Automated program repair

Beiträge zum Software Engineering

Sönke Schmidt
(Soenke.Schmidt@fu-berlin.de)

Motivation

Was wäre, wenn eine KI

- defekten oder fehlerhaften Code von Dir berichtigen könnte?
- deinen Code verbessern und optimieren könnte?
- Logikfehler anzeigt?

Die Papers (ICSE 2019)

[1] Learning to spot and refactor inconsistent method names

(https://dl.acm.org/doi/10.1109/ICSE.2019.00019)

Hier wird angegeben, wie wichtig der Methodenname ist, um die Lesbarkeit und Softwarewartung zu gewährleisten. Dazu wird anhand eines Convolutional Neural Networks überprüft, ob der Methodenname zum Methodeninhalt passt.

[2] On learning meaningful code changes via neural machine translation

(https://dl.acm.org/doi/10.1109/ICSE.2019.00021)

Via Deep Learning sollen automatisch Codeänderungen stattfinden, um eine Fehlerbehebung oder Refactoring des Codes (genauer gesagt einer Methode) zu ermöglichen. Dafür wird eine Neural Machine Translation angelegt und lernt anhand von Pull-Requests die vorher und nachher Änderungen am Code, um so diese Änderungen dann selbstständig zu replizieren.

Methodenname – erstes Paper

Methodennamen sind wichtig

- für die Lesbarkeit des Codes
- um die Softwarewartung zu erleichtern

Man erwartet,

- das Methodennamen mit Methodenimplementierungen übereinstimmen

Methodenname – erstes Paper

Methodennamen sind wichtig

- für die Lesbarkeit des Codes
- um die Softwarewartung zu erleichtern

Man erwartet,

- das Methodennamen mit Methodenimplementierungen übereinstimmen

```
public static int quersumme(int n)
{
  int qs = 0;
  while (n > 0)
  {
    qs += n % 10;
    n /= 10;
  }
  return qs;
}
```

Methodenname – erstes Paper

Methodennamen sind wichtig

- für die Lesbarkeit des Codes
- um die Softwarewartung zu erleichtern

Man erwartet,

- das Methodennamen mit Methodenimplementierungen übereinstimmen

```
public static int quersumme(int n)
{
  int qs = 0;
  while (n > 0)
  {
    qs += n % 10;
    n /= 10;
  }
  return qs;
}
```

"If you have a good name for a method, you don't need to look at the body." — Fowler et al.

Was führt zu schlechten Methodennamen?

- Benennung von Bezeichnern ist die schwierigste Aufgabe von Programmierern (laut P. Johnson, siehe [1] S. 1).

- Inkonsistente Namen:
  - Kein guter Thesaurus vorhanden
  - Widersprüchliche Stile bei einer Zusammenarbeit mehrerer Entwickler
  - Falsches klonen von Code
  - Zeitmangel oder mangelnde Konzentration („nicht aufpassen“)

Debuggen von Methodennamen

- Per Hand (Code Review)
- Gemeinsamkeiten zwischen Methodennamen in einem großen Datensatz analysieren
  - Code selbst wird nicht analysiert
- Analyse ob die Konsistenz zwischen Methodennamen und Methodencode gegeben ist

Debuggen von Methodennamen

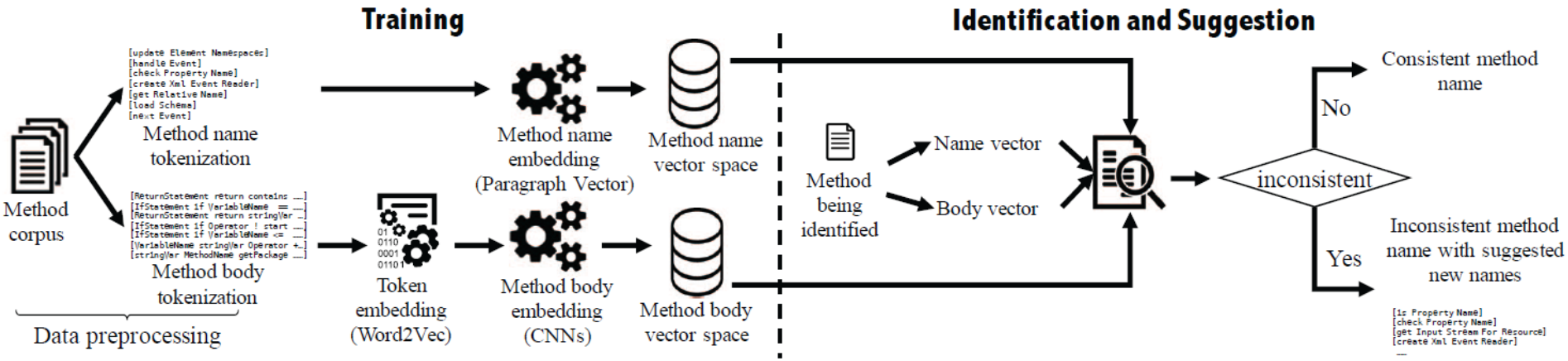
- Per Hand (Code Review)
- Gemeinsamkeiten zwischen Methodennamen in einem großen Datensatz analysieren
  - Code selbst wird nicht analysiert
- Analyse ob die Konsistenz zwischen Methodennamen und Methodencode gegeben ist

hasHeaderPatterns()

```
private boolean blub() {
    String[] parts = filter.split(",");
    for (String pattern : parts) {
        if (pattern.startsWith("--")) {
            continue;
        }
        if (pattern.startsWith("header:")
        ;
```

Angepasstes Beispiel von: https://github.com/apache/camel/blob/master/core/camel-core-processor/src/main/java/org/apache/camel/processor/ClaimCheckAggregationStrategy.java

Wie funktioniert das?



