

Künstliche Intelligenz als helfende Hand im Test

Kann KI bei funktionalen Systemtests den Tester unterstützen?

KI-Assistenten unterstützen die Softwareentwicklung schon an vielen Stellen. Wir wollen wissen, ob KI auch beim Design von Testfällen im Systemtest hilfreich sein kann, wo ihre Grenzen liegen und wie man den Prozess optimieren kann. Dazu befragen wir eine generative KI anhand einer Beispielanwendung, bewerten die Ergebnisse und untersuchen die Einschränkungen.



Die Qualitätssicherung von Software steht vor stetig wachsenden Herausforderungen angesichts der zunehmenden Komplexität moderner Anwendungen. Einer der grundlegenden Aspekte dieser Qualitätssicherung ist das Testen von Software, um sicherzustellen, dass die Software den Anforderungen und Erwartungen der Benutzer gerecht wird. Innerhalb des Tests nimmt das Testfalldesign eine zentrale Rolle ein, da es den Prozess steuert, wie

verschiedene Szenarien und Funktionalitäten der Software überprüft werden. Testfälle repräsentieren dabei konkrete Szenarien oder Abläufe, die die Software während des Testens durchlaufen soll. Sie bestehen aus einer präzisen Beschreibung der auszuführenden Schritte sowie den erwarteten Ergebnissen.

Die Erstellung von Testfällen erfolgt üblicherweise durch Testanalysten oder Entwickler mit der Rolle Test. Bei dem Pro-

zess fließen die Kenntnis des Systems unter Test (SUT) und der Rahmenbedingungen ebenso ein wie die fachliche Expertise und Erfahrung des Testers.

Die Testfälle und ihre Erstellung unterscheiden sich bei den unterschiedlichen Teststufen: So werden Komponententests (White-Box-Tests) eher an der Implementierung ausgerichtet, während Systemtests sich an der Spezifikation und der Benutzersicht orientieren. Die Anzahl und der Aufwand für die unterschiedlichen Testarten werden oftmals mithilfe der Testpyramide (siehe Abbildung 1), die auf Mike Cohn zurückgeht, dargestellt. Wir sehen in der Darstellung, dass Systemtestfälle von der Anzahl her am wenigsten vertreten sind, aber den höchsten Aufwand erfordern. Aus diesem Grund betrachten wir gerade diese Testart, um Arbeiten durch KI zu unterstützen.

In diesem Artikel liegt der Fokus auf der Anwendung von Künstlicher Intelligenz (KI) im Kontext des Systemtests. Insbesondere wird untersucht, inwieweit KI-gesteuerte Ansätze das Testfalldesign und die Testdatenerstellung optimieren können.

Wieso sollte man KI im Testen einsetzen?

In den letzten Monaten ist die Hürde zum Einsatz von KI-Assistenten in der täglichen Arbeit durch deren einfache Zugänglichkeit deutlich gesunken. Freie Angebote wie OpenAI ChatGPT (vgl. [OpenAI]) oder Coding-Assistenten wie Github Copilot (vgl. [CoPilot]) liefern im Rahmen ihrer Fähigkeiten erstaunlich gute Ergebnisse. Im Bereich des Software-Developments haben sich damit ganz neue Möglichkeiten für Entwickler eröffnet, die zu einer deutlichen Zeitersparnis führen können. Aber kann auch der Software-Tester davon profitieren und zeitaufwendige Testdesign-Aufgaben teilweise oder ganz an KI-Tools auslagern? Wie gut sind die Antworten der KI und welche Fragen müssen wir überhaupt stellen?

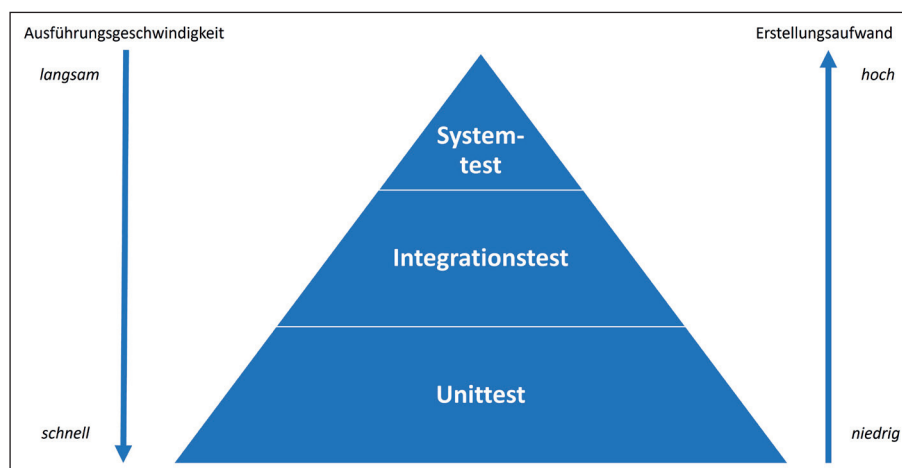


Abb. 1: Testpyramide (vgl. [Coh09])

Um eine gute Abdeckung mit funktionalen Testfällen in dieser Teststufe zu erreichen, ist eine detaillierte Kenntnis der Anforderungen nötig. Die Beschreibung der Testfälle in Textform ist oft repetitiv und durch damit nachlassende Konzentration fehleranfällig. Hier sehen wir eine gute Möglichkeit, durch den Einsatz von KI-Tools eine Erleichterung herbeizuführen. Wir wollen anhand einer kleinen Beispielanwendung herausfinden, welche Daten wir der KI zur Verfügung stellen und wie wir diese Anfragen formulieren müssen, damit die erzeugten Antworten möglichst direkt nutzbar sind.

