

CHATGPT IN DER SOFTWAREENTWICKLUNG

Pair Programming mit der KI

Der erste Kontakt verläuft eher überraschend, gibt aber einen Ausblick auf Möglichkeiten und Einschränkungen.

Wieder hat eine Umwälzung begonnen und wir können noch nicht absehen, was sie mit uns machen wird. Für mich ist sie vergleichbar mit dem Erscheinen des ersten iPhones. Was haben Sie damals im November 2007 darüber gedacht? Ich habe nichts gedacht, wenn ich mich richtig erinnere. Für mich war das kein Thema. Vor 2009 habe ich kein iPhone in der Hand gehabt. Und auch dann war ich noch nicht überzeugt; es hat weitere zwei Jahre gedauert, bis ich mir eines gekauft habe.

Disruptor iPhone – die Handys werden smart

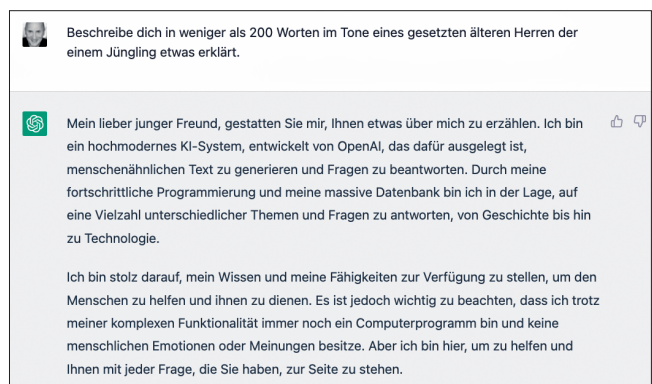
Die ersten iPhones ließen einiges vermissen. Für den Preis wurde zu wenig geboten, fand ich. Da bin ich lieber bei meinem Samsung-Handy geblieben. Das war kleiner und völlig ausreichend.

Doch die Umwälzung hatte begonnen. Die Beziehungen zwischen den Menschen haben sich verändert durch das iPhone beziehungsweise folgende Smartphones von Apple und anderen Anbietern. Das ist in jeder U-Bahn zu sehen, im Restaurant, am Nachttisch, am Arbeitsplatz, in Meetings. Smartphones haben das Leben durchdrungen. Sie sind zwischen unsere Beziehungen geschlüpft. Wer nicht sehr bewusst mit ihnen umgeht, wer nicht aktiv gegen ihre Benachrichtigungen und Ablenkungsversprechen etwas tut, wird von ihnen verschlungen. Und das fängt bei Kleinkindern an, die von ihren Eltern davor geparkt werden.

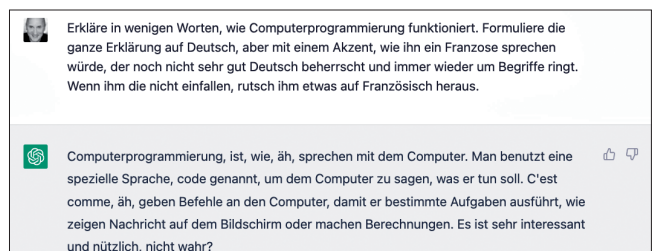
Das meine ich gar nicht als Gesellschaftskritik. Mir geht es nicht darum, ob Smartphones gut oder böse sind. Ich will nur eine tiefgreifende Veränderung skizzieren, die 2007 mit einem iPhone begonnen hat, das viel zu wünschen übrig ließ.

In einem war es jedoch schon ausgereift. Disruptiv anders war, dass es eine neue User Experience (UX) bot: Die Tasten waren abgeschafft. Damit war die Hardware des Handys im Grunde flüssig geworden. Wie der T-1000 in „Terminator 2“ konnte sie sich (an der Oberfläche) in alles verwandeln, was App-Programmierern vorstellbar war. Im Sonderfall sah das Handy aus wie ein traditionelles mit Tasten zum Wählen einer Nummer. Aber meistens hatte es eine ganz andere Form. Damit war das Handy so formbar, dass es jedem zu jeder Zeit ein nützlicher, ja unersetzlicher Gefährte werden konnte.

Aus einem Spezialgerät fürs Telefonieren war ein Universalgerät geworden, für alles Denkbare und noch viel mehr,



ChatGPT beschreibt sich selbst (Bild 1)



ChatGPT erklärt Programmierung – nur ein wenig schräg (Bild 2)

was noch nicht gedacht werden konnte. Das war die potenzierte Idee des general purpose computer, des PC.

Disruptor ChatGPT – die Chats werden smart

Für mich war das iPhone also eine Disruption, die in einem gänzlich anderen UX als bisher bestand.

