

Der Mensch in der Entwicklung



Welche
persönlichen
Herausforderungen wir
alle am Arbeitsplatz zu meistern
haben – und wie dies gelingt. Ab Seite 6

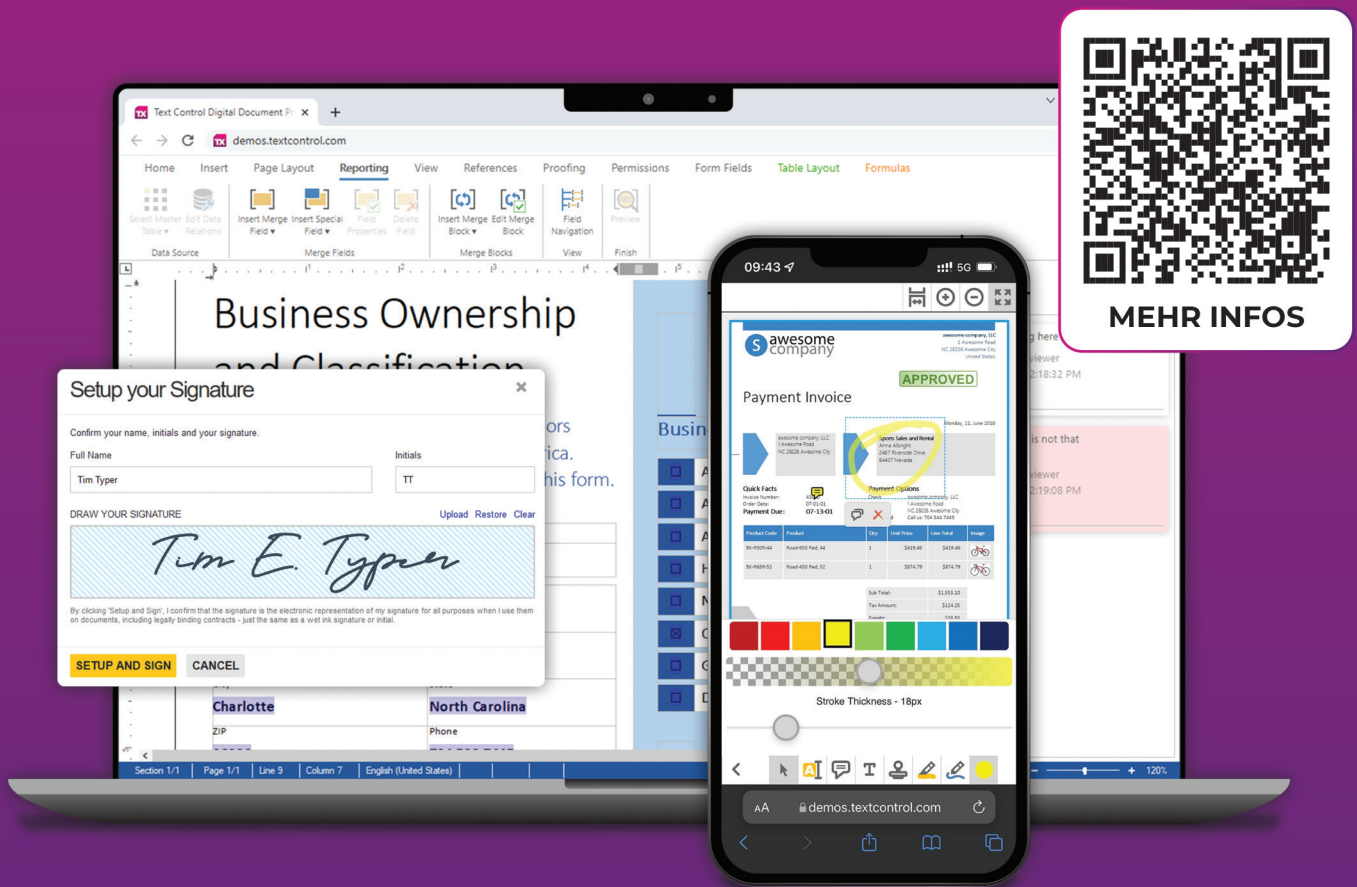


.NET DEVELOPER CONFERENCE '24

Jetzt Programm ab S.23 entdecken und auf dotnet-developer-conference.de

Wir denken Dokumente neu!

Integriere die Bearbeitung, Anzeige, Erstellung und Automatisierung digitaler Dokumente in deine .NET-Anwendungen.





Zum Glück

Zufriedenheit und Wertschätzung sind starke Triebfedern bei allem, was wir tun. Das gilt nicht zuletzt auch für unseren Arbeitsalltag.

Was war das gestern nur wieder für ein Meeting! Zweieinhalb Stunden endlose Diskussionen, und am Ende nur wachsweiße Ergebnisse. Ob sich danach etwas ändert und wir die Probleme im Projekt jetzt endlich in den Griff bekommen?

Solche und viele ähnliche Situationen sind keine Seltenheit. Früher, so mein Gefühl, wurde eher darüber hinweggeschaut. Nach dem Motto: War so, ist so, wird so bleiben, wir bekommen das schon hin. Augen zu und durch.

In meinem allerersten Editorial für die dotnetpro hatte ich geschrieben, dass die Anforderungen an Softwareentwickler stetig wachsen. Nun, an diesem Umstand hat sich nichts geändert, im Gegenteil – die Anforderungen steigen weiter. Was sich aber verändert hat, ist der Umgang damit: Immer häufiger wird nicht mehr einfach hingenommen, dass die Dinge so sind, wie sie früher waren, dass sie aus dem Ruder laufen. Nein, inzwischen wird das Kind beim Namen genannt. Und das mitunter recht deutlich.

„Der wertschätzende Umgang mit den Mitarbeitern zählt zu den Schlüsselfaktoren für Unternehmen.“

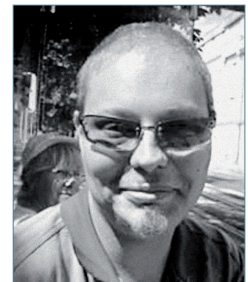
Dieses Umdenken, das ich in den letzten Jahren wahrgenommen habe, hat meiner Einschätzung nach zwei wesentliche Ursachen: Die Erkenntnis, dass der zunehmende Druck im beruflichen Alltag zu negativen Auswirkungen führt, die so nicht mehr akzeptabel sind. Und eine wachsende Zahl vor allem jüngerer Softwareentwicklerinnen und -entwickler, die sich vom absoluten Primat dessen, was sie Tag für Tag leisten (sollen), ein Stück weit emanzipieren und für Bedingungen eintreten, die zu einer besseren Balance zwischen den Anforderungen einerseits und eigenen Bedürfnissen andererseits führen.

Weil das so wichtig ist, widmet sich der Schwerpunkt der dotnetpro diesem Thema. Seine Beiträge kreisen dabei um zwei aus meiner Sicht zentrale Aspekte: Was macht uns eigentlich „happy at work“, und welche Vorteile hat das? Und: Wie sorgen wir selbst in unserem Arbeitsumfeld für Bedingungen, die uns Zufriedenheit geben und motivieren?

Ich wünsche Ihnen eine schöne Zeit mit der neuen dotnetpro!

Herzlichst

Fernando Schneider
Chefredakteur dotnetpro



Klaus
Aschenbrenner

erklärt in Teil 8 seiner Serie, wie Sie asynchrone Programmierung in Rust realisieren können (S. 113)



Christian
Jacob

sorgt in Teil 6 seiner MonoGame-Serie für eine Wiedergeburt des Players und lässt ihn endlich tanzen (S. 122)



INHALT

Schwerpunkt

Happiness at Work

Die Kraft hinter erfolgreichen Transformationen

Die unsichtbare Blockade

Zwischen Technik und Emotionen

Planung

Time-out: Just don't blame

Wie wir unproduktive Schuldzuweisungen hinter uns lassen

Ich packe meinen Koffer ...

Sleepy-Hollow-Architektur, Teil 2

Frontend

Gestaltung mit System

Designsysteme und .NET MAUI

Core

NETirol: PDFs generieren

QuestPDF

Abseits des Trampelpfads

.NET Aspire in Legacy-Applikationen einsetzen

Rx.NET Deep Dive

Reaktiv programmieren, Teil 3

Fehlersuche

Messaging mit Wolverine und .NET, Teil 6

Frameworks: Der Code im Hintergrund

Webanwendungen mit CodeBehind

KI

Über Sprache lernen

KI und Data, Teil 2

6

15

28

30

Foto: Shutterstock / GoodIdeas; Aleksandr_Lysenko



Happiness at Work und Reflexion sind zwei Schlüsselfaktoren für den Erfolg von Softwareprojekten.

6

40

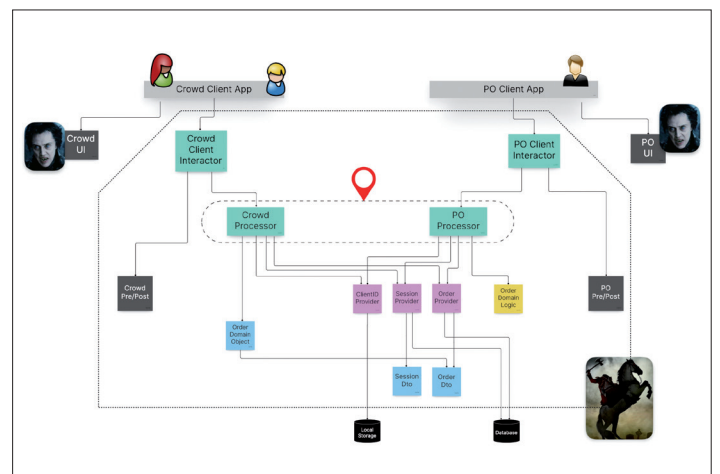
48

51

56

64

68



Slicing und Sleepy Hollow spannen den Bogen von der Anforderungsskizze hin zum Architekturmuster.

30

Experten in dieser Ausgabe



Christian Giesswein zeigt Ihnen, wie Sie mithilfe von QuestPDF in C# PDF-Dokumente erstellen können.

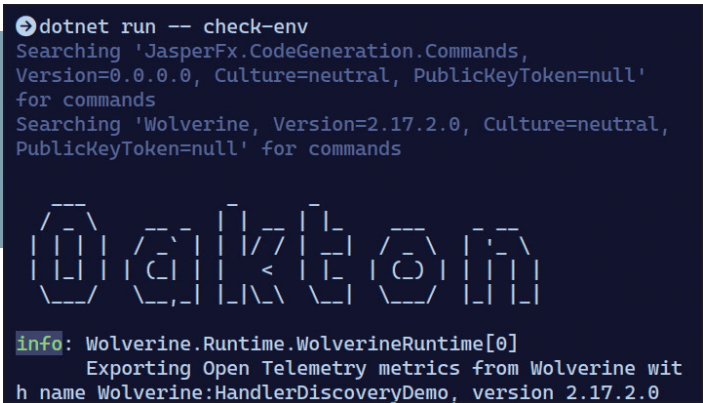
48



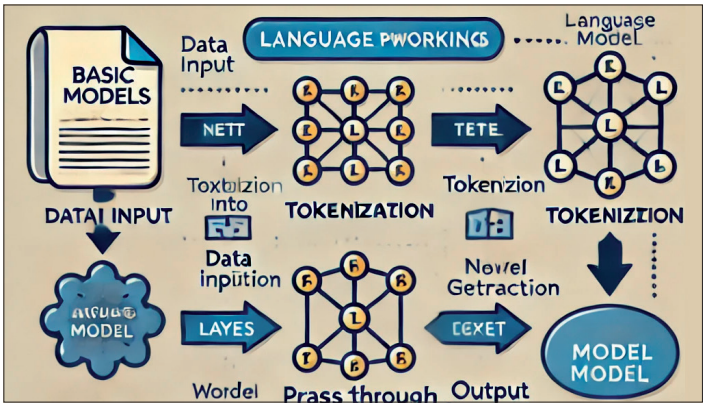
Daniel Basler entwickelt Graphen-Algorithmen für die Tiefen- und die Breitensuche in C#.