Als KI-Assistenten haben wir uns die frei verfügbare Version ChatGPT ausgesucht, die auf GPT-3.5 basiert. Wir haben auch einige Funktionen von GPT-4 getestet, hier ist insbesondere die Möglichkeit hervorzuheben, mithilfe von Plug-ins auf aktuelle Webinhalte zuzugreifen und spezielle Ausgabeformate anzubieten.

Grundlagen Generativer KI

Bevor wir uns der Beispielanwendung zuwenden, ein kleiner Exkurs zum Thema Generative KI. Die von uns genutzten Modelle GPT-3.5 und GPT-4 sind sogenannte Large Language Models (LLM). Ein LLM ist ein Sprachmodell mit einer sehr großen Datenbasis. Es ist darauf trainiert, Texte in natürlicher Sprache zu verstehen und zu erzeugen. Dadurch kann es genutzt werden, Texte zu generieren, Fragen zu beantworten, Übersetzungen anzufertigen oder sogar Programmcode zu verstehen und neuen zu erzeugen (vgl. [Wiki1]). GPT steht für „Generative Pre-Trained Transformer“ und bezeichnet die Art des Modells, nämlich ein generatives (erzeugt Texte, Bilder, Code), vortrainiertes neuronales Netz, welches auf der Transformer-Architektur basiert (vgl. [Wiki2]). Essenziell für den Erfolg solcher Modelle ist die Datenbasis, auf der sie trainiert wurden. Eine übliche Quelle für das Training ist das Internet, in dem nicht nur eine große Menge an Textmaterial zu finden ist, sondern auch frei verfügbare Sammlungen von Quellcode in vielen unterschiedlichen Programmiersprachen. Die KI gewichtet die Daten aus den Quellen mit verschiedenen Parametern und „lernt“ dadurch, Anfragen durch Antworttexte zu ergänzen, die auf ihren Quelldaten basieren. Bei großen Modellen wie dem von uns genutzten GPT-3.5 von OpenAI umfasst das Quellmaterial Hunderte Gigabytes Text, die durch diverse Trainingsmethoden in etwa 175 Milliarden Parametern resultieren (vgl. [Arx1]).

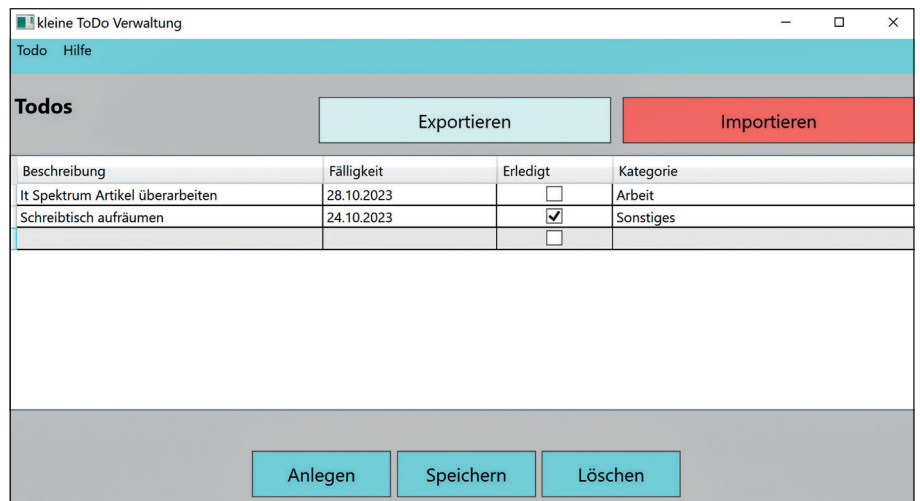


Abb. 2: Hauptseite der Anwendung



Abb. 3: Seite zum Anlegen einer neuen Aufgabe

In den letzten Jahren hat die Forschung zu solchen Modellen große Fortschritte gemacht. So hat die Qualität der Modelle inzwischen ein Niveau erreicht, welches sie auch für alltägliche Aufgaben nützlich macht. Man muss sich aber immer vor Augen halten, dass LLMs nicht wirklich verstehen, was sie gefragt werden. Aus ihrer Funktionsweise resultiert, dass ihre Antworten ein Produkt der Trainingsdaten und der angewandten Algorithmen sind, die die Antworten daraus erzeugen.