Und genau das ist nun wieder passiert. Wir sind im Jahr null der Smartchats. Bisher gab es Chats aller Art auf allen möglichen Plattformen: von SMS über Whatsapp und Telegram zu Slack, Facebook und Co. Bisher waren Chats mit Menschen oder mit simplen Chat-Bots.

Doch mit OpenAIs [1] ChatGPT [2] als Benutzerschnittstelle für seine künstliche Intelligenz ist eine neue Qualität für Chats entstanden. Die Disruption des iPhones bestand in der Verflüssigung der UX; die Disruption von ChatGPT besteht in der Verflüssigung der UX.

Ja, wieder ist es die User Experience, die den Unterschied macht. Dieses Mal fehlt nicht eine fixe Menge an Tasten, die Interaktionen in ein Korsett zwängen. Dieses Mal fehlt eine fixe Befehlssprache. Um ChatGPT benutzen zu können, müssen Benutzer nicht mehr können, als ihre Muttersprache zu beherrschen. Die Interaktion ist natürlichsprachlich. Sie erfolgt sogar in einem echten Dialog, bei dem ChatGPT alles bisher Gesagte in seine Antworten mit einbezieht.

Aber nicht nur fehlt jede limitierende Befehlssprache, es gibt auch keine explizite thematische Grenze mehr. ChatGPT kann zu allem etwas sagen. Es kann erklären, was es selbst ist (Bild 1). Oder es kann erklären, worum es bei der Programmierung geht (Bild 2).

ChatGPT kann zusammenfassen, umwandeln, übersetzen, erfinden. ChatGPT beherrscht natürliche Sprache und formale Sprachen. ChatGPT kann daher auch selbst programmieren.

Wie gesagt, für mich ist die Barrierefreiheit die Disruption. Es gibt keine Sprachbarriere mehr, es gibt keine inhaltliche Barriere mehr. Was das mit uns, mit der Gesellschaft machen wird, ist nicht abzusehen. Schon jetzt ist mindestens die Lehre von Grundschule bis Universität in Aufruhr. Was muss sich ändern, wenn ChatGPT jede Hausaufgabe genauso gut oder besser als Schüler und Studenten erledigen kann?

DeepL [3] war ein Meilenstein für die Übersetzung von Texten. Kaum ein Übersetzer mehr, der heute nicht einen Text zuerst durch DeepL laufen lassen würde, um anschließend dessen Ergebnis nur noch zu polieren.

Doch mit ChatGPT kann nicht nur übersetzt, sondern gleichzeitig verfeinert werden.

Wenn ich ChatGPT bitte: „Formuliere die obigen Absätze so, dass auch jemand sie versteht, der kein Computerenthusiast ist“, liefert es die Vereinfachung wie in Bild 3. Und wenn ich dann bitte: „Übersetze den umformulierten Text ins Englische“, kommt das Ergebnis in Bild 4 heraus.

Mit ChatGPT kann also Chefredakteur Tilman Börner nicht sicher sein, welche Teile meines Artikels wirklich von mir stammen. Es sei denn, er bemüht eine andere KI, um KI-generierte Texte zu identifizieren.

Aber letztlich ist es einerlei, ob diese Zeilen direkt von mir geschrieben wurden oder ob ChatGPT sie auf ein sogenanntes Prompt hin generiert hat. Das Kriterium für Tilman sollte nicht die unmittelbare Textquelle sein, sondern der Inhalt. Passt der in Form und Inhalt? Schreiben war immer schon ein Akkordgeschäft – wie die Programmierung. Wer mehr ausrei-

chend guten Text in kürzerer Zeit produzieren konnte, hat mehr verdient.

Für mich ist ChatGPT deshalb ein universelles Produktivitätstool. Es potenziert die Fähigkeit, Texte zu extrahieren, zu analysieren, zu präsentieren, zu transformieren, zu generieren. Es ist gleichermaßen analytisch wie kreativ. Die Kunst ist, mit dieser Power umzugehen. Wer es schafft, auf der kraftvollen Welle ChatGPT zu surfen, hat einen riesigen Vorsprung.

Und das wird die Welt in einer Weise verändern, glaube ich, wie wir es uns noch nicht vorstellen können. So wie nach dem Erscheinen des iPhones 2007 nicht klar war, dass es einmal dazu beitragen würde, dass die Suizidraten bei Mädchen in den USA steigen oder Menschen eine Partnerwahl mit einem Swipe treffen.

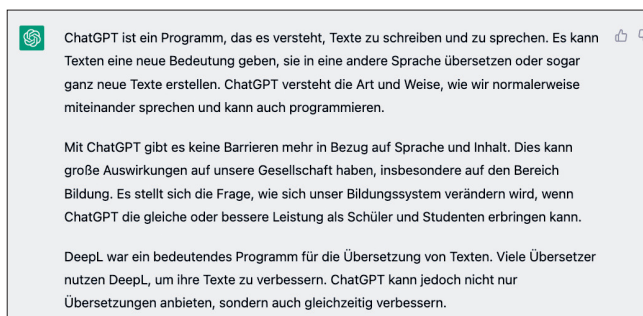
ChatGPT kann programmieren

Welchen Einfluss wird ChatGPT aber auf die Softwareentwicklung haben? Nicht weniger als einen großen, behaupte ich. Dass ChatGPT heute keine perfekten Antworten liefert, Fehler hat, aussetzt, manchmal einfach nicht verstehen will, tut dem keinen Abbruch.

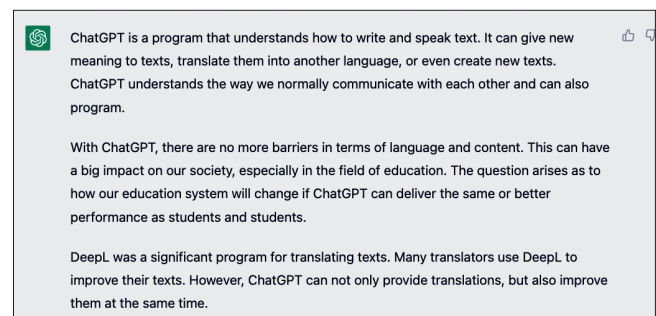
ChatGPT ist Anfang 2023 auf einem Niveau wie das iPhone 2007: vielversprechend, aber für viele noch zu lückenhaft, als dass sie es für ihr Leben für relevant hielten.

Das verstehe ich sehr gut. Das ist normal. Allerdings sollte es nicht als grundsätzliche Limitation missverstanden werden. Wir sprechen über ein MVP, ein Minimum Viable Product, eine erste Version, auf die viele weitere folgen werden. Denn ChatGPT ist eine Lizenz zum Gelddrucken. Google ist schon in heller Aufregung. Denn wer wollte noch googeln, um sich dann mühsam durch Suchergebnisse zu klicken, wenn er von ChatGPT eine Antwort auf den Punkt in verständlicher Sprache bekommen kann? Bild 5 spricht für mich Bände – und dem tut es auch keinen Abbruch, falls Sie meinen, dass ChatGPT vielleicht das eine oder andere neue C#-Feature unterschlagen hätte. Die Antwort enthält bereits einen inhaltlichen Überblick, wohingegen Google keine Eigenleistung erbringt.

Und bei ChatGPT habe ich jederzeit die Möglichkeit, nachzufragen (Beispiel in Bild 6). Ich kann Bezug nehmen auf das, was vorher im Dialog war. ChatGPT legt gern nach. Der Dialog kann in die Tiefe gehen, in die Breite und wieder in die Höhe. Dass am anderen Ende kein Mensch antwortet, ist unverkennbar. Aber das mindert den Wert schon dieser Ver- ►



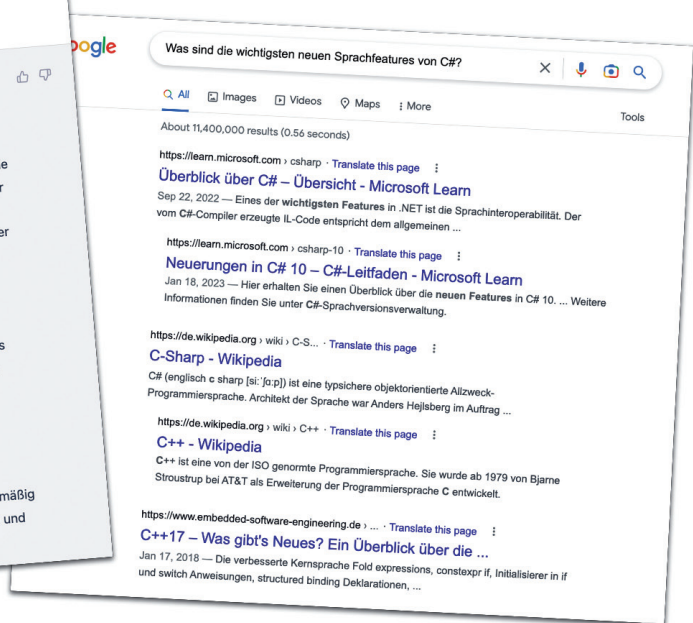
ChatGPT hat einige Absätze aus diesem Artikel vereinfacht (Bild 3)



Den Text aus Bild 3 hat ChatGPT ins Englische übersetzt (Bild 4)



ChatGPT versus Google (Bild 5)



sion von ChatGPT nicht. Wir befinden uns noch in einer Art Uncanny Valley [4] der KI: Ihre Reaktionen sind nicht mehr augenfällig künstlich oder blechern, aber auch noch nicht ununterscheidbar vom Menschen. Irgendwo dazwischen, schon sehr nah dran am Menschen, bewegt sich ChatGPT – und das macht einen Teil seiner Unheimlichkeit aus.

