertreffen.

Wenn wir alle Experimente zusammen betrachten, können wir die folgende Beobachtung machen: In 71,4 % (15/21) aller Experimente erreichen die Modelle zur Fehlervorhersage, die unsere semantische Fehlervorhersage verwenden, eine bessere Leistung als diejenigen, die die syntaktischen Metriken verwenden. Genauer gesagt, können wir das sehen:

- ***SDG + ADTree vs. Syntaktisch + ADTree.*** In 4 von 7 Vergleichen schneiden die SDG-basierten semantischen Metriken besser ab als die syntaktischen Metriken der Studien. Die Durchschnittswerte in der letzten Zeile zeigen, dass das durchschnittliche F-Maß der Vorhersagemodelle, die SDG-basierte Metriken verwenden, 0,593 beträgt, was 16,7 % höher ist als die Modelle, die syntaktische Metriken verwenden.
- ***SDG + NB vs. Syntaktisch + NB.*** In 5 von 7 Vergleichen schneiden die SDG-basierten semantischen Metriken besser ab als die ausgewählten syntaktischen Metriken. Das durchschnittliche F-Maß der Vorhersagemodelle, die die SDG-basierten Metriken verwenden, liegt bei 0,633, was 31,3 % höher ist als die Modelle, die syntaktische Metriken verwenden.
- ***SDG + LR vs. Syntaktisch + LR.*** In 6 von 7 Vergleichen schneiden die SDG-basierten semantischen Metriken besser ab als die syntaktischen Metriken der Studien. Das durchschnittliche F-Maß der Vorhersagemodelle, die die SDG-basierten Metriken verwenden, liegt bei 0,731 und damit 47,1 % höher als bei den Modellen, die syntaktische Metriken verwenden.

Nach den obigen Beobachtungen glauben wir, dass unsere SDG-basierten semantischen Metriken die ausgewählten syntaktischen Metriken bei der Fehlervorhersage übertreffen.

Antwort auf Frage 2: Die Modelle, die die von uns vorgeschlagenen SDG-basierten semantischen Metriken verwenden, könnten eine bessere Leistung bei der Fehlervorhersage erzielen als die Modelle, die traditionelle syntaktische Metriken verwenden.

Diese syntaktischen Metriken beinhalten Codezeilen, Anzahl der Operanden und Operatoren, Fan-in, Fan-out, C&K Metriken [9] und McCabe's Komplexität [8], etc. Alle diese Metriken wurden in früheren Studien zur Fehlervorhersage [5], [31], [10], [6], [7] gut beschrieben und weithin verwendet.

RQ3: Haben die SDG-basierten semantischen Metriken eine bessere Leistung bei der Fehlervorhersage im Vergleich zum Stand der Technik?

Um die Effektivität unserer SDG-basierten semantischen Metriken bei der Vorhersage von Fehlern weiter zu evaluieren, nutzen wir zwei moderne Ansätze als Basis für die Durchführung von Vergleichen. Die erste Basislinie ist das DBN-basierte (Deep Belief Network) Fehlerprognosemodell, das von Wang et al. [7] vorgeschlagen wurde. Ihre Arbeit verwendet ein DBN-Modell, um automatisch semantische Merkmale aus dem Quellcode zu generieren, und wendet ADTree-, NB- und LR-Klassifikatoren an.

TABELLE IV: Leistung von Modellen mit SDG-basierten Metriken und syntaktischen Metriken

Projekt	Datensätze	SDG-basiert			Syntaktische		
		DT	NB	LR	DT	NB	LR
Log4j	1.0→1.1	0.620	0.560	**0.650**	0.687	0.689	0.535
Lucene	2.0→2.2	**0.582**	**0.803**	**0.847**	0.502	0.500	0.598
Lucene	2.2→2.4	0.545	**0.743**	**0.756**	0.605	0.378	0.694
Synapse	1.0→1.1	**0.616**	**0.551**	**0.616**	0.476	0.508	0.316
Synapse	1.1→1.2	**0.583**	0.546	0.530	0.530	0.565	0.533
Xalan	2.4→2.5	0.444	**0.497**	**0.938**	0.518	0.398	0.540
Xerces	1.2→1.3	**0.761**	**0.731**	**0.777**	0.238	0.333	0.266
Durchschnitt		0.593	0.633	0.731	0.508	0.482	0.497

zur Vorhersage fehleranfälliger Dateien. Die zweite Grundlage ist das von [27] vorgeschlagene CAP-CNN-Modell. Das CAP-CNN-Modell ist ein Deep-Learning-Modell, das automatisch semantische Merkmale aus dem Quellcode generieren kann, um fehleranfällige Stellen vorherzusagen.