Wie funktioniert das?

Method corpus

[update Element Namespaces]
[handle Event]
[check Property Name]
[create Xml Event Reader]
[get Relative Name]
[load Schema]
[next Event]

Method name tokenization

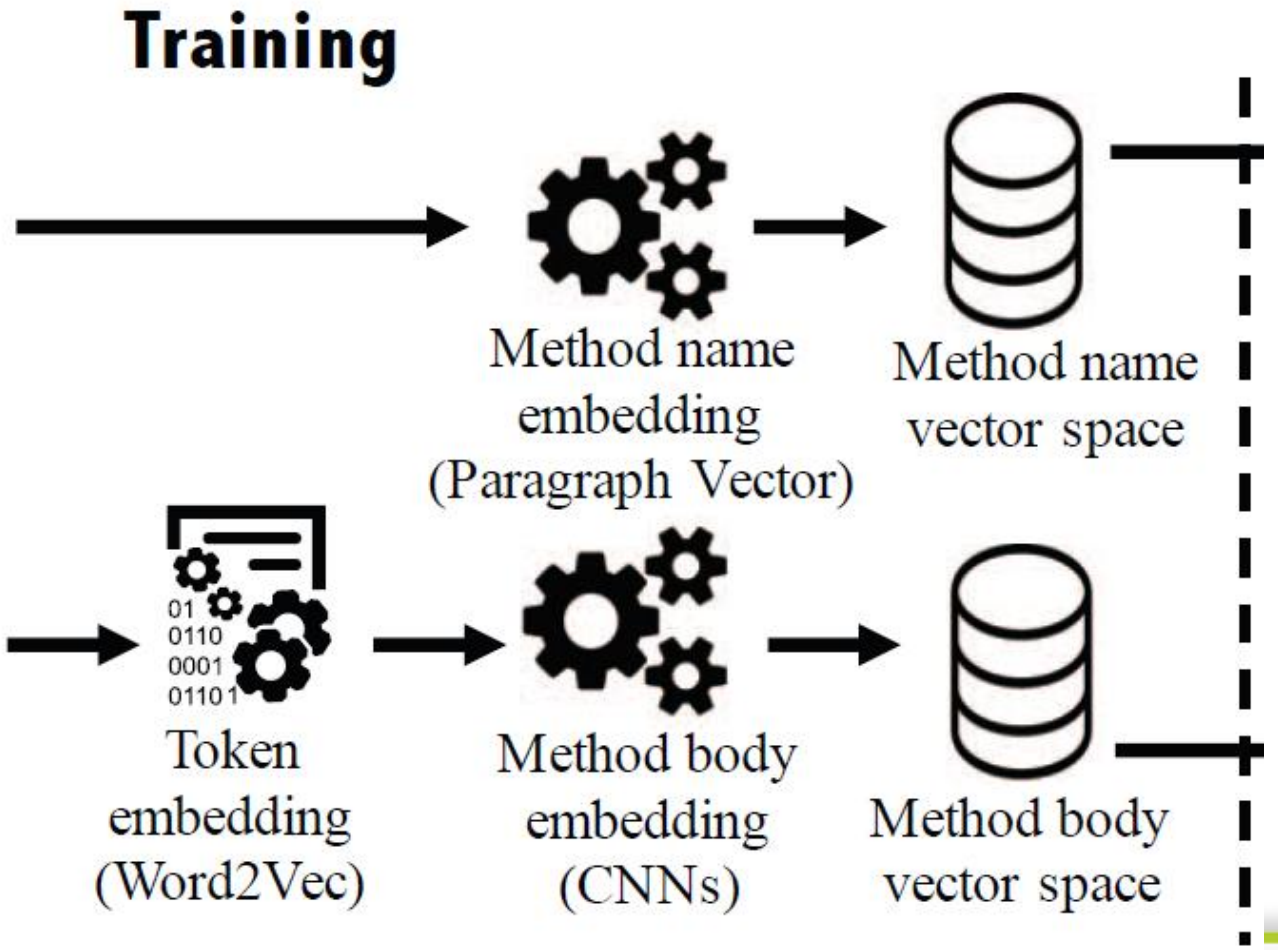
[ReturnStatement return contains .....]
[IfStatement if VariableName == .....]
[ReturnStatement return stringVar ...]
[IfStatement if Operator ! start .....]
[IfStatement if VariableName <= .....]
[VariableName stringVar Operator +...]
[stringVar MethodName getPackage .....]

Method body tokenization

Data preprocessing

Wie funktioniert das?