Aber damit nicht genug: ChatGPT kann auch programmieren, das heißt Code für Anforderungen generieren. Welches Beispiel sich dafür anbietet, weiß ChatGPT selbst am besten (Bild 7). Es soll ja nicht so schwer sein. Auch eine KI will nicht überfordert werden, oder?

Auf geht's also. Bitten wir ChatGPT, Code für diese einfachen Anforderungen zu generieren (Bild 8). Ich finde, das Ergebnis kann sich sehen lassen. ChatGPT benutzt zwar nicht die neuesten Möglichkeiten der C#-Syntax, aber das halte ich für in Ordnung. Auch in der `Main()`-Methode eingepackt ist die korrekte Lösung erkennbar. ChatGPT hat uns verstanden und ist dabei sogar kreativ geworden: Das abschließende `ReadLine()` war nicht gefragt, ist allerdings typisch und wird am Ende auch begründet.

ChatGPT jenseits der Muster

Aber wer kann wissen, ob das, was ChatGPT hier geliefert hat, nicht einfach abgeschrieben ist? Hello-World-Programme in C# gibt es zuhauf im Internet. Hat ChatGPT die Anforderungen wirklich verstanden?

Prüfen wir es nach. Wünschen wir uns ein nächstes Feature (Bild 9). Das geht jetzt über das übliche „Hello World“ hinaus, denke ich. Verständnis ist nötig – und Kreativität.

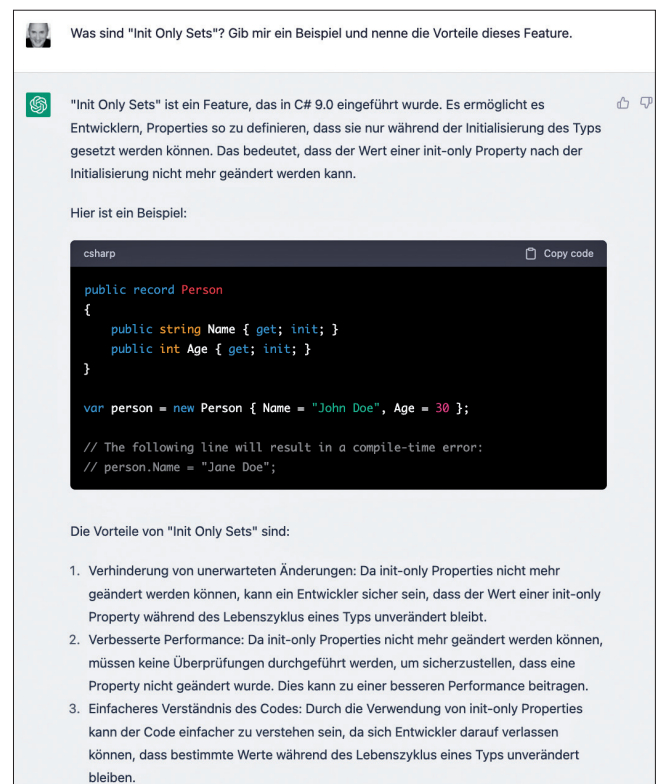
Ich finde bemerkenswert, dass nun das `ReadLine()` am Ende fehlt. ChatGPT weiß, dass das Programm nicht damit am Laufen gehalten werden muss. Es hat also nicht einfach die vorherige Version erweitert, sondern auch etwas vorher Relevantes entfernt.

Außerdem hat ChatGPT meine Formulierung „wird mit 'Hello' und seinem Namen begrüßt“ übersetzt in `"Hello, " + name + "!"`. Der Name wird sinnvoll mit einem Komma getrennt und mit einem Ausrufezeichen abgeschlossen. Formal

richtig wäre auch `"Hello" + name` gewesen. Die Endlosschleife ist eine simple Lösung für meine Anforderung, den Benutzernamen immer wieder zu erfragen. Mehr habe ich nicht gewollt. Aber ChatGPT hätte auch eine Frage einfügen können, ob noch ein nächster Name erfragt werden soll.

ChatGPT nimmt sich also gewisse Interpretationsfreiheit; darin unterscheidet es sich nicht von menschlichen Entwicklern. Im Guten wie im Schlechten, würde ich sagen.

Okay, lassen wir das Problem etwas kniffliger werden. Jetzt sollen die Begrüßungen wechseln, je nachdem, wie oft ein Name schon eingegeben wurde (Bild 10). Dafür findet sich kei-



Nachgefragt bei ChatGPT (Bild 6)

ne Vorlage im Internet, da bin ich mir sicher. ChatGPT muss also ganz allein das Problem lösen.

Und prompt fällt ChatGPT auf die Nase. Der Code tut nicht, was er tun soll, wie der Output zeigt. Als der Name „Maria“ zum ersten Mal eingegeben wird, erscheint dennoch die Begrüßung für eine wiederholte Eingabe. Das ist nicht richtig. Das habe ich nicht gemeint.