Tabelle V zeigt die Vergleichsergebnisse zwischen den SDG-basierten Modellen und den DBN-basierten Modellen, aus denen wir die folgenden Beobachtungen ableiten können:

- ***SDG+ADTree vs. DBN+ADTree.*** In 3 von 7 Vergleichen schneiden die SDG-basierten Modelle bei der Fehlervorhersage besser ab. Am Beispiel von *Log4j 1.0 1.1* beträgt das F-Maß des SDG+DT-Modells 0,62 und das F-Maß von DBN+DT 0,701. Anhand des Durchschnittswerts können wir sehen, dass die durchschnittliche F-Bewertung der SDG-basierten Modelle 0,593 beträgt. Er ist ähnlich wie der Durchschnittswert der DBN-basierten Modelle, der 0,608 beträgt.
- ***SDG+NB vs. DBN+NB.*** In 5 von 7 Vergleichen konnten die SDG-basierten Modelle eine bessere Vorhersageleistung erzielen. Das durchschnittliche F-Maß der SDG-basierten Modelle beträgt 0,633, was 11,2 % höher ist als der Durchschnittswert (0,569) der DBN-basierten Modelle.
- ***SDG+LR vs. DBN+LR.*** Die SDG-basierten Modelle erreichen auch in 5 von 7 Vergleichen eine bessere Vorhersageleistung. Das durchschnittliche F-Maß der SDG-basierten Modelle liegt bei 0,731, was 29,6 % über dem Durchschnittswert (0,564) der DBN-basierten Modelle liegt.

Betrachtet man alle Experimente zusammen, so sind die SDG-basierten Modelle in 61,9% (13 von 21) aller Experimente besser als DBN-basierte Modelle bei der Vorhersage von Fehlern. Für 5 von 7 experimentellen Datensätzen konnten die Modelle, die unsere SDG-basierten semantischen Metriken verwenden, die beste Vorhersageleistung in Bezug auf die F-Maß-Werte erzielen (*das höchste F-Maß wurde in jedem Satz von Experimenten farbig markiert*). Am Beispiel von *Lucene 2.0 2.2* konnte SDG+LR die beste Leistung mit einem F-measure-Wert von 0,847 erzielen. Daher glauben wir, dass unsere SDG-basierten Metriken die von DBN-basierten Modellen gelernten semantischen Merkmale bei der Fehlervorhersage übertreffen.

Tabelle VI enthält die Vergleichsergebnisse zwischen den SDG-basierten Modellen und den CAP-CNN-Modellen. Das CAP-CNN-Modell ist ein Deep-Learning-Modell, das automatisch semantische Merkmale aus dem Quellcode eines Projekts lernen kann. Im Gegensatz zu den anderen Ansätzen, die für die Vorhersage von Fehlern externe Klassifikatoren verwenden müssen, kann das CAP-CNN-Modell die gelernten Merkmale direkt für die Vorhersage von fehleranfälligen Stellen nutzen. Aus Tabelle VI können wir die folgenden Beobachtungen ablesen:

- Im Vergleich zu den CAP-CNN-Modellen konnten beide Arten von SDG+ADTree- und SDG+NB-Modellen nur in 3 von 7 Vergleichsfällen eine bessere Vorhersageleistung erzielen.
- Allerdings konnten die SDG+LR-Modelle in 4 von 7 Vergleichen eine bessere Vorhersageleistung erzielen als die CAP-CNN-Modelle.

Wenn man alle Experimente berücksichtigt, sind die SDG-basierten Modelle den CAP-CNN-Modellen in nur 10 von 21 Experimenten überlegen. Allerdings konnten die Modelle, die unsere SDG-basierten semantischen Metriken verwenden, in 5 von 7 Experimenten eine bessere Vorhersageleistung erzielen als die CAP-CNN-Modelle (*das höchste F-Maß wurde in jedem Experimentsatz farbig markiert*). In den Experimenten mit *Log4i 2.0 2.2 und Lucene 2.2 2.4*, konnte CAP-CNN die beste Leistung erzielen. Aber die besten Leistungen in den anderen 5 Versuchsreihen stammen von den SDG-basierten Modellen.