Unsere Beispielanwendung

Als exemplarische Anwendung, anhand derer wir die Versuche durchführen, dient ein schlanker Aufgaben-Planer. Er zeigt angelegte Aufgaben in Form einer Liste an. Technologisch handelt es sich um eine C#-Anwendung mit einer WPF-Oberfläche. Die Datenhaltung ist durch die Ablage der Datensätze in Form von Dateien realisiert. Die Aufgaben können mithilfe der Schaltflächen exportiert oder importiert werden. Die importierten Auf-

gaben erscheinen, wie die neu erstellten, in Listenform im Hauptfenster (siehe Abbildung 2) der Anwendung. Neben dem Hauptfenster verfügt die Anwendung noch über ein Fenster, mit dessen Hilfe man die neuen Aufgaben anlegen kann (siehe Abbildung 3).

Wir betrachten hier den Workflow zur Anlage neuer Aufgaben als Testumfang. Damit der Test im Bereich Testdaten einige typische Herausforderungen beinhaltet, wurden zwei Pflichtfelder (Beschreibung, Kategorie) definiert. Eine fehlende Eingabe erzeugt hier eine Fehlermeldung, die das Speichern unmöglich macht. Das Fälligkeitsdatum erzeugt bei Daten in der Vergangenheit eine Warnung, nach Bestätigung kann es aber bei der Anlage übernommen werden.

Eingaben

Wie oben beschrieben, erzeugen LLMs ihre Antworten als Ergänzung der vorgelegten Anfrage. Die Formulierung dieser Anfrage ist daher essenziell für das Resultat.

Ergänzung der Anfrage mit Template

„Bitte beachte dabei folgendes Template:

- Testfallnummer: eine eindeutige Nummerierung.
- Titel: Das ist die Kurzbeschreibung, sodass der Tester in einem Satz sieht, um was es in dem Testfall geht.
- Zu testende Funktion: Welche Funktion der Anwendung wird getestet.
- Priorität: Wie hoch ist die Priorität des Testfalls in Hinblick auf seine Ausführung für die zu testende Funktion beziehungsweise den zu testenden Use-Case.
- Vorbedingungen: In diesem Feld werden die Vorbedingungen erfasst.
- Testschritte: eine genaue Beschreibung der während des Tests durchzuführenden Schritte. Je nach Kenntnisstand des späteren Testers müssen Sie hier genauer werden oder können auch nur Stichworte definieren.
- Erwartete Ergebnisse: Eine Beschreibung dessen, was Sie als gültiges Ergebnis betrachten, also wie ein Testdurchlauf aussieht, wenn das Ergebnis korrekt ist.
- Testdaten: Welche Testdaten werden für welche Eingaben benötigt?
- Abhängige Tests: Optional kann hier definiert werden, welche anderen abhängigen Tests es zum Testfall gibt.“

Kasten 1

Ergänzung der Anfrage mit Details zur Maske

„Bitte beachte, dass der Titel des neuen Fensters ‚neuesToDo‘ sein sollte, dass es dort die Felder ‚Beschreibung‘, ‚Fälligkeit‘, eine Checkbox ‚Erledigt‘ und ein Auswahlfeld ‚Kategorie‘ gibt. Bitte erzeuge Testfälle für positive und für negative Fälle (alle Fälle, in denen eine Warn- bzw. Fehlermeldung erscheint).

Die Testdaten kannst Du mithilfe von Äquivalenzklassen angeben. Folgende Felder stehen in der Kategorie zur Auswahl: ‚Sport‘, ‚Arbeit‘, ‚Einkauf‘, ‚Sonstiges‘. Bei ‚Erledigt‘ kann der Wert ‚ja‘ oder ‚nein‘ sein. Die Beschreibung ist ein Freitextfeld. Bei dem Datum kannst Du mithilfe eines Datumpickers wählen. In dem Hauptfenster sollte die Überprüfung der Daten erfolgen. Bitte berücksichtige nur den Use-Case ‚Neue ToDo-Aufgabe hinzufügen‘.“

Kasten 2

Um der KI die erforderlichen Informationen zur Erstellung von Testfällen zur Verfügung zu stellen, gibt es mehrere Möglichkeiten. Wir möchten mit unterschiedlichen Artefakten als Eingabeparameter für die KI testen, wie gut das Testfalldesign und die Testdatenerstellung mit ihrer Hilfe funktioniert. Dazu nutzen wir eine textuelle Spezifikation der Anwendung in Form eines einfachen Benutzerhandbuchs, Aktivitätsmodelle im UML-Format und Teile des Quellcodes der Anwendung.

Wir haben die Erfahrung gemacht, dass die Ergebnisse deutlich nachvollziehbarer werden, wenn man das Ausgabeformat in Form eines Templates vorgibt. Daher stellen wir den Textblock aus **Kasten 1** den Anfragen voran. Damit erreicht man, dass alle erzeugten Testfälle die gleiche Struktur haben. Sie können dann leichter kontrolliert und verglichen werden.