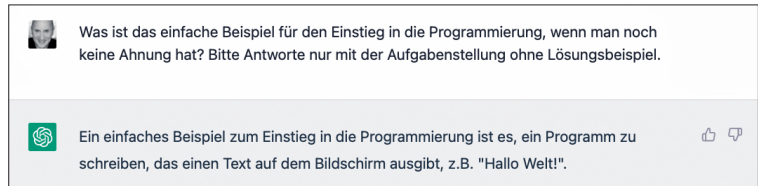
Und genau da liegt das Problem: Es gab einen Unterschied zwischen dem, was ich meinte und dem, was ich schrieb. Meine Formulierung war, „dass die Begrüßung wechselt, je nachdem, wie oft ein Benutzer schon seinen Namen eingegeben hat“.

Für ChatGPT gibt es nur einen Benutzer, der immer wieder aufgefordert wird, seinen, nein, einen Namen einzugeben. Ob er sich erst Peter und dann Maria nennt, ist ChatGPT einerlei. Die Begrüßung verändert sich nach Anzahl der Eingaben überhaupt eines Namens, nicht nach der Anzahl der Eingaben eines bestimmten Namens. Für mich waren es verschiedene Benutzer, die nacheinander das Programm während desselben Laufs nutzen: Zuerst tritt Peter an den Computer, dann Maria. Jeder von ihnen soll passend begrüßt werden, je nachdem, wie oft sie ihre Namen eingeben.

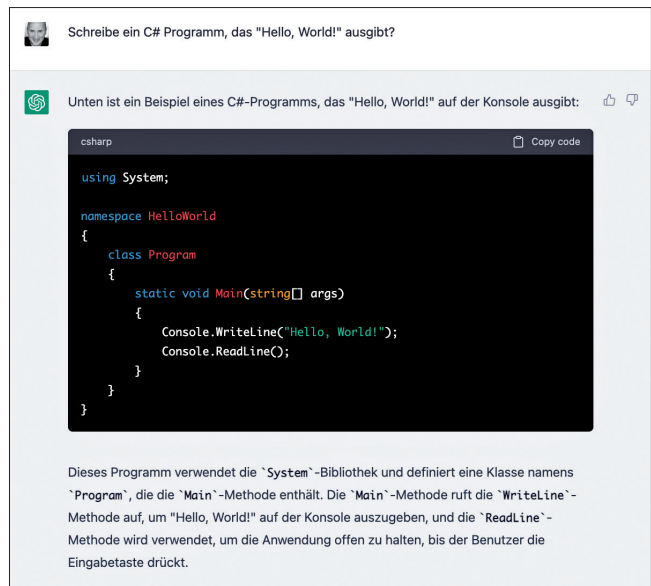
Nicht die KI hat also einen Fehler gemacht. Der Mensch war es. Ich war zu ungenau in meiner Formulierung. Deshalb hat ChatGPT nur eine Zählervariable *counter* eingeführt, um die Anzahl der Begrüßungen zu halten.

In seinem Kommentar sagt ChatGPT auch ganz klar, dass es nur einen Benutzer erwartet.

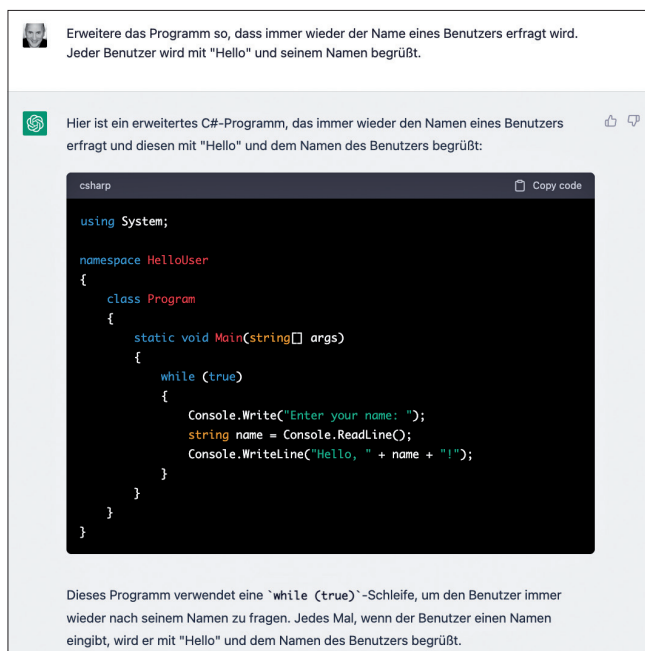
Hier zeigt sich schon das zentrale Problem beim Pair Programming mit KI: Als menschlicher Entwickler, der die KI als Codegenerierungswerkzeug nutzen will, muss ich die Anforderungen sehr, sehr präzise formulieren. Garbage in, garbage out bekommt mit ChatGPT eine Bedeutung, die mir in