Basierend auf den obigen Ergebnissen glauben wir, dass unsere SDG-basierten semantischen Metriken die semantischen Merkmale übertreffen, die von den modernsten Modellen zur Fehlervorhersage gelernt werden.

RQ4: Wie leistungsfähig ist unser Ansatz in Bezug auf die Ausführungszeit und den Platzbedarf?

Um diese Frage zu beantworten, untersuchen wir die Zeit- und Speicherplatzkosten für den Generierungsprozess der SDG-basierten semantischen Metriken. Wie in Abschnitt IV eingeführt, besteht der gesamte Prozess aus der LSA-Implementierung, der Generierung der semantischen Ähnlichkeitsmatrix und der SDG sowie der Extraktion der semantischen Metriken. Die Zeitkomplexität der SDG-Generierung und der semantischen Metriken Extraktionen sind beide $O(n^2)$, wobei n die Anzahl der files in ein Projekt. Für jedes Projekt führen wir jedes Experiment wiederholt durch 10 Mal und berechneten die durchschnittlichen Zeit- und Raumkosten. In jedem Fall müssen wir die semantischen Metriken sowohl für die Trainingsmenge als auch für die Testmenge berechnen. Daher nehmen wir die Summe der Zeitkosten in zwei Schritten als Ausführungszeit und verwenden die maximalen Raumkosten in zwei Schritten als Ausführungsraum. Unser laufender Rechner ist ein Laptop mit i5-8265U CPU@1.6GHz und 8 GB RAM.

Tabelle VII zeigt den Zeitaufwand und den verwendeten Speicherplatz

Antwort auf Frage 3: Im Vergleich zum Stand der Technik schneiden die Modelle, die unsere SDG-basierten semantischen Metriken verwenden, bei der Fehlervorhersage für die meisten experimentellen Datensätze besser ab.

TABELLE V: Leistung von SDG-basierten Modellen und DBN-basierten Modellen

Projekt	Datensätze	SDG-basiert			DBN-basiert		
		DT	NB	LR	DT	NB	LR
Log4j	1.0→1.1	0.620	0.560	0.650	0.701	0.725	0.682
Lucene	2.0→2.2	0.582	**0.803**	**0.847**	0.651	0.632	0.630
Lucene	2.2→2.4	0.545	**0.743**	**0.756**	0.773	0.738	0.629
Synapse	1.0→1.1	**0.616**	**0.551**	**0.616**	0.544	0.479	0.423
Synapse	1.1→1.2	**0.583**	0.546	0.530	0.583	0.579	0.541
Xalan	2.4→2.5	0.444	**0.497**	**0.938**	0.595	0.452	0.565
Xerces	1.2→1.3	**0.761**	**0.731**	**0.777**	0.411	0.380	0.475
Durchschnitt		0.593	0.633	0.731	0.608	0.569	0.564

TABELLE VI: Leistung von SDG-basierten Modellen und CAP-CNN-Modelle

Projekt	Datensätze	SDG-basiert			CAP-CNN
		DT	NB	LR	
Log4j	1.0→1.1	0.620	0.560	0.650	0.754
Lucene	2.0→2.2	0.582	0.803	0.847	0.743
Lucene	2.2→2.4	0.545	0.743	0.756	0.771
Synapse	1.0→1.1	0.616	0.551	0.616	0.577
Synapse	1.1→1.2	0.583	0.546	0.530	0.555
Xalan	2.4→2.5	0.444	0.497	0.938	0.631
Xerces	1.2→1.3	0.761	0.731	0.777	0.609

bei der Generierung semantischer Metriken. Am Beispiel von *Log4j 1.0 1.1* lässt sich feststellen, dass die Generierung semantischer Metriken für alle Niveaus im Durchschnitt 2,9 Minuten und 111,1 MB Speicherplatz erfordert. Wenn wir alle untersuchten Projekte zusammen betrachten, können wir zusammenfassen, dass die Zeitkosten für die automatische Generierung semantischer Metriken zwischen 2,9 Minuten (*Log4j 1.0 1.1*) und 111,6 Minuten (*Xalan 2.4 2.5*) liegen. Die Kosten für den Speicherplatz reichen von 106,3 MB (*Synapse 1.0 1.1*) bis 146,0 MB (*Xalan 2.4–2.5*) wurde für jede der empirischen Studien verwendet. Angesichts dieser Ergebnisse sind wir der Meinung, dass der Zeit- und Platzbedarf unseres Ansatzes erschwinglich ist und unser Ansatz in der Praxis angewendet werden kann.