100

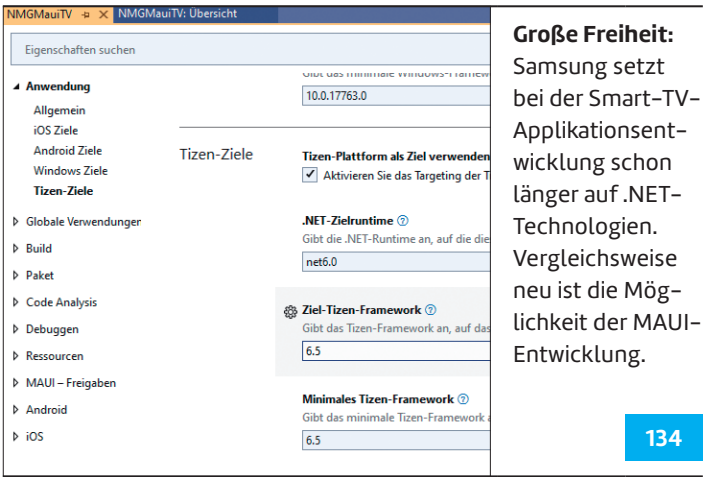
74



Fehlersuche: Was zu tun ist, wenn Wolverine einmal nicht so funktioniert, wie man es eigentlich erwarten würde. 64



Sprachmodelle sind die Basis für leistungsfähige KI-Tools. Ein Überblick über die verschiedenen Modelle und Ansätze. 74



Quellcode gefällig?
Sie möchten den Quellcode eines Artikels ausprobieren? Oder den Artikel als PDF herunterladen?
Dann achten Sie auf den dnpCode im Autorenkasten am Ende des Artikels. Hängen Sie diesen Code an www.dotnetpro.de/ an. Dieser Link führt Sie zur passenden Seite im dotnetpro-Archiv.

Backend

Data Access: Verlangsamte Fahrt
Neuerungen in System.Text.Json 9.0 82

Handlich verpackt
DACPAC und BACPAC als Transfer-Dateiformate für den SQL Server 87

Tools

Tresore per Schnittstelle
Ein Passwortmanager aus Entwicklersicht 92

Workout

Knoten- und Kantenkunde
C#-Graphen-Algorithmen 100

Soll es Block oder Text sein?
Lego Education Spike Prime im Blick 106

Asynchron
Rust-Kurs, Teil 8 113

Wie neu geboren, und das im Loop
MonoGame für Einsteiger, Teil 6 122

Die große Freiheit
.NET MAUI auf Samsung-Smart-TVs 134

Editorial 3

Lernstoff 90

Arbeitsmarkt 140

Impressum / Inserentenverzeichnis 144

Anbieterverzeichnis 145

Vorschau 146



Foto: Shutterstock / GoodIdeas

DIE KRAFT HINTER ERFOLGREICHEN TRANSFORMATIONEN

Happiness at Work

Wie arbeitet man an Zufriedenheit, Glück und Wohlbefinden im Job?

Softwareentwicklung ist für viele, die sie betreiben, Hobby und Beruf gleichermaßen. Doch manchmal sind die Arbeitsumgebungen nicht gut geeignet, Glücksgefühle bei Entwicklern hervorzurufen. Dagegen kann jeder Einzelne etwas unternehmen. Und den Firmen kann das auch finanziell sehr nutzen.

Vor ungefähr vier Jahren haben mein Team und ich begonnen, mit unseren Kunden an „Happiness at Work“ zu arbeiten. Der Kundenauftrag ist immer irgendeine Art von Transformation, von Veränderung, und natürlich immer zum Besseren. Manchmal nennt der Kunde es Agile Transformation, manchmal Digitale Transformation, manchmal Teamentwicklung, manchmal Leadership-Beratung, manchmal Innovationsentwicklung. Im Auftrag ist also gar nicht von „Happiness“ die Rede. Wie gelangen wir dann zu diesem Thema?

Es gibt mehrere konkrete Auslöser, die man ungefähr so zusammenfassen kann: Wenn eine Firma sich ein paar Jahre lang an einer Transformation versucht, liegen irgendwann genug Lösungsansätze auf dem Tisch. Bleibt eine nachhalti-

ge, tiefgreifende Verbesserung trotzdem aus, weil niemand ins Handeln kommt, entsteht Frustration. Hier beginnen wir unsere Arbeit rund um „Happiness“. Am Startpunkt sind oft Teams anzutreffen, die nicht besonders glücklich wirken. Wenn wir in dieser Situation direkt fragen: „Was braucht ihr, um glücklicher zu sein?“, sind die Antworten oft vage, sie kommen nur spärlich, und man weiß nicht so recht, wie man daraus eine Verbesserung ableitet. Also haben wir die Fragestellung verändert: „How to destroy happiness at work?“ Auf diese Frage prasseln die Antworten nur so auf uns ein. Was wir aus den Beiträgen und den daran anschließenden Verbesserungsmaßnahmen im Kontext von Softwareentwicklungsteams lernen [1], bildet die Grundlage für diesen Artikel.

Ist Happiness relevant?

Die Frage „Ist Happiness relevant?“ wird entweder als triviale oder als verrückte Frage angesehen. Die einen antworten etwas wie: „Natürlich ist das wichtig; schließlich verbringe ich acht Stunden des Tages mit der Arbeit.“ Die anderen, und

dazu gehören manche Geschäftsführer, Unternehmer und Beschäftigte im oberen Management, sagen: „Ein Unternehmen hat andere Aufgaben, als sich um das Glück der Mitarbeiter zu kümmern. Wir sorgen für Arbeit und Geld und sichern damit den Lebensunterhalt unserer Belegschaft. Um das eigene Glück sollte sich jeder selbst kümmern.“

Beide Perspektiven haben ihre Berechtigung. Doch sind sie miteinander vereinbar? Was hat das Thema mit Softwareentwicklung zu tun? Und warum ist die ganze Zeit vom englischen „Happiness“ die Rede? Der vorliegende Artikel beantwortet diese Fragen und gibt darüber hinaus Anregungen, was man als Softwareentwickler tun kann, um die Happiness at Work in der eigenen Firma zu erhöhen.

Dass es für jeden Einzelnen wichtig ist, sich mit und bei der eigenen Arbeit wohlfühlen, liegt auf der Hand. Doch wie sieht es mit der Relevanz des Themas für Arbeitgeber und Unternehmen aus? Ist es aus wirtschaftlicher Sicht eine gute Idee, dass sich eine Firma auch um das Wohlbefinden der Mitarbeiter kümmert?

Die kurze Antwort lautet: Ja. Und für eine lange Antwort gibt es eine ganze Reihe an wissenschaftlich fundierten Erkenntnissen. Die Übertragbarkeit der Ergebnisse aus der Forschung auf das eigene Unternehmen kann man durchaus hinterfragen. Sind Ergebnisse aus dem produzierenden Gewerbe auch für die Softwareentwicklung gültig? Sind Studien aus anderen Ländern auch für Deutschland relevant? Nicht immer sind die Antworten klar. Manche Aussagen dürften aber offensichtlich grundsätzlich gelten. So haben Studien etwa gezeigt, dass die Gesundheitskosten in Unternehmen sinken, in denen die Mitarbeiterzufriedenheit hoch ist. Das Forschungsunternehmen Gallup kommt in konkreten Fällen auf um 62 Prozent geringere Kosten [2]. Auch wenn dieser Wert, der im US-amerikanischen Kontext ermittelt wurde, nicht pauschalisierbar ist, sollte allein die Aussicht auf Senkung von Fehlzeiten einen großen Anreiz für die Unternehmensführung liefern.

Ebenfalls nachgewiesen wurde bereits, dass die Zahl der Qualitätsmängel bei der Produktentwicklung in einem Umfeld mit hoher Mitarbeiterzufriedenheit signifikant sinkt. In der Softwareentwicklung ist Qualität ein Thema mit hohem Stellenwert. Wenn die Fehlerzahl steigt, rufen viele Firmen reflexartig nach besserem Testing, ausgefeilteren Teststrategien, größerer Testabdeckung und dergleichen mehr. Vielleicht erschafft man dann die Position des Test-Managers oder will Test-Driven Development (TDD) einführen. Aber wer fragt nach den Ursachen, wenn die Bugzahl nach oben schnell? Und wer kommt auf die Idee, dass es vielleicht die 23 Stunden an langweiligen Regelmeetings pro Woche sind, die den Entwicklern zu schaffen machen? Welche Wirkung hat der ständige Druck, zu willkürlichen Terminen unklare oder gar sinnlose Anforderungen zu implementieren? Wie viele Bugs entstehen, weil man als Entwickler keine Zeit bekommt, technische Schulden abzubauen, und stattdessen Features

quasi am Fließband produzieren soll? Es sind keine Studien bekannt, die Antworten für alle Unternehmen in allen Ländern liefern. In konkreten Fällen konnten Studien jedoch eine Reduzierung der Qualitätsmängel in der Produktentwicklung um 40 Prozent nachweisen, wenn die Mitarbeiterzufriedenheit steigt.

Firmen, die auf das Wohlbefinden der Mitarbeiter achten, weisen eine höhere Wertschöpfung (in einer Studie: +6,6 Prozent [3]) und eine höhere Produktivität (+12 Prozent) auf [4].

Wenig überraschend ist, dass die Mitarbeiterfluktuation erheblich sinken kann. In einem Fall konnte eine Kostensenkung von 52 Prozent durch verringerte Fluktuation, weniger Onboarding und weniger Einarbeitung nachgewiesen werden [2].

Weitere Unternehmenszahlen, die einen positiven Zusammenhang mit der Happiness at Work zeigen, sind steigende Dividende, steigender Aktienkurs, höhere Innovationstätigkeit und -erfolg. Es gibt also mehr als genug betriebswirtschaftliche Gründe, sich des Themas anzunehmen. Wer Happiness at Work für ein „softes“ Thema hält, liegt falsch. Vereinfacht ausgedrückt: Wer mehr Geld machen will, sollte sich um das Wohlbefinden der Mitarbeiter kümmern.

Happiness, Joy, Fun

Was heißt eigentlich Happiness? Fangen wir mit der Sprache an. Das Wort hat im Deutschen ein Problem. Es wird nämlich oft missverstanden. Manche übersetzen es mit Spaß oder Freude, was im Englischen wohl eher Joy und Fun entspricht. Wenn man Happiness mit Glück oder „Glücklichsein“ übersetzt, kann man zu Recht fragen, ob das nicht jedem Menschen selbst überlassen sein sollte. Arbeit muss nicht immer angenehm sein, lautet eine weitverbreitete Ansicht. Manchmal hören wir, dass Arbeit doch wohl mit Anstrengung, gar Kampf zu tun hat. Manch ein Azubi dürfte auch in unserer Zeit den althergebrachten Spruch „Lehrjahre sind keine Herrenjahre“ zu hören bekommen. Wenn das Projekt unter Druck gerät, meinen sogar ganz vernünftige Menschen, man müsse eben „die Pobacken zusammenkneifen“, und nennen als Standardlösung: „Da muss man durch.“

Ich räume gerne ein, dass es konkrete Situationen gibt, in denen diese Äußerungen ein Fünkchen Wahrheit enthalten. Aber drei wesentliche Punkte sind: Erstens gibt es hier ein fundamentales Missverständnis, was „Happiness at Work“ bedeutet. Zweitens steckt, wenn ein Projekt unter Druck gerät und deshalb mehr gearbeitet werden soll, automatisch die Annahme darin, dass vorher nicht genug gearbeitet worden sei und dass mehr Arbeit das Projekt rette. Ist das überhaupt richtig? Und drittens schwingt in den genannten Aussagen immer die grundlegend falsche Auffassung mit, dass Happiness at Work bedeute, die Mitarbeiter machten sich ein nettes Leben zum Schaden des Unternehmens.