ChatGPT weiß, womit seine Programmierkompetenz getestet werden sollte (Bild 7)



ChatGPT programmiert wie ein Anfänger (Bild 8)



ChatGPT kann mehr, als nur mit Mustern zu antworten (Bild 9)

20 Jahren Agilität ein wenig in Vergessenheit geraten scheint. Mit agilen Iterationen lassen sich unklare Anforderungen doch schnell ausbügeln; einfach noch eine Runde drehen. Agilität bedeutet doch, im Gespräch zu bleiben. Oder?

Ja genau: Im Gespräch bleiben. Doch die Turnaround-Zeit beim Gespräch zwischen Product Owner und Entwickler ist lang. Und der Entwickler ist ein Mensch, der schon bei Ungenauigkeiten selbstständig nachschärfen wird. Oder? Das darf man doch als Product Owner und Kunde erwarten. Und so hat es sich eingebürgert, dass sich Entwickler vielfach auf unklare Anforderungen committen – was notwendig zu Nachbesserungen führt. Doch das wurde unter Agilität verbucht und fiel nicht weiter auf.

Damit ist bei ChatGPT Schluss. Die KI ist gnadenlos in ihrer Ernsthaftigkeit bei der Interpretation von Anforderungen. Wenn Sie aus ihr einen Vorteil ziehen wollen, müssen Sie viel, viel genauer als früher wissen, was Sie wollen beziehungsweise was der Product Owner will.

Wie das Navigationssystem im Auto ist ChatGPT allerdings auch emotionslos. Wenn der vorgeschlagene Weg nicht gefällt, ist es nur zu gern bereit, eine Korrektur vorzunehmen. Also legen wir einfach mit einer Verfeinerung der Anforderungen nach (Bild 11).

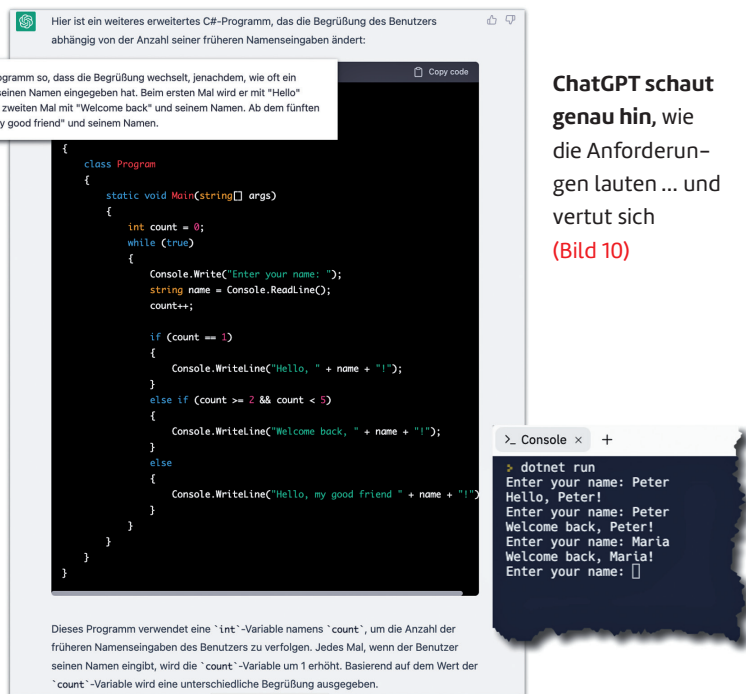
Und schon sieht der Output aus wie ursprünglich gedacht: Maria wird als Neue begrüßt, Peter wird beim fünften Mal als guter Freund erkannt. ▶

Was ist hier bemerkenswert? Ich hoffe, Ihnen fällt auf, dass es eigentlich nichts Auffälliges gibt. Wir unterhalten uns mit ChatGPT über Anforderungen. Es ist ein Dialog. Wir folgen keinem Skript, wir benutzen keine spezielle Sprache. Wir reden, wie uns der Schnabel gewachsen ist – mit allen Konsequenzen. ChatGPT ist hier der Partner im Pair, der codiert. Wir wünschen uns nur etwas von der Rückbank. Wie bei einer Taxifahrt.

Wenn wir aber genauer hinschauen, dann gibt es schon ein paar Dinge, die verwundern sollten:

- Wir gehen in Iterationen innerhalb eines Kontextes vor. Den teilen wir mit ChatGPT. Auch ChatGPT behält alle Gesprächsbeiträge im Hinterkopf. Ich habe nur gesagt „Berücksichtige ...“, und ChatGPT wusste, worauf sich das bezieht. Das macht einen guten Teil der disruptiven UX aus, finde ich.
- Als Benutzer von ChatGPT muss ich mir wenig Gedanken machen, wie ich mit dem Tool umgehe. Ich habe im Grunde nur die Erwartungen, die ich an jeden Menschen habe: Dass er mir in einem Gespräch zuhört und sich erinnert, was schon gesagt wurde.
- Ich muss mir kein Mindset zulegen für den Umgang mit ChatGPT, ich muss keine Sprache lernen. Im Detail stimmt das nicht so ganz, aber für einen Einstieg hier reicht diese Anspruchshaltung aus. ChatGPT wird ihr in Zukunft auch mehr und mehr gerecht werden.

ChatGPT hat nun eine nicht mehr so triviale Lösung erdacht: Die Zahl der Eingaben wird für jeden Namen in einem



ChatGPT schaut genau hin, wie die Anforderungen lauten ... und vertut sich (Bild 10)

Dictionary geführt. Es versteht die Abstraktion, die diese Datenstruktur bietet, und hat sie als hilfreich für die Lösung erkannt. Das schafft nicht jeder Programmieranfänger – ChatGPT ist also schon weiter.

Es erklärt auch – im Kommentar unter dem Code –, was es sich dabei gedacht hat.

Jetzt die letzte Runde. Wie wäre es, wenn wir die Lösung persistent machen würden? Dann kann das Programm beendet werden, ohne dass die Namen vergessen werden. Oder es könnte ohne großen Schaden abstürzen. In Bild 12 formuliere ich bewusst laienhaft: „Die Namen sollen nur erinnert werden.“ Ob ChatGPT weiß, was das bedeutet?

Aber sicher doch! ChatGPT übersetzt „erinnern“ in eine persistente Speicherung der Namensliste. Deren Datenstruktur wird nicht verändert, aber nun zu Beginn geladen und nach jeder Änderung gespeichert.

Bitte beachten Sie: Dabei werden selbstverständlich auch Typen berücksichtigt. Der Zählerstand wird beim Laden von einer Zeichenkette in eine Zahl umgewandelt!

Der Output zeigt, dass die Änderungen den gewünschten Effekt haben. Meine Anforderungsergänzungen wurden verstanden. ChatGPT hat sogar die Persistenz in zwei eigene Funktionen herausgezogen. Das wäre für die Funktionalität nicht nötig gewesen. Einen gewissen Anspruch an Code-Sauberkeit scheint die KI zu haben.

Auch die Namensgebung ist nicht ganz schlecht, finde ich. Zur Auslagerung in ein eigenes Modul hat sich ChatGPT allerdings noch nicht hinreißen lassen. Aber man kann nicht gleich alles haben wollen.

Oder muss man das überhaupt haben? Ist es wichtig, ob ChatGPT wirklich sauberen Code generiert? Sind Namen wichtig, sind Funktionen wichtig, Module, Abstraktionen im Allgemeinen? Ich meine, nein. Nein, das ist alles überhaupt nicht mehr wichtig.



ChatGPT bessert nach (Bild 11)

Um die Namen über die Programmlaufzeit hinaus zu erinnern, müssen wir die Namen-Zähler-Informationen speichern, wenn das Programm beendet wird, und sie beim nächsten Start des Programms wieder laden. Eine einfache Möglichkeit, dies zu tun, besteht darin, die Informationen in einer Textdatei zu speichern und zu laden. Hier ist ein weiteres erweitertes C#-Programm, das die Namen-Zähler-Informationen in einer Textdatei speichert und lädt:

```
using System;
using System.Collections.Generic;
using System.IO;

namespace HelloUser
{
    class Program
    {
        static void Main(string[] args)
        {
            Dictionary<string, int> nameCounts = LoadNameCounts();

            while (true)
            {
                Console.WriteLine("Enter your name: ");
                string name = Console.ReadLine();

                int count = 1;
                if (nameCounts.ContainsKey(name))
                {
                    count = nameCounts[name] + 1;
                }
                nameCounts[name] = count;

                SaveNameCounts(nameCounts);

                if (count == 1)
                {
                    Console.WriteLine("Hello, my good friend " + name + "!");
                }
            }
        }
    }
}
```

Ändere das Programm so, dass die Namen über die Programmlaufzeit hinaus erinnert werden.

```
Console.WriteLine("Hello, my good friend " + name + "!");
}
}

private static Dictionary<string, int> LoadNameCounts()
{
    Dictionary<string, int> nameCounts = new Dictionary<string, int>();

    if (File.Exists("namecounts.txt"))
    {
        string[] lines = File.ReadAllLines("namecounts.txt");
        foreach (string line in lines)
        {
            string[] parts = line.Split(':');
            string name = parts[0];
            int count = int.Parse(parts[1]);
            nameCounts[name] = count;
        }
    }

    return nameCounts;
}

private static void SaveNameCounts(Dictionary<string, int> nameCounts)
{
    List<string> lines = new List<string>();
    foreach (KeyValuePair<string, int> nameCount in nameCounts)
    {
        lines.Add(nameCount.Key + ":" + nameCount.Value);
    }
    File.WriteAllLines("namecounts.txt", lines);
}
}
```

Dieses Programm verwendet die 'LoadNameCounts'- und 'SaveNameCounts'-Methode um die Namen-Zähler-Informationen zu laden und zu speichern. Die 'LoadNameCounts'-Methode liest die Informationen aus der Textdatei 'namecounts.txt' und

```
>. Console x +
> dotnet run
Enter your name: Peter
Hello, Peter!
Enter your name: Maria
Hello, Maria!
Enter your name: exit status 130
> dotnet run
Enter your name: Peter
Welcome back, Peter!
Enter your name: █
```

ChatGPT macht die Lösung persistent (Bild 12)

Ausblick

Für ein erstes Experiment mit ChatGPT war es nett, den generierten Code zu sehen. Das habe ich auch wertgeschätzt, als ich 1990 das erste Mal mit einem C++-Compiler gearbeitet habe: Der hat mich seinen Zwischencode in C sehen lassen. Das war spannend. Was macht der Compiler aus diesen damals für mich neuen objektorientierten Sprachfeatures? Als bald jedoch war das nicht mehr interessant. Ich habe mich auf die OO-Abstraktionen in C++ konzentriert und wollte direkt ein lauffähiges Programm. Wie der Compiler das macht, wie viele Arten Zwischencode er erzeugt – das war mir egal.

Und so sehe ich das auch bei ChatGPT. Im Moment ist es interessant zu sehen, wie die KI ihrer Aufgabe Codierung nachkommt. Ich habe noch nicht so viel Vertrauen in ihre Programmierfähigkeit. Da will ich lieber einmal mehr als einmal zu wenig ihren Output in Augenschein nehmen.

Ich erwarte aber, dass dieses Bedürfnis schwindet. Die KI wird schon beweisen, dass sie etwas kann. Da muss ich mir nicht auch noch den ganzen Code anschauen, den sie generiert. Das verlangsamt auch die Iterationen. „Trust is speed“ wird auch für die Zusammenarbeit mit ChatGPT gelten.

Aus der White Box muss eine Black Box werden. Über kurz oder lang erwarte ich, dass ChatGPT beziehungsweise seine Nachfolger und Verwandten ausführbaren Code erzeugen. Der interessiert mich dann nur im Hinblick auf seine Korrektheit. Sauberkeit ist für mich einerlei. Ich muss ihn ja nicht ändern. Wenn ich neue Anforderungen nachreiche, kann ChatGPT komplett neuen Code generieren. Auch ChatGPT muss seinen eigenen Code nicht verstehen.

Aber wie können Sie feststellen, ob ChatGPT einen guten Job macht? Es ist heute und auf absehbare Zeit kein perfektes Tool. Blindes Vertrauen ist sicherlich fehl am Platz. Alle mal, weil nicht nur ChatGPT missverstehen kann; auch Sie

können Anforderungen missverständlich, lückenhaft, gar fehlerhaft formulieren. Wie damit umgehen, wenn ChatGPT Code generieren soll? Die Antwort ist für mich sehr einfach: So wie mit jedem Code, den Sie schreiben oder jemand anderes. Es braucht auch, nein, gerade in Zeiten von ChatGPT automatisierte Tests mit einer relevanten Abdeckung.

Meiner Meinung nach verschiebt sich mit ChatGPT sogar der Ansatz für die Softwareentwicklung massiv hin zu Test-first-Codierung [5]. Wie das mit ChatGPT funktioniert, zeige ich in einem nächsten Artikel.

Warten Sie jedoch mit eigenen Experimenten mit ChatGPT nicht bis dahin. Laden Sie die KI ein zum Pair Programming. Es lohnt sich schon jetzt, mit der Disruption vertraut zu werden. In zehn Jahren können Sie sich dann erinnern, dieses Mal am Anfang mit dabei gewesen zu sein. ■

[1] OpenAI, <https://openai.com>

[2] ChatGPT, <https://chat.openai.com>

[3] DeepL, www.deepl.com

[4] Uncanny Valley, www.dotnetpro.de/SL2306ChatGPT1

[5] Ralf Westphal, *Test-first Codierung*, 2020, 493 Seiten, www.dotnetpro.de/SL2306ChatGPT2



Ralf Westphal

ist Trainer, Berater, Autor, Sprecher und Mitgründer der Clean Code Developer Initiative. Seine Schwerpunkte sind dauerhaft hohe Produktivität für die Softwareentwicklung und zukunftsfähige Teamorganisation.

<https://ralfw.de>

dnpCode

A2306ChatGPT