Seminararbeit

Rahmenthema des Wissenschaftspropädeutischen Seminars

[redacted]

Leitfach: Informatik

KI in agiler Softwareentwicklung

Verfasser/in: [redacted]

Kursleiter/in: [redacted]

Abgabetermin: [redacted]

Bewertung	Note	Notenstufe in Worten	Punkte		Punkte
Schriftliche Arbeit				· 3	
Abschlusspräsentation				· 1	
				Summe:	
				Gesamtleistung nach § 29 (7) GSO = Summe : 2 (gerundet)	

(Datum, Unterschrift der Kursleiterin bzw. des Kursleiters)

Inhaltsverzeichnis

1. Prolog

> *[Some] software development activities [...] need intellectual human skills, which are not currently available as computer software. To solve this shortcoming, Artificial Intelligence (AI) approaches are the ones that can be used to develop software tools imitating these intellectual skills.*[1]

Experten sehen Künstliche Intelligenz als den nächsten großen Meilenstein in der Industrie 4.0[2]. Schon bald wird jede Branche und jedes Fachgebiet Künstliche Intelligenz integrieren müssen, um am Markt weiterhin konkurrieren zu können. Doch keine Branche wird und ist schon so sehr von Künstlicher Intelligenz betroffen wie die Informatik. Besonders in der Softwareentwicklung findet Künstliche Intelligenz Verwendung, da diese Tools ermöglicht, die den Arbeitsalltag von Programmierenden effizienter oder angenehmer machen. Solche Tools heißen allgemein "Computer Aided Software Engineering"-Tools (oder CASE-Tools) und Intelligent CASE-Tools wenn sie von Künstlicher Intelligenz Gebrauch machen[3].
Ziel dieser Arbeit ist die Untersuchung von Intelligent CASE-Tools, ihren Hintergründen, ihren Einflüssen auf den Alltag der Softwareentwicklung und relevante zukünftige Entwicklungen.
Der Fokus liegt auf der Entwicklung eines eigenen Intelligent CASE-Tools zur Lösung eines bisher übersehenen, fundamentalen Problems in der Softwareentwicklung. Nämlich das automatische Umwandeln von UML-Diagrammen zu objektorientiertem Code. Das Problem wird genau erklärt und die bisherigen Lösungsansätze und Forschungen aufgezeigt. Auf Basis der Probleme und Erkenntnisse der bisherigen Forschungen wird ein neues, besseres System entwickelt und getestet, das basierend auf Bildern im PNG/JPEG Format ein vollständiges Java-Code-Skelett erstellen kann. Allgemein wird also die Frage untersucht, inwieweit künstliche Intelligenz Entwickler*innen in der agilen Softwareentwicklung unterstützen kann.

> *[...]the software engineer's own material, software, can be used to attack the challenges posed by the production of systems in this very same material.*[4]

---

[1] Quelle 3, A. T. Imam, A. J. Alnsour, und A. Al-Hroob, "The Definition of Intelligent Computer Aided Software Engineering (I-CASE) Tools" p1
[2] Quelle 13, M. Javaid, A. Haleem, R. P. Singh, R. Suman, "Artificial Intelligence Applications for Industry 4.0: A Literature-Based Study" p2
[3] Quelle 3, A. T. Imam, A. J. Alnsour, und A. Al-Hroob, "The Definition of Intelligent Computer Aided Software Engineering (I-CASE) Tools" p1
[4] Quelle 4, M. Harman, "The role of Artificial Intelligence in Software Engineering," p1

2. Der Stand von KIs in der Implementationsphase von agiler Softwareentwicklung

2.1 Künstliche Intelligenzen ohne hidden Layer

Die ersten CASE-Tools der Geschichte befinden sich in der Implementations- und Programmierungs Phase der Softwareentwicklung. Diese Tools, sogenannte “Quality of Life" -Tools, übernehmen Refactoring und Reformatting und ermöglichen damit den Entwickler*innen, sich auf die tatsächliche Entwicklung zu fokussieren und die Formatierung automatisiert den integrierten Entwicklungsumgebungen zu überlassen.

> *Refactoring is [...] a crucial tool to maintain high-quality software and [...] [has] been implemented as actionable tools in modern Integrated Development Environments (IDEs), providing developers with an automatic and safe way to apply these predefined code transformations.*[5]

Diese simplen und sehr hilfreichen Features sind unter anderem: Semikolons setzen, Klammern vervollständigen, HTML-Tags schließen, Variablennamen formatieren und Blank-Line-Intendierung.

> *Our first evaluation study shows that 88.7% of the time, the system-generated blank lines are as good as original developer-written ones or even better.*[6]

Diese Tools sind ein perfektes Beispiel, wie Technologien den Programmier-Alltag vereinfachen können. Sie sind aber keine vollwertigen Künstlichen Intelligenzen trainiert mit Deep-Learning Algorithmen, sondern sind meist simple Algorithmen oder sogar Regex Gleichungen. Sie können in keinster Weise Menschen ersetzen, helfen diesen aber sehr. Vor allem stellen sie den historisch ersten Schritt dieser Art von Tools dar und sind die Grundbausteine moderner KI-Tools.

---

[5] Quelle 11, Yaroslav Golubev, Zarina Kurbatova, et al. “One thousand and one stories: a large-scale survey of software refactoring” p1

[6] Quelle 10, X. Wang, L. Pollock and K. Vijay-Shanker, "Automatic Segmentation of Method Code into Meaningful Blocks to Improve Readability," p43

2.2 Künstliche neuronale Netze

Die neueste Entwicklung unter den CASE Tools sind die Intelligent CASE Tools, die künstliche neuronale Netze verwenden, um vor allem bekannte Probleme zu erkennen und zu lösen.

2.2.1 Language Models

69,24% aller professionellen Programmierer*innen benutzen oder planen künstliche neuronale Netze in ihrem Berufsalltag zu verwenden[7]. 47,79% verwenden dabei künstliche Sprachmodelle (LMs) als Coding Helfer. Dies ist besonders interessant, da Sprachmodelle nicht dafür erstellt worden sind, Code zu generieren, sondern dies auf Token-Probability Basis geschieht, genauso wie z.B. Gedichte oder Kochrezepte generiert werden[8]. Das Sprachmodell versteht also nicht wirklich den Code, sondern ist nur darauf trainiert, das nächste Wort so gut wie möglich zu erraten.

LMs haben den großen Vorteil ihrer Dialog Funktionen und Textverständnis, anders als andere neuronale Netze basieren sie nämlich auf menschlichem Sprach-Input und menschlichem Code-Output, was den generierten Code verständlicher und intuitiver macht.

> *LM-produced suggestions must also be readable, robust, fast, and secure*[9]

Ein Aspekt der gegen die ausschließliche Nutzung von LMs spricht, ist, dass sie an Effektivität und Korrektheit verlieren, desto größer und komplexer der Input wird, da die Token Anzahl des Inputs die Wahrscheinlichkeiten und damit die Konkretheit des Outputs verstreut.

> *Other models trained specifically for code also perform better than general models on these tasks*[10]

2.2.2 Code-Generators

Code Generatoren wie Github Copilot oder Codex sind keine Sprachmodelle, sondern neuronale Netze, die spezialisiert auf das Programmieren sind. Diese werden immer besser und hilfreicher, je mehr Kontext vom Menschen bereitgestellt wird und desto größer der Umfang der gesamten Codebase ist.

---

[7] Quelle 21, Stackoverflow survey 2023, #AI-Developer-Tools
[8] Quelle 12, Hugo Touvron et al. “LLaMA: Open and Efficient Foundation Language Models,“ p. 2
[9] Quelle 5, A. Chowdhery, S. Narang, J. et al. "PaLM: Scaling Language Modeling with Pathways" p. 27
[10] Quelle 12, Hugo Touvron et al. “LLaMA: Open and Efficient Foundation Language Models,“ p. 6

> *[Github] Copilot currently offers three main functionalities: convert comments to code, suggest tests matching implementation code, and autofill for repetitive code.*[11]

Github Copilot scannt zum Beispiel das gesamte Projekt mit allen Ordnern und Unterordnern, um den gesamten Kontext zu verstehen und so den Programmierer*innen möglichst effizient helfen zu können. Code Generatoren haben ihren großen Vorteil also in ihrer Akkuratheit und in ihrem Verständnis für Komplexität. In diesen Aspekten übertreffen sie Sprachmodelle deutlich.

> *OpenAI's Codex model [...] is able to correctly solve 30–70% of novel Python problems, while DeepMind's AlphaCode ranked in the top 54.3% among 5000 human programmers [...]*[12]

Das Verständnis für Komplexität von Code Generatoren entsteht aber nur, wenn ein kompetenter Mensch die Aufgabe korrekt und vollständig erklärt. Code Generatoren sind also sehr mächtig in den Händen von erfahrenen Fachkräften, aber deutlich nutzloser als Sprachmodelle in den Händen von Laien.

> *[...]the treated group that has access to GitHub Copilot was able to complete the task 55.8% faster than the control group.*[13]

In diesem Kontext wird also deutlich, dass selbst die stärksten Künstlichen Intelligenzen den Programmierer*innen nur assistieren und sie nicht ersetzen können.

3. Ungelöstes Problem in der Softwareentwicklung

3.1 Problem Einführung

Sogenannte UML-Diagramme (Unified Modelling Language Diagramme) werden in der agilen Softwareentwicklung für die Planung gesamter Projekte verwendet.[14] Es ist dann die Aufgabe der Programmierer*innen, diese Diagramme in nützlichen und funktionierenden Code zu verwandeln. Der erste Schritt ist dabei immer die Object Design Modells aus der Solution Domain heraus zu lesen und sie zu konvertieren.

---

[11] Quelle 15, N. Nguyen, S. Nadi, (2022) "An Empirical Evaluation of GitHub Copilot's Code Suggestions" p. 2
[12] Quelle 8, Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. "Grounded Copilot: How Programmers Interact with Code-Generating Models" p1
[13] Quelle 14, Peng, Sida, et al. (2023) "The impact of ai on developer productivity: Evidence from github copilot." p7
[14] Quelle 2, S. Krusche und B. Bruegge, (2017) "CSEPM - A Continuous Software Engineering Process Metamodel"

One particular kind of model transformation is devoted to code generation with model-to-text transformations[15]

Der Code, der aus solchen Diagrammen abzulesen ist, ist nur ein strukturelles Skelett des Endproduktes und erfordert kein tieferes Verständnis aus der Application Domain. Das Vorgehen der Umwandlung ist also sehr simpel und repetitiv, weswegen es oft sehr viel Zeit und Geld in Anspruch nimmt ohne den tatsächlichen entsprechenden Mehrwert.
Der folgende Teil der Arbeit befasst sich mit der Entwicklung des Programms VULCAN (Visual Uml Link and Class Analyzing Neural-network), welches in der Lage ist, Object Design Modells als pures PNG/JPEG Bild einzulesen und alle herauslesbaren Informationen zur Generierung von Java-Code Skeletten zu verwenden.

3.2 Definition des Materials

Die in agiler Softwareentwicklung benutzten Diagramme sind seit 1996 standardisiert durch die Unified Modelling Language (UML)[16] in der ISO/IEC 19501. Im Folgenden wird sich auf die UML-Klassendiagramme fokussiert. Des Weiteren unterteilen sich Klassendiagramme in "Application Domain" und "Solution Domain".[17]

3.2.1 Application Domain

Addressed in requirements elicitation and analysis[18]

In der Application Domain werden mit "Analysis Object Modells" informelle Konzepte und Ideen dokumentiert. Das beinhaltet unter anderem: Aggregation, Komposition und Kardinalität.

3.2.2 Solution Domain

Addressed in system design, object design and implementation[19]

In der Solution Domain werden mit "Object Design Modells" formell Strukturen und Hierarchien definiert, die per Definition direkt zu Code konvertiert werden können. Das beinhaltet unter anderem: Klassennamen, Methoden, Methodensignaturen, Attribute und Vererbungen.

---

[15] Quelle 7, Syriani, E., Luhunu, L., & Sahraoui, H. (2018). "Systematic mapping study of template-based code generation" p4
[16] Quelle 22 OMG UML Superstructure, Version 2.3: Mai 2010, p.1
[17] Quelle 23 S. Krusche, "Einführung in die Softwaretechnik" Sommersemester 2022, p. 73ff
[18] Quelle 23 S. Krusche, "Einführung in die Softwaretechnik" Sommersemester 2022, p. 73ff
[19] Quelle 23 S. Krusche, "Einführung in die Softwaretechnik" Sommersemester 2022, p. 73ff

3.3 Bisherige Forschung

Die bisherige Forschung, welche sich spezifisch mit dem Einlesen und Analysieren von UML-Diagramm-Bildern befasst, endet 2014 mit der Entwicklung von Img2UML.[20] Diese Arbeit bewertet die eigenen Ergebnisse als zufriedenstellend, veröffentlicht aber nie die Software oder den Quellcode, was die Ergebnisse schwer zu verifizieren und die Software schwer zu verwenden macht.

> *As a result, the accuracy of the Img2UML system is: for classes, 95% of the rectangles that denote classes are recognized, for relationships it is 80% and for text recognition it is 92%*[21]

Die Ergebnisse von VULCAN sowie das Programm selbst werden vollumfänglich veröffentlicht und sind auch, basierend auf 10 Jahren des Fortschritts in KI und vor allem Text- und Strukturen-Erkennen, als akkurater und schneller einzuschätzen.

3.4 Probleme und Einschränkungen

Das größte Problem bei der Entwicklung war das Fehlen von vollständig standardisierten UMLs. Verschiedene Quellen, Institutionen und Zeichenprogramme beziehen sich auf leicht unterschiedliche Standards hinsichtlich, zum Beispiel Linien oder Pfeilen. Deswegen wurde ein allgemeiner Standard für die Input-UML-Diagramme festgelegt.

3.5 Standardisierung der Object Design Modells

Im Folgenden werden die Object Design Modells definiert, die VULCAN am idealsten konvertieren kann. Abweichung von diesen Standards führt vor allem zu Fehlern in der Erbschafts-Erkennung, welche schnell und problemfrei manuell nachkorrigiert werden können.

Die Standards basieren auf den Programmen wie Apollon sowie internen Konventionen der Technischen Universität München:[22]

---

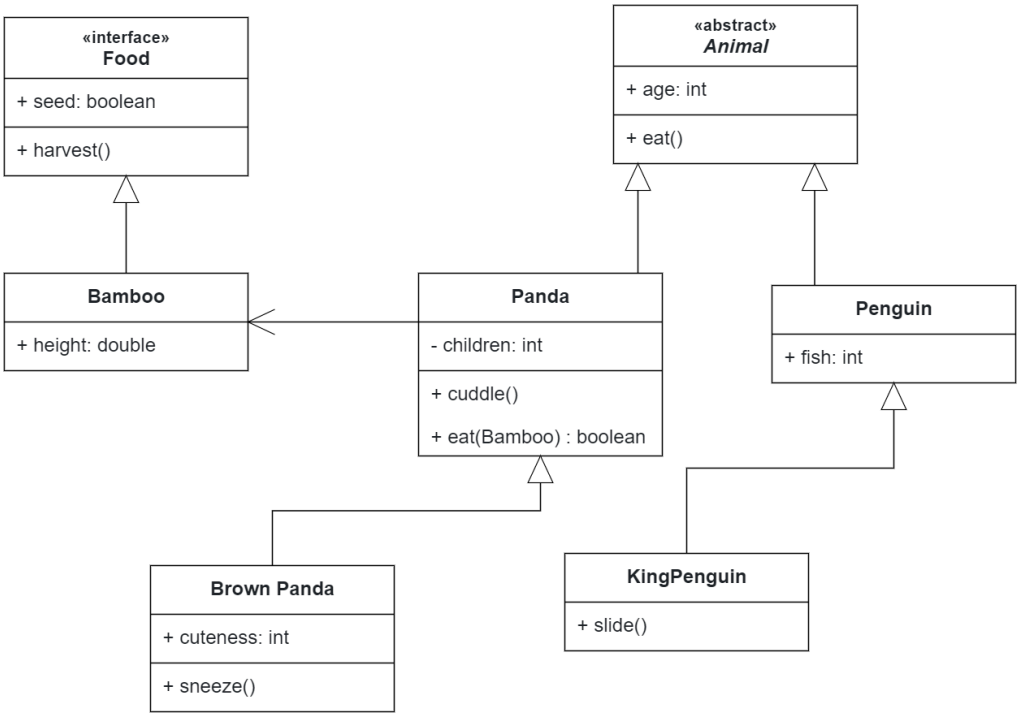
[20] Quelle 9, B. Karasneh and M. R. V. Chaudron, "Img2UML: A System for Extracting UML Models from Images"
[21] Quelle 9, B. Karasneh and M. R. V. Chaudron, "Img2UML: A System for Extracting UML Models from Images" p136
[22] Quelle 23, S. Krusche, "Einführung in die Softwaretechnik" Sommersemester 2022, p. 73ff

3.5.1 Klassen

- Jede Klasse ist ein Rechteck bestehend aus Klassenname, Attributliste, Methodenliste.
- Abstrakte Klassen werden mit “«abstract»” gekennzeichnet.
- Interfaces werden mit “«Interface»” gekennzeichnet.
- Wenn eine Methode public ist, wird sie mit “+” gekennzeichnet, sonst gilt sie standardmäßig als mit “-” gekennzeichnet, was sie private macht.
- Auf das “+” oder “-” einer Methodensignatur folgt der Methodenname.
- Auf den Methodennamen folgt in Klammern ein Auflistung, durch Kommata getrennt, der Parameter (oder nur der Parametertyp).
- Optional folgt auf die restliche Methodensignatur ein Doppelpunkt und der Rückgabetyp der Methode, der sonst standardmäßig “void” ist.
- Wenn eine Attribut public ist, wird es mit “+” gekennzeichnet, sonst gilt es standardmäßig als mit “-” gekennzeichnet, was es private macht.
- Auf das “+” oder “-” der Attributsignatur folgt der Attributname.
- Auf den Attributnamen folgt ein Doppelpunkt und der Typ des Attributs.

3.5.2 Verbindungen

- Per Definition sind bei Object Design Modells nur unidirektionale und bidirektionale Assoziations-Pfeile sowie Vererbungs-Pfeile erlaubt.
- Ein Assoziations-Pfeil ist eine direkte, horizontale oder vertikale, durchgezogene Linie mit 0; 1 oder 2 offenen Spitzen.
- Ein Vererbung Pfeil ist immer eine Aneinanderreihung von durchzogenen, alternierend horizontalen und vertikalen Linien, die vom Kind zur Elternklasse leiten und bei der Elternklasse eine geschlossene, weiße Spitze haben.



[Abbildung eines Beispiels eines standardisierten Klassendiagramms, erstellt mit TUM: Apollon]

4. Image Analysis

Das gesamte Projekt VULCAN wird in Python 3.9.13 entwickelt mit den Libraries CV2 und OCR.

4.1 Image Conversion

Zuerst muss das Bild eingelesen und dann mit CV2 konvertiert werden.

Dabei werden 2 Algorithmen verwendet, um das Schwarz-Weiß Bild für die folgenden Analysen brauchbarer zu machen, nämlich der "Binary Threshold Algorithm" und der "OTSU Algorithm".[23]

Der Binary Threshold Algorithmus untersucht iterativ die Helligkeit jedes Pixels und konvertiert diesen dann basierend auf dem Verhältnis zu einem standardisierten Grenzwert zu entweder weiß oder schwarz, dies bewirkt, dass das entstandene Bild nicht nur aus Grautönen besteht, sondern tatsächlich nur aus schwarzen und weißen Pixeln, was die Untersuchung des Bildes stark vereinfacht, da es nun als eine Bitmap repräsentiert werden kann.

[23] Quelle 16, J. Yousefi, "Image binarization using Otsu thresholding algorithm"

Dieser für den Binary Threshold Algorithmus benötigte Schwellenwert **σ** wird durch den OTSU Algorithmus bestimmt. Dafür werden wieder alle Pixel i des Bildes iterativ nach ihrer Intensität $p(i)$ analysiert, basiert auf welcher dann eine allgemeine Intensitätsverteilung erstellt wird, die das Bild basierend auf den Medianen der Intensitäten μT optimal in "Vordergrund" und "Hintergrund" klassifizieren.[24]

4.2 Rectangle Detection

Für das Erkennen von Rechtecken innerhalb des Bildes wird der Ramer-Douglas-Peucker Algorithmus[25] verwendet:

Der Ramer-Douglas-Peucker Algorithmus vereinfacht polygonale Kurven innerhalb des Bildes, indem er iterativ die Punkte reduziert, die innerhalb der Toleranz Epsilon liegen. Nach mehreren Versuchen hat sich Epsilon gleich 1% der Bogenlänge eingependelt, um die bestmöglichen Ergebnisse zu garantieren. Wenn nun mit dem Algorithmus die komplexen Konturen extrahiert wurden, ohne die fundamentale Form signifikant zu verändern, müssen die Eckpunkte des entstandenen Polygons gezählt werden. Wenn diese Anzahl 4 entspricht, so ist die Kontur ein Rechteck-ähnliches Polygon.

4.3 Text Recognition

Innerhalb jedes Rechteckes befindet sich jetzt Text , der entweder Klassennamen, Methoden oder Attribute beschreibt, dieser Text ist aber in verschiedenen Größen, Schärfen und Schriftarten vorzufinden, weswegen eine OCR(optical character recognition) Engine verwendet werden muss, um den Text zu erkennen.

Die Engine für VULCAN ist die Tesseract OCR Engine, welche Open-Source, gut dokumentiert und Python kompatibel ist.[26]

Die konkret verwendete Version wurde von der Universität Mannheim zur Verfügung gestellt:

https://github.com/UB-Mannheim/Tesseract_Dokumentation [23.9.2023]

[24] Quelle 16, J. Yousefi, "Image binarization using Otsu thresholding algorithm"
[25] Quelle 19, D. Douglas, T. Peucker: "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature"
[26] Quelle 18, R. Smith, "An Overview of the Tesseract OCR Engine"

4.4 Zuordnung

Jedes Rechteck wurde in den vorherigen Schritten also erkannt und hat eine Position und einen Text zugeordnet bekommen. Im nächsten Schritt muss zugeordnet werden, welches Rechteck Klassennamen, Methoden oder Attribute enthält und welche Rechtecke zueinander gehören.

Der Typ des Rechtecks wird durch seine Y-Position bestimmt, das oberste Rechteck ist natürlich immer das Klassennamen-Rechteck. Wenn in direkter Nähe 2 Rechtecke darunter sind, dann ist das obere das Attribut-Rechteck und das unterste ist das Methoden-Rechteck. Wenn nur ein unteres Rechteck existiert, muss durch Analyse des Textes entschieden werden, um welches es sich handelt.

Rechtecke gehören genau dann zusammen, wenn sich genau 2 Ecken in direkter Umgebung voneinander befinden.

Basierend auf diesen Regeln werden alle Rechtecke eingeordnet und ihr Typ wird bestimmt, basierend auf diesen Informationen können die folgenden Schritte ausgeführt werden.

4.5 Klassenstruktur in JSON

Jede Rechteck Gruppe wird nun einzeln untersucht und in eine generelle Struktur unterteilt, um sie in folgenden Schritten zu Code konvertieren zu können.

Diese Untersuchung basiert auf dem Text innerhalb der Rechtecke sowie auf ihrem Typ, entsprechend muss hierfür wieder die OCR-Engine Tesseract verwendet werden.[27]

Die für diese Arbeit arbiträr gewählte Struktur ist wie folgt definiert:

{ ClassName : String, Attributes: String[], Methods: String[] }

zum Beispiel:

{'ClassName': 'Panda', 'Attributes': ['- children: int'], 'Methods': ['+ cuddle()', '+ eat(Bamboo) : boolean']}

[27] Quelle 18, R. Smith, "An Overview of the Tesseract OCR Engine"

4.6 Verbindungs Erkennung

Eine fundamentale Funktion von Sprachen wie Java ist die Polymorphie und Vererbung, diese Vererbungen können auch in UML repräsentiert werden, sind aber schwer zu erkennen.

4.6.1 geschlossene Pfeilspitzen

Wie in 3.5.2 definiert werden Vererbungen durch geschlossene Pfeilspitzen gekennzeichnet, diese lassen sich erkennen, indem wieder der Ramer-Douglas-Peucker Algorithmus[28] verwendet wird, diesmal aber 3 statt 4 als Ergebnis bei Epsilon = 1% erwartet wird.

4.6.2 Pfeile

Da ein Pfeil als Menge an konkatenierten, horizontalen und vertikalen Linien zwischen einer Klasse und einer Pfeilspitze definiert ist, kann zur vollständigen Rückverfolgung von Pfeilen folgender rekursiver Algorithmus verwendet werden:

Jeder Pfeil ist eine Menge L an horizontalen oder vertikalen Linien l aus der Menge HVL, die initial die Pfeilspitzen sind und für die es eine Linie p in der Menge L gibt, bei der die Distanz d der Startpunkte p zu l < 3 ist:

$$d(a,\ b,\ c,\ d)\ \Rightarrow \sqrt{(a\ -\ c)^2\ +\ (b\ -\ d)^2}$$

$$L\ =\ \{Pfeilspitzen\} \cup \{Klassenkanten\}$$

$$L\ =\ L \cup \{l \mid l \in HVL \wedge (\exists p \in L:\ (d(l.x1,\ l.y1,\ p.x1,\ p.y1)\ <\ 3 \vee d(l.x1,\ l.y1,\ p.x2,\ p.y2)\ <\ 3 \vee d(l.x2,\ l.y2,\ p.x1,\ p.y1)\ <\ 3 \vee d(l.x2,\ l.y2,\ p.x2,\ p.y2)\ <\ 3))$$

4.6.3 Pfeil Implikationen

Sobald bekannt ist, welche Pfeil Linien zu welchen Pfeilspitzen gehören, kann eine direkte Verbindung von Start zu Endpunkt gezogen werden, dabei ist die Klasse unterhalb des Startpunkts immer das Kind und die Klasse oberhalb der Pfeilspitze immer die Elternklasse. Diese Beziehungen können für den späteren Code in JSON gespeichert werden.

[28] Quelle 19, D. Douglas, T. Peucker: "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature"

5. Code Conversion

UML diagrams [..] are used as metamodels for code generation[29]

Jetzt existiert eine Liste an Klassen sowie eine Liste an Beziehungen zwischen diesen Klassen, jetzt müssen nur noch die Klassen, repräsentiert durch das in 4.5 beschriebene Schema, zu einem String an Java Code zusammengebaut werden.

Es wird also geschaut wie die Klasse heißt, wenn sie ein interface ist, so sind alle Methoden abstrakt, wenn eine Methode mit "+" anfängt ist sie public, mit einer Verkettung dieser und ähnlicher Bedingungen wird im Detail eine Klassenstruktur aus den Strings heraus gefiltert.

Zum Beispiel:
{'classname': 'public class Panda', 'extends': 'Animal', 'attributes': ['private int children'] 'methods': [{'name': 'public void cuddle', arguments': []}, { 'name': 'public boolean eat' 'arguments:' ['Bamboo']}]

5.1 Code Conventions

Jeder professionell produzierter Code muss syntaktischen Standards, sogenannten "Code Conventions", folgen. Diese haben keinen Einfluss auf die Funktion, erhöhen aber die Lesbarkeit, Wartbarkeit und Developer Experience. Jede Programmiersprache, jede Firma und jedes Team hat eigene Standards, VULCAN benutzt dabei den "Kernighan and Ritchie Indentation Style" aus dem Buch der gleichnamigen Autoren[30], auch bekannt als Egyptian Style. Dieser Style ist auch als Standard von Java definiert[31] und wird von Firmen wie Google LLC verwendet[32]. Es wird empfohlen, den "camelCase" als Variablen und Methodennamen zu verwenden.

[29] Quelle 7, Syriani, E., Luhunu, L., & Sahraoui, H. (2018). "Systematic mapping study of template-based code generation" p16
[30] Quelle 1, Brian W. Kernighan and Dennis M. Ritchie, "The C Programming Language"
[31] Quelle 20, Achut Reddy, "Java Coding Style Guide" p26
[32] Quelle 24, Alphabet "Google JavaScript Style Guide" 4.1.2 Nonemty blocks: K&R style

5.2 Abschließende Auswertung der Ergebnisse

Es ist nicht möglich, das in dieser Arbeit erstellte Programm direkt mit den Ergebnissen der bisherigen Forschung zu vergleichen, da diese Programme nicht öffentlich sind. Nach eigenen Angaben erreicht das fortgeschrittenste Programm, Img2UML, aber 95% Erfolg bei Rechtecken, 80% bei Beziehungen und 92% Korrektheit bei Texten.[33]

Basierend auf einem Jahrzehnt an Fortschritt erreicht schon OCR[34] von Tesseract über 99.2% Korrektheit bei Texten[35], was schon an sich ein immenser Erfolg ist.

Ähnliche Verbesserungen sind auch in anderen Kategorien zu sehen, auch wenn diese nicht so akkurat nachzuprüfen sind, da die bei Img2UML verwendeten Metriken und "Benchmarks" nicht veröffentlicht wurden.

VULCAN erreicht ca. 90% Korrektheit beim Erkennen von Beziehungen, diese (im Vergleich zu Img2UML) höhere Quote ist zwar ein Erfolg, lässt aber dennoch Raum für zukünftige Innovation und Verbesserungen. Da aber die händische Korrektur von falsch/nicht erkannten Beziehungen nur sehr wenig Zeit in Anspruch nimmt, sind Verbesserungen in dieser Hinsicht auf Kosten von Schnelligkeit des Programms nicht nötigerweise zielführend.

VULCAN erreicht des Weiteren bei korrekten Diagrammen 99% Korrektheit beim Erkennen von Klassen, diese Verbesserung ist vor allem dem verwendeten Ramer-Douglas-Peucker-Algorithmus zuzurechnen, welcher mit heutiger Rechenkapazität und modernen Verbesserungen, Polygon-Erkennung um ein vielfaches effizienter und akkurater gemacht hat.

Basierend auf diesen Fortschritten wird die Entwicklung von VULCAN somit mit zufriedenstellenden Ergebnissen abgeschlossen.

---

[33] Quelle 9, B. Karasneh and M. R. V. Chaudron, "Img2UML: A System for Extracting UML Models from Images" p136

[34] Quelle 18, R. Smith, "An Overview of the Tesseract OCR Engine"

[35] Quelle 25, Cem Dilmegani, "OCR in 2023: Benchmarking Text Extraction/Capture Accuracy"

6. Epilog

Künstliche Intelligenz wird immer besser darin, Code zu generieren und diese Arbeit hat gezeigt, dass vor allem einfacher, simpler und repetitiver Code vermutlich bald ausschließlich von KIs geschrieben werden wird.

Die erfolgreiche Entwicklung des eigenen Intelligent CASE-Tools VULCAN zeigt beispielhaft, dass auf dem Gebiet der Künstlichen Intelligenz in der Softwareentwicklung noch viele ungelöste Probleme und zu automatisierende Aufgaben existieren. Es ist zu erwarten, dass in naher Zukunft immer mehr Forschung auf diesem Gebiet passieren wird.

KIs werden immer besser werden, einfache Programmieraufgaben zu lösen oder wie VULCAN ganze "Code Bases" zu generieren. Sie werden aber vermutlich nicht komplexe, neue und kreative Lösungsansätze finden können. Kreativität und Innovation bleibt, was den Menschen abhebt und definiert.

Des Weiteren ist der Beruf der Softwareentwicklung vielfältiger und komplexer als das pure Programmieren von Code, es geht um die Entwicklung, das "Engineering" der Software, Künstliche Intelligenz kann nur auf Ideen und Strukturen von Menschen aufbauen, so braucht VULCAN ein von Menschen erstelltes Diagramm, das im Kern eine direkte Anleitung für den Code ist. Es ist aber dennoch der Mensch, der die Struktur vorgibt, der das UML-Diagramm zeichnet und der alles mit Kollegen und anderen Beteiligten kommuniziert.

Abschließend kann also festgestellt werden, dass Künstliche Intelligenz den Menschen trotz immenser Fortschritte nicht ersetzen wird, es heißt "Computer *Aided* Software Engineering" und das Ziel sollte immer sein, das Künstliche Intelligenz und Computer dem Menschen weiterhelfen, ihn effektiver machen, sodass er sich auf seine Stärken, seine Kreativität und Menschlichkeit fokussieren kann.

7. Glossar

Industrie 4.0	Die Industrie Phase der Digitalisierung, Modernisierung und künstlichen Intelligenz
CASE-Tools	Computer Aided Software Engineering Tools
HTML-Tags	In der Programmiersprache HTML verwendete Strukturindikatoren, die immer paarweise auftreten. Zum Beispiel: Text
Blank-Line-Intendierung	Programmiersprachen wie Python basieren auf Leerräumen als Vorgabe für die Semantik, zusammengehörige Code-Abschnitte müssen gleich intendiert sein
Regex Gleichungen	Regex ist ein einheitliches System für die semantische Formatierung von Zeichenketten.
Tokens	Neuronale Netze formatieren den puren Input in einzelne Teile (=Tokens), um den gesamten Kontext besser verstehen zu können.
Developer Experience	zu deutsch “Entwickler Erfahrung”, wie angenehm es für die entwickelnde Person ist, am Projekt zu arbeiten, beeinflussbar zum Beispiel durch gut benannte Methoden/Attribute.
Aggregation	Das Verhältnis zweier Klassen, bei der die aggregierende Klasse existentiell von der aggregierten Klasse abhängig ist.
Komposition	Das Verhältnis zweier Klassen, bei der die eine Klasse die andere Klasse ergänzt oder vervollständigt, nicht aber von ihr abhängig ist.
JSON	JavaScript Object Notation ist eine Konvention und ein Datentyp zur Speicherung von hierarchisch strukturierter Objekte.
CV2	Eine Open-Source Python Library zur Analyse von Bildern und visuellen Elementen.

8. Quellenverzeichnis

8.1 Buch Quellen

1) Kernighan, B.W. & Ritchie, D.M., 2006. The C programming language Link: https://venkivasamsetti.github.io/ebookworm.github.io/Books/cse/C%20Programming%20Language%20(2nd%20Edition).pdf [23.9.2023]
2) S. Krusche und B. Bruegge, "CSEPM - A Continuous Software Engineering Process Metamodel". 2017 IEEE/ACM 3rd International Workshop on Rapid Continuous Software Engineering (RCoSE), 1 (2017). Link: https://ase.in.tum.de/lehrstuhl_1/research/paper/krusche2017csepm.pdf [14.11.2022]

8.2 Wissenschaftliche Veröffentlichungen

3) A. T. Imam, A. J. Alnsour, und A. Al-Hroob, "The Definition of Intelligent Computer Aided Software Engineering (I-CASE) Tools".Journal of Information Engineering and Applications Vol.5, No.1 (2015). Link: https://deliverypdf.ssrn.com/delivery.php?ID=836003098074000078010102120085067081113072089033089044065118018103090064064017084096019037058000058022054080012009084066103067056045086013087028085026105076015027065021029028125095116069120102111096103124109094119094030095029007116065026127084093103115&EXT=pdf&INDEX=TRUE [14.11.2022]
4) M. Harman, "The role of Artificial Intelligence in Software Engineering," 2012 First International Workshop on Realizing AI Synergies in Software Engineering (RAISE), (2012). Link: http://www0.cs.ucl.ac.uk/staff/mharman/raise12.pdf [14.11.2022]
5) Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, et al. "PaLM: Scaling Language Modeling with Pathways". Google Research v. 5 (2022). Link: https://doi.org/10.48550/arXiv.2204.02311 [4.1.2023]
6) M. Barenkamp, J. Rebstadt, und O. Thomas, "Applications of AI in classical software engineering". AI Perspect 2, 1 (2020). Link: https://doi.org/10.1186/s42467-020-00005-4 [14.11.2022]
7) Syriani, E., Luhunu, L., & Sahraoui, H. (2018). “Systematic mapping study of template-based code generation”. Computer Languages, Systems & Structures. Link: https://arxiv.org/abs/1703.06353 [17.4.2023]

8) Shraddha Barke, Michael B. James, and Nadia Polikarpova. 2023. “Grounded Copilot: How Programmers Interact with Code-Generating Models”. Proc. ACM Program. Lang. 7, OOPSLA1, Article 78 (April 2023). Link: https://doi.org/10.1145/3586030 [27.4.2023]
9) B. Karasneh and M. R. V. Chaudron, "Img2UML: A System for Extracting UML Models from Images," 2013 39th Euromicro Conference on Software Engineering and Advanced Applications, Santander, Spain, 2013, pp. 134-137. Link: https://doi.org/10.1109/SEAA.2013.45 [27.4.2023]
10) X. Wang, L. Pollock and K. Vijay-Shanker, "Automatic Segmentation of Method Code into Meaningful Blocks to Improve Readability," 2011 18th Working Conference on Reverse Engineering, Limerick, Ireland, 2011, pp. 35-44, doi: 10.1109/WCRE.2011.15. Link: https://recipp.ipp.pt/bitstream/10400.22/13921/3/DM_ClaudioPinto_2018_MEI.pdf.txt [3.5.2023]
11) Yaroslav Golubev, Zarina Kurbatova, Eman Abdullah AlOmar, Timofey Bryksin, and Mohamed Wiem Mkaouer. 2021. “One thousand and one stories: a large-scale survey of software refactoring”. Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2021). Association for Computing Machinery, New York, NY, USA, 1303–1313. Link: https://doi.org/10.1145/3468264.3473924 [1.6.2023]
12) H .Touvron* , T .Lavril* , G. Izacard* , X. Martinet Marie-Anne Lachaux, T. Lacroix, B. Rozière, N. Goyal Eric Hambro, F. Azhar, A. Rodriguez, A. Joulin Edouard Grave* , G. Lample. “LLaMA: Open and Efficient Foundation Language Models,” Meta AI, Feb 2023. Link: https://arxiv.org/pdf/2302.13971.pdf [22.6.2023]
13) Javaid Mohd, Haleem Abid, Singh Ravi Pratap, Suman Rajiv “Artificial Intelligence Applications for Industry 4.0: A Literature-Based Study”, Journal of Industrial Integration and Management, March 2021. Link: https://www.worldscientific.com/doi/epdf/10.1142/S2424862221300040 [9.7.2023]
14) Peng, Sida, et al. "The impact of ai on developer productivity: Evidence from github copilot." arXiv preprint arXiv:2302.06590 (2023). Link: https://doi.org/10.48550/arXiv.2302.06590 [27.4.2023]
15) N. Nguyen und S. Nadi, "An Empirical Evaluation of GitHub Copilot’s Code Suggestions", MSR '22: Proceedings of the 19th International Conference on Mining Software Repositories. (2022). Link: https://dl.acm.org/doi/proceedings/10.1145/3524842 [9.7.2023]

16) J. Yousefi, “Image binarization using Otsu thresholding algorithm”. University of Guelph, 2011. Link: https://www.researchgate.net/profile/Jamileh-Yousefi-2/publication/277076039_Image_Binarization_using_Otsu_Thresholding_Algorithm/links/55609b1408ae8c0cab31ea42/Image-Binarization-using-Otsu-Thresholding-Algorithm.pdf [10.9.2023]
17) Wu, Shin-Ting; Marquez, Mercedes "A non-self-intersection Douglas-Peucker algorithm". *16th Brazilian Symposium on Computer Graphics and Image Processing (2003)*. Sao Carlos, Brazil Link: https://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.73.5773. [10.9.2023]
18) R. Smith, "An Overview of the Tesseract OCR Engine," *Ninth International Conference on Document Analysis and Recognition (ICDAR 2007)*, Curitiba, Brazil, 2007, pp. 629-633. Link: https://static.googleusercontent.com/media/research.google.com/de//pubs/archive/33418.pdf [10.9.2023]
19) David Douglas, Thomas Peucker: Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. In: The Canadian Cartographer. Band 10, Nr. 2, 1973, ISSN 0008-3127, S. 112–122. Link: http://www2.ipcku.kansai-u.ac.jp/~yasumuro/M_InfoMedia/paper/Douglas73.pdf [23.9.2023]
20) Achut Reddy “Java Coding Style Guide”, 1998 Server Management Tools Group Sun Microsystems, Inc. Link: https://web.archive.org/web/20060228095122/http://developers.sun.com/prodtech/cc/products/archive/whitepapers/java-style.pdf [23.9.2023]

8.3 Weitere Quellen

21) Stackoverflow 2023. Stackoverflow Survey 2023. Link: https://survey.stackoverflow.co/2023/#section-sentiment-and-usage-ai-tools-in-the-development-process [22.6.2023]
22) OMG Unified Modeling Language (OMG UML) Superstructure, Version 2.3: Mai 2010. link: https://www.omg.org/spec/UML/2.3/Superstructure/PDF/ [10.9.2023]
23) S. Krusche, "Einführung in die Softwaretechnik" LS 1 Vorlesung TUM, Sommersemester 2022, [10.9.2023]

24) Alphabet “Google JavaScript Style Guide” Link: https://google.github.io/styleguide/jsguide.html [23.9.2023]
25) Cem Dilmegani “OCR in 2023: Benchmarking Text Extraction/Capture Accuracy” 2023 Link: https://research.aimultiple.com/ocr-accuracy/ [23.9.2023]

9. Anhang

Der Quellcode des entwickelten Programms VULCAN (Visual Uml Link and Class Analyzing Neural-network) zur Konvertierung von UML-Diagrammen hin zu Java-Code kann dem physisch beigelegten Datenträger im Anhang entnommen werden.

10. Erklärung

Ich versichere, dass ich die vorliegende Arbeit über das Thema “KI in agiler Softwareentwicklung” in der gesetzten Frist selbständig verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe.

Alle Stellen der Arbeit, die anderen Werken wörtlich oder sinngemäß entnommen sind, sind unter Angaben der Quelle als Entlehnung kenntlich gemacht.

Die bildlichen Darstellungen und Statistiken sind von mir verfasst, soweit nicht anders als Entlehnung gekennzeichnet.