TABELLE VII: Ausführungszeit und Speicherplatz für die Generierung SDG-basierter semantischer Metriken

Projekt	Datensätze	Zeit(min)	Platz (MB)
Log4j	1.0→1.1	2.9	111.1
Lucene	2.0→2.2	9.0	116.6
Lucene	2.2→2.4	16.8	118.9
Synapse	1.0→1.1	13.5	106.3
Synapse	1.1→1.2	18.5	110.5
Xalan	2.4→2.5	111.6	146.0
Xerces	1.2→1.3	59.4	126.0

Antwort auf RQ4: In Bezug auf die Ausführungszeit und den Platzbedarf ist unser Ansatz zur Erzeugung von SDG-basierten semantischen Metriken in der Praxis anwendbar.

A. Begrenzung

Erstens haben wir bisher nur elf semantische Metriken definiert, was im Vergleich zur Menge der vorhandenen Metriken nur eine kleine Menge ist.

In diesem Abschnitt gehen wir kurz auf die Grenzen unseres Ansatzes und die Gefahren für die Validität ein.

syntaktische Metriken. Wir können daher nicht behaupten, dass diese elf Metriken die semantischen Merkmale eines Quelltextes vollständig abbilden können. Wir haben jedoch sowohl die direkten Abhängigkeiten zwischen den Flees als auch die Transitivität der semantischen Abhängigkeit berücksichtigt, und die Vorhersageergebnisse haben gezeigt, dass die elf semantischen Metriken zumindest geeignet sind, fehleranfällige Flees vorherzusagen. Darüber hinaus ist unser Ansatz skalierbar. Wann immer wir neue Metriken identifizieren, ist es einfach, unseren Ansatz zu erweitern, um die neuen Metriken für Experimente einzubeziehen.

Zweitens: Da die ursprünglichen Implementierungen der verglichenen Ansätze [7], [27] nicht veröffentlicht sind, übernehmen wir die Vorhersageergebnisse aus den verglichenen Studien direkt in unsere Vergleiche, ohne ihre Ansätze neu zu implementieren. Dies schränkt die Auswahl unserer experimentellen Datensätze ein. Um einen fairen Vergleich zu gewährleisten, wählen wir in dieser Arbeit nur den Datensatz aus, der in der Basisstudie verwendet wird. Da wir die Datensätze der Projekte Ivy und Poi nicht finden konnten und die Datensätze des Projekts Camel nicht übereinstimmen, analysieren wir nur fünf Open-Source-Projekte mit sieben Versuchsreihen. Wir geben zu, dass dies sowohl eine Einschränkung unserer Studie als auch eine Gefahr für die externe Validität darstellt.

B. Bedrohung der Gültigkeit

Erstens ist die Konstruktvalidität durch die von uns verwendeten Leistungsindikatoren des Modells gefährdet. Die Leistungsindikatoren könnten die Schlussfolgerung der Fehlervorhersage beeinflussen. In dieser Arbeit verwenden wir das F-Maß, um die Leistung der Fehlervorhersage anzuzeigen, da es in vielen früheren Studien [31], [10], [7], [27] verwendet wurde.

Zweitens haben wir ein Code-Analyse-Tool, Understand, verwendet, um die Lexikoninformationen jeder Quelldatei zu erfassen. Folglich könnte sich jede Ungenauigkeit des Tools negativ auf unsere Analyse auswirken. Dies stellt eine Gefahr für die interne Validität dar. Unser Ansatz ist jedoch nicht zwangsläufig von Understand abhängig: Jedes Code-Analyse-Tool, das Lexikoninformationen, z. B. Identifier, Kommentare usw., extrahieren und in lesbare Formate exportieren kann, kann für unseren Ansatz verwendet werden.

Drittens liegt eine Gefahr für die externe Validität in unserem Datensatz. Wie im obigen Abschnitt erwähnt, haben wir nur fünf Open-Source-Projekte mit sieben Versuchsreihen analysiert, so dass wir nicht sicherstellen können, dass die Bewertungsergebnisse auf alle Projekte verallgemeinert werden können. Wir planen, unseren Ansatz in Zukunft auf

weitere Open-Source- oder Industrieprojekte mit unterschiedlichen Größen und in verschiedenen Bereichen anzuwenden.

VIII. SCHLUSSFOLGERUNG

In dieser Arbeit haben wir eine Reihe von semantischen Metriken formal definiert und ein Werkzeug entwickelt, mit dem sich die semantischen Metriken automatisch aus Quelltexten generieren lassen. Genauer gesagt haben wir zunächst vorgestellt, wie die LSA zur Erstellung semantischer Informationen aus Quelltexten implementiert wird. Anhand der semantischen Informationen erstellten wir eine semantische Ähnlichkeitsmatrix und schlugen ein neuartiges Modell des Semantic Dependency Graph (SDG) vor, das semantische Abhängigkeiten zwischen Quelltexten explizit darstellt. Auf der Grundlage des SDG konnten wir die definierten semantischen Metriken extrahieren. Schließlich haben wir gezeigt, wie die SDG-basierten semantischen Metriken zur Erstellung von Fehlervorhersagemodellen verwendet werden können.

Wir haben die Effektivität der von uns vorgeschlagenen semantischen Metriken zur Vorhersage von Fehlern durch sieben Experimente validiert. Die Bewertungsergebnisse haben gezeigt, dass: 1) die von uns vorgeschlagenen SDG-basierten semantischen Metriken zur Entwicklung effektiver Fehlerprognosemodelle verwendet werden können; 2) die SDG-basierten semantischen Metriken die traditionellen syntaktischen Metriken bei der Fehlerprognose übertreffen; 3) unser SDG-basierter Ansatz im Vergleich zum Stand der Technik, der se- mantische Merkmale automatisch erlernt, Fehlerprognosemodelle mit besserer Leistung erstellen kann; 4) unser Ansatz in Bezug auf Ausführungszeit und Platzbedarf in der Praxis erschwinglich ist. Zusammenfassend glauben wir, dass unser Ansatz die Lücke zwischen Semantik und Fehlervorhersage überbrückt. Die von uns vorgeschlagenen SDG-basierten semantischen Metriken sind nützlich, um vielversprechende Modelle zur Fehlervorhersage zu erstellen.

DANKSAGUNGEN

Diese Arbeit wird von der National Natural Science Foundation of China unter der Förderungsnummer 62002129, der Hubei Provincial Natural Science Foundation of China unter der Förderungsnummer 2020CFB473 und den Fundamental Re- search Funds for the Central Universities unter der Förderungsnummer CCNU19TD003 unterstützt.

REFERENZEN

[1] T. Gyimothy, R. Ferenc, and I. Siket, "Empirical validation of object-oriented metrics on open source software for fault prediction," *IEEE Transactions on Software Engineering*, vol. 31, no. 10, pp. 897-910, 2005.

[2] N. Nagappan, T. Ball, und A. Zeller, "Mining metrics to predict component failures," in *Proc. 28th International Conference on Software Engineering*, pp. 452-461, 2006.

[3] T. Menzies, J. Greenwald und A. Frank, "Data Mining statischer Code-Attribute zum Erlernen von Fehlerprädiktoren", *IEEE Trans. Softw. Eng.* Bd. 33, Nr. 1, S. 2-13, 2007.

[4] M. Tim, B. Andrew, M. Andrian, Z. Thomas, and C. David, "Local vs. global models for effort estimation and defect prediction," in *Proc. 26thIEEE/ACM International Conference on Automated Software Engineering*, pp. 343-351, 2011.

[5] E. Giger, M. D'Ambros, M. Pinzger, and H. C. Gall, "Method-level bug prediction," in *Proceedings of the ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '12, pp. 171-180, 2012.

[6] X.-Y. Jing, S. Ying, Z.-W. Zhang, S.-S. Wu, and J. Liu, "Dictionary learning based software defect prediction," in *Proceedings of the 36th International Conference on Software Engineering*, p. 414423, 2014.

[7] S. Wang, T. Liu, and L. Tan, "Automatically learning semantic features for defect prediction," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 297-308, 2016.

[8] T. J. McCabe, "A complexity measure," *IEEE Transactions on Software Engineering*, Bd. 2, S. 308-320, Dez. 1976.

[9] S. R. Chidamber und C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, pp. 476- 493, June 1994.

[10] Z. He, F. Peters, T. Menzies, and Y. Yang, "Learning from open-source projects: An empirical study on defect prediction," in *2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 45-54, 2013.

[11] T. T. Nguyen, H. A. Nguyen, N. H. Pham, J. M. Al-Kofahi, und T. N. Nguyen, "Graph-based mining of multiple object usage patterns," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, S. 383392, 2009.

[12] A. T. Nguyen und T. N. Nguyen, "Graph-based statistical language model for code," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, S. 858868, 2015.

[13] Z. Li und Y. Zhou, "Pr-miner: Automatically extracting implicit programming rules and detecting violations in large software code," *SIGSOFT Softw. Eng. Notes*, vol. 30, no. 5, p. 306315, 2005.

[14] A. Hindle, E. T. Barr, Z. Su, M. G. Gabel, und P. Devanbu, "On the naturalness of software," in *Proceedings of the 34th International Conference on Software Engineering*, S. 837847, 2012.

[15] T. K. Landauer und S. T. Dumais, "A solution to Plato's problem: The latent semantic analysis theory of the acquisition, induction, and representation of knowledge," *Psychological Review*, vol. 104, pp. 211- 240, 1997.

[16] K. S. Jones, "Index term weighting," *Information Storage and Retrieval*, Bd. 9, Nr. 11, S. 619 - 633, 1973.

[17] G. H. Golub und C. Reinsch, "Singular value decomposition and least squares solutions", *Numer. Math.* , vol. 14, no. 5, p. 403420, 1970.

[18] B. Croft, D. Metzler, und T. Strohman, *Search Engines: Information Retrieval in Practice*. Pearson, 1. Aufl., 2009.

[19] M. Lan, C. L. Tan, J. Su, and Y. Lu, "Supervised and traditional term weighting methods for automatic text categorization," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 31, no. 4, pp. 721- 735, 2009.

[20] R. W. Selby und V. R. Basili, "Analyzing error-prone system structure," *IEEE Transactions on Software Engineering*, vol. 17, pp. 141-152, Feb. 1991.

[21] T. J. Ostrand, E. J. Weyuker, and R. M. Bell, "Predicting the location and number of faults in large software systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 4, pp. 340-355, 2005.

[22] S. Lessmann, B. Baesens, C. Mues, and S. Pietsch, "Benchmarking classification models for software defect prediction: A proposed framework and novel findings," *IEEE Transactions on Software Engineering*, vol. 34, no. 4, pp. 485-496, 2008.

[23] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fi x e d ? more accurate information retrieval-based bug localization based on bug reports," in *2012 34th International Conference on Software Engineering (ICSE)*, pp. 14-24, 2012.

[24] X. Li, W. Li, Y. Zhang, and L. Zhang, "Deepfl: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, S. 169180, 2019.

[25] X. Ye, H. Shen, X. Ma, R. Bunescu, and C. Liu, "From word embeddings to document similarities for improved information retrieval in software engineering," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*, pp. 404-415, 2016.

[26] Y. Xiao, J. Keung, Q. Mi, and K. E. Bennin, "Improving bug localization with an enhanced convolutional neural network," in *2017 24th Asia-Pacific Software Engineering Conference (APSEC)*, pp. 338-347, 2017.

[27] X. Huo, Y. Yang, M. Li, and D.-C. Zhan, "Learning semantic features for software defect prediction by code comments embedding," in *2018 IEEE International Conference on Data Mining (ICDM)*, pp. 1049- 1054, 2018.

[28] G. E. Box und R. D. Meyer, "An analysis for unreplicated fractional factorials," *Technometrics*, vol. 28, no. 1, pp. 11-18, 1986.

[29] M. Jureczko und L. Madeyski, "Towards identifying software project clusters with regard to defect prediction," in *Proceedings of the 6th International Conference on Predictive Models in Software Engineering*, 2010.

[30] F. Peters, T. Menzies, L. Gong, and H. Zhang, "Balancing privacy and utility in cross-company defect prediction," *IEEE Transactions on Software Engineering*, vol. 39, no. 8, pp. 1054-1068, 2013.

[31] J. Nam, S. J. Pan, and S. Kim, "Transfer defect learning," in *2013 35th International Conference on Software Engineering (ICSE)*, pp. 382-391, 2013.
[32] N. Chinchor, "Muc-4 evaluation metrics," in *Proceedings of the 4th Conference on Message Understanding*, pp. 22-29, 1992.
[33] M. Sokolova und G. Lapalme, "A systematic analysis of performance measures for classification tasks", *Inf. Process. Manage.* , vol. 45, no. 4, pp. 427-437, 2009.
[34] T. Zimmermann, R. Premraj, and A. Zeller, "Predicting defects for eclipse," in *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, 2007.
[35] F. Rahman und P. Devanbu, "How, and why, process metrics are better," in *Proceedings of the 2013 International Conference on Software Engineering*, S. 432441, 2013.