Ich verwende gerne folgendes Beispiel, um klarzumachen, dass Happiness, wie ich es meine, nicht so eindimensional ►

„Auf die Frage ‚How to destroy happiness at work?‘ prasseln die Antworten nur so auf uns ein.“

ist, wie es manchmal verstanden wird: Angenommen, man läuft einen Marathon, und bei Kilometer 38 kommt diese wirklich gemeine Steigung. Die Muskeln brennen. Jede Faser des Körpers schmerzt. Dann ist dieser Augenblick kein Spaß und keine Freude. Aber es ist Happiness. Es ist Erfüllung. Vielleicht auch erst am Ende der Steigung. Doch egal wie es sich anfühlt: Man macht es freiwillig. Und man will es wieder tun.

Ich könnte nun versuchen, das Wort „Happiness“ ins Deutsche zu übersetzen. Mir kommen Wohlbefinden (Well-being), Erfüllung (Fulfillment) und natürlich auch Glück oder Glücklichkeit in den Sinn. Doch weil Menschen unter Glück und Glücklichkeit verschiedene Dinge verstehen, bleibe ich gerne bei Happiness, um die Unschärfe des Begriffs zu akzeptieren. Wer die (englischsprachige) Fachliteratur zu diesem Thema liest, sollte unbedingt darauf achten, welches Verständnis des Begriffs die jeweiligen Autoren haben. Für manche Autoren ist Happiness „Glück“, und sie stellen fest, dass auch noch andere Aspekte wichtig sind. Dann ist Happiness nicht ausreichend. So kann „Glücklichkeit“ den schmerzvollen Moment im obigen Marathon-Beispiel nicht erklären. Andere Autoren verstehen – so wie ich – unter Happiness ein vielschichtiges Phänomen, das aus mehreren Bausteinen besteht. In der Arbeit mit unseren Kunden verwenden wir deshalb folgende vier Säulen, die für Happiness at Work unverzichtbar sind und im Wesentlichen auf die Darstellung von Happiness at Work an der Uni Berkeley zurückgehen [5]:

- Zweck, Sinn, Antrieb (Purpose)
- Engagement
- Resilienz
- Integration, Vertrauen und Kooperation (Kindness)

Wenn Purpose fehlt

Softwareentwicklern fehlt es nicht an intrinsischer Motivation. Die Kreativität von Softwareentwicklung verstehen viele

Außenstehende nicht. Sie denken, Software könne quasi am Fließband produziert werden. Es ist jedoch eine schöpferische Tätigkeit, die ihren Reiz unter anderem daraus zieht, etwas zu erschaffen, das funktioniert und läuft. Im kreativen Prozess ist die Kommunikation zwischen allen Beteiligten essenziell. Damit meine ich keineswegs nur den Austausch zwischen den Entwicklern, sondern ganz im Gegenteil den zwischen Entwicklern und allen anderen, die an der Entstehung beteiligt sind (Stakeholder, Kunden, Business-Abteilungen und andere). Eine weitreichende Darstellung dieser Sichtweise findet sich bereits vor rund 30 Jahren in einem Lehrbuch des MIT, das auch die TU Dortmund in den 1990er-Jahren für die Einführung in Programmierung verwendete. Die Autoren schreiben im Vorwort [6]:

„We want to establish the idea that a computer language is not just a way of getting a computer to perform operations but rather that it is a novel formal medium for expressing ideas about methodology. Thus, programs must be written for people to read, and only incidentally for machines to execute.“

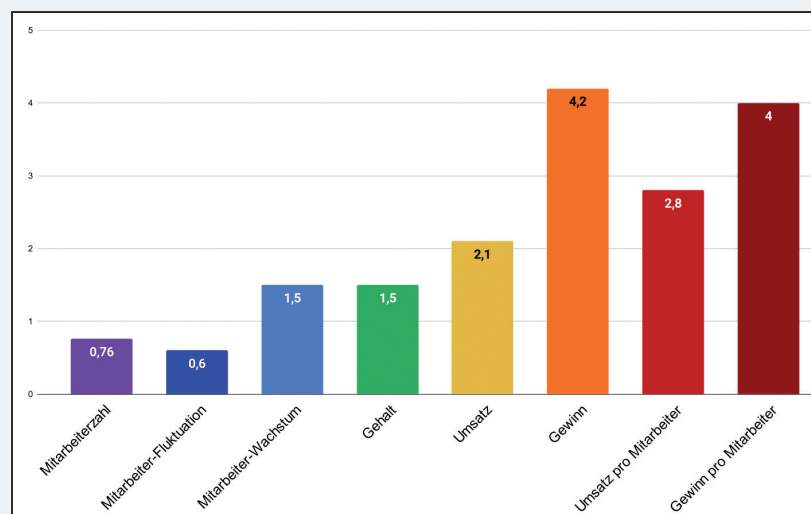
Bewegungen wie Clean Code zeigen, dass es Entwicklern darüber hinaus um eine innere Schönheit ihrer Programme geht. Eine Erzählung über Steve Jobs besagt, dass er die Elektronikentwickler eines neuen Macs noch einmal ins Labor geschickt hat, weil ihm der Verlauf der Leiterbahnen auf dem Mainboard nicht ästhetisch genug war [7]. Die meisten Nicht-Tekkies halten das für „weird“. Clean-Code-Entwickler verstehen das.

Begegnen solche Menschen nun Managern, die ihre Abteilungen wie Maschinen betrachten, auslasten und monitoren, gelegentlich nachjustieren und bei Bedarf mal ein Ersatzteil austauschen wollen, ist der Konflikt vorbestimmt.

Eine häufige Ursache von Problemen in der Softwareentwicklung besteht im Aufbau von Unternehmen. Früher war es üblich, von Abteilungen zu sprechen. Heute spricht man meist von Teams, auch wenn die Menschen, die dazugehö-

● Employee Experience beeinflusst zentrale Kennzahlen von Firmen

Unternehmen, die nicht auf die Employee Experience achten, erhalten in der nebenstehenden Grafik in allen Kategorien den Grundwert 1 zugeordnet. Die Balken der Chart-Darstellung zeigen die entsprechenden Vergleichswerte für Firmen, die Employee Experience als wichtig erachten. Diese Firmen sind kleiner (76 Prozent der Größe) als andere betrachtete Unternehmen, weisen eine geringere Fluktuation (60 Prozent) auf, wachsen schneller (150 Prozent), bezahlen ihre Mitarbeiter besser (150 Prozent) und machen mehr Umsatz (210 Prozent) sowie Gewinn (420 Prozent). Die beiden letztgenannten Werte sind auch pro Kopf höher (280 Prozent beziehungsweise 400 Prozent).



Quelle: Jacob Morgan [16]

ren, nicht oder nicht viel zusammenarbeiten. Die Umbenennung ist insofern schade, als das Wort „Abteilung“ wenigstens seinen Zweck verrät: Menschen voneinander abzutheilen. Der Unternehmensaufbau hat für den Einzelnen großen Einfluss darauf, worin die Arbeit besteht und wie sie sich anfühlt. Wenn es eine Entwicklungsabteilung und eine nachgeschaltete Testabteilung gibt, ist die Chance, dass man Test-Driven Development betreibt, ziemlich gering. Nicht dass TDD pauschal der richtige Ansatz ist. Aber man sollte zumindest die Option haben, wenn man sie braucht.

Die oben zitierte Auffassung, dass eine Programmiersprache ein Medium zum Austausch von Ideen ist, findet sich in der Kommunikation zwischen Business-Abteilungen und der Entwicklung nicht wieder. Wenn Business von Entwicklung getrennt ist, erwartet man offensichtlich nicht, dass die Entwickler etwas vom Geschäft verstehen (sollen). Die Folge ist oft, dass „das Business“ seine Requirements über den Zaun wirft und erwartet, dass die Entwicklung daraus Programmcode macht. Die Organisation wird dann sehr leicht eine „Feature Factory“ – ohne erkennbaren Purpose.

In einem TED-Beitrag mit dem Titel „8 lessons on building a company people enjoy working for“ betont Patty McCord [8]: *„Everyone in your company should understand the business. Now, based on the assumption that we’ve got smart adults here, the most important thing we can teach them is how our business works. When I look at companies that are moving fast, that are really innovative and that are doing amazing things with agility and speed, it’s because they’re collaborative.“*

Wenn wir diese Kollaboration durch Abteilungen und Silos verhindern, sind die Konsequenzen für alle Beteiligten unschön, auch für das Unternehmen. Die Entwickler haben keine Ahnung, an was sie eigentlich arbeiten. Oder genauer: Für wen lösen sie welches Problem? Welches Ziel ist zu erreichen? Wem helfen sie? Wer ist die Person, die vielleicht mal danke sagen würde, wenn sie nur Kontakt zu den Entwicklern hätte?

Statt eines Ziels bekommen die Entwickler Arbeitspakete. Die Requirements sind früh mit Abnahmekriterien vollgepackt, die womöglich noch jemand ohne technischen Sachverstand formuliert hat. Die oben genannte Kreativität hat keine Überlebenschance.

Doch glücklicherweise sind Softwareentwickler hart im Nehmen und kennen das Spiel ja – getreu dem Motto: „Wenn ich schon nicht weiß, was ich mache, mache ich es wenigstens richtig gut.“ Womit wir wieder bei Clean Code wären. Man spricht in der Abteilung (aka „Team“) über Coding Guidelines, Coding Principles, Quellcode-Konventionen, Code-Metriken und dergleichen.

Auf der Business-Seite platzt da schon mal jemandem der Kragen. Vorwürfe werden laut, die Entwickler würden den Maschinenraum polieren. Aha! Der Quellcode ist also der Maschinenraum, und der darf unordentlich, ölverschmiert und dreckig sein. Na, wenn das nicht motivierend ist! Dass mit mangelnder Fürsorge für den Quellcode auch die Grundlage für technische Schulden oder gar den technischen Bankrott gelegt wird, sei nur am Rande erwähnt. Ohne Zwei-

fel kann man zu viel am Quellcode oder am ganzen Produkt arbeiten. Um das zu erkennen, muss man aber das Business verstehen, die Ziele kennen und klare Rückmeldungen über Erfolg oder Nichterfolg der eigenen Arbeit haben.



MITARBEITER-FLUKTUATION

-52%

Wohlbefinden am Arbeitsplatz senkt die Kosten durch Mitarbeiter-Fluktuation

Eine Untersuchung hat eine Senkung der Kosten durch Mitarbeiter-Fluktuation von 52 Prozent für Unternehmen nachgewiesen, die Workplace Happiness als Ziel mit einbeziehen.

Quelle: Gallup [2], Grafik: Kutura [1]

Was kann man als Softwareentwickler dagegen tun?

So weit zur Problemstellung und ihren Auswirkungen. Wenn ich mich als Entwickler in solch einer Feature Factory wiederfinde, was kann ich tun?

Wenn nicht klar ist, wofür ein Ticket gut ist, sollte man nachfragen. „Warum machen wir das? Wozu dient es? Wer hat einen Nutzen davon?“ Wichtig ist, darauf zu achten, dass man eine „echte“ Antwort bekommt. Wie sieht das Gegenteil aus?

In vielen Firmen wird die Arbeit durch sogenannte Prozesse geordnet. Dabei entsteht in erstaunlich vielen Fällen ein Cargo Cult [9]. Kennzeichnend ist, dass es wichtiger ist, den Prozess und die Regeln einzuhalten, als ein Ziel zu erreichen.

Bei der Formulierung von User Stories tritt oft ein Cargo Cult in Erscheinung. Während wirkliche User Stories sehr wertvoll sind, sind es Tickets, die nur so genannt werden und einem Muster folgen, nicht.