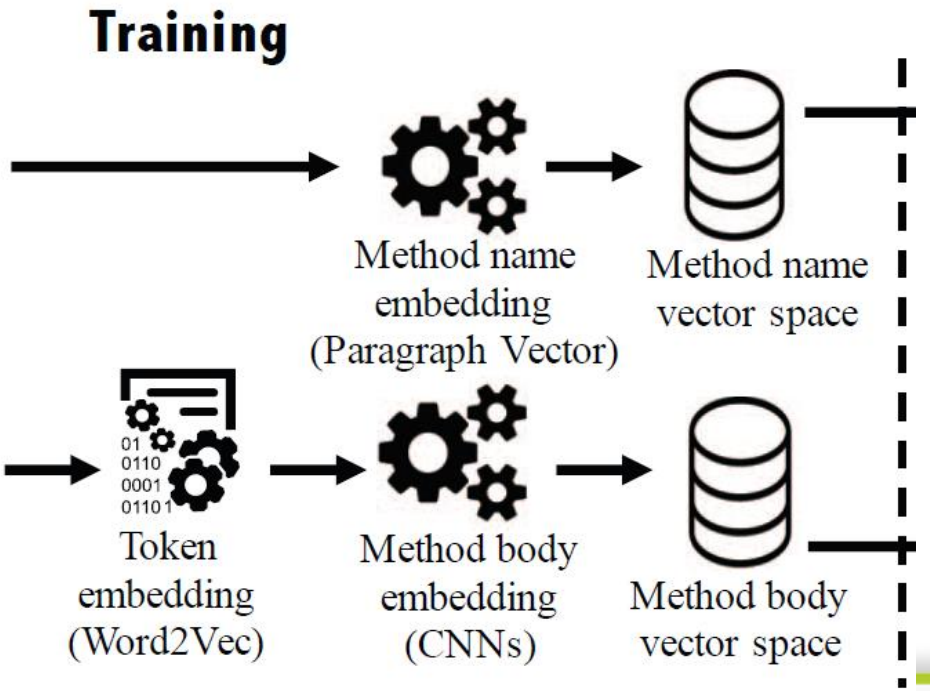
Die Analyse der Konsistenz zwischen Methodennamen und Methodencode beruht auf:



Wie funktioniert das?

Die Analyse der Konsistenz zwischen Methodennamen und Methodencode beruht auf:

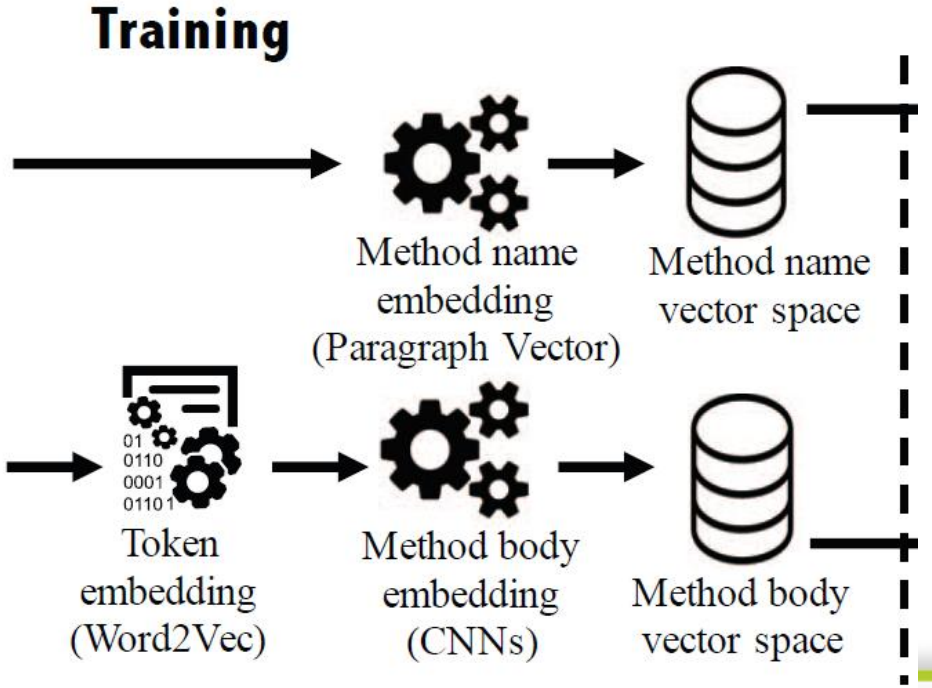
- Paragraph Vector
  - „bag-of-words“ Problem



Wie funktioniert das?

Die Analyse der Konsistenz zwischen Methodennamen und Methodencode beruht auf:

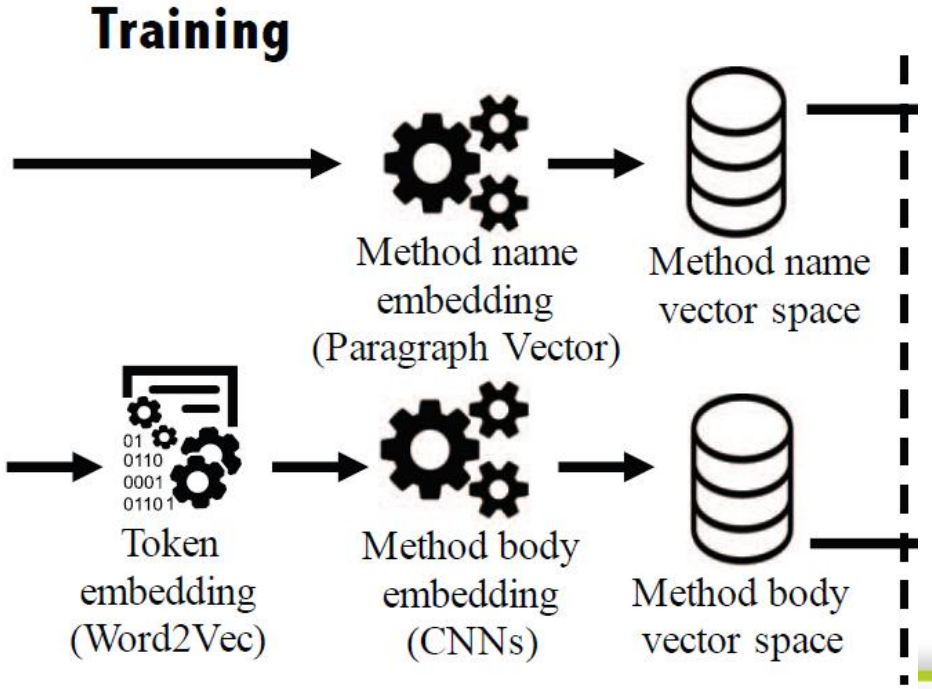
- Paragraph Vector
  - „bag-of-words“ Problem
- Convolutional Neural Networks (= CNN)
  - LeNet5



Wie funktioniert das?

