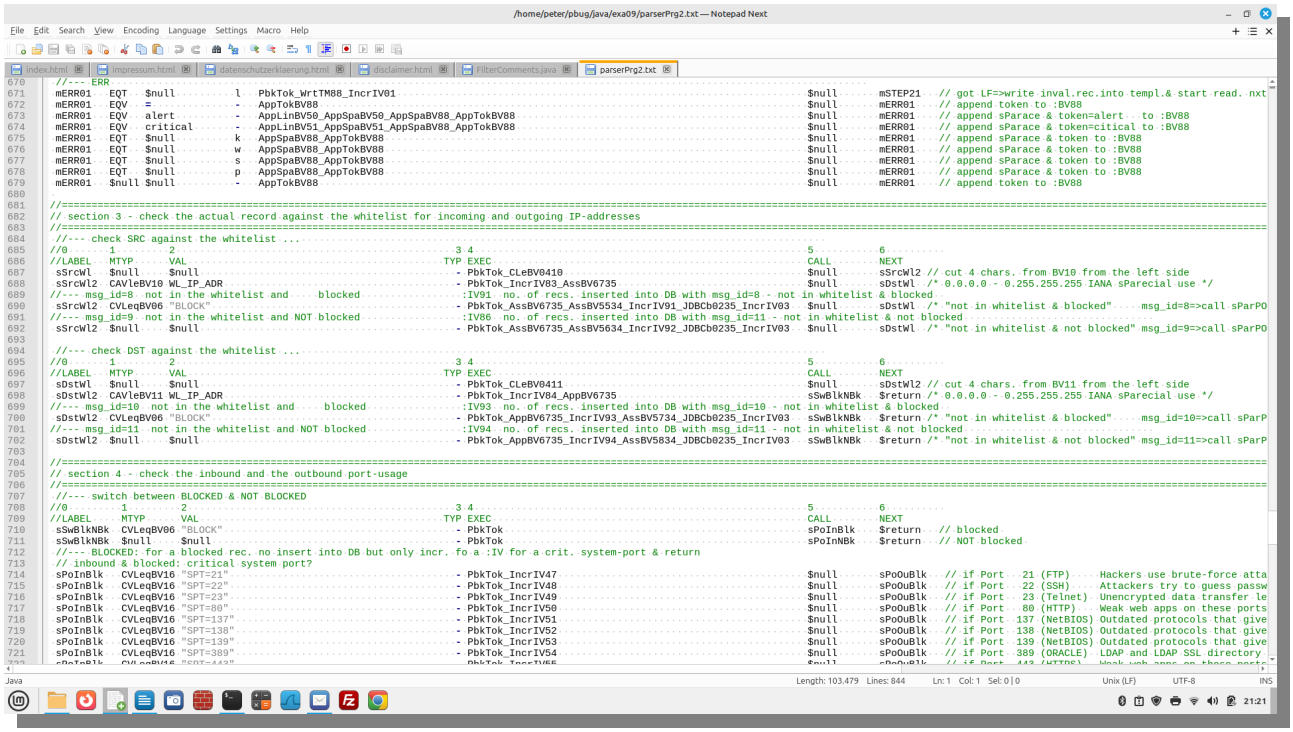
Table of Contents

Softwareentwicklung mithilfe von KI....2
Training der KI und Fragen zum geistigen Eigentum....2
Form des erzeugten Codes....2
Codequalität des erzeugten Codes....3
Wartbarkeit des erzeugten Codes....3
Ein einfaches Beispiel für den Entwurf einer Softwarearchitektur....3

Softwareentwicklung mithilfe von KI



Training der KI und Fragen zum geistigen Eigentum

Generative KI-Tools erstellen eigenständig Code, der auf Mustern aus großen Datenmengen gewonnen wird. Damit werden generative KI-Modelle mit öffentlich verfügbarem und proprietärem Code trainiert. Es besteht die Gefahr, dass Code mit unbekannter Herkunft verwendet wird und Eigentumsrechte verletzt werden. Die Einhaltung von Rechten des geistigen Eigentums und von Lizenzen ist jedoch ein wichtiger Aspekt. Viele generative KI-Tools behalten sich das Recht vor, mit dem vom Nutzer bereitgestellten Daten zu trainieren. Es besteht das Risiko, dass Code, der einer Open Source Lizenz unterliegt, öffentlich zugänglich gemacht wird. Dies kann zur Verletzung von gesetzliche Vorschriften und von unternehmensinternen Richtlinien führen. Daher ist Aufsicht und Analyse erforderlich, um den generierten Code zu überprüfen.

Form des erzeugten Codes

Die Form des erzeugten Codes richtet sich eher nach den Trainigsdaten und nicht nach den Style Guides des Unternehmens, das generative KI-Tools einsetz.
Coding Style Guides machen Vorgaben zu

- Namensgebung für benutzerdefinierte Bezeichner, erlaubte Sprachen, Präfixe und Suffixe, Singular/Plural, Vorgaben zur Groß-/Kleinschreibung und zur Lesbarkeit der Bezeichner,

- Groß-/Kleinschreibung von Schlüsslewörtern,
- Vorgaben zum Geltungsbereich von Variablen (global,lokal)
- Formatierung, Einrückung und die Verwendung von Tabulatoren (damit der Code immer gleich aussieht), Leerzeichen zwischen Operatoren und Variablen, Zeilenlängen
- Modularisierung zur Einhaltung von Komplexitätsmassen und
- Richtlinien zu Kommentaren im Code, um daraus auch ggf. automatisiert eine technische Dokumentation zu gewinnen.

Die Anpassung von erzeugtem Code an solche Vorgaben ist sehr zeitaufwendig und kann zu Fehler führen.

Codequalität des erzeugten Codes

Ggf. ist nicht klar erkennbar, dass die erzeugte Softwarearchitektur erprobt, robust und performant ist. Der von KI-Tools erzeugte Code entspricht möglicherweise nicht immer den Qualitätsstandards eines Unternehmens. Generative KI-Tools basieren auf Mustern und Vorgehensweisen, die sie aus Trainingsdaten entnommen haben. Falls die Trainingsdaten nicht auf ihre Qualität überprüft worden sind, kann dies zu qualitativ minderwertigem oder ineffizientem Code führen. Minderwertiger oder ineffizientem Code führen zu Fehler, Sicherheitslücken und zu einer miderwertigen Softwarearchitektur.
Falls ein erzeugter Code eine spezielle Sicherheitslücke aufweist, kann man leicht auf die Quelle schließen, die zum Training des generativen KI-Tools verwendet worden ist.
Dann können zum Schaden, der durch die Sicherheitslücke entstanden ist noch eine Anklage wegen der Verletzung von Rechten am geistigen Eigentums des Codes führen.

Wartbarkeit des erzeugten Codes

Die Wartbarkeit des erzeugtem Code kann komplex und anspruchsvoll sein. Selbst wenn die Software gemäss der Spezifikation funktioniert, kann es schwierig sein, die zugrunde liegende Logik nachzuvollziehen oder zu verstehen. Mangelnde Transparenz kann dazu führen, dass der erzeugten Code nicht gewartet werden kann. Ein Refactoring oder die Fehlersuche (Debugging) ist dann nicht möglich, falls Fehler auftreten. Der Versuch, mangelnde Erfahrung von Softwareentwicklern durch KI auszugleichen, führt zu erheblichen Risiken für Unternehmen. Die Einhaltung von Prinzipien zu sauberem Code („Clean Code“) führt zu zufriedenen Mitarbeitern und zu einer niedrigen Fluktuationsrate im IT-Bereich. Der Verlust von erfahrenen Mitarbeitern ist für Unternehmen oft mit hohen Kosten und Risiken verbunden.

Ein einfaches Beispiel für den Entwurf einer Softwarearchitektur

Wie oben bereits beschrieben ist Softwareentwicklung mittels KI im Grunde Softwareentwicklung unter Einsatz von Codegeneratoren – ein Ansatz, der mit erheblichen Risiken verbunden ist. Es besteht die Gefahr, dass das System Codefragmente erzeugt, die nicht vollständig verstanden werden, wodurch es unmöglich wird, die Korrektheit des resultierenden Codes zu überprüfen. Zudem besteht das Risiko, dass die generierte

Softwarearchitektur eine Komplexität erreicht, die ihre Wartung erschwert oder unmöglich macht. In der Softwareentwicklung wird ein solcher Code auch als „Spaghetti-Code“ bezeichnet. Mit diesem Artikel möchte ich Ratschläge geben, wie sich Spaghetti-Code vermeiden lässt.

Softwarearchitekturen variieren je nach Anwendungsbereich:

- Echtzeitsysteme, etwa operative Dashboards und IoT-Anwendungen, Messaging- und Chat-Plattformen (wie WhatsApp) oder Handelsplattformen im Finanzsektor (wie Robinhood).
- Stapelverarbeitung (Batch Processing), beispielsweise bei der Lohn- und Gehaltsabrechnung, dem Bankabgleich und Tagesabschlussverfahren, der Abrechnung von Versorgungsleistungen oder Kreditkartentransaktionen sowie bei ETL-Prozessen (Extract, Transform, Load) und Data Warehousing.

Im Folgenden wird ein Ansatz für die Stapelverarbeitung beschrieben. Mithilfe der „Jackson Structured Programming“-Methodik lassen sich die einzelnen Komponenten identifizieren, aus denen das zu entwerfende System bestehen soll. (Einen ausführlichen Beitrag zu diesem Thema finden Sie auf Wikipedia.) Jede Komponente kann oft als Arbeitsschritt betrachtet werden, die mehrere technische Funktion umfasst, die darauf ausgelegt sind, jeweils genau eine spezifische technische Aufgabe zu erfüllen.

```
71  //-------------------------------------------------------------------------------------------------------------
72  // returning a write-command for the act. and the prev.char
73  //-------------------------------------------------------------------------------------------------------------
74  final private static int automat22() {
75      /*
76      | / | * | - | # | lf | ' | " | so |
77      +---+---+---+---+---+---+---+---+
78      1 | 2n | 1a | 5n | 6n | 1a | 7n | 8n | 1a | / - # ' " (unmaskiert) Statusänderung
79      +---+---+---+---+---+---+---+----------+
80      2 |6n/2v| 3n | 5v | 6v | 1z | 7v | 8v | 1z | / * - # ' " (unmaskiert) Statusänderung - // Einzeilenkommentar=6n c
81      +---+---+---+---+---+---+---+----------+
82      3 | 3n | 4n | 3n | 3n | 3n | 3n | 3n | 3n | * Statusänderung - Mehrzeilenkommentar ein
83      +---+---+---+---+---+---+---+----------+
84      4 | 1n | 4n | 3n | 3n | 3n | 3n | 3n | 3n | / Statusänderung - Mehrzeilenkommentar aus
85      +---+---+---+---+---+---+---+----------+
86      5 | 2v | 1z | 6n | 6v | 1z | 7v | 8v | 1z | / - # ' " (unmaskiert) Statusänderung - Einzeilenkommentar mit Start
87      +---+---+---+---+---+---+---+----------+
88      6 | 6n | 6n | 6n | 6n | 1n | 6n | 6n | 6n | lf Statusänderung - Einzeilenkommentar aus ?
89      +---+---+---+---+---+---+---+----------+
90      7 | 7n/a| 7n/a| 7n/a| 7n/a| 7n/a| 7n/a| 7n/a| 7n/a| ' aus ?
91      +---+---+---+---+---+---+---+----------+                filDouSla   // double slash
92      8 | 8n/a| 8n/a| 8n/a| 8n/a| 8n/a| 8n/a| 8n/a| 8n/a| " aus ?        filDouHyp   -- double hyphen
93      +---+---+---+---+---+---+---+----------+                filNumSgn   # number sign
94      | / | * | - | # | lf | ' | " | so |                      filSinQuo   ' single quotation
95      n nichts schreiben                                      filDouQuo   " double quotation
96      v vorheriges Zeichen schreiben                          mask_q    b/d mask single quote by backslash
97      a aktuelles Zeichen schreiben                           mask_Q    b/d double-quote by doubling
98      z vorheriges und aktuelles Zeichen schreiben
99      */
```

Ein endlicher Automat (oder kurz: „Zustandsautomat“) eignet sich hervorragend für diesen Zweck. (Auch Wikipedia bietet einen ausführlichen Artikel zu diesem Thema.) Indem der aktuelle Zustand auf einer Festplatte oder in einer Datenbank gespeichert wird, kann das System nach einem Ausfall neu gestartet werden; nach Behebung des Fehlers lässt sich die Verarbeitung genau an dem Punkt fortsetzen, an dem sie zuvor unterbrochen wurde. Diese Funktionalität ist sehr zu empfehlen. Dieser Ansatz führt zu einer robusten Softwarearchitektur. Sobald die Struktur steht, können Sie die einzelnen technischen Funktionen von einem KI-Agenten generieren lassen – es sei denn natürlich, du verfügst bereits über bewährte, zuverlässige Module in deiner eigenen Softwarebibliothek.

Es empfiehlt sich, für jede Verarbeitungsschicht einen Zustandsautomaten zu entwerfen. Jeder Schicht sollte eine eindeutige Kennung (oder Abkürzung) zugewiesen werden, die in

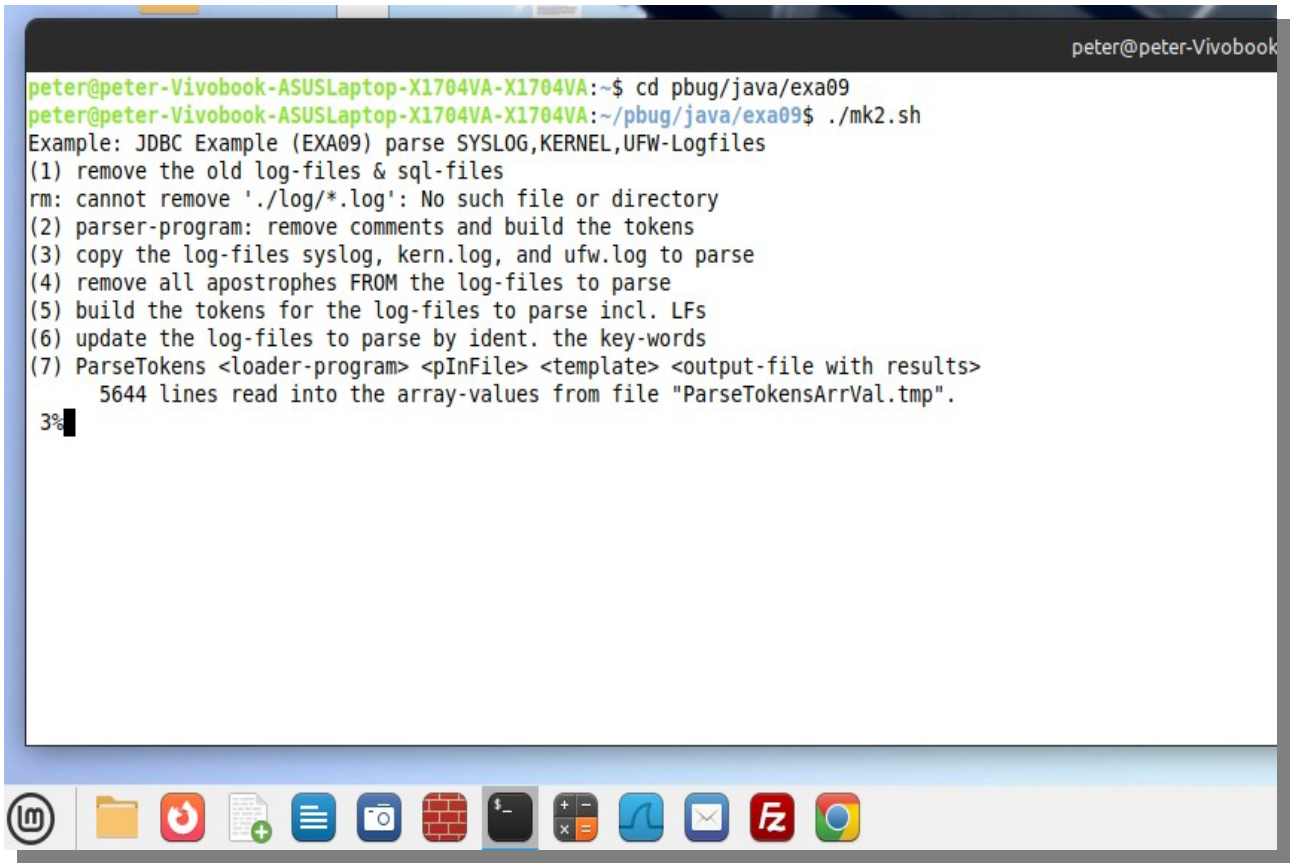
jedem Protokolleintrag und jeder Fehlermeldung enthalten ist. Zudem sollte jede Meldung eine im gesamten System eindeutige Meldungsnummer erhalten. Dies ermöglicht es, die exakte Stelle im Code zu identifizieren, die eine bestimmte Meldung ausgelöst hat; führt ein Fehler darüber hinaus zum Programmabbruch, lässt sich zudem der Aufrufstapel (Call Stack) einsehen.

Das hier vorgestellte Framework zur Textanalyse ist wie folgt strukturiert:
1. Filterung von Kommentaren
2. Identifizierung von Tokens
3. Identifizierung von Schlüsselwörtern
4. Parsing der Tokens mithilfe einer Skriptsprache
Jede Ebene umfasst ein Programm, das von einem Zustandsautomaten gesteuert wird. Der Parser arbeitet auf der Grundlage eines Zustandsautomaten, der einen Stack (Stapelspeicher) einbezieht, was das Parsen verschachtelter SQL-Anweisungen ermöglicht.

```
703
704  //=============================================================================================================================================================
705  // section 4 - check the inbound and the outbound port-usage
706  //=============================================================================================================================================================
707   //--- switch between BLOCKED & NOT BLOCKED
708  //0       1         2                                                  3 4                                                  5           6
709  //LABEL     MTYP      VAL                                            TYP EXEC                                             CALL        NEXT
710   sSwBlkNBk  CVLeqBV06 "BLOCK"                                          - PbkTok                                           sPoInBlk    $return   // blocked
711   sSwBlkNBk  $null     $null                                            - PbkTok                                           sPoInNBk    $return   // NOT blocked
712   //--- BLOCKED: for a blocked rec, no insert into DB but only incr. fo a :IV for a crit. system-port & return
713   // inbound & blocked: critical system port?
714   sPoInBlk   CVLeqBV16 "SPT=21"                                         - PbkTok_IncrIV47                                  $null       sPoOuBlk  // if Port  21 (FTP)
715   sPoInBlk   CVLeqBV16 "SPT=22"                                         - PbkTok_IncrIV48                                  $null       sPoOuBlk  // if Port  22 (SSH)
716   sPoInBlk   CVLeqBV16 "SPT=23"                                         - PbkTok_IncrIV49                                  $null       sPoOuBlk  // if Port  23 (Telnet)
```

Definiert man die notwendigen Komponenten einer 3GL (Sprache der 3. Generation) als
• Ausdrücke (oder Sequenzen),
• Bedingungen,
• Schleifen sowie
• Unterprogrammaufrufe,
dann lässt sich die Skriptsprache des Parsers als eine einfache 3GL klassifizieren. Diese Sprache ermöglicht die Erstellung maßgeschneiderter Parser.

Fazit: Durch den Entwurf einer geeigneten Softwarearchitektur und die Beschränkung des KI-Einsatzes auf die Generierung von Code für präzise definierte technische Funktionen sollten die mit der Nutzung dieser Technologie verbundenen Risiken überschaubar bleiben.



Ein Beispiel:

Ein endlicher Automat mit einem Stack – in der Theorie auch als Kellerautomat (englisch: Pushdown Automaton, PDA) bezeichnet – speichert den Aufruf-Stack (Call-Stack).

Ein Beispiel für einen Kellerautomaten findet sich im TAFr-Handbuch, im Kapitel „The Parser Program“ unter „Introduction“:

- **Ein Ereignis ist ein Befehl aus der Arbeitsliste, der den Aufruf einer Anwendung oder eines Arbeitsschritts (einer Funktion in einer 3GL-Anwendung) darstellt:**

event / state	event-1	event-2	event-3	event-4
1	START	START		
2	STP00			
3	STP01	STP01	STP01	
4	STP02	STP02	STP02	
5	STP03	STP03	STP03	
.....				

- **Ein Aufruf (grün) springt an eine andere Stelle in der aktuellen oder einer anderen Übergangstabelle. Um an die aufrufende Stelle zurückzukehren, muss ein RETURN-Befehl ausgeführt werden:**

state \ event	event-1	event-2	event-3	event-4
1				
2	STP02	STP02 ERR01	STP02 ERR01	
3				
4				
5				

```
Layers          Technical Functions
Pre-processing  : check of the work list
                  check of the hard drive storage space
                  check of the condition of the database
                  etc.
Input           : read the next task from the work list
Process         : process the actual task
Output          : write the output into the database
                  write the data into the store on the hard disk
                  etc.
Post-processing : Write log-messages into the log-file,
                  process clean up (delete temp. file)
                  etc.
```

Das finale Design überlasse ich nun dem interessierten Leser.