Ein beliebtes Muster für User Stories lautet: „Als [Person X] möchte ich [Y], um [Z] zu erreichen.“ Der letzte Teil wird manchmal weggelassen, was den Satz sogar als Muster unbrauchbar macht. Das Format hat seine Vorzüge. Aller- ►

dings wird die Story vollkommen sinnlos, wenn man die Variablen X, Y und Z nur pro forma füllt. Zwei Beispiele:

Beispiel 1: „Als Kunde möchte ich einen Login-Button rechts oben haben, um mich schnell einloggen zu können.“

Auf den ersten Blick sieht der Satz ganz gut aus. Zu klären ist hier die Frage: Welcher Kunde möchte das so haben? Gibt es eine Mehrheit von Kunden? Und was hat es mit dem „schnell einloggen“ auf sich? Geht es um wenige Klicks oder eine schnelle Response des Systems? Für Letzteres bringt es vielleicht mehr, wenn das Backend-System schneller reagiert. Für wenige Klicks könnte ein „Sign-in with Google“ oder ein „Eingeloggt bleiben“, wie es einige Webseiten anbieten, mehr bringen.

Vielleicht ist die Story aber nur eine Erfindung des Product Owners oder irgendeines Stakeholders, der/die einfach den Login-Button rechts oben haben möchte. Durch Nachfragen kann man diesen Sachverhalt zumindest sichtbar machen.

Beispiel 2: „Als Servicetechniker möchte ich 1 GByte Speicherplatz auf dem Gerät zur Verfügung haben.“

Hierbei handelt es sich um ein echtes Beispiel, das mir bei einem Kunden im Bereich Embedded Software Development begegnet ist. Diese sogenannte „User Story“ nennt nicht einmal pro forma den Zweck oder Nutzen. Das Team, das diese „Story“ bekommen hat, konnte nie mit dem „Servicetechniker“ sprechen; kein Wunder, er hätte auch nichts über 1 GByte Speicherplatz auf dem zu entwickelnden Gerät sagen können. Es handelte sich einfach um eine unklare, wirre Story, die nicht einmal der Product Owner erklären konnte. Man hatte ihm das Ticket vorgesetzt, so wie er es dem Entwicklerteam vorgesetzt hat. Das Team hat glücklicherweise nicht locker gelassen, nachgefragt und gebohrt. Nach circa zwei Wochen (sic!) war klar, was eigentlich das Ziel war. Die Geräte, die der Kunde baut, werden im Betrieb einer regelmäßigen Wartung unterzogen. Das Wartungsprotokoll der früheren Gerätegeneration wurde immer in ein Buch eingetragen. Die neue Generation kommt ohne Buch aus, und das Protokoll wird im Gerätespeicher abgelegt. Dafür braucht es Speicherplatz. Der Speicherbedarf von 1 GByte war dafür übrigens viel zu hoch angegeben. Durch Klärung des Ziels des Tickets konnten die Entwickler die Aufgabe mitgestalten und eine sehr gute Lösung entwickeln. Eine bessere Formulierung für diese Story hätte viel Zeit und Mühe gespart. Sie lautet zum Beispiel:

„Als Servicetechniker möchte ich die von mir durchgeführten Arbeiten protokollieren, sodass der Wartungsverlauf zu einem späteren Zeitpunkt nachvollziehbar ist. Dazu möchte ich meine Notizen auf dem Gerät speichern, sodass man sie nach dem Speichern nicht mehr manipulieren kann.“

Wenn ich als Entwickler solche vermeintlichen User Stories bekomme, sollte ich nachhaken, bis ich verstanden habe, worin der Nutzen besteht. Erst aus dem Erzielen eines Nutzens für einen echten Menschen entsteht meine eigene Zufriedenheit.

Im letztgenannten Beispiel konnte der Product Owner die Fragen des Teams nicht beantworten. Natürlich kann der Product Owner in so einem Fall „die Frage mitnehmen“ [10]. Im Allgemeinen bin ich jedoch ein Freund von direkter Kommunikation. Wenn ich in einem Meeting sitze und mein Ge-

sprächspartner „nimmt die Frage mit“, heißt das, dass ich später in ein weiteres Meeting gehen muss, um mir die Antwort anzuhören. Wie viel besser wäre es doch, wenn ich gleich mit dem richtigen Gesprächspartner reden könnte! Es gibt Firmen, die etwas so Einfaches wie ein Gespräch durch Organisationsstrukturen verhindern oder zumindest behindern. In einem extremen Fall, den ich erlebt habe, hat eine Mitarbeiterin der UX-Abteilung ein Jahr lang versucht, ihre Erkenntnisse aus User-Tests dem zuständigen Product Owner direkt mitzuteilen. Verhindert wurde es durch die Vorgesetzte mit dem Hinweis auf die etablierten Kommunikationskanäle. Der PO wusste nicht einmal, dass es eine UX-Abteilung gibt. Herausgekommen ist die Sache, als die UX-Mitarbeiterin das Handtuch geworfen und gekündigt hat.



Aus meiner Sicht hätten UX und Product Ownership gar nicht in getrennten Abteilungen verortet sein sollen. Trotzdem rate ich einzelnen Teammitgliedern meist davon ab, die bestehenden Strukturen, also den Zuschnitt von Abteilungen und Bereichen, infrage zu stellen. Nicht, weil es für diese Firmen nicht sinnvoll wäre, Strukturen zu verändern, sondern weil es aus der Entwicklerposition ein sehr mühevolleres Unterfangen ist, an den Strukturen zu rütteln. Neben den formalen Strukturen, die in einem Firmenorganigramm zu finden sind, gibt es die tatsächliche „Struktur“. Sie wird bestimmt durch die Kommunikation im Unternehmen: Wer redet mit wem? An dieser Stelle kann man gut ansetzen:

- „Mit wem müsste mein Team sprechen, um die Zielsetzung hinter unseren Tickets besser zu verstehen?“ Diese Perso-

nen sind gute Gesprächspartner in einem Ticket-Refinement.

- „Welche Stakeholder können uns aus erster Hand berichten, welche Fortschritte unsere Arbeit für das Business liefert?“ Diese Personen sind gute Gesprächspartner in einem Review.
- „Mit welchen Teams sollten wir zusammenarbeiten, um das große Bild zu sehen?“ Diese Frage eignet sich für Komponententeams, die ihr Produktverständnis ausbauen wollen.

Alle diese Fragen kann man in einem Satz zusammenfassen: Es geht darum, über Grenzen (Team, Abteilung, Silo) hinweg zu kommunizieren. Wie einfach man das als Entwickler tun „darf“, hängt von der konkreten Situation ab. Manchmal gibt es Vorgesetzte oder Product Owner, die gerne Gatekeeper sind; siehe oben. Sie sind das einzige Sprachrohr des Teams nach außen. 100 Prozent der Kommunikation laufen über sie. In so einem Fall dürfte ein Konflikt vorprogrammiert sein, wenn man als Entwickler einfach mal ein paar Stakeholder zum Review einlädt. Hier empfiehlt es sich natürlich, vorab mit dem PO zu sprechen, auf ihn/sie einzuwirken und den Nutzen des direkten Austauschs klar zu machen. Gelingt das nicht, bleibt noch der Weg als „mutiger Rebell“. Dazu gehört auch, sich nicht ganz exakt an die bestehenden Regeln zu halten. Es geht darum, trotz der Regeln und gegebenenfalls Verbote etwas Positives zu erreichen. Und manchmal muss man dafür Regeln überschreiten. Den Begriff „mutiger Rebell“ habe ich einem TED-Talk entliehen, dessen Titel lautet: „Plucky rebels: Being agile in an un-agile place“ [11]. Zu den Ratschlägen des Vortrags gehört, manche Aktivitäten nicht an die große Glocke zu hängen, sondern für eine gewisse Zeit im kleinen Kreis zu behandeln. Konkret kann das bedeuten, die Stakeholder nicht zum Review einzuladen, um dem Konflikt mit dem Gatekeeper aus dem Weg zu gehen. Stattdessen trifft man sich mit ihnen zu einem Gespräch unter vier Augen, vielleicht auch beim Mittagessen.

Wie die Firmenkultur Engagement reduziert

Der zweite unverzichtbare Baustein für Happiness at Work heißt „Engagement“. Es ist wichtig, mit Energie und Hingabe für die Arbeit unterwegs zu sein. Man kann nun einwenden, dass es sich ja wohl um eine innere Haltung jedes Einzelnen handelt. Folglich können das Unternehmen und das Management nichts dafür oder dagegen tun. Das ist nicht richtig.

Stellen wir uns einmal einen Entwickler namens Bob vor. Er liest eine Stellenanzeige, die voller schöner Aussagen ist. Ein paar Stichworte:

- verantwortungsvolle Aufgabe
- selbstständige Arbeitsweise gewünscht, viele Freiheiten geboten
- kurze Entscheidungswege
- wertschätzender Umgang
- und mehr in dieser Art

Bob ist begeistert, und die Interviews laufen gut. Er will den Job und bekommt ihn. Für das Onboarding wird ihm eine Pa-

tin an die Seite gestellt. Sie stellt schon in der ersten Woche sicher, dass alle Regel-Meetings an ihn weitergeleitet werden. Nach ein paar Tagen zeigt sein Kalender nur noch wenige Lücken. Er lernt schnell alle Kolleginnen und Kollegen kennen. Schrittweise arbeitet er sich in den Technologie-Stack ein, und es dauert nur ein paar Sprints, bis er eigenständig arbeiten kann.

Im Planning finden regelmäßig viel zu viele Tickets ihren Weg ins Sprint-Backlog. Als er das infrage stellt, weist der PO auf den Termindruck hin und darauf, dass das Projekt schon verspätet ist. Ein Release war für letztes Jahr geplant, und das Management hat jetzt ein Auge auf das Projekt. Die Statistiken der zurückliegenden Sprints zeigen, dass das Team sein Commitment noch nie geschafft hat.

Mit dem Fortschreiten des Projekts nimmt die Zahl der Krisenmeetings zu. Bobs PO muss nun täglich dem Steering Committee reporten. Zur Unterstützung stellt das Management dem Team erfahrene Entwicklungs- und Testleiter aus anderen Bereichen zur Verfügung. Sie beginnen, die Tickets immer kleinteiliger zu schneiden und bis auf technische Ebene zu detaillieren. Bob versteht die Zusammenhänge der Tickets immer weniger. Das Team fordert seit Langem Veränderungen, aber die Entscheidungswege sind sehr lang, und seit Monaten wurde nichts entschieden. Seine Bereitschaft, „mehr“ zu geben, nimmt immer mehr ab. Er macht, was man ihm sagt, und stellt das eigenständige Denken ein.

So oder so ähnlich kann man hohe Leistungsbereitschaft, Motivation und Engagement zunichtemachen. Eigene Umfragen zeigen, dass es häufige Muster gibt, die das Engagement der Mitarbeitenden beschädigen [1]. Dazu gehören:

- Micromanagement
- Zu viele Meetings
- Zu viele Arbeitspakete
- Man darf nicht an bevorzugten Themen arbeiten
- Entscheidungen bleiben in zu vielen Hierarchieebenen stecken.
- Zweck und Nutzen der gestellten Aufgaben werden nicht erklärt.
- Keine Bereitschaft zur Veränderung
- Hoher Leistungs- und Konkurrenzdruck

Wer sich in einer solchen Umgebung wiederfindet, sollte unbedingt handeln; zum eigenen Wohlbefinden und nebenbei auch im Firmeninteresse. Andernfalls droht eine pathologische Dienst-nach-Vorschrift-Haltung.