Eingabeform: Natürliche Sprache

Ein großer Vorteil von Implementierungen wie ChatGPT ist ihre Möglichkeit, die natürliche Sprache zu verstehen. Im

Rahmen der Testfallerstellung können wir diese Fähigkeit nutzen und dem Modell eine Beschreibung der Anwendung in Form eines Benutzerhandbuchs vorlegen. Bei unserem Beispiel haben wir ein kurzes Handbuch erstellt und dieses mit den Anweisungen zum Template (**siehe Kasten 1**) übergeben. Vor das Manual haben wir noch einen Text gesetzt, der der KI Hinweise gibt, wie die Eingabemaske aussieht, da dies aus dem Handbuch nicht ausreichend hervorgeht (**siehe Kasten 2**).

Auf dieser Basis erzeugte ChatGPT bei verschiedenen Läufen entweder fünf oder sechs Testfälle. Hier ein Beispiel eines Positiv-Testfalls:

„Testfall 1: Neue ToDo-Aufgabe erfolgreich hinzufügen

- Testfallnummer: 001
- Titel: Neue ToDo-Aufgabe erfolgreich hinzufügen
- Zu testende Funktion: Neue ToDo-Aufgabe hinzufügen
- Priorität: Hoch
- Vorbedingungen: Die Anwendung ist geöffnet

■ Testschritte:

1. Klicken Sie auf die Schaltfläche ‚Anlegen‘ im Hauptfenster.
 2. Geben Sie eine kurze Beschreibung in das Textfeld ‚Beschreibung‘ ein.
 3. Wählen Sie ein Fälligkeitsdatum mithilfe des Datumsauswahl-Widgets aus.
 4. Wählen Sie eine Kategorie aus der Dropdown-Liste.
 5. Setzen Sie bei Bedarf ein Häkchen im Kontrollkästchen ‚Erledigt‘.
 6. Klicken Sie auf ‚Speichern‘.
- Erwartete Ergebnisse: Die neue ToDo-Aufgabe wird der Liste hinzugefügt und in der Tabelle angezeigt.“

Auch die gewünschten Negativ-Testfälle wurden erzeugt, hier ein Beispiel:

„Testfall 3: Neue ToDo-Aufgabe ohne Beschreibung hinzufügen

- Testfallnummer: 003
- Titel: Neue ToDo-Aufgabe ohne Beschreibung hinzufügen
- Zu testende Funktion: Neue ToDo-Aufgabe hinzufügen
- Priorität: Hoch
- Vorbedingungen: Die Anwendung ist geöffnet
- Testschritte:
 1. Klicken Sie auf die Schaltfläche ‚Anlegen‘ im Hauptfenster.
 2. Lassen Sie das Textfeld ‚Beschreibung‘ leer.
 3. Wählen Sie ein Fälligkeitsdatum mithilfe des Datumsauswahl-Widgets aus.
 4. Wählen Sie eine Kategorie aus der Dropdown-Liste.
 5. Klicken Sie auf ‚Speichern‘.
- Erwartete Ergebnisse: Eine Warnmeldung wird angezeigt, die den Benutzer auffordert, eine Beschreibung einzugeben.“

Am Ende der Liste wies die KI auf die Testdaten hin, während sie selber in den Testfällen eine konkrete Angabe von Testdaten vermied:

„Diese Testfälle decken verschiedene Szenarien ab und sollen sicherstellen, dass die ToDo-Verwaltungsanwendung ordnungsgemäß funktioniert und Benutzerhinweisen korrekt folgt. Denken Sie daran, Testdaten aus verschiedenen Äquivalenzklassen zu verwenden, um die Abdeckung zu maximieren.“

Interessant war, dass ein Negativ-Testfall mit dem Titel „ToDo-Aufgabe mit ungültigem Fälligkeitsdatum hinzufügen“ erzeugt wurde, in dem der Tester ein Fälligkeitsdatum aus der Vergangenheit ein-