Die Analyse der Konsistenz zwischen Methodennamen und Methodencode beruht auf:

- Paragraph Vector
  - „bag-of-words“ Problem
- Convolutional Neural Networks (= CNN)
  - LeNet5
- Word2Vec

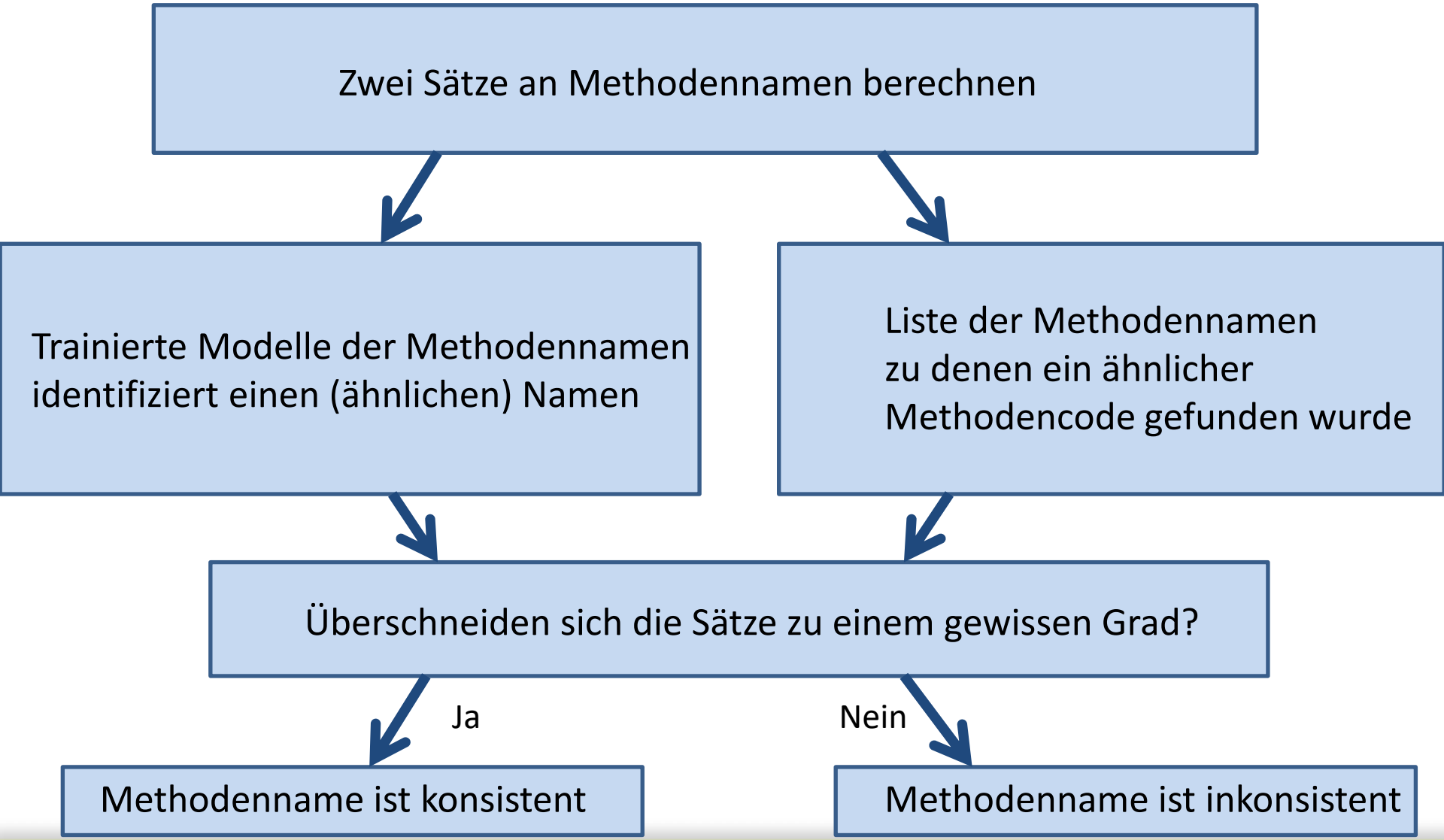


Wie funktioniert das?

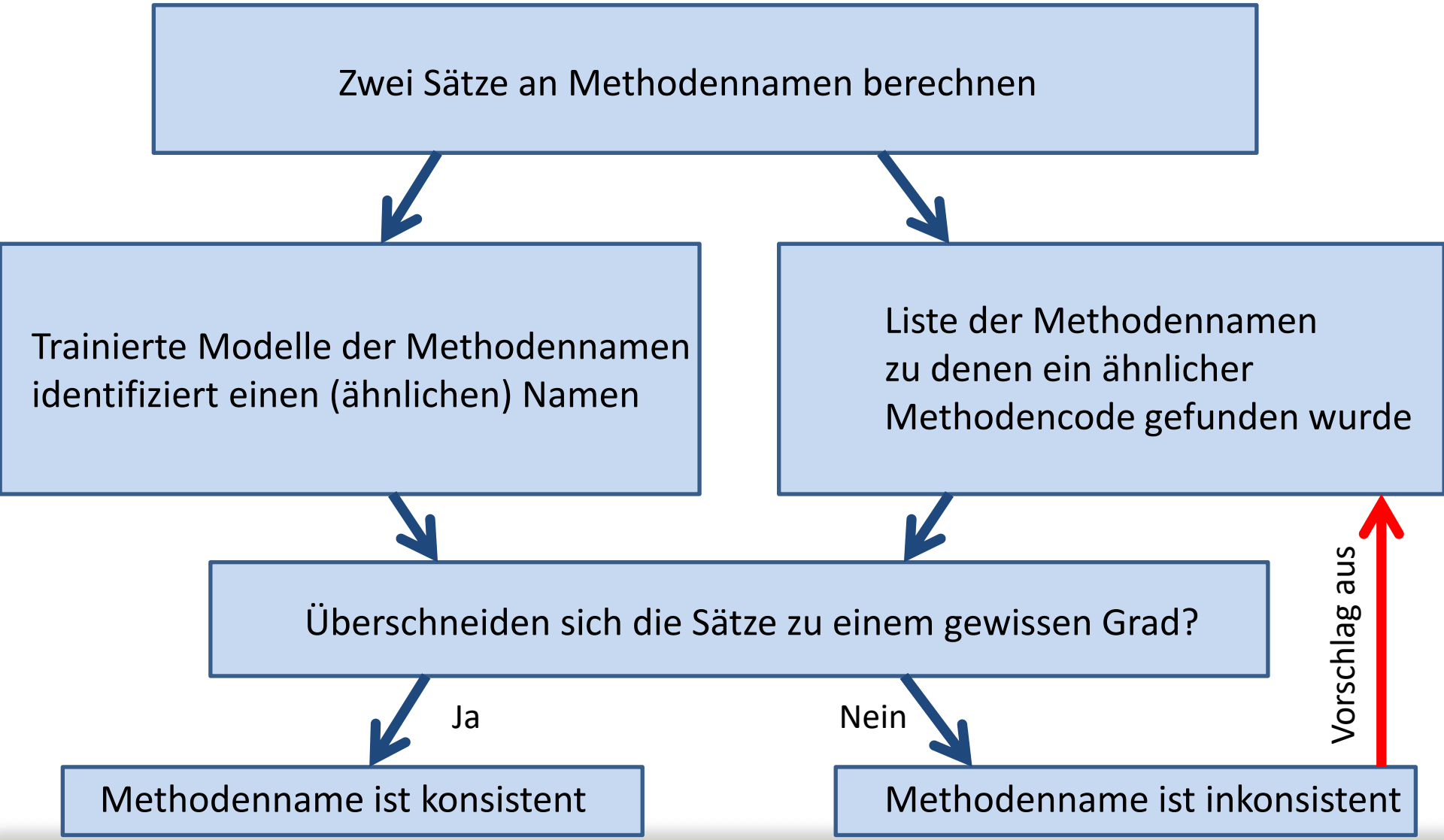
Es sollen zwei Ziele erreicht werden:

- Einbetten von Token aus Methodennamen und Methodencodes in numerische Vektorformen
- Extrahieren von Merkmalsdarstellungen zur genauen Identifizierung ähnlicher Methodennamen und Methodencodes

Wie funktioniert das?



Wie funktioniert das?



Ergebnisse und Fakten

- Aus 430 Open-Source-Java-Projekten wurden
  - 2.116.413 Methoden für Trainingsdaten verwendet

- 2.805 Methoden mit geänderten Namen an Testdaten

- Erreichtes F1-Maß von 67,9 %
- Genauigkeit beim Vorschlagen von 34 bis 50 % der ersten Sub-Token
- Genauigkeit von 16 bis 25 % beim Vorschlagen genauer vollständiger Namen für inkonsistente Methodennamen

16 % bei nur einem Namenstreffer

Bis zu 25 % bei fünf vorgeschlagenen Namen

F-Maß ist bei der statistischen Analyse der binären Klassifizierung ein Maß für die Genauigkeit eines Tests.
Siehe:
https://en.wikipedia.org/wiki/F-score

Ergebnistabelle – erst eine kurze Erklärung

T1 = Vermeidung von Inkonsistenzen: Es wird bewertet, inwieweit sich der vorgeschlagene Name von dem fehlerhaften Namen der Eingabe unterscheidet.

T2 = Genauigkeit des ersten Tokens: Es wird bewertet, inwieweit die vorgeschlagenen Namen mit dem Namen identisch sind, die Entwickler beim Debuggen von Änderungen vorgeschlagen haben.

T3 = Vollständige Namensgenauigkeit: Es wird bewertet, inwieweit der vollständige Name jedes vorgeschlagenen Namens mit dem Namen identisch ist, den Entwickler beim Debuggen von Änderungen vorgeschlagen haben.

Der Schwellenwert „k“ stellt die Größe der Sätze benachbarter Vektoren dar.

Der Schwellenwert „thr“ (für "threshold") gibt die Anzahl der vorgeschlagenen Namen an.

Ergebnistabelle

Accuracy	T1		T2		T3	
	thr = 1	thr = 5	thr = 1	thr = 5	thr = 1	thr = 5
conv_attention	78.4%	27.6%	22.3%	33.6%	0.3%	0.6%
copy_attention	77.2%	38.9%	23.5%	44.7%	0.4%	1.1%
R1 (k = thr)	86.9%	**69.7%**	36.4%	47.2%	16.5%	22.9%
R2 (k = 10)	88.5%	67.5%	34.8%	50.2%	**17.0%**	25.4%
R3 (k = 10)	**88.6%**	67.5%	34.7%	50.3%	16.9%	25.5%
R4 (k = 10)	77.0%	67.3%	**35.4%**	**50.5%**	16.0%	**25.7%**

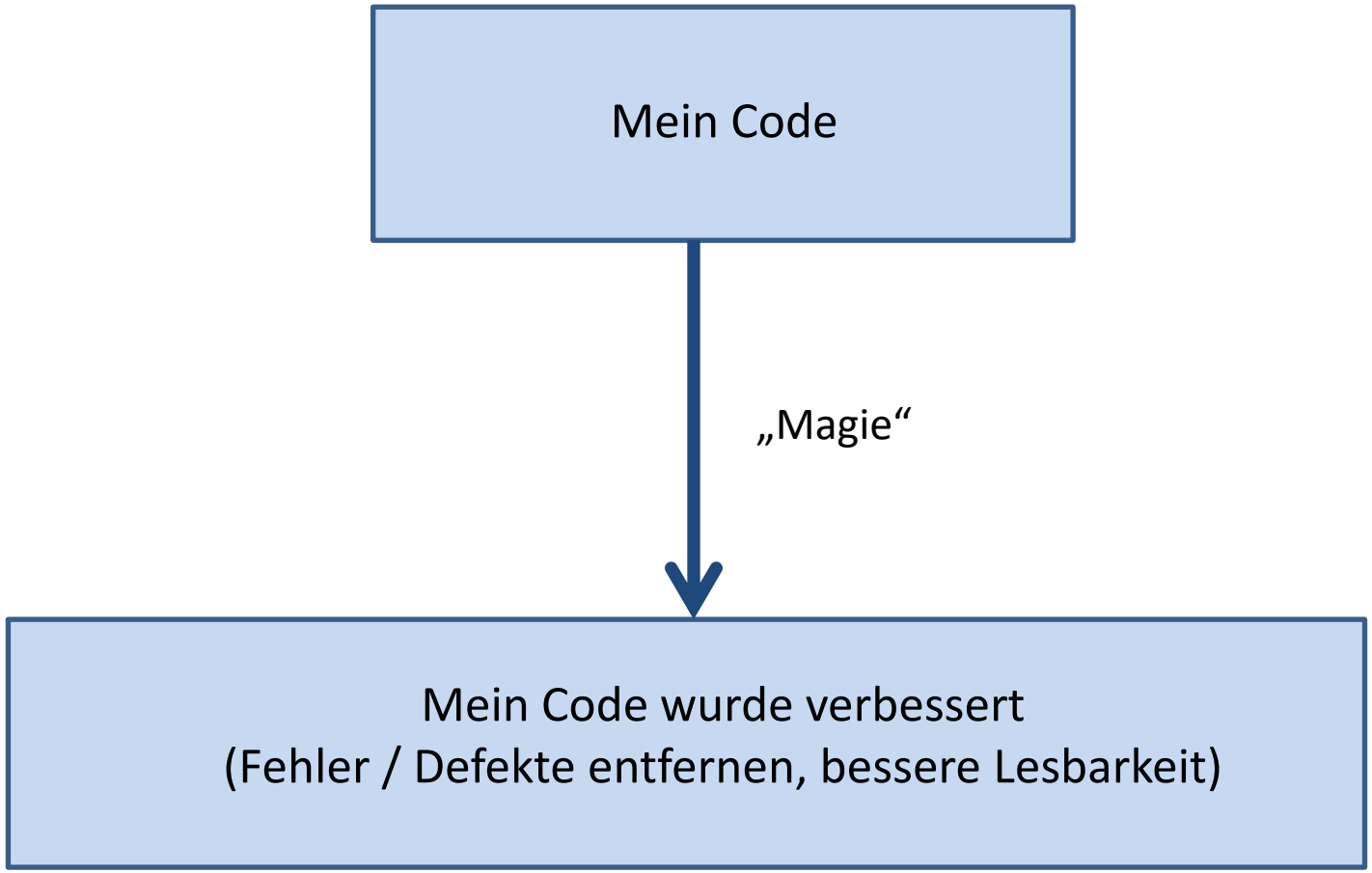
Methodencode – zweites Paper

Methodencode

- beinhaltet die Logik zur Ausführung des Codes
- ist fehleranfällig
- kann ggf. verbessert werden
  (Lesbarkeit oder Code-Optimierungen => Refactoring)

Bester Weg wäre gute Lesbarkeit, einfache Wartbarkeit und hohe Effizienz

Die Idee


Mein Code
„Magie“
Mein Code wurde verbessert
(Fehler / Defekte entfernen, bessere Lesbarkeit)

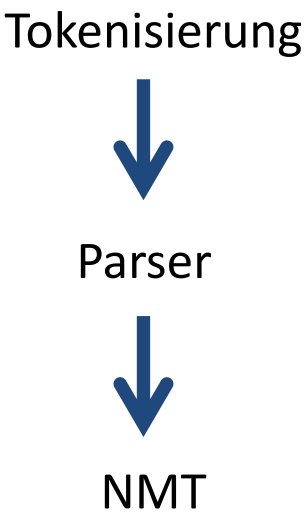
Die „Magie“ oder besser die KI

Es wird eine neuronale maschinelle Übersetzung
(Neural Machine Translation = NMT) verwendet. Diese wird auch häufig für die Übersetzung verschiedener (Programmier-) Sprachen verwendet.

"Neuronale maschinelle Übersetzung ist ein Ansatz zur maschinellen Übersetzung, bei dem ein künstliches neuronales Netzwerk verwendet wird, um die Wahrscheinlichkeit einer Folge von Wörtern vorherzusagen, wobei typischerweise ganze Sätze in einem einzigen integrierten Modell modelliert werden."

Quelle: https://en.wikipedia.org/wiki/Neural_machine_translation

Die „Magie“ oder besser die KI



Wo kommen die Trainingsdaten her?

Drei große Code-Review-Repositories wurden durchsucht:

- Android
- Google Source
- Ovirt

Insgesamt haben diese Repositorys 339 Teilprojekte.

Aus diesen wurden 78.981 merged pull requests (PRs) gesammelt und einer Codeüberprüfung unterzogen.

Dabei wurden nur kleine bis mittelgroße Methoden verwendet:

- Einfacher zum Lernen/Trainieren für die KI
- So wurden überwiegend Fehlerbehebungs- und Refactoring-Aktivitäten gelernt
- Implementierung neuer Funktionen wurde nicht gelernt

Warum merged PRs?

Aus folgenden Gründen wurden nur zusammengeführte (merged) und überprüfte (reviewed) PRs verwendet:

- NMT-Modell soll aussagekräftigere Änderungen lernen
- Welche Art von Codeänderung wird gemacht? Verstehen der verschiedenen Arten an Codeänderungen
- Gründe für die implementierten Änderungen nachvollziehbar
- Merged PRs haben den Code vor sowie nach der Zusammenführung und können so verglichen/ausgewertet werden

Was wird mit den Daten gemacht?

Es wurden abstrakte Syntaxbaum-Bearbeitungsoperationen auf Methodenebene aus diesen PRs unter Verwendung feinkörniger Quellcode-Differenzierung extrahiert:

- Das führte zu 239.522 Methodenpaaren
  - bestehend aus vor dem PR und nach dem merged PR

Ein Encoder-Decoder Recurrent Neural Network (= RNN) wurde dann verwendet, um die Codetransformationen zu lernen, die von Entwicklern während der PR-Aktivitäten durchgeführt wurden.

Was wird mit den Daten gemacht?

Danach wird das NMT-Modell quantitativ und qualitativ bewertet

- Quantitative Analyse: Wie ist die Fähigkeit den Projektcode genau so zu ändern, wie es Entwickler während realer PRs getan haben?

- Qualitative Analyse: Destillieren einer Taxonomie aussagekräftiger Code-Transformationen, die das Modell automatisch aus den Trainingsdaten lernen konnte.

Ergebnisse

- „Die extrahierte Taxonomie zeigt, dass das Modell in der Lage ist, eine Vielzahl aussagekräftiger Code-Transformationen zu erlernen, Fehler automatisch zu beheben und Code wie Menschen umzugestalten.“ ([2] S. 26)
- Das NMT-Modell kann in 16-36 % der Fälle dieselben Codetrans-formationen einfügen, die von Entwicklern in PRs implementiert wurden

16-21 % bei nur einer vorgeschlagenen Code-Änderung

30-36 % bei zehn vorgeschlagenen Code-Änderungen

Ergebnisse

PERFECT PREDICTIONS

Dataset	Beam	M_{small}	M_{medium}
Google	1	10 (4.62%)	7 (3.07%)
	5	17 (7.87%)	13 (5.70%)
	10	20 (9.25%)	17 (7.45%)
Android	1	40 (9.61%)	51 (14.12%)
	5	71 (17.06%)	73 (20.22%)
	10	79 (18.99%)	76 (21.05%)
Ovirt	1	55 (12.35%)	60 (11.78%)
	5	93 (20.89%)	90 (17.68%)
	10	113 (25.39%)	102 (20.03%)
All	1	228 (21.16%)	178 (16.21%)
	5	349 (32.40%)	306 (27.86%)
	10	388 (36.02%)	334 (30.41%)

DATASETS

Dataset	M_{small}	M_{medium}
Google	2,165	2,286
Android	4,162	3,617
Ovirt	4,456	5,088
All	10,783	10,991

VOCABULARIES

Dataset	Vocabulary	Abstracted Vocabulary
Google	42,430	373
Android	266,663	429
Ovirt	81,627	351
All	370,519	740

Richtig oder Falsch

```
public void reset(int i) {
      if ((i < 0) || (i >= mLen)) { [...] }
}
```

>= wird zu >

richtig? falsch?

```
public void reset(int i) {
      if ((i < 0) || (i > mLen)) { [...] }
}
```

Richtig oder Falsch

```
public void reset(int i) {
      if ((i < 0) || (i >= mLen)) { [...] }
}
```

>= wird zu >

richtig? falsch?

```
public void reset(int i) {
      if ((i < 0) || (i > mLen)) { [...] }
}
```

Richtig (16-21 % Wahrscheinlichkeit)!

Es ist richtig!

Der Entwickler erläutert den Bugfix:

“a fix to the FieldPacker.reset() API, which was not allowing the FieldPacker to ever point to the final entry in its buffer” ([2] S. 32)

Daher wurde der Operand von >= in > geändert

Richtig (16-21 % Wahrscheinlichkeit)!

Es ist richtig!

Der Entwickler erläutert den Bugfix:

“a fix to the FieldPacker.reset() API, which was not allowing the FieldPacker to ever point to the final entry in its buffer” ([2] S. 32)

Daher wurde der Operand von >= in > geändert

Eigene Meinung

Woher soll jemand das denn wissen?
Woher soll jemand den Kontext kennen?
Woher soll jemand das Verhalten der Klasse(n) und das der anderen Methoden kennen?

Woher soll es die KI kennen?

Eigene Meinung

- Änderungen können als sinnvoll erachtet werden, wenn sie so interpretiert werden. Müssen es aber nicht sein!
- Auch andersherum möglich: Es kann eine gute Änderung sein, die man aber selber nicht erkennt bzw. weiß, wieso? Und die Änderung somit verwirft.
- Wenn unter zehn Vorschlägen eine 30-36 %-ige Chance besteht, dass die Änderung richtig ist, dann ist es ziemlich mühsam, diese zehn Vorschläge durchzugehen.
  - Besonders, da das Verfahren nur kleine bis mittlere Methoden abdeckt. Der Änderungsvorschlag müsste überaus wertvoll sein, um es dennoch zu machen.
- Es braucht viele Daten, die per Hand eingegeben und überprüft werden müssen, damit das Training stattfinden kann
  - Hier stellt sich die Frage der Unterstützbarkeit…

Eigene Meinung

- Die PR geben noch eine Erklärung über die Änderungen, die KI gibt nur Lösungsvorschläge ohne Erklärungen…
  - Vom Menschen könnte man noch lernen und die Änderungen nachvollziehen und auch überprüfen.
- Im Umkehrschluss kann die KI aus den Texten der PR nicht lernen und sich verbessern. Die KI ignoriert die Erklärungen im PR

„Moreover, the use of frequent identifiers and literals allows the NMT model to learn typical changes (e.g., if(i>1) to if(i>0)) and introduce API calls based on other API calls already available in the code.” ([2] S. 27)

- Wann sind diese typischen Änderungen angebracht? Jedes zweite Mal? Eine Logik sollte hier aber zum Kontext passen
  - Logischer Zusammenhang wird ggf. nicht erkannt und es kann auch nicht erkannt werden, ob der Programmierer absichtlich so gehandelt hat.

Eigene Meinung

Beispiel:

Die KI sieht häufig:

for (int i = 0; i <= count -1; i++)

for (int i = 0; i <= size -1; i++)

Jemand programmiert nun:

for (int i = 0; i <= lastIndex; i++) → for (int i = 0; i <= lastIndex -1; i++)

Die KI verbessert -1

Eigene Meinung – gibt es auch positives?

- Besonders die Vorschläge der Methodennamen könnten sinnvoll sein:
  - Leichteres Verstehen von Methoden anderer Programmierer, da Methodenname ersten Aufschluss geben kann (jedoch vermutlich nur über trivialen und eher kürzeren Code)
  - Geeignet evtl. für Anfänger, die (noch) nicht verstehen, was die Methode machen wird (dennoch hohe Fehlerquote der KI)
- Bei Sicherheitskritischen Methoden können zusätzliche Vorschläge noch Hinweise geben, ob an spezielle Fälle gedacht wurde. Auch eine Erinnerung an Ausnahmen bzw. Problemfällen erfolgt so:
  - Wo Menschenleben von abhängen, ist eine mehrfache Überprüfung des Codes sicher noch akzeptabel
- Durch Nachvollziehbarkeit des Handelns der KI vielleicht besser nutzbar
  - Das Fraunhofer HHI arbeitet an dem Nachvollziehen bzw. Anzeigen der KI Entscheidung. Leichteres Nachvollziehen der Entscheidung für den Menschen

Vielen Dank für Ihre Aufmerksamkeit.

Haben Sie

Fragen?

Diskussion

- Zwingen mich die Vorschläge nun dazu ein Code Review zu machen?
  - Zum Überprüfen auf Richtigkeit muss der Zusammenhang verstanden werden

Diskussion

- Zwingen mich die Vorschläge nun dazu ein Code Review zu machen?
  - Zum Überprüfen auf Richtigkeit muss der Zusammenhang verstanden werden
- Wer profitiert nun davon, solche Vorschläge zu erhalten?
  - Können Anfänger einschätzen, ob der Vorschlag gut/richtig ist?
  - Grob jeder fünfte bis sechste Vorschlag ist richtig oder besser als der aktuelle Code. Kann man noch erkennen, welcher der richtige Vorschlag bei so einer Quote ist?
  - Nützt es professionellen Entwicklern noch als Hinweis? Oder ist man bei so einer hohen Fehlerquote eher „genervt“?
  - Verschlimmert man seinen Code damit eher nur (besonders Anfänger, die die Richtigkeit nicht einschätzen können)?

Diskussion

- Zwingen mich die Vorschläge nun dazu ein Code Review zu machen?
  - Zum Überprüfen auf Richtigkeit muss der Zusammenhang verstanden werden
- Wer profitiert nun davon, solche Vorschläge zu erhalten?
  - Können Anfänger einschätzen, ob der Vorschlag gut/richtig ist?
  - Grob jeder fünfte bis sechste Vorschlag ist richtig oder besser als der aktuelle Code. Kann man noch erkennen, welcher der richtige Vorschlag bei so einer Quote ist?
  - Nützt es professionellen Entwicklern noch als Hinweis? Oder ist man bei so einer hohen Fehlerquote eher „genervt“?
  - Verschlimmert man seinen Code damit eher nur (besonders Anfänger, die die Richtigkeit nicht einschätzen können)?
- Was passiert bei neuem Code, den die KI so noch nie gesehen hat?
  - Beispiel bei Sockets Binding…