Wie beim Purpose bereits genannt, ist es unverzichtbar, zu wissen, welchem Ziel eine Aufgabe dient. Wem nützt das Ergebnis? Von wem kann ich aufrichtigen Dank für meine Arbeit bekommen? Langfristig kann Engagement keinen Bestand haben, wenn eine echte Wertschätzung fehlt. Der Psychologe Dan Ariely berichtet in einem TED-Talk von einer Studie, in der die Teilnehmer aus Lego-Steinen Figuren bauen und dafür bezahlt werden [12]. Der Preis wird mit jeder Figur gesenkt, bis die Teilnehmer aussteigen. Bemerkenswert ist, dass das Engagement deutlich sinkt, wenn der „Auftraggeber“ die Figuren unmittelbar nach dem Zusammenbau wieder auseinandernimmt. Offensichtlich macht es in dieser ►

Situation einen Unterschied, wenn man sieht, dass die eigene Arbeit ohne Wert ist. Trotz der Bezahlung.

Ein weiterer Faktor, der oft als Engagement-Killer genannt wird, ist die unproduktive Meetingkultur. Das Phänomen dürfte so weit verbreitet sein, dass es unnötig ist, Beispiele zu nennen. Doch was kann man selbst tun?



**+12%
PRODUKTIVITÄT**

Wohlbefinden und Glück steigern die Produktivität

Eine Studie hat Wohlbefinden und Glück mit einer Steigerung der Produktivität um 12 Prozent in Verbindung gebracht. Die Autoren schreiben: „Geringere Zufriedenheit ist systematisch mit geringerer Produktivität verbunden. Diese verschiedenen Beweise, die sich in ihren Stärken und Schwächen ergänzen, sprechen dafür, dass es einen kausalen Zusammenhang zwischen menschlichem Wohlbefinden und menschlicher Leistung gibt.“

Quelle: Oswald/Proto/Sgroi [4], Grafik: Kutura [1]

Der häufigste Tipp: Man lehnt eine Teilnahme ab, wenn man erwarten kann, dass die Besprechung der eigenen Arbeit nicht hilft. Das Statement dahinter ist unmissverständlich. Wer es etwas weniger drastisch handhaben möchte, versucht es konstruktiv. Man bittet den Einladenden rechtzeitig um eine Agenda und hinterfragt, welcher Beitrag von einem selbst erwartet wird. Zu Beginn des Meetings kann man noch einmal nach der Agenda und nach Regeln fragen, die einen reibungslosen und produktiven Verlauf des Meetings sicherstellen. Gibt es zum Beispiel Besprechungsregeln, die Endlosgespräche verhindern? Manche Teams nutzen dafür das Schlüsselwort ELMO als Abkürzung für „Enough. Let's move on.“ Bei einem Online-Meeting signalisiert der kurze Hinweis „ELMO“ im Chat allen Sprechenden, dass man keinen Fortschritt sieht. Da Veränderungen von Meetingkulturen, die über Jahre entstanden sind, nicht immer leicht erzielbar

sind, sollte die Option, den nutzlosen Meetings fernzubleiben, immer auf dem Tisch liegen und auch genutzt werden. Dass diese Vorgehensweise möglicherweise Konflikte hervorruft, liegt auf der Hand.

Konflikte sind auch zu erwarten, wenn man im Fall von zu vielen Arbeitspaketen anfängt, Aufgaben abzulehnen. Doch seien wir ehrlich. Wer mit Arbeit überlastet wird, befindet sich schon im Konflikt; vermutlich aber ganz allein. Den Konflikt öffentlich zu machen, indem man Arbeit ablehnt, eröffnet zumindest die Chance einer Verbesserung. Das Maximieren der Auslastung mag für Maschinen, die eine gleichförmige Arbeit leisten, eine gute Idee sein, für Menschen aber nicht. Und auch nicht für die Unternehmen, in denen die Menschen arbeiten. Für dieses Thema sei das Buch „Schwarmdumm“ von Gunter Dueck empfohlen [13].

Es führt kein Weg daran vorbei, Grenzen rechtzeitig aufzuzeigen, um eigene gesundheitliche Konsequenzen zu vermeiden. Rechtzeitig bedeutet: bevor gesundheitliche Folgen zu spüren sind.

Resilienz

In der Softwareentwicklung, speziell in der Entwicklung neuer Produkte, gibt es immer wieder Rückschläge. Manchmal funktioniert ein gewählter Ansatz nicht. Manchmal ändern sich über die Projektlaufzeit die Anforderungen, sodass zum Beispiel die einstmals gute Architektur nicht mehr passt. Das ist normal. Folglich ist es eine gute Idee, Resilienz zu entwickeln: die Fähigkeit, mit diesen Widrigkeiten umzugehen, nach Rückschlägen wieder aufzustehen und Konflikte am Arbeitsplatz mit Anstand zu lösen. Auch hier könnte man auf den ersten Blick denken, das sei eine individuelle Fähigkeit und das Unternehmen hätte damit nichts zu tun.

Und genau wie beim vorhergehenden Thema gibt es übliche Verhaltensweisen im Unternehmen, die die Resilienz der Mitarbeiter strapazieren und eigentlich ein Versagen der Unternehmenskultur darstellen. Der erste und schwerwiegendste Punkt lautet „öffentliche Schuldzuweisung“.

Ein Beispiel dafür ist folgender Verlauf: Für die Entwicklung eines neuen Produkts gibt der Chef einen Zieltermin vor. Die genauen Anforderungen sind nicht klar; ihm nicht und auch sonst niemandem. Erfahrungen mit dem Produkt gibt es natürlich nicht. Es ist ja neu. Eine Rücksprache mit den Entwicklerteams findet nicht statt. Der Termin steht im Raum, wird Kunden und anderen Stakeholdern versprochen. Vertriebs- und Marketingmaßnahmen werden geplant und kommen ins Rollen. Und dann?

Zum geplanten Termin ist die Software nicht fertig. Moment mal. Wieso „geplant“? So heißt das in den Unternehmen dann. „Geplanter“ Termin. Richtig wäre: „Aus der Luft gegriffener Termin“. Wenn der Termin nun platzt, wer bekommt die Kritik zu hören? Genau: Die Teams, die den Termin nicht geschafft haben. Das allein ist schon schädlich genug. Langfristiger Schaden entsteht besonders dann, wenn die Verantwortlichen aus der Situation nichts lernen und der nächste Termin genauso willkürlich festgelegt wird.

Was kann der Einzelne in so einer Situation tun? Kurz gesagt: Aufstehen, das Problem klar und deutlich ansprechen

und in den Konflikt gehen. Wer das macht, übernimmt Führungsverantwortung für sich und sein Team. „Führung“ heißt hier nicht formale Macht und die Position eines Vorgesetzten. Oft sind es die Silent Leaders im Team, die dazu bereit sind [14]. Als praktisches Handwerkszeug empfehle ich an dieser Stelle einmal mehr die GFK, die Gewaltfreie Kommunikation; dazu unten mehr.

Integration, Vertrauen und Beteiligung

Damit die eigene Arbeit zu Wohlbefinden, Erfüllung und einem Glücksgefühl beiträgt, sind Vertrauen, Kooperation und Integration notwendig.

Unrealistische Termine passen nicht dazu, denn offensichtlich wird man an der Terminfindung nicht beteiligt. Verallgemeinert geht es um fehlende Mitsprache. Wer sagt: „Ich darf selbst nichts entscheiden“, fühlt sich mit Sicherheit nicht in-

tegriert. Das gilt auch für Meetings, in denen über die eigene Arbeit gesprochen wird und zu denen man gar nicht eingeladen ist. Oder man ist eingeladen, über Dinge zu sprechen, die die Arbeit von abwesenden Personen betreffen. Das sind keine fiktiven Aussagen, sondern tatsächliche Beobachtungen, die Menschen in Unternehmen machen, in denen nicht besonders viel auf Happiness at Work geachtet wird.

Zu einer integrativen Kultur gehört auch Feedback von und für Vorgesetzte. In Umfragen vermissen Mitarbeitende immer wieder Feedback oder berichten von Feedback in einer unangemessenen Form. Dazu gehören Äußerungen, die als passiv-aggressiv empfunden werden, ebenso wie überwiegend negatives und nur wenig positives Feedback. An der Grenze zum Micromanagement liegt Feedback der Art „Ich erzähle dir, wie dein Job funktioniert“. Vorgesetzte, die nie Feedback geben und nie nach Feedback für ihr eigenes Handeln fragen, tragen ebenfalls zur Distanzierung und nicht zur Integration bei.

Wie kann man als Entwickler damit umgehen? Soweit es das Feedback betrifft, gibt es eine geradlinige Antwort: Man fragt nach Feedback und man gibt (ungefragt) Feedback. Beim Geben von Feedback ist Fingerspitzengefühl gefragt. In reifen Unternehmenskulturen wird Feedback nicht als Konflikt empfunden, sondern als Beitrag zur Verbesserung.

Dort, wo Unsicherheit und Ungewissheit vorherrschen, mag es sein, dass Feedback, das sachlich vorgetragen wird, trotzdem als persönliche Kritik aufgefasst wird. Wenn diese Gefahr besteht, sollte die Kommunikation wie im Konfliktfall geführt werden. Eine sehr gute Methode, in solchen Fällen erfolgreich zu kommunizieren, ist die Gewaltfreie Kommunikation (GFK) [15]. Es handelt sich um ein einfaches Vorgehen, das trotz seiner Einfachheit Übung erfordert. Da man GFK für alle möglichen kniffligen Situationen gebrauchen kann, in denen das Potenzial für einen (gegebenenfalls unterschweligen) Konflikt besteht, empfiehlt es sich, bei vielen Gelegenheiten zu üben. Ohne hier eine umfangreiche Einführung in die GFK zu geben, sei zumindest auf einen wesentlichen Punkt hingewiesen: Es ist wichtig, eine Beobachtung von der eigenen Interpretation zu trennen. Ein Beispiel hierfür liefert der Kasten **GFK im Teamgespräch**.

Fazit

In vielen Unternehmen gibt es systemische und strukturelle Eigenschaften, die nicht zu „Happiness at Work“ beitragen. Diese Eigenschaften fallen in die Zuständigkeit von Managern. Aus diesem Grund erscheinen Defizite, die man als Mitglied eines Entwicklerteams sieht, häufig als Konflikt zwischen Mitarbeitern und Managern. Diese Sichtweise ist irreführend. Bis auf eine oder sehr wenige Personen haben alle Menschen im Unternehmen eine Vorgesetzte oder einen Vorgesetzten; sie sind also ebenfalls einer Person mit mehr formaler Macht untergeordnet. Kurz gesagt gilt meistens: Mein Chef hat auch einen Chef; wir sitzen alle im selben Boot.

Ich halte nichts davon, im Unternehmen Fronten zu bilden, die zu einer „Wir gegen sie“-Sichtweise beitragen. Vielmehr halte ich es für zielführend, immer wieder die Frage zu stellen: „Was kann ich selbst zur Verbesserung beitragen?“ ►

● GFK im Teamgespräch

Das Handlungskonzept der Gewaltfreien Kommunikation basiert auf vier Schritten, deren erster es ist, eine Beobachtung von der eigenen Interpretation zu trennen. Ein Beispiel aus dem Entwickleralltag soll dies erläutern: Alice gibt Bob Feedback zu seinem Verhalten im Teamgespräch.