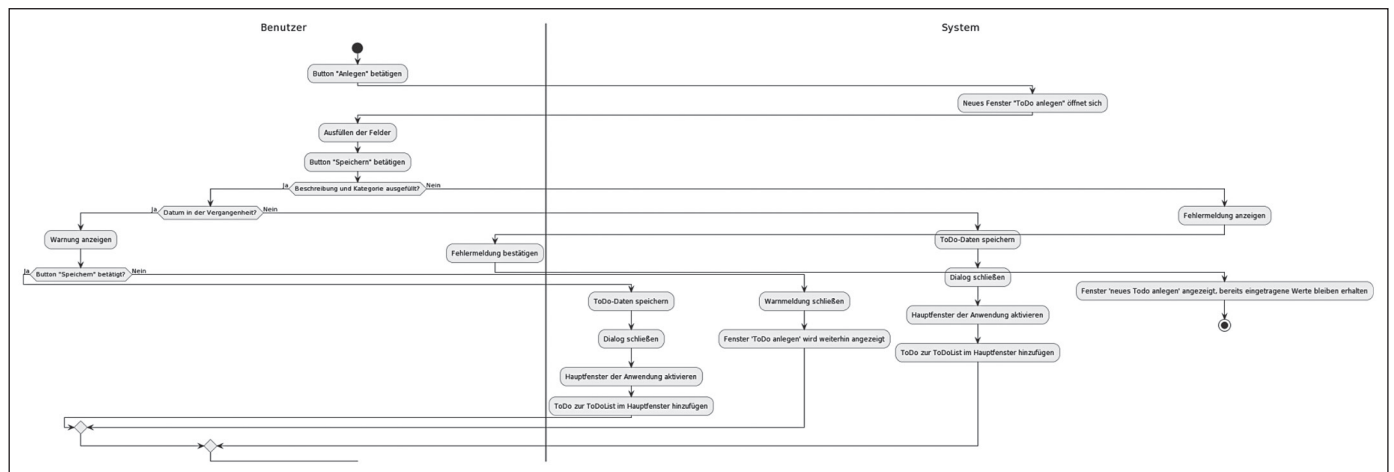


Abb. 4: Aktivitätsdiagramm „neue Aufgabe“ anlegen

geben sollte. Das erwartete Ergebnis war eine Warnmeldung, dass dies nicht gültig sei. Eine solche Beschränkung wurde im Benutzerhandbuch, welches als Eingabe diente, aber an keiner Stelle erwähnt, die KI hat also von selbst angenommen, dass ein solches Datum in der Zukunft liegen muss.

Bewertung:

- Die Testfälle sind größtenteils plausibel und decken die gewünschten Funktionalitäten ab.
- Sequenzielle Läufe in neuen Sitzungen erzeugen verschiedene Ergebnisse, sowohl in der Menge der generierten Testfälle als auch in der Ausprägung.
- Konkrete Testdaten, die über die genannten Werte in den Vorgaben hinausgehen, gibt die KI nicht an.
- Auch wenn die Testfälle plausibel aussehen, sollte man sich nicht blind auf sie verlassen, siehe dazu das Beispiel mit dem Fälligkeitsdatum.

Ein wichtiges Kriterium, welches insbesondere bei langen Anfragen in natürlicher Sprache zum Tragen kommt, wie zum Beispiel bei Benutzerhandbüchern, ist die resultierende Kontextlänge der Anfrage.

Sprachmodelle wie ChatGPT zerlegen die Anfrage in sogenannte Token. Diese sind die kleinsten Einheiten, die das Modell verarbeiten kann. Ein Token kann dabei ein Zeichen oder ein Teil eines Worts sein. Die maximale Kontextlänge stellt die Menge an Tokens dar, die das Modell innerhalb einer Anfrage verarbeiten kann. Je größer sie ist, desto mehr Informationen kann die KI in die Verarbeitung einschließen und desto genauer kann sie ihre Antwort formulieren. Die Antwort zählt dabei zur gesamten Anfrage dazu, ihre Länge wird durch die maximale Größe

des Kontexts begrenzt. Die aktuelle freie von uns genutzte ChatGPT Version hat eine Kontextlänge von 4096 Tokens, bei der Nutzung der kommerziellen API sind Kontexte bis zu 32768 Tokens möglich. In unserer deutschen Anfrage wurde ein Wort in durchschnittlich 2,8 Tokens zerlegt, sodass unsere Anfrage aus etwa 2690 Tokens bestand. Der gesamte Kontext inklusive Antwort enthielt etwa 4700 Tokens, was knapp über der maximalen Größe liegt (vgl. [POA]). Wenn nun in der gleichen Sitzung eine Verfeinerung der Antwort angefragt wird, rutscht das von der KI nutzbare Kontextfenster nach unten und die Daten außerhalb des Fensters sind nicht mehr für die Ergänzung der Anfrage verfügbar.

Eingabeform: UML-Diagramm

Als nächsten Versuch haben wir eine formalisierte Eingabe in Form eines UML-Diagramms gewählt. Da es sich bei Systemtestfällen um die Nachbildung eines Ablaufs durch die Software handelt, fiel die Wahl auf ein Ablaufdiagramm und hier auf ein Aktivitätsdiagramm (siehe

Abbildung 4). Die Möglichkeit, das Modell mithilfe von PlantUML (vgl. [PLA]) zu definieren und so an ChatGPT zu übergeben, machte die Spezifikation lesbar für die KI.

Bei der Abfrage arbeiten wir mit der bereits bekannten Abfrage, ergänzen diese lediglich um die Angabe des PlantUML-Modells. Auf eine Beschreibung der Oberfläche wurde hier jedoch verzichtet, sodass unsere Anfrage das Template aus **Kasten 1** enthielt, ergänzt um die Informationen aus **Kasten 3**.

Ein Testfall, der im Folgenden dargestellt wird, prüft eine sehr seltene Eingabesituation, bei der die Warnmeldung nicht bestätigt wird, jedoch wird beim Test nicht auf das erwartete Fenster geprüft:

„Testfall 3

- Testfallnummer: 003
- Titel: Negativer Fall: ToDo anlegen mit Datum in der Vergangenheit ohne Bestätigung
- zu testende Funktion: ‚Anlegen‘ eines ToDo
- Priorität: Mittel

Ergänzung der Anfrage zum PlantUML-Modell

```

„Bitte erzeuge Testfälle für einen positiven und auch für negative Fälle (alle Fälle, in denen
eine Warn- bzw. Fehlermeldung erscheint). Die Testdaten kannst Du mithilfe von Äqui-
valenzklassen angeben. In dem Hauptfenster sollte die Überprüfung der Daten erfolgen.
Berücksichtige die durch folgendes Aktivitätsdiagramm beschriebene Funktion „Anlegen“.
@startuml
|Benutzer| start :Button „Anlegen“ betätigen;
|System| :Neues anlegen“ öffnet sich;
|Benutzer| :Ausfüllen der Felder;
|Button „Speichern“ betätigen;
if (Beschreibung und Kate-
gorie ausgefüllt?) then (Ja)
if (Datum in der Vergangenheit?) then (Ja)
|Benutzer| :Warnung
anzeigen;
if (Button „Speichern“ betätigt?) then (Ja)
|System| :ToDo-Daten speichern;
|Dialog
schließen;
:Hauptfenster der Anwendung aktivieren;
:ToDo zur ToDoList im Hauptfenster
hinzufügen;
else (Nein)
|System| :Warnmeldung schließen;
:Fenster „ToDo anlegen“ wird
weiterhin angezeigt;
endif
else (Nein)
|System| :ToDo-Daten speichern;
|Dialog schließen;
:Hauptfenster der Anwendung aktivieren;
:ToDo zur ToDoList im Hauptfenster hinzufügen;
endif
else (Nein)
|System| :Fehlermeldung anzeigen;
|Benutzer| :Fehlermeldung bestätigen;
|System| :Fenster „neues ToDo anlegen“ angezeigt, bereits eingetragene Werte bleiben erhalten;
stop @enduml

```

Kasten 3

- Vorbedingungen: Hauptfenster des ToDo-Planers ist offen und funktionsbereit
- Testschritte:
 1. Button ‚Anlegen‘ betätigen
 2. In Fenster ‚ToDo anlegen‘ Beschreibung und Kategorie ausfüllen
 3. Vergangenes Datum eingeben
 4. Button ‚Speichern‘ betätigen
- Erwartete Ergebnisse:
 1. Neues Fenster ‚ToDo anlegen‘ öffnet sich
 2. Warnung wird angezeigt
 3. Fenster ‚ToDo anlegen‘ bleibt offen
- Testdaten:
 1. Beschreibung: ‚Einkaufen‘
 2. Kategorie: ‚Privat‘
 3. Datum: (Ein Datum in der Vergangenheit) Abhängige Tests: Keine“

Bewertung:

- Die Testfälle wurden in der gewünschten Form erzeugt. Es gab positive und negative Testfälle.
- Die Beschreibung der Testdaten war etwas unspezifisch.
- Ein Teil der Testfälle prüfte das Verhalten wie gewünscht ab, abweichend hiervon war der dargestellte Testfall.
- Testschritte sind gut verständlich.

Ein Vorteil der Nutzung der PlantUML-Notation ist die im Vergleich zur natürlichen Sprache deutlich verkürzte genutzte Kontextlänge von etwa 3100 Tokens inklusive Antwort. Dadurch können auch komplexere Abläufe innerhalb einer Anfrage behandelt werden. Das Ergebnis kann als Ausgangsbasis gut verwendet werden, einige Angaben benötigten jedoch eine Anpassung.

Eingabeform: UML-Diagramm mit Oberflächenbeschreibung

Der letzte Versuch zeigte, dass wir durch die Ablaufbeschreibung Testfälle erhalten haben, die die Testschritte gut nachgebildet haben. Jedoch ergaben sich bei der Anbindung für die notwendigen Testdaten einige Unschärfen und die zu prüfenden Fenster waren nicht immer korrekt bezeichnet beziehungsweise die Prüfschritte hier nicht detailliert. Aus diesem Grund haben wir die Beschreibung der Anwendung mit dem PlantUML-Modell (siehe **Kasten 3**) in der Anfrage an ChatGPT um die Beschreibung der Oberfläche der beiden genutzten Fenster ergänzt, analog zu der Abfrage in natürlicher Sprache (siehe **Kasten 2**). Wir erhalten zum Beispiel folgenden Testfall:

„Testfall 2

- Testfallnummer: 002
- Titel: Anlegen eines ToDo-Eintrags ohne Beschreibung
- zu testende Funktion: Anlegen eines ToDo
- Priorität: Hoch
- Vorbedingungen: Hauptfenster des ToDo-Planers ist offen und funktionsbereit
- Testschritte:
 1. Button ‚Anlegen‘ betätigen
 2. Nur das Feld ‚Kategorie‘ ausfüllen
 3. Button ‚Speichern‘ betätigen
- Erwartete Ergebnisse:
 1. Fehlermeldung erscheint
 2. Fenster ‚neuesToDo‘ bleibt offen und bereits eingetragene Werte bleiben erhalten
- Testdaten:
 1. Beschreibung: ‚
 2. Kategorie: ‚Sport‘
 3. Abhängige Tests: Keine“

Bewertung:

- Mit dieser Angabe konnte bei der Erzeugung der Testfälle die Unschärfe, gerade bei den Negativtestfällen, genommen werden. Wir erhalten bessere Testdaten mithilfe der genauen Beschreibung der Oberfläche.
- Des Weiteren werden die Fenster wie erwartet besser bezeichnet.
- Die erzeugten Testschritte gleichen erwartungsgemäß denen der vorherigen Abfrage.

Eingabeform: Quellcode

Eine weitere Eingabeform kann der erstellte Quellcode sein, dies ermöglicht natürlich nicht die entwicklungsbegleitenden Tests (es muss für den Systemtest bereits der Quellcode eines Gesamtprozesses verfügbar sein), aber bei Legacy-Software wäre diese Eingabeform eine Möglichkeit, Testfälle zu erstellen.

Die Begrenzung der Eingabe- beziehungsweise Ausgabegröße muss hierbei beachtet werden. Aus diesem Grund haben wir die Eingabe des Quellcodes auf die XAML-Oberfläche der Seite zur Anlage neuer Aufgaben begrenzt und für die Hauptseite (siehe **Abbildung 2**) sowohl den C#-Code als auch die XAML-Oberfläche übermittelt. Wir ergänzten den Quellcode mit der bereits bekannten Anfrage zum Template und der Oberflächenbeschreibung (siehe **Kasten 1 und 2**). Die KI erzeugte daraus wie schon bei den anderen Anfragen zwischen vier und sechs Testfälle, hier ein Beispiel:

„Testfall 5: ToDo-Eintrag mit vergangenem Fälligkeitsdatum speichern

- Testfallnummer: 005
- Titel: ToDo-Eintrag mit vergangenem Fälligkeitsdatum speichern
- Zu testende Funktion: addToDo()
- Priorität: Hoch
- Vorbedingungen: Keine
- Testschritte:
 1. Öffne das Hauptfenster der Anwendung.
 2. Klicke auf den ‚Neu erstellen‘-Button.
 3. Gib eine gültige Beschreibung ein.
 4. Wähle die Kategorie ‚Arbeit‘ aus.
 5. Wähle ein vergangenes Fälligkeitsdatum aus.
 6. Setze den Erledigt-Status auf ‚Ja‘.
 7. Klicke auf ‚Speichern‘.“
- Erwartete Ergebnisse: Eine Warnmeldung wird angezeigt, die den Benutzer fragt, ob er das vergangene Fälligkeitsdatum übernehmen möchte.“

Bewertung:

- Dieser Versuch hat dazu geführt, sehr detaillierte Testschritte zu erhalten, jedoch „unterschlug“ ChatGPT die gewünschten Testdaten komplett.
- Diese Form der Eingabe hat prinzipbedingt zur Folge, dass entgegen der Spezifikation gegen den Quellcode getestet wird, damit ist der Systemtest aus dem Bereich Black- in den Bereich White-Box-Test gelangt.
- Die Beschreibung der Testdaten entfiel.
- Die Testschritte wurden sehr gut beschrieben.
- Diese Methode ist nur für kleine Bereiche der Anwendung geeignet, da die maximale Eingabelänge schnell erreicht wird.

Fazit & Ausblick

Über alle Versuche hinweg konnten wir feststellen, dass der Einsatz von ChatGPT zur Unterstützung im Bereich Testfalldesign sehr gewinnbringend sein kann. Die erstellten Artefakte müssen jedoch immer noch ein Review durchlaufen, da das Risiko für sogenannte „Halluzinationen“ [Aix], Antworten für eher irrelevante Fälle, Ungenauigkeiten in den Antworten und Ähnliches zu groß ist. Wir haben zum Beispiel bemerkt, dass ChatGPT Anforderungen der Software voraussetzt, die nicht explizit in den Quelldaten erwähnt wurden.

Der Aufwand der Prüfung liegt hierbei unter dem Erstellungsaufwand für Testfälle. Eine genaue Bezifferung fällt hierbei

Literatur & Links

- [Arx1] Arxiv, Language Models are Few-Shot Learners, siehe: <https://arxiv.org/abs/2005.14165>
- [Arx2] Arxiv, Survey of Hallucination in Natural Language Generation, siehe: <https://arxiv.org/abs/2202.03629>
- [Coh09] M. Cohn, Succeeding with Agile: Software Development Using Scrum, Addison-Wesley, 2009
- [CoPilot] Github CoPilot, siehe: <https://github.com/features/copilot>
- [OpenAI] OpenAI ChatGPT, siehe: <https://chat.openai.com/>
- [PLA] PlantUML, siehe: <https://plantuml.com/de/>
- [POA] Offizieller Kontextrechner von OpenAI, siehe: <https://platform.openai.com/tokenizer>
- [Wiki1] Wikipedia, Large Language Models, siehe: https://en.wikipedia.org/wiki/Large_language_model
- [Wiki2] Wikipedia, GPT-3, siehe: <https://en.wikipedia.org/wiki/GPT-3>

natürlich schwer, da die Erstellung durch menschliche Tester in der Geschwindigkeit sehr variiert. Die Analyse der Dokumente nimmt bei komplexen Programmen einen großen Teil der Arbeit an, hier kann die KI mit Geschwindigkeit punkten.

Aber es gilt auch hier: „Shit in, Shit Out“. Gibt man der KI zu ungenaue Angaben, entweder in Bezug auf das Format der zu erstellenden Antwort oder bei den benötigten Informationen zur Anwendung, dann erhält man ungenaue Antworten beziehungsweise solche in einem für den Zweck ungeeigneten Format. Grundsätzlich ist, bei der Erstellung der Anfrage die maximale Kontextgröße des benutzten Modells zu beachten. Dieser Kontext umfasst die gesamte Chat-Session, wird diese zu lang, dann werden Informationen vom Anfang ignoriert und nicht mehr in die Antwort einbezogen.

Eine Nutzung der OpenAI-API ermöglicht deutlich längere Kontexte als die Chat-Schnittstelle. Pauschal lässt sich keine Empfehlung für eine passende Kontextgröße machen. Wir empfehlen, die Größe der Anfrage und die Menge der erwarteten Testfälle bei der Berechnung einzubeziehen und sich iterativ anzunähern. Tools wie die unter [POA] genannten dabei sehr hilfreich.

Bei komplexen Anwendungen wird die Anzahl an Codedateien beziehungsweise eine umfassende Beschreibung in Form von Prosa oder auch modellbasierter Darstellung schnell sehr groß. Eine schrittweise Generierung von Testfällen kann dann auch problematisch sein, weil gegebenenfalls Abhängigkeiten nicht beachtet werden.

Im Unterschied zum manuellen Vorgehen bei der Erstellung von Tests sind die erzeugten Testfälle nicht unbedingt aufeinander abgestimmt. Für die Priorisierung der Testfälle benötigt die KI mehr Informationen, als dies bei Fachexperten notwendig wäre. In unseren Experimenten haben wir gezeigt, dass ChatGPT eine bestimmte Anzahl an Testfällen für

einen Anwendungsfall erstellen kann. Die Möglichkeit, eine hohe Testabdeckung zu erreichen beziehungsweise die Testfälle aufeinander abzustimmen und so risikobasiert zu testen, konnten wir nicht nachweisen.

Die erzeugten Testfälle waren jedoch so überzeugend, dass wir beim Ausgabeformat für die Testfälle unterschiedliche Anfragen integriert haben. User-Stories stellen für ChatGPT kein Problem dar, mithilfe einer Definition des gewünschten Templates können auch diese einschließlich einer Definition of Done und Kriterien zum Test erstellt werden. ChatGPT bietet auch Unterstützung für die Generierung von Testfällen im Cucumber-Format. Die präsentierten Testfälle entsprechen dem allgemeinen Muster von Cucumber, doch es besteht die Möglichkeit von Abweichungen und Ungenauigkeiten. Diese können auftreten, da ChatGPT auf gelernten Mustern und vorhandenem Wissen basiert, aber nicht optimal auf

spezifische Anforderungen oder Details eingestellt ist.

Das Werkzeug kann helfen, unterschiedliche Formate, Testfälle, User-Stories und Cucumber-Code zu erzeugen und schnelle Ergebnisse zu liefern, die einen guten Einstiegspunkt liefern.

Die Beschränkung der Token-Größe haben wir als Einschränkung bereits erwähnt, dies erfordert ein besonderes Vorgehen für das Arbeiten mit ChatGPT bei größeren Anwendungen beziehungsweise bei der Erstellung von Systemtests für die Gesamtmenge aller Anforderungen einer Anwendung. Ein Vorgehen, das Zerlegen in Anwendungsfälle wie in unserem Beispiel das Anlegen eines Tasks und das schrittweise Abarbeiten dieser haben wir in diesem Artikel gezeigt. Die Größe der Use-Cases kann hierbei variieren. Jedoch müssen wir im Systemtest darauf achten, dass diese den Testfokus des Systemtests, nämlich einen Gesamtprozess aus Nutzersicht, nicht verlieren. ||

Die Autoren



Dr. Jessica Schiffmann

(jessica@schiffmann.it)

promovierte im Bereich der Systemtestautomatisierung an der TU Bergakademie Freiberg und arbeitet als IT-Spezialistin bei der Festo SE & Co. KG. Neue Technologien, wie KI, sowie Automatisierung von Aufgaben und Softwaretest interessieren sie besonders.



Christian Giudici

(chris@giudici.de)

ist Testmanager bei der eurodata AG. Er und sein Team stellen sicher, dass Software funktional, performant und sicher an die Kunden ausgeliefert wird. Seine Schwerpunkte sind neben der Testautomatisierung insbesondere Last- und Performancetests.