2021 Internationale IEEE-Konferenz über Software-Analyse, -Evolution und -Reengineering (SANER)

Evaluierung der Fehlervorhersage unter realistischen Bedingungen

Sho Ogino* , Yoshiki Higo *und Shinji Kusumoto *
*Graduiertenschule für Informationswissenschaft und -technologie, Universität Osaka, Japan
{s-ogino, higo, kusumoto}@ist.osaka-u.ac.jp

***Abstrakt* - Die Vorhersage von *Fehlern* soll die Kosten der Qualitätssicherung senken. Um ein zuverlässiges Fehlervorhersagemodell zu erstellen, sollten wir realistische Einstellungen verwenden, die alle drei der folgenden Bedingungen erfüllen. (1) Wir sollten einen Datensatz so erstellen, dass wir die Vorhersageleistung des Modells korrekt bewerten können.**
(2) Wir sollten die optimale Granularität der Fehlervorhersage wählen, um die Kosten der Qualitätssicherung zu minimieren. (3) Wir sollten eine abhängige Variable verwenden, die das Vorhandensein oder Nichtvorhandensein von Fehlern in den zu prognostizierenden Softwaremodulen korrekt wiedergibt. Es gibt jedoch noch keine Forschungsarbeiten zu Fehlervorhersagemodellen, die unter den oben genannten realistischen Bedingungen erstellt wurden. Daher haben wir uns in dieser Untersuchung die folgenden zwei Ziele gesetzt.
(1) Wir bewerten experimentell die Vorhersageleistung von Fehlervorhersagemodellen, die unter realistischen Bedingungen erstellt wurden. (2) Wir schlagen Techniken zur Verbesserung der Vorhersageleistung von Fehlervorhersagemodellen vor, die unter realistischen Bedingungen erstellt wurden. Das erste Ziel ist nun erreicht. Unsere experimentellen Ergebnisse zeigen, dass das F-Maß der unter realistischen Bedingungen erstellten Fehlervorhersagemodelle nur 0,19 beträgt. Es gibt also noch einige Probleme, die gelöst werden müssen, um ein leistungsfähiges Modell zur Vorhersage von Fehlern unter realistischen Bedingungen zu erstellen.

***Index* Begriffe-Qualitätssicherung; Fehlervorhersage; maschinelles Lernen**

I. EINFÜHRUNG

In den letzten Jahren hat der Umfang der Softwareentwicklung stetig zugenommen. In dieser Situation sind Techniken zur Senkung der Entwicklungskosten unerlässlich. Techniken zur Senkung der Kosten für die Qualitätssicherung, wie z. B. Reviews, sind besonders wichtig, da die Qualitätssicherung einen großen Teil der Entwicklungskosten ausmacht [1].

Die Fehlervorhersage ist eine Technik zur Senkung der Kosten für die Qualitätssicherung. Unter Fehlervorhersage versteht man die Vorhersage des Vorhandenseins oder Nichtvorhandenseins von Fehlern in den Softwaremodulen (z. B. Quelltexten). Die Identifizierung fehlerhafter Module und deren intensive Überprüfung kann die Kosten der Qualitätssicherung senken.

Um die Leistung eines Fehlervorhersagemodells richtig zu bewerten, muss der Datensatz richtig aufgebaut sein. Traditionell wurde ein Datensatz erstellt, indem abhängige Variablen für Softwaremodule berechnet wurden, die zu einem bestimmten Zeitpunkt in einem Projekt vorhanden waren, und dann unabhängige Variablen anhand der Entwicklungsgeschichte vom Beginn des Projekts bis zum Zeitpunkt berechnet wurden [2], [3].

Pascarella et al. bezeichneten jedoch Datensätze, die auf diese Weise erstellt wurden, als unrealistisch, da sie Daten enthalten, die realistischerweise nicht gewonnen werden können [4]. Pascarella et al. schlugen die Release-by-Release-Technik vor, um einen Datensatz zu erstellen, der dieses Problem vermeidet, und untersuchten die Vorhersageleistung mit einem realistischen Datensatz. Sie kamen zu dem Schluss, dass die gemessene Leistung gering ist und dass die Erstellung eines leistungsstarken Fehlervorhersagemodells mit einem realistischen Datensatz eine Herausforderung darstellt.

Wir sind der Ansicht, dass die Fehlerprognosemodelle von Pascarella et al., die mit der Release-by-Release-Technik erstellt wurden, umstritten sind. Das Problem mit ihrem Modell ist, dass die abhängige Variable nicht der Indikator ist, "ob ein Fehler in dem Softwaremodul zu diesem Zeitpunkt existiert (*isBuggy*)", sondern der Indikator, "ob ein Fehler in der vergangenen Periode mindestens einmal behoben wurde (*hasBeenFixed*)". Dies liegt daran, dass es eine Situation geben kann, in der ein Softwaremodul "falsch für hasBeenFixed, aber wahr für isBuggy" ist, was bedeutet, dass ein Modell, das mit einem Datensatz erstellt wurde, dessen abhängige Variable *hasBeenFixed* ist (hasBeenFixed-Modell), Fehler möglicherweise nicht korrekt vorhersagt.

Wie bereits erwähnt, wurden in früheren Studien Modelle mit unrealistischen Einstellungen für den Aufbau von Datensätzen und abhängigen Variablen erstellt. Um zuverlässige Fehlervorhersagemodelle zu erstellen, sollten wir realistische Einstellungen verwenden. Es wurde jedoch keine Forschung zu Fehlervorhersagemodellen durchgeführt, die unter realistischen Bedingungen erstellt wurden. In dieser Studie wurden die folgenden zwei Ziele festgelegt und untersucht.

1) Wir bewerten experimentell die Leistung von Fehlervorhersagemodellen, die unter realistischen Bedingungen erstellt wurden.
2) Wir schlagen eine Technik zur Verbesserung der Leistung von Fehlervorhersagemodellen vor, die unter realistischen Bedingungen erstellt wurden.

Das erste Ziel wurde in diesem Moment erreicht. Unsere experimentellen Ergebnisse haben gezeigt, dass das F-Maß des unter realistischen Bedingungen erstellten Fehlervorhersagemodells nur 0,19 beträgt, und es gibt noch einige Probleme zu lösen, um leistungsstarke Fehlervorhersagemodelle unter realistischen Bedingungen zu erstellen. In naher Zukunft werden wir einen Weg zur Verbesserung der Leistung vorschlagen.

II. HINTERGRUND

Es gibt verschiedene Einstellungen für die Erstellung von Fehlervorhersagemodellen. Wir stellen hier zwei Einstellungen vor, (1) die Granularität des Vorhersageziels und (2) die Art der Erstellung eines Datensatzes, die eng mit dieser Studie zusammenhängen.

A. Granularität des Vorhersageziels

Es gibt drei Zielgranularitäten, die häufig in Fehlervorhersagemodellen verwendet wurden: Quellcode, Methode und Commit. In dieser Studie konzentrieren wir uns auf die Vorhersage von Fehlern auf Methodenebene, da Pascarella et al. die Methode als Vorhersagegranularität gewählt haben und eines der Ziele dieser Studie darin besteht, den Wahrheitsgehalt der Schlussfolgerungen von Pascarella et al. zu bestätigen.

B. Art der Erstellung eines Datensatzes

Um die Leistung eines Modells zur Vorhersage von Fehlern richtig zu bewerten, muss der Datensatz auf geeignete Weise erstellt werden. Traditionell,

978-1-7281-9630-5/21/$31.00 ©2021 IEEE
DOI 10.1109/SANER50967.2021.00052

2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER) | 978-1-7281-9630-5/20/$31.00 ©2021 IEEE | DOI: 10.1109/SAN

Die unabhängigen Variablen wurden anhand der Entwicklungsgeschichte vom Beginn des Projekts bis zu einem bestimmten Zeitpunkt berechnet [2], [3]. Pascarella et al. bezeichneten den Aufbau eines Datensatzes auf diese Weise als unpraktisch, da der Datensatz Daten enthält, die realistischerweise nicht abgerufen werden können.

Pascarella et al. schlugen das Release-by-Release-Verfahren als geeignete Methode zur Erstellung eines Datensatzes vor, der ein solches Problem vermeidet. *Release-by-Release* berechnet abhängige und unabhängige Variablen für die Softwaremodule, die in einem bestimmten Release des Projekts vorhanden sind. Die unabhängigen Variablen werden anhand der Entwicklungshistorie von der vorherigen Version bis zur aktuellen Version berechnet. Das Verfahren zur Erstellung eines Datensatzes auf der Grundlage des *Release-by-Release* wird in Abschnitt V-B ausführlich beschrieben.

III. FORSCHUNGSFRAGEN

In diesem Papier untersuchen wir die folgenden vier Forschungsfragen.

RQ1. Ist *hasBeenFixed* als abhängige Variable für die Erstellung von Fehlervorhersagemodellen geeignet?

Durch RQ1 versuchen wir quantitativ zu beurteilen, ob *hasBeenFixed* als abhängige Variable für die Erstellung von Fehlervorhersagemodellen geeignet ist. Wenn die meisten fehlerhaften Methoden nicht von *hasBeenFixed* erfasst werden, dann können Modelle zur Vorhersage von *hasBeen-Fixed* (hasBeenFixed-Modelle) fehlerhafte Methoden nicht korrekt vorhersagen, was bedeutet, dass *hasBeenFixed* als

RQ2. Ist die Vorhersageleistung von isBuggy-Modellen geringer als die von hasBeenFixed-Modellen?

Durch RQ2 vergleichen wir hasBeenFixed-Modelle und isBuggy-Modelle aus der Perspektive der Vorhersageleistung. Wenn die Leistung von isBuggy-Modellen geringer ist als die von hasBeenFixed-Modellen, gibt es in diesem Forschungsbereich noch einige Probleme.

RQ3. Welcher Algorithmus für maschinelles Lernen liefert unter realistischen Bedingungen die beste Vorhersageleistung?

Durch RQ3 identifizieren wir den optimalen Algorithmus für maschinelles Lernen unter den folgenden realistischen Bedingungen.

Methode der **Granularität**
Erstellung von Datensätzen, *Release für Release*
Abhängige Variable *istBuggy*

RQ4. Wie verändert sich die Vorhersageleistung unter realistischen Bedingungen, wenn die Entwicklungsgeschichte zunimmt?

Mit Frage 4 versuchen wir herauszufinden, ob die Algorithmen des maschinellen Lernens in Abhängigkeit von der Länge der Entwicklungsgeschichte variiert werden sollten.

IV. KONFIGURATIONEN

In diesem Abschnitt werden die Versuchsaufbauten beschrieben, die den in den folgenden Abschnitten beschriebenen Experimenten gemeinsam sind.

A. Zielvorhaben

Die Zielprojekte dieser Studie sind in Tabelle I aufgeführt. Diese Projekte wurden aus den folgenden vier Gründen ausgewählt.

- Ihre Git-Repositories sind verfügbar.
- Sie sind in Java geschrieben.
- Sie übernehmen die semantische Versionierung.
- Es gibt mindestens vier Veröffentlichungen in ihren Projekten.

B. Wie man Freisetzungen identifiziert

In dieser Studie erstellen wir Datensätze nach dem Release-by-Release-Verfahren. Für die Anwendung der Release-by-Release-Technik ist es erforderlich, den Zeitpunkt der einzelnen Releases zu ermitteln. Wir folgen der von Pascarella et al. [4] vorgeschlagenen Technik, um die Versionen der Projekte zu identifizieren, die semantische Versionierung als Versionierungsschema verwenden. Die semantische Versionierung drückt Versionen in der Form X.Y.Z aus (z. B. 1.2.1), wobei X die Hauptversion darstellt. Wir identifizieren nur Veröffentlichungen von Hauptversionen, d. h. Versionen, bei denen sowohl Y als auch Z gleich Null sind (z. B. 2.0.0).

C. Wie wird hasBeenFixed berechnet?

Für jede Methode wird *hasBeenFixed* wie folgt berechnet.

Schritt1 Wir sammeln das Projektarchiv und die Fehlerberichte, deren Status *FIXED* ist.

Schritt2 Fehlerberichte werden für zuvor entdeckte Fehler ausgegeben, und jeder Fehlerbericht hat eine ID. In diesem Experiment wird ein Commit, dessen Commit-Nachricht die Fehlerberichts-ID enthält, als Bug-fixing-Commit betrachtet. Wir durchsuchen das Repository nach den Bug-fixing-Commits.

Schritt3 Angenommen, eine Zeile der Methode A wird in einer fehlerbehebenden Übertragung geändert. Fällt der Commit in das Intervall von
die $(n\ r)$-te Freigabe (R_{n-1}) zur n-ten Freigabe (R_n), dann ist Methode A *wahr* für *hasBeenFixed* bei R_n.

step4 Wenn die oben genannten Operationen für alle Fehlerberichte durchgeführt werden, werden Methoden, die für *hasBeenFixed* nicht *wahr* sind, als falsch für *hasBeenFixed* bewertet.

TABELLE I
ZIELPROJEKTE

Name des Projekts	Entwicklungszeitraum	veröffentlicht	verpflichtet sich	Fehlerberichte	Methoden	Prozentsatz der fehlerhaften Methoden
lucen-solr(LUC)	6.967 Tage	9	34,319	6,232	72,356	2.3 %
Pforte(WIC)	5.860 Tage	7	20,972	2,871	22,961	2.5 %
cassandra(CAS)	4.239 Tage	4	25,671	5,402	28,567	5.6 %
linuxtools(LIN)	4.197 Tage	8	10,733	2,201	28,644	1.2 %
egit(EGI)	4.021 Tage	5	6,473	2,529	8,082	7.2 %

jgit(JGI)	4.029 Tage	5	8,014	700	14,098	1.1 %
eclipse.jdt.core(ECL)	7.062 Tage	4	24,804	9,716	23,910	7.9 %
poi(POI)	6.824 Tage	4	10,421	2,620	34,218	3.6 %

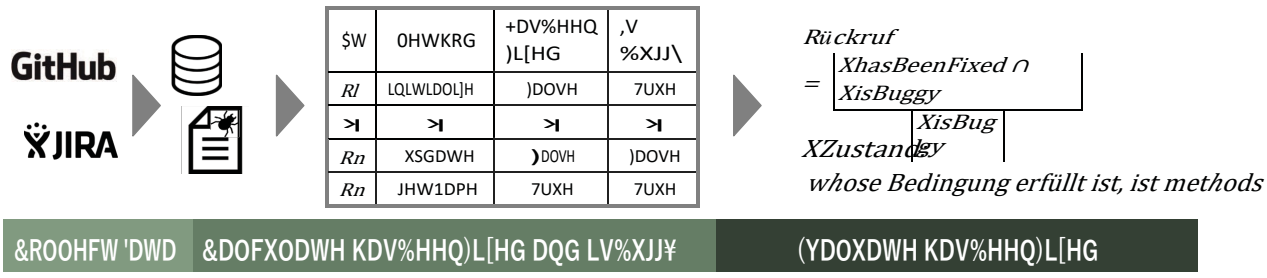

Abb. 1. Ablauf des Experiments für RQ1

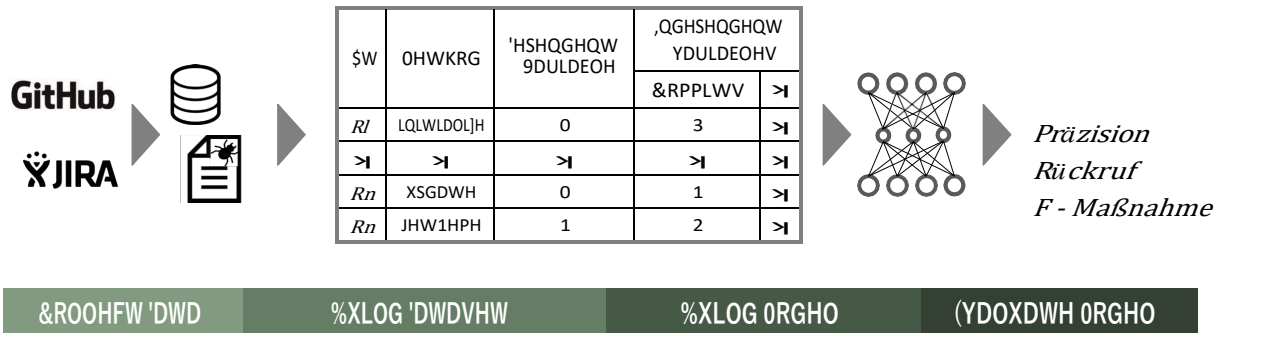

Abb. 2. Ablauf des Experiments für RQ2

D. *Wie berechnet man isBuggy*

Wir haben die in der Literatur [5] beschriebene Implementierung eines SZZ-Algorithmus [6] verwendet, um *isBuggy* wie folgt zu berechnen.

step1 Wie Schritt 1 der hasBeenFixed-Berechnung. **step2** Wie Schritt 2 der hasBeenFixed-Berechnung. **step3** Angenommen, eine Zeile der Methode A wird in einer Fehler-fixing commit. Wir identifizieren den Commit, der die geänderte Zeile eingefügt hat, mit dem "git blame" und betrachten den Commit als einen fehlerverursachenden Commit.

Schritt 4 Wenn ein Bug-fixing-Commit für Methode A nach R_n und der Bug-verursachende Commit für den Bug wurde zwischen R_{n-1} und R_n gemacht, dann ist Methode A *wahr* für *isBuggy* bei R_n.

Schritt 5 Wenn die obigen Operationen für alle Fehlerberichte durchgeführt werden, werden Methoden, die für *isBuggy* nicht *wahr sind*, als falsch für *isBuggy* bewertet.

V. EXPERIMENTELLE ERGEBNISSE

A. *RQ1*

In RQ1 untersuchen wir, ob *hasBeenFixed* fehlerhafte Methoden genau erfasst. Ein Überblick über das Experiment ist in Abbildung 1 dargestellt.

Vorgehensweise. Zur Beantwortung von RQ1 haben wir folgende Schritte durchgeführt: (1) Wir haben die Repositories und die Fehlerberichte der Zielprojekte gesammelt, (2) *isBuggy* und *hasBeenFixed* für jede Zielmethode berechnet und (3) die Korrektheit von *hasBeenFixed* quantitativ bewertet.

Bewertungsmaßstab. Zur Bewertung der Eignung von *hasBeenFixed* als abhängige Variable für die Erstellung von Fehlervorhersagemodellen verwenden wir die nachfolgend beschriebene *Rückrufquote*.

$XhasBeenFixed$ ist eine Menge von Methoden, die für *has- wahr* sind.

$$Rückruf = \frac{|XhasBeenFixed \cap XisBuggy|}{|XisBuggy|}$$

BeenFixed, und $XisBuggy$ ist ein Satz von Methoden, die für *isBuggy wahr sind*. Eine Methode namens "initialize", die in Abbildung 1 gezeigt wird, ist wahr für *isBuggy*, aber falsch für *hasBeenFixed* bei

R_1 . Die "Initialisierung" bei R_1ist also ein Element von $XisBuggy$,
aber kein Element von $XhasBeenF_{ixed}$. Angenommen, die ausgewertete
recall kleiner als 0,5 ist. In diesem Fall halten wir *hasBeenFixed* als abhängige Variable für Fehlervorhersagemodelle für ungeeignet, da hasBeenFixed-Modelle nicht mehr als die Hälfte der fehlerhaften Methoden vorhersagen können.

Ergebnisse. Die Ergebnisse des Experiments sind in Tabelle II aufgeführt. In Tabelle II ist der Recall, d. h. der Prozentsatz der von *hasBeenFixed* erfassten fehlerhaften Methoden, für jedes Projekt aufgeführt. Die Wiederfindungsraten liegen alle signifikant unter 0,5. Wir schließen daraus, dass *hasBeenFixed* als abhängige Variable für Fehlerprognosemodelle ungeeignet ist, weil *hasBeenFixed* nicht mehr als die Hälfte der fehlerhaften Methoden erfassen kann.

B. RQ2

In RQ2 erstellen wir hasBeenFixed- und isBuggy-Modelle, um ihre Leistung zu vergleichen. Ein Überblick über das Experiment ist in Abbildung 2 dargestellt.

Vorgehensweise. Wir haben die folgenden Schritte durchgeführt, um RQ2 zu untersuchen: Wir haben (1) die Repositories und die Fehlerberichte der Zielprojekte gesammelt, (2) Datensätze auf der Grundlage der gesammelten Daten erstellt, (3) Fehlervorhersagemodelle auf der Grundlage der Datensätze erstellt und
(4) die Leistung der Modelle zur Fehlerprognose bewertet.

Abhängige Variable. Wir nahmen *hasBeenFixed* und *isBuggy* als abhängige Variablen an und erstellten Regressionsmodelle für

TABELLE II
RÜCKRUF VON HASBEENFIXED FÜR ISBUGGY

Projekt	LUC	WIC	CAS	LIN	EGI	JGI	ECL	POI	ALLE
Rückruf	0.22	0.22	0.36	0.16	0.34	0.16	0.43	0.20	0.28

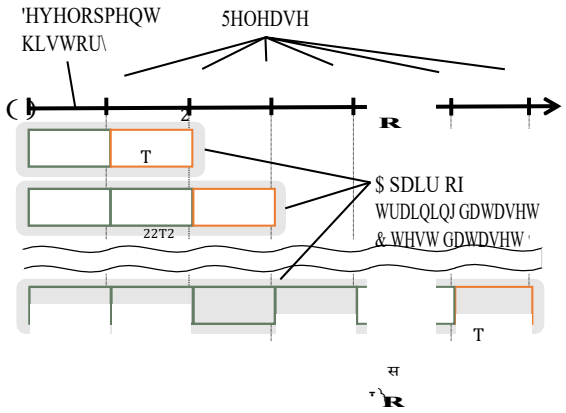

Abb. 3. Übersicht der Datensätze für jedes Projekt

jede von ihnen. Die Definitionen der Metriken sind in Abschnitt IV-C bzw. IV-D enthalten.

Unabhängige Variablen. Wir verwenden die von Giger et al. [2] definierten Produkt- und Prozesskennzahlen als unabhängige Variablen. Die Metriken sind in den Tabellen III und IV aufgeführt.

Erstellung eines Datensatzes. In dieser Studie werden Paare aus einem Trainingsdatensatz und einem Testdatensatz mit Hilfe der Release-by-Release-Technik berechnet. Auf der Grundlage dieser Technik können wir n 1 Paare aus einem Trainingsdatensatz S_k und einem Testdatensatz T_k aus einem Projekt bilden, das n Mal veröffentlicht wurde, wie in Abbildung 3 dargestellt. Die Paare werden wie folgt berechnet.

1) Wir berechnen Paare von abhängigen und unabhängigen Variablen (Instanzen) unter Verwendung der Entwicklungsgeschichte zwischen R_k und R_k für jedes Release R_k (R_0 ist der Beginn der Entwicklung).
2) Wir erstellen einen Testdatensatz T_k aus den Instanzen, die aus der Entwicklungsgeschichte zwischen R_{k-1} und R_k berechnet wurden.
3) Wir erstellen einen Trainingsdatensatz S_k aus den Instanzen, die aus der Entwicklungsgeschichte vor R_{k-1} berechnet wurden.
4) Wir nehmen eine Überstichprobe von Instanzen in S_k, die für eine bestimmte Variable wahr sind, um das Problem des Klassenungleichgewichts zu lösen [7].

TABELLE III
UNABHÄNGIGE VARIABLEN (PRODUKTKENNZAHLEN)

Methode verweisen	FanInAnzahl der Methoden, die auf eine bestimmte
	FanOutAnzahl der Methoden, auf die eine bestimmte
	LocalVarAnzahl der lokalen Variablen im Körper einer
Methode verweist	ParameterAnzahl der Parameter in der Deklaration
	CommentRatioVerhältnis der Kommentare zum
Quellcode (zeilenbasiert)	CountPathAnzahl der möglichen Pfade im Körper
	KomplexitätMcCabe Zyklomatische Komplexität einer
Methode	execStmtAnzahl der ausführbaren Quellcode-
	maxNestingMaximale Verschachtelungstiefe aller
Kontrollstrukturen	

TABELLE IV
UNABHÄNGIGE VARIABLEN (PROZESSKENNZAHLEN)

	MethodHistoriesAnzahl der Änderungen einer
Methode	
	AuthorsAnzahl der eindeutigen Autoren, die eine
	StmtAddedSumme aller hinzugefügten Quellcode-
Methode geändert haben	MaxStmtAddedMaximum StmtAdded
Anweisungen	AvgStmtAddedDurchschnitt von StmtAdded
	StmtDeletedSumme aller gelöschten
	MaxStmtDeletedMaximum von StmtDeleted
Quellcode-Anweisungen	
	ChurnSum von stmtAdded - stmtDeleted
	AvgStmtDeletedDurchschnitt von
	AvgChurnDurchschnittliche Abwanderung pro
Methode	
	CondAnzahl der Konditionsänderungen über alle
Revisionen hinweg	
	ElseDeletedAnzahl der gelöschten else-parts über
alle Revisionen	

Algorithmus für maschinelles Lernen. Der *Random Forest* (RF) [8] wurde als maschineller Lernalgorithmus gewählt, da RF in früheren Studien [2]-[4] häufig verwendet wurde und die Der Zeitaufwand für die Erstellung von Modellen ist gering.

Abstimmung der Hyperparameter. Die Abstimmung von Hyperparametern ist für die Erstellung leistungsstarker Modelle unerlässlich [9]. Wir haben für jedes Modell 10 Stunden Hyperparameter-Tuning durchgeführt.

Bewertungsmaßstäbe. Wir haben die folgenden Maßnahmen ergriffen
die Vorhersageleistung zu bewerten.

$$Precision = \frac{|TP|}{}$$

S rn für alle Methodenhistorien DeclAnzahl der
t Änderungen der Methodendeklaration
m ElseAddedAnzahl der hinzugefügten else-
t parts über alle Revisionen
D
e
l
e
t
e
d

M
a
x
C
h
u
r
n
M
a
x
i
m
a
l
e
r

C
h
u

TP (True Positive) ist eine Menge von Methoden, die von einem Modell als fehlerhaft eingestuft wurden und tatsächlich fehlerhaft sind. FN (False Negative) ist eine Menge von Methoden, die von einem Modell als nicht fehlerhaft eingestuft wurden und tatsächlich fehlerhaft sind. FP (False Positive) ist eine Menge von Methoden, die von einem Modell als fehlerhaft eingestuft wurden und in Wirklichkeit nicht fehlerhaft sind.

Ergebnisse. Die Ergebnisse des Experiments sind in Tabelle V aufgeführt. Die Tabelle zeigt die durchschnittliche Vorhersageleistung sowohl für die hasBeenFixed-Modelle als auch für die isBuggy-Modelle.

Das F-Maß der Modelle, die *isBuggy* als realistische abhängige Variable verwenden, ist etwa halb so hoch wie das von *hasBeenFixed*. Wir kommen zu dem Schluss, dass die Leistung der isBuggy-Modelle deutlich geringer ist als die der hasBeenFixed-Modelle und dass es in diesem Forschungsbereich noch Probleme gibt.

C. RQ3

In Frage 3 untersuchen wir, welcher maschinelle Lernalgorithmus unter realistischen Bedingungen die beste Leistung erbringt. Die Konfigurationen für die Erstellung und Bewertung der Modelle sind mit Ausnahme der abhängigen Variable und des maschinellen Lernalgorithmus die gleichen wie in Frage 2.

Abhängige Variable. Wir setzen *isBuggy* als abhängige Variable ein. Die Definition ist in Abschnitt IV-D enthalten.

Algorithmus für maschinelles Lernen. Wir testen *Random Forest* (RF) und *Deep Neural Network* (DNN) [10] als maschinelle Lernalgorithmen. DNN wurde ausgewählt, weil DNN aussagekräftige Modelle berechnen kann, auch wenn sie in Bezug auf die Vorhersageleistung nicht immer besser sind als RF-Modelle.

Ergebnisse. Die Ergebnisse der Experimente sind in Tabelle VI aufgeführt. Die Tabelle zeigt die durchschnittliche Vorhersageleistung sowohl für RF als auch für DNN. Wir kommen zu dem Schluss, dass RF-Modelle aus Sicht des F-Measure besser sind als DNN-Modelle.

TABELLE V
VORHERSAGELEISTUNG FÜR ABHÄNGIGE VARIABLEN

	Präzision	Rückruf	F-Maßnahme
HasBeenFixed	0.33	0.75	0.40
IsBuggy	0.15	0.61	0.19

TABELLE VI
VORHERSAGELEISTUNG FÜR ALGORITHMEN DES MASCHINELLEN LERNENS

	Präzision	Rückruf	F-Maßnahme
RF	0.15	0.61	0.19
DNN	0.13	0.64	0.18

$$F\,Measure = 2 \times \frac{Recall \times Präzision}{Wiedererkennungswert + \frac{1}{Präzision}}$$

Fig. 4. performance transition as the development history accumulates

D. RQ4

Bei der Erstellung der Datensätze verwenden wir die Release-by-Release-Technik. Bei dieser Technik ist die Größe des Trainingsdatensatzes proportional zur Länge der Entwicklungsgeschichte (der Anzahl der Versionen). In Frage 4 untersuchen wir, wie sich die Vorhersageleistung der unter realistischen Bedingungen erstellten Modelle mit zunehmender Entwicklungshistorie verändert. Die Konfigurationen für die Erstellung der Modelle sind die gleichen wie bei RQ3.

Ergebnisse.

Abbildung 4 zeigt die durchschnittliche Leistung der Modelle zur Vorhersage fehlerhafter Methoden bei der Veröffentlichung. Bei den Ergebnissen des Experiments sind zwei Punkte besonders erwähnenswert.

Erstens nimmt die Vorhersageleistung mit zunehmender Entwicklungshistorie ab. Dies könnte darauf zurückzuführen sein, dass sich die Eigenschaften der fehlerhaften Methoden mit zunehmender Entwicklungsgeschichte ändern. Dies ist jedoch nur eine Hypothese, die wir in naher Zukunft überprüfen werden.

Zweitens sind DNN-Modelle in einem frühen Entwicklungsstadium leistungsfähiger, während RF-Modelle in einem späteren Entwicklungsstadium leistungsfähiger sind. Dies könnte daran liegen, dass sich die Merkmale der fehlerhaften Methoden mit zunehmender Entwicklungsgeschichte ändern und DNN-Modelle die Merkmale der fehlerhaften Methoden in der frühen Entwicklungsphase stark gelernt haben.

Wir kommen zu dem Schluss, dass DNN besser für die Erstellung von Fehlervorhersagemodellen unter realistischen Bedingungen in der zweiten Version und RF in späteren Versionen geeignet ist.

VI. BEDROHUNGEN DER VALIDITÄT.

Im Folgenden beschreiben wir mögliche Gefahren für unsere Studie.

Abhängige Variable. In dieser Studie verwenden wir Fehlerberichte mit dem Status *FIXED*, um *hasBeenFixed* und *isBuggy* zu berechnen. Andererseits können die Zielprojekte latente Fehler enthalten, die noch nicht aufgedeckt sind und für die noch keine Fehlerberichte erstellt wurden. Das Vorhandensein von latenten Fehlern kann die Genauigkeit der abhängigen Variablen beeinträchtigen.

Abstimmung der Hyperparameter. Die Abstimmung von Hyperparametern ist für die Erstellung leistungsstarker Modelle mit Algorithmen des maschinellen Lernens unerlässlich. In dieser Studie haben wir für jedes Modell 10 Stunden Hyperparameter-Tuning durchgeführt. Wenn die Modelle länger abgestimmt werden, können leistungsfähigere Modelle erstellt werden.

SZZ-Algorithmus. Zur Berechnung von *isBuggy* haben wir die in der Literatur [5] beschriebene Implementierung eines SZZ-Algorithmus [6] verwendet. Allerdings ist die Genauigkeit des SZZ-Algorithmus noch verbesserungswürdig [11], [12]. Dieses Problem kann die Genauigkeit von *isBuggy beeinträchtigen*.

VII. SCHLUSSFOLGERUNG

Diese Arbeit ist der e r s t e Schritt zur Erstellung zuverlässiger Fehlervorhersagemodelle unter realistischen Bedingungen. Wir untersuchten zunächst die Eignung der Metrik *hasBeenFixed* als abhängige Variable für Fehlervorhersagemodelle. Dann erstellten wir Fehlervorhersagemodelle unter realistischen Bedingungen und bewerteten ihre Leistung.

Infolgedessen war *hasBeenFixed* in der Lage, nur 28 % der fehlerhaften Methoden zu erfassen. Daher ist *hasBeenFixed* als abhängige Variable für Fehlerprognosemodelle ungeeignet. Das F-Maß der Fehlervorhersagemodelle, die unter realistischen Bedingungen erstellt wurden, betrug nur 0,19, und es erwies sich als schwierig, praktische Fehlervorhersagemodelle unter realistischen Bedingungen zu erstellen. Aus diesen Ergebnissen können wir schließen, dass es noch einige Probleme zu lösen gibt, um leistungsstarke Fehlerprognosemodelle unter realistischen Bedingungen zu erstellen.

Wir versuchen derzeit, eine neue Technik zu entwickeln, um die Leistung von Fehlervorhersagemodellen unter realistischen Bedingungen zu verbessern. Konkret wollen wir untersuchen, wie sich die Vorhersageleistung ändert, wenn neue unabhängige Variablen eingeführt werden.

ANERKENNTNIS

Diese Arbeit wurde von MEXT/JSPS KAKENHI 20H04166 unterstützt.

REFERENZE N

[1] T. Britton, L. Jeng, G. Carver, P. Cheak und T. Katzenellenbogen, "Increasing software development productivity with reversible debugging," accessed 2020-10-16, https://undo.io/media/uploads/files/Undo ReversibleDebugging Whitepaper.pdf. –

[2] E. Giger, M. D'Ambros, M. Pinzger, and H. Gall, "Method-level bug prediction," in *International Symposium on Empirical Software Engineering and Measurement*, 2012, pp. 171-180.

[3] H. Hata, O. Mizuno, and T. Kikuno, "Bug prediction based on fine-grained module histories," *Proceedings - International Conference on Software Engineering*, pp. 200-210, 2012.

[4] L. Pascarella, F. Palomba, and A. Bacchelli, "Re-evaluating method-level bug prediction," in *2018 IEEE 25th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, 2018, pp. 592-601.

[5] M. Borg, O. Svensson, K. Berg, und D. Hansson, "Szz unleashed: An open implementation of the szz algorithm - featuring example usage in a study of just-in-time bug prediction for the jenkins project," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, 2019, pp. 7-12.

[6] J. Sliwerski, T. Zimmermann, and A. Zeller, "When do changes induce fixes?" *SIGSOFT Software Engineering Notes*, vol. 30, no. 4, pp. 1-5, 2005.

[7] N. V. Chawla, *Data Mining and Knowledge Discovery Handbook*. Springer, 2010, S. 875-886.

[8] A. Liaw und M. Wiener, "Classification and regression by randomforest," in *R news*, vol. 2, no. 3, 2002, S. 18-22.

[9] J. Bergstra und Y. Bengio, "Random search for hyper-parameter optimization", *Journal of Machine Learning Research*, Bd. 13, S. 281-305, 2012.

[10] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems*, vol. 1, 2012, pp. 1097-1105.

[11] C. Williams und J. Spacco, "Szz revisited: Verifying when changes induce fixes," in *Proceedings of the 2008 Workshop on Defects in Large Software Systems*, 2008, S. 32-36.

[12] D. A. da Costa, S. McIntosh, W. Shang, U. Kulesza, R. Coelho, and A. E. Hassan, "A framework for evaluating the results of the szz approach for identifying bug-introducing changes," *IEEE Transactions on Software Engineering*, vol. 43, no. 7, pp. 641-657, 2017.