- **Variante 1:** Alice beschwert sich bei Bob mit folgenden Worten: „Du hast die Teamkollegen im Meeting aufgrund ihrer mangelnden Git-Kenntnisse heruntergemacht. Hör auf damit.“ Bob geht in eine Verteidigungshaltung: „Ich habe niemanden heruntergemacht. Niemand hat sich bei mir beschwert. Kümmer dich um deinen eigenen Kram.“ Und schon sind die beiden im Konflikt.
- **Variante 2:** Alice spricht mit Bob und wiederholt zunächst, was sie im Meeting beobachtet hat: „Bob, du hast einigen Teamkollegen geraten, das Auflösen von Merge-Konflikten mit Git zu üben. Für mich klang es so, dass du sie wegen mangelnder Git-Kenntnisse heruntermachst.“ Bob reagiert und stellt seine Sicht dar: „Mein Vorschlag, Auflösen von Git-Konflikten zu üben, klang für dich nach Heruntermachen? Das war nicht meine Absicht. Ich wollte ihnen einen Hinweis geben, wie sie besser werden können. Es hat einige Probleme auf dem Master-Branch gegeben, weil die Juniorkollegen bei dem Versuch, Merge-Konflikte aufzulösen, eine wichtige Code-Änderung von mir ‚überschrieben‘ haben.“

Der wesentliche Unterschied beider Varianten ist der Auftakt von Alice. Im ersten Fall beginnt sie mit ihrer Interpretation des Sachverhalts und macht Bob einen direkten Vorwurf. Im zweiten Fall trennt sie ihre Beobachtung von ihrer Interpretation. Und sie macht klar, dass es sich um eine Interpretation handelt, während sie in der ersten Variante ein faktisches Fehlverhalten feststellt, das Bob als Angriff auffasst. Die Trennung von Beobachtung und Interpretation gibt Bob die Gelegenheit, seine Perspektive darzustellen. Sie folgt den beiden ersten Schritten der GFK.

Die im Lauf des Textes gemachten, möglichst konkreten Vorschläge anhand von Beispielen lassen sich wie folgt verallgemeinern:

Zunächst sollte jeder Mensch im Unternehmen Verantwortung für sich selbst übernehmen. Das bedeutet unter anderem, sich selbst klarzumachen:

- Welche Umgebung, welche Informationen, welche Voraussetzungen brauche ich, um mich wohlfühlen und produktiv arbeiten zu können?
- Welchen Beitrag zum Unternehmenserfolg möchte ich leisten? Wie will ich mich einbringen?
- Welche Defizite oder Konflikte sind derzeit gegeben, und wie würde meine (Arbeits-)Welt ohne diese Probleme aussehen?

Wenn die Antworten bekannt sind, bleibt die Frage nach der Umsetzung. Dafür ist es unvermeidlich, alle Punkte professionell kommunizieren zu können. In vielen Fällen heißt das: Kommunikation mit dem/der Vorgesetzten. Aber auch: Konfliktkommunikation mit Kollegen. Wer das nicht gerne macht oder glaubt, es nicht gut zu können, hat eine Aufgabe identifiziert: Üben und Lernen. Viele Unternehmen bieten interne Fortbildungen an, und die Möglichkeit einer externen Schulung ist immer gegeben.

Nach meiner Erfahrung lassen sich durch offene Ansprache mit dem richtigen Ton die meisten Anliegen und Wünsche, um zu mehr Happiness at Work zu kommen, angehen. Die dazu nötigen Veränderungen brauchen oft viel Zeit, weil viele Defizite ihre Ursache in der Unternehmenskultur und in Glaubenssätzen wie „So machen wir das hier schon immer“ haben. Geduld und Hartnäckigkeit sind gefragt.

In der Arbeit mit unseren Kunden haben wir neben den oben genannten spezifischen Antworten auf die Frage „Was kann jeder Einzelne tun?“ auffallend oft folgende zwei Reaktionen bekommen:

- Lernt einander kennen.
- Bau dein Netzwerk auf.

Diese beiden Punkte sind so universell, dass man sie gar nicht hoch genug bewerten kann. Aus diesem Grund markieren sie auch den Abschluss des Artikels.

Unsere Unternehmen sind so aufgebaut, dass es Positionen mit formaler Macht gibt. Diese Positionen sind durch Titel und weitere Äußerlichkeiten gekennzeichnet. Firmen sind jedoch auch soziale Systeme. In ihnen gibt es Einfluss durch persönliche Beziehungen. Wer damit anrühige Seilschaften verbindet, die im Hintergrund die Fäden zum eigenen Vorteil und an der Grenze der Illegalität ziehen, begeht zwei schwerwiegende Fehler:

- Das Knüpfen von persönlichen Beziehungen sagt nichts über die moralisch-ethische Verwendung dieser Beziehungen aus.
- Solange Menschen zusammenarbeiten, werden persönliche Beziehungen existieren. Die Frage ist nur, ob ich sie als wesentlichen Teil meiner Arbeitswelt begreife und sie selbstbestimmt gestalte und nutze, oder ob ich sie vernachlässige und meinen Einfluss freiwillig begrenze.

Da wir in unserer Zeit mehr als genug Herausforderungen haben, freue ich mich über jeden Menschen, der seinen Einfluss nutzt, um an Verbesserungen zu arbeiten. ■

- [1] Stefan Mintert, *How to destroy happiness at work?*, Kutura 2024, <https://kutura.digital/happiness-at-work>
- [2] Gallup, *The Economics of Wellbeing*, Gallup Press, 2010, https://www.gallup.com/file/services/177050/The_Economics_of_Wellbeing.pdf
- [3] Petri Böckerman, Pekka Ilmakunnas, *The Job Satisfaction-Productivity Nexus: A Study Using Matched Survey and Register Data*, in: *Industrial and Labor Relations Review* 65(2): 244–262, April 2012
- [4] Andrew J. Oswald, Eugenio Proto, Daniel Sgroi, *Happiness and Productivity*, *Journal of Labor Economics*, Volume 33, Number 4, October 2015
- [5] Emiliana Simon-Thomas, Dacher Keltner, *The Foundations of Happiness at Work*, BerkeleyX GG201x, www.dotnetpro.de/SL2412Happiness1
- [6] Harold Abelson, Gerald Jay Sussman, Julie Sussman, *Structure and Interpretation of Computer Programs*, 2. Auflage, MIT Press, 1996, ISBN 978-0-262-51087-5
- [7] Walter Isaacson, *Steve Jobs*, Bertelsmann (E-Book), 2011, ISBN 978-3-641-07462-3
- [8] Patty McCord, *8 lessons on building a company people enjoy working for*, TED 2019, www.dotnetpro.de/SL2412Happiness2
- [9] Cargo-Kult bei Wikipedia, www.dotnetpro.de/SL2412Happiness3
- [10] Stefan Mintert, *Meetingkultur: „Ichnehm‘ das mit“ darf nicht zur Regel werden*, heise online 2024, www.dotnetpro.de/SL2412Happiness4
- [11] Peter Biddle, *Plucky Rebels – Being agile in an un-agile place*, TED 2013, www.dotnetpro.de/SL2412Happiness5
- [12] Dan Ariely, *What makes us feel good about our work?*, TED 2012, www.dotnetpro.de/SL2412Happiness6
- [13] Gunter Dueck, *Schwarmdumm: So blöd sind wir nur gemeinsam*, Goldmann, 2018, ISBN 978-3-442-15950-5
- [14] Stefan Mintert, Marco Maria Gerlach, *Silent Leaders im Team*, Kutura 2024, www.dotnetpro.de/SL2412Happiness7
- [15] Marshall B. Rosenberg, *Gewaltfreie Kommunikation: Eine Sprache des Lebens*, Junfermann, 2016, ISBN 978-3-95571-572-4
- [16] Jacob Morgan, *The Employee Experience Advantage*, Wiley, 2017, ISBN 978-1-119-32162-0



Stefan Mintert

begleitet Organisationen in der agilen Transformation und auf dem Weg zu Agile Leadership. Seine Mission sieht er darin, Menschen dabei zu helfen, sich im Rahmen ihrer Arbeit zu entwickeln und zu wachsen.

stefan.mintert@kutura.digital

dnpCode

A2412Happiness

ZWISCHEN TECHNIK UND EMOTIONEN

Die unsichtbare Blockade

Transformationsprojekte verzögern sich selten aufgrund der Technik. Erfahren Sie mehr über die Ursachen und darüber, wie Sie sinnvoll und sorgsam damit umgehen.

Ständige Veränderung, Weiterbildung und Neudefinition gehören zum alltäglichen Leben eines Entwicklers und eines Softwareunternehmens. Trotzdem muss man oft nicht lange suchen, um herauszufinden, warum manchmal tiefgreifendere Transformationen in einem Softwareprojekt notwendig werden. Einige Unternehmen geraten beispielsweise durch zu schnell wachsenden Erfolg in Zugzwang für neue Features, während das Team nicht schnell genug wächst und daher Wartbarkeit und Zukunftsstabilität auf der Strecke bleiben. Andere verschließen zu lange die Augen vor dem Zeitgeist der Anwender und werden dann von einer notwendigen Transformation wie der von einer Desktop-zentrierten Welt ins Web und auf mobile Geräte überrascht.

Stellen wir uns das vielleicht einmal an einem Extrembeispiel aus der Praxis vor: Ein Unternehmen, das bislang eine Desktop-Lösung angeboten hat – vielleicht sogar noch auf Basis von WinForms – und jetzt den Entschluss gefasst hat, jene Lösung ins Web – modernes Layout, Single-Page-Applikation, mobil – zu portieren. Wer das schon mal gemacht hat, weiß, dass das im Grunde den maximalen Kurswechsel bedeutet. In der Realität ist das kein seltenes Szenario.

Die übliche Vorgehensweise

Jedem wird schnell klar, dass diese Umstellung massive Veränderungen im gesamten Unternehmen bedeutet. Neben dem technischen Umbau hat das sicher auch einen größeren Einfluss auf den Marketing- und Vertriebsapparat, da unter Umständen auch das gesamte Vertriebsmodell auf links gekrempelt wird. Darum wird hier, wenn alles gut läuft, eine gute Planung vorausgehen. Man wählt eine neue Zieltechnologie, denkt sich ein machbares Transformationsmodell aus, das zum aktuellen Softwaredesign passt, und los gehts. All diese Schritte bergen eine große Komplexität und zahlreiche Fallstricke. Aber darum soll es hier nicht gehen. Viel wichtiger für diesen Artikel ist es, was in dieser Phase meist überhaupt nicht berücksichtigt wird: Wie geht die Belegschaft mit den Entscheidungen um? Konkreter: Wie reagieren die Entwickler auf so eine krasse Veränderung der gewohnten technischen Umgebung? Denn was passiert hier effektiv? Man erwartet vom einzelnen Entwickler, dass er seine bevorzugte Wunschtechnologie verlässt und gegen die – möglicherweise fremdbestimmte – Zieltechnologie eintauscht.

„Was kaum je berücksichtigt wird: Wie geht die Belegschaft mit den Entscheidungen um?“

Dieser Aspekt wird in den meisten Fällen tatsächlich außer Acht gelassen. Im Extremfall läuft es auf eine „Friss oder stirb“-Strategie hinaus, bei der der Mitarbeiter das Erlernen der neuen notwendigen Fähigkeiten eigenständig – Googling by doing sozusagen – im Lauf der Zeit erledigt. Das hat zwar den Charme eines hemdsärmeligen Start-ups, ist jedoch in einem etablierten, professionell arbeitenden Unternehmen fehl am Platz.

Vielleicht gibt es aber dann doch eine Mitarbeiterschulung, um die Kollegen zumindest über die wichtigsten Basics des neuen Arbeitsumfelds in Kenntnis zu setzen. Wenn das gut geplant ist, passiert es auch zum richtigen Zeitpunkt. Also nicht zu früh, damit die Entwickler bis zum praktischen Einsatz nicht schon alles vergessen haben. Aber auch nicht zu spät, damit das Kind nicht bereits im Brunnen liegt. Doch das ist eine andere Geschichte.

Für uns im Artikel interessant: Wie gehen die Softwareentwickler emotional mit dieser Thematik um? Denn dieser Aspekt findet selten Platz in solchen Transformationsplanungen. Im Grunde gibt es drei Arten, wie Entwickler auf diese Situation reagieren, wobei die Übergänge fließend sind.

Gruppe eins wird sich über eine zukunftsgerichtete Strategie freuen und sich auf neues Wissen, neue Wege und neue Lösungen einstellen. Das sind meist die Entwickler, die es gewohnt sind, neue Dinge zu lernen, die auch privat noch tausend Projektideen haben und gefestigt in ihrer Persönlichkeit sind. Es sind meist die jüngeren Mitarbeiter, die noch nicht

zu sehr festgefahren in ihrer Wunschtechnologie sind, das Lernen womöglich noch aus der Ausbildung oder dem Studium kennen und sich auch in ihrer Freizeit ausgiebig mit Side Quests beschäftigen. Sie bringen eine positive Grundeinstellung im Hinblick auf eine lernorientierte Unternehmenskultur mit, nutzen Fortbildungsangebote regelmäßig

und haben Spaß am spielerischen Umgang mit anderen Dingen. Diese Gruppe zeigt sich flexibel und unkompliziert, wenn es auf zu neuen Ufern geht.

Gruppe zwei reagiert skeptisch und verunsichert, wenn neue Themen um die Ecke kommen. Diese Gruppe ist gefestigter in den Arbeitsweisen, die sich über Jahre eingefahren haben. Möglicherweise ist hier auch die Neugier auf Neues bereits abgeebbt, und die Prioritäten im Leben haben sich verschoben – entweder durch eine Familie oder aufgrund ►

anderer Themen. Neue Dinge sind aus deren Sicht schwer abzuschätzen und daher mit Vorsicht anzuwenden.

Wir sprechen hier von einer erfahrenen Skepsis, die durchaus gesund und sinnvoll ist. Es handelt sich um Menschen, die auf der Basis ihrer Lebenserfahrung Dinge gerne hinterfragen und solche Fragestellungen an dieser Stelle auch im Positiven ins Projekt einbringen können. Tonangebend ist hier in der Regel eine gedämpfte Zuversicht im Hinblick darauf, ob das Projekt „einfach so“ gelingen kann und ob die eigenen Fähigkeiten sich schnell genug anpassen.

Damit sind in der Regel die Leute gemeint, die diverse Management-Ratgeber oder Transformationshandbücher unter dem Kapitel „Mitarbeiter ins Boot holen“ adressieren. Und in der Tat handelt es sich hier um Menschen, die keineswegs gegen das Projekt arbeiten, sondern einfach nur Vorbehalte haben. Diese lassen sich mit einer transparenten Kommunikation, Ansprache auf Augenhöhe und einem überzeugenden Plan durchaus in das besagte Boot holen.

Sicher darf man in dieser Gruppe nicht die Menschen vergessen, die sich aufgrund ihrer Lebenserfahrung und persönlichen Präferenz auf einen bestimmten Technologiebereich spezialisiert haben und diesen ganz bewusst nicht verlassen wollen. Rechnen Sie also in einem solchen Mammutprojekt auch mit Personalfluktuations. Denn es gibt neben all den zwischenmenschlichen Themen auch harte Grenzen im fachlichen Bereich, die Mitarbeiter nicht überschreiten wollen. Das ist auch verständlich, wenn man gestern noch nach einem C#-Entwickler mit 20 Jahren Berufserfahrung mit WinForms gesucht hat und eben diesem Mitarbeiter dann JavaScript aufs Auge drücken will.

Gruppe drei ist es, die uns Sorgen machen sollte. Sie zeigt sich sichtlich verängstigt, wenn so grundlegende Veränderungen im Arbeitsumfeld anstehen. Die Reaktionen fallen hier unterschiedlich aus, sind aber durch sonderbare Extreme geprägt, die sich recht klar von der gesunden Skepsis der vorherigen Gruppe unterscheiden. Dies äußert sich mal durch tiefgreifenden Pessimismus, mal resultiert es aber auch in ei-

ner übereifrigen Mitgestaltung des Endprodukts. Beides auf spürbar ungesunde Weise. Auf diese Gruppe konzentrieren wir uns in den folgenden Absätzen.

Was passiert da?

Wo die zweite Gruppe mit sachlich begründbaren Sorgen argumentiert – und eben auch mit geeigneten Gegenargumenten überzeugt werden kann –, scheint in der letztgenannten Gruppe irgendetwas anders zu laufen. Zwar werden auch hier Argumente vorgebracht, doch diese sind meist oberflächlicher Art, und mit etwas Feingefühl spürt man, dass dahinter ganz andere Triebfedern am Werk sind. Der Mitarbeiter zeigt sich getrieben und steigert sich mehr in Argumente hinein, als es eigentlich angebracht wäre.

Die Folge sind emotionsgeladene Diskussionen, scheinbar dünne Argumente, Wortgefechte unter der Gürtellinie – häufig auch lautstark – und ein recht verkrampft arbeiten. Genauso, wie das offensichtlich nicht mehr sachlich zum Thema passt, gehen im entsprechenden Mitarbeiter auch ganz andere Dinge vor. Hier geht es in der Tat nicht mehr um das Transformationsprojekt – und doch ist es der Auslöser.

Was hier passiert, kennen wir alle: Irgendeine Situation sorgt dafür, dass wir irgendwie anders reagieren, als es unser rationales Selbst es eigentlich getan hätte – freundlich ausgedrückt. Das äußert sich dann in Wut, einer spitzen Bemerkung oder in Form einer lauterer Reaktion, als man das beabsichtigt hat. Man ertappt sich hinterher dabei, dass man „nicht man selbst“ war und dass einem die Reaktion auch leidtut. Wir sprechen dann entschuldigend davon, dass wir „getriggert“ wurden.

Was steckt dahinter? Offensichtlich schaltet das Hirn hier auf eine Art Autopilot und ignoriert, was der rationale Teil unseres Gehirns von der Situation hält. Hier klinkt sich das gute alte Unterbewusstsein ein, das berühmt für seine evolutionären Überlebensreflexe ist. In einer für unsere Urahnen vor Tausenden von Jahren bedrohlichen Situation musste schnell entschieden werden, ob man fliehen, erstarren oder

Wie Trigger-Reize ausgelöst werden

In Trigger-Situationen können physiologische Reaktionen des menschlichen Nervensystems unmittelbar zu Verhaltensweisen führen, die aus rationaler Sicht im jeweiligen Kontext unverständlich erscheinen.



kämpfen sollte – dies wird auch als Fight-Flight-Freeze-Prinzip bezeichnet und oft im Zusammenhang mit Säbelzähntigern erwähnt.

Jetzt sitzt da aber kein Säbelzähntiger. Und jeder würde für den heutigen Projektalltag in den allermeisten Fällen bestätigen, dass keine unmittelbare Gefahr besteht, für die sich diese Abwägung lohnen würde. Warum wirkt dieser Reflex also trotzdem?

Das liegt daran, dass die Mustererkennung für Gefahren eben nicht hart verdrahtet ist, sondern im Lauf unseres Lebens neue Muster hinterlegt, die einmal als Gefahr eingestuft wurden und seither als solche wiedererkannt werden sollen. Das Gehirn tut also, was eine seiner wichtigsten Aufgaben ist: Aus Situationen lernen, um zukünftige gleichartige Situationen zu vermeiden oder besser zu meistern.

Um Ihnen diesen Mechanismus zu demonstrieren, möchte ich gerne eine Reihe von Beispielen dafür anführen, wie diese Automatismen unschwerflich wirken. Beginnen wir mit einem Beispiel, das sich auf der rationalen Ebene abspielt, denn das kennen wir als erwachsene Menschen aus dem alltäglichen Arbeitsleben. Wir beginnen eine Tätigkeit, von der wir bislang nicht viel verstehen: Wir kochen zum ersten Mal im Leben Kaffee.

Der Ablauf ist grob klar (Filter, Kaffeepulver rein, Wasser in die Maschine, einschalten). Durch die Analyse des Ergebnisses verfeinern wir den Prozess und lernen über viele Iterationen, den besten Kaffee zu produzieren.

Beim Erstellen des Kaffees sind wir uns über jeden Schritt bewusst, erinnern uns an die Fehlschläge und haben dennoch volle Handlungsfreiheit, hier entgegen unserer Erfahrung auch einmal Orangensaft anstatt Wasser in die Maschine zu füllen.

Gehen wir eine Ebene tiefer: Sie beschließen eines Tages, den längst abgelaufenen Fisch im Kühlschrank doch noch zu essen, weil Sie in diesem Zusammenhang mutig und ohne schlechte Erfahrungen durchs Leben gegangen sind. Unerwarteterweise wird Ihnen übel, und der Körper stößt dieses unverträgliche Lebensmittel auf mehreren Wegen wieder ab. Dies bleibt Ihnen definitiv im kognitiven Gedächtnis. Aber nicht nur dort. Sollten Sie kurze Zeit später wieder das Vergnügen haben, zu lange gelagerten Fisch zur Auswahl ihrer Nahrung zu haben, wird Ihnen nicht nur Ihr Verstand sagen, dass das möglicherweise eine dumme Idee ist, sondern auch der Körper. Sie werden nach dieser Erfahrung vorerst keinen Appetit mehr auf vergleichbare Nahrungsmittel haben – Ihnen wird schlecht beim Geruch. Und selbst wenn Sie auf Ihren freien Willen zurückgreifen und den Fisch gegen jede Vernunft essen, werden Ihre Körperfunktionen frühzeitig in Gegenwehr gehen.

Das mag jetzt zwar nicht unbedingt derselbe Mechanismus sein, auf den ich im dritten Beispiel hinauswill, demonstriert aber auf plakative Weise, wie stark Körperfunktionen und Unterbewusstsein ein Mitspracherecht haben, wenn es um unmittelbare Gefahren geht.

● Vorschnelle Schlüsse

Die in den Fallbeispielen genannten Ursachen, Symptome und Reaktionsmuster sind Zusammenstellungen, die bei jedem Menschen vollkommen unterschiedlich auftreten können. Lassen Sie sich bitte nicht dazu verleiten, über Ihre Arbeitskollegen vorschnelle Beurteilungen auszusprechen. Diese Beispiele sollen nur verdeutlichen, mit welchen unerwarteten inneren Dämonen einige Menschen durchs Leben gehen.

Für das dritte Beispiel stellen wir uns einmal vor, dass sich die Muster, die eine Gefahr beschreiben, zum einen in einem Alter geprägt werden, an das wir uns nicht mehr so recht erinnern können, und andererseits keine konkreten Objekte (Säbelzähntiger, Herdplatte ...), sondern eher vage Situationen mit sehr weit gefächertem Wiedererkennungswert beinhalten. Ich weiß, es wird ein bisschen abstrakt. Aber ich verspreche, dass wir bald die Kurve zurück zum Transformationsprojekt bekommen.

Eine solche Situation könnte zum Beispiel sein, dass eine Person in einem Elternhaus groß geworden ist, in dem ihr das Gefühl gegeben wurde, dass sie etwas leisten muss, um liebenswert zu sein. Das Kind lernt hier, dass Zugehörigkeit und Liebe erst verdient werden müssen. Für ein Kind ist das Gefühl, dazuzugehören und liebenswert zu sein, jedoch ein emotionales Grundbedürfnis, ohne das keine gesunde emotionale Entwicklung stattfinden kann.

Es wird das Gefühl, „immer leisten zu müssen“, als Kompensationsstrategie und als den tiefsitzenden Glaubenssatz „Nur wenn ich leiste, bin ich richtig“ abspeichern.

Es geht hier im Allgemeinen um die Grundbedürfnisse des Menschen wie Selbstwert, Sicherheit, Zugehörigkeit. Also Dinge, die für das gesunde emotionale Überleben eines von Erwachsenen abhängigen Kindes unabdingbar sind.

Ich denke, Sie stimmen mir zu, dass dies für das Kind klar nachvollziehbar als eine bedrohliche Situation wahrgenommen wird.

Ein Mensch, der in einem emotional bedrohlichen Umfeld aufwächst, entwickelt zwangsläufig Überlebensstrategien, die in solchen Situationen für ihn funktionierten. Die Strategien, die sich aus jener beschriebenen Situation ergeben, können grundunterschiedlich sein. Das Kind könnte in die Offensive gehen und besonders laut werden, um auf seinen emotionalen Mangel aufmerksam zu machen. Es könnte sich aber ebenso gut in den Rückzug flüchten und das Vertrauen in Beziehungen grundlegend verlieren. Im schlimmsten Fall aber passt sich das Kind an und definiert seinen Wert über die äußerlichen Umstände – und zwar ein Leben lang. Es leistet und übergeht dabei seine Bedürfnisse.

Ausgangssituationen und die daraus gebildeten Verhaltensweisen können von Mensch zu Mensch grundunter- ►

„Der passende Kampf, der diese Reaktion rechtfertigte, liegt lange zurück.“

schiedlich sein. Auch die Intensität von Gefahr und die Auswirkungen sind sehr individuell. Perfider noch: Es muss sich dabei noch nicht mal um eine tatsächlich lebensbedrohliche Situation gehandelt haben. Allein die persönliche Wahrnehmung des Kindes ist hier ausschlaggebend, um diese Muster ein Leben lang vorzuhalten.

Das verdeutlicht, dass die Beweggründe einzelner Menschen tief vergraben liegen können. Einerseits, weil es sich möglicherweise um eine lebensbedrohliche, also wirklich einschneidende Erfahrung gehandelt hat. Und andererseits, weil die Erinnerung daran nicht präsent in Bildern vorhanden ist, um der ganzen Handlung eine „Begründung“ mitzugeben. Man ist also den überwältigenden Empfindungen ausgeliefert, die einem das Unterbewusstsein sendet.

Um nun also den Link zum Jetzt und Heute zu geben: Je einschneidender eine solche frühe Erfahrung war, umso gefestigter sind die entsprechenden Reaktionsmuster. Entwickelt man dafür kein Bewusstsein, halten diese sich ein Leben lang. Gleichzeitig sind die Trigger-Muster recht weit gefächert. Anstatt also auf exakt einen knallroten VW Käfer zu reagieren, der genau auf einen zurast, springen die Trigger schon bei vage ähnlichen Situationsmustern an. Um beim Beispiel zu bleiben: Empfindet der Betroffene in einem Gespräch, dass sein Gegenüber die eigene Meinung nicht ausreichend angehört hat, kommt unterschwellig das Gefühl von Wut auf – ohne dass es vordergründig einen Grund dafür gibt. Denn auch hier ist wieder wichtig: Nicht die objektive Situation ist hier ausschlaggebend, sondern das eigene subjektive Empfinden. Das wird später noch wichtig, wenn es um mögliche Maßnahmen geht.

Womit wir nun wieder beim Transformationsbeispiel landen. Inzwischen sollte allen Lesern klar sein, dass das konkrete Beispiel der Transformation nur stellvertretend für alle einschneidenderen Ereignisse im (Arbeits-)Leben von betroffenen Personen ist. Solche großen Veränderungen bergen besonders großes Potenzial, entsprechende Mitarbeiter aus der berühmten Komfortzone zu locken und sich den unterschiedlichsten Triggern zu stellen und in Angstsituationen getrieben zu werden. Sei es die Angst vor Konflikten, die Sorge um die eigene Wertschätzung oder die fundamentale Definition des eigenen Selbstwerts.

Konkret ausgedrückt kann die Veränderung, die ein Transformationsprozess mit sich bringt, die Überlebensstrategie eines Mitarbeiters gefährden. Definiert ein Mitarbeiter zum Beispiel seine Autonomie darüber, immer dagegen zu sein, oder seinen Selbstwert darüber, niemals Fehler machen zu dürfen, dann versteht sich von selbst, dass Veränderung unerwünschte Muster aufdecken kann.

Plakativ gesagt geht es hier um menschliche Grundbedürfnisse, in denen sich der Mitarbeiter eingeschränkt oder sogar bedroht fühlt.

Spätestens jetzt sollte jeder eine grobe Vorstellung davon vor Augen haben, warum Menschen in

manchen Situationen eine scheinbar vollkommen unangemessene emotionale Reaktion zeigen – der passende Kampf, der diese Reaktion rechtfertigte, liegt lange zurück.

Auswirkungen im Team

Wie also hat das einen Einfluss auf das Team? Natürlich ist es erst einmal unangenehm, wenn sich einzelne Teammitglieder mit entsprechenden Entscheidungen nicht wohlfühlen. In der Regel ist so ein Umstand erst einmal kein Grund zur Sorge. Sicher werden ständig technische Entscheidungen getroffen, mit denen einzelne Mitglieder des Teams nicht einverstanden sind. Aber der Kompromiss und das Akzeptieren von Entscheidungen gehören nun mal zum gemeinschaftlichen Arbeiten dazu. Betrifft es dann auch noch weniger wichtige Themen, ist schnell Gras über die Sache gewachsen, und alle Entwickler ziehen am Ende wieder an einem Strang.

Betrachten wir das Ganze jetzt aber einmal aus der Brille von jemandem, der von oben beschriebenen Ängsten getrieben wird. Konstruieren wir vielleicht einen weiteren Fall, um das an einem konkreten Beispiel durchzuspielen. Jemand, der in seiner Kindheit wenig Wertschätzung entgegengebracht bekommen hat, findet in seinem Beruf eine Erfüllung,

Stresstoleranzfenster

Der Begriff Stresstoleranzfenster (Window of Tolerance) wurde von Daniel J. Siegel geprägt, um die Reaktionen des Nervensystems auf Stress durch die Beschreibung verschiedener Niveaus nervöser Erregungen zu erklären. Wann immer wir durch Stressoren den individuellen Toleranzbereich nach oben (Übererregung) oder unten (Untererregung) verlassen haben, ist es wichtig, über kurz oder lang wieder in den emotionalen Intensitätsbereich zurückzukehren, in dem wir uns sicher und wohl fühlen. Nur innerhalb dieses Fensters können wir rational denken, Neues lernen und das Leben genießen. Die Bandbreite reicht von angenehmer Aufregung und Freude bis zu tiefer Entspannung und einem Gefühl von Verbundenheit.

ÜBERERREGUNG | Kampf oder Flucht

Angst, Panik, Wut, Zwänge, Selbstmedikation

AUSGEGLICHENER ZUSTAND

Erschöpfung, Burn-out, Depression, Dissoziation, Lustlosigkeit, Selbstmedikation

UNTERERREGUNG | Erstarrung

die den einstigen Mangel an Wertschätzung im gegenwärtigen Leben kompensiert. Indem er durch überdurchschnittlich viel Fleiß seine eigenen Erfolge schafft, besser ist als alle anderen in seinem Umfeld und im Ergebnis von Kollegen und dem Arbeitsumfeld Zuspruch erfährt, schafft sich dieser Mensch mithilfe seiner Expertise eine neue Daseinsberechtigung für seine Existenz.

Sie müssen sich also vorstellen, dass im Wertesystem eines solchen Menschen die Lebensberechtigung unmittelbar an seine überdurchschnittliche Schaffenskraft und sein Können geknüpft ist. Dies ist leider ein sehr weit verbreitetes Phänomen der aktuellen Leistungsgesellschaft – und noch extremer unter uns verkopften Informatikern.

Nehmen wir also an, dass dieser Mitarbeiter, der seinen aktuellen Wissensstand über die letzten 10 bis 15 Jahre zur Perfektion ausgebaut hat, nun vor die Aufgabe gestellt wird, von vorne anzufangen – gemeinsam und gleichgestellt mit allen anderen im Team. Dies gleicht dem Entzug der Wertigkeit seiner Person.

In den Augen des Betroffenen beginnt hier der Überlebenskampf erneut – der Kampf, den er durch die viele Arbeit der vergangenen Jahre vermeintlich bereits gemeistert hatte. Wie diese Person schlussendlich konkret reagiert, hängt von seiner individuellen Überlebensstrategie ab. Aber naheliegend ist natürlich, dass die drohende Gefahr mit allen Mitteln bekämpft wird. Das Projekt wird sabotiert, Argumente werden verdreht, das Team mit endlosen Diskussionen müde gemacht.

Befinden sich diese „Saboteure“ in eher ranghöheren Positionen (Architekten, Senior Devs ...), haben deren Bemühungen einen entsprechend längeren Hebel, was unter Umständen das gesamte Team lahmlegt. Es ist dann nicht nur das Problem eines einzelnen Mitarbeiters, sondern betrifft die gesamte Gruppe.

Das Perfide daran: Betroffenen Menschen ist in diesem Augenblick überhaupt nicht bewusst, was da in ihnen abläuft. Würden Sie oberflächlich nachfragen, würde Ihnen die Person klar bestätigen, dass sie im besten Wissen und Gewissen für das Unternehmen und das Team agiert. Das bedeutet, dass hier keineswegs wissentlicher Vorsatz im Spiel ist. Diese Menschen können in diesen Augenblicken nicht anders!

Die Überlebensstrategie kann auch in eine ganz andere Richtung umschlagen: Jener Mitarbeiter könnte beschließen, möglichst schnell wieder Vorreiter in der neuen Technologie zu werden, um seine Position im neuen Arbeitsumfeld so früh wie möglich zu sichern.

Was zuerst einmal eher positiv klingt, endet dann aber in einem krampfhaften Einarbeiten, oft auf Kosten täglicher Aufgaben, durchgearbeiteten Nächten und unvollständigen Fakten. Sobald sich der Mitarbeiter für „überlegen“ hält, wird er eine führende Rolle bei der Transformation anstreben, um seine Position erneut zu festigen.

Bei den umschriebenen Charakteren wird beim Lesen möglicherweise offensichtlich, dass diese Personen unter

Umständen nicht nur bei Großprojekten problematisch werden, sondern auch in der alltäglichen Arbeit. Wer seine Überlegenheit ständig unter Beweis stellen muss, um sich als Mensch einen Wert zu geben, hat auch Schwierigkeiten, sich in ein gleichberechtigtes Team einzubringen. Hier wird „Besserwisserei“ zum Problem, da man sich ja um jeden Preis abheben muss.

Was kann das Team tun?

Was also tun, wenn man als Teamleiter, Projektmanager, Manager oder Teamkollege bemerkt, dass sich im Team Menschen befinden, die aus unterschiedlichen Gründen von heftigen Aufgaben verängstigt, verunsichert oder anderweitig beeinträchtigt werden? Sollten Sie sich selbst in dem einen oder anderen Fallbeispiel wiedererkannt haben, finden Sie im nächsten Kapitel ein paar Vorschläge, was Sie für sich selbst tun können.

Vielleicht die schlechte Nachricht zuerst: Es ist hier mit ein paar Handlungshinweisen leider nicht getan. Ein antrainiertes Schulterklopfen, weil im Managerhandbuch steht, dass man Mitarbeiter auch mal loben soll, führt meist zum gegenteiligen Effekt.

Um möglichst viele der eigenen Mitarbeiter in den Flow zu holen, braucht es eine wohlwollende und vertrauensvolle Arbeits- und Teamatmosphäre. Das ist leichter gesagt als getan. Denn am Ende handelt es sich ja streng genommen doch um eine recht inhomogen zusammengewürfelte Gruppierung von Menschen, die primär aus wirtschaftlichen Gründen vonseiten des Arbeitgebers gemeinsam an grob demselben Machwerk arbeiten sollen.

Die Frage, warum man als Chef über das Gehalt hinaus sich auch noch um das übrige Wohlbefinden der eigenen Mitarbeiter kümmern sollte, stellen sich zum Glück immer weniger Unternehmer – zumindest in der IT. In der ersten Denkschleife ist es ja in der Tat auch das eigene Problem des Mitarbeiters, sich täglich zu motivieren und etwas für sein Geld zu machen. Und ja, diese Denkweise mag auch eine Zeit lang gut funktionieren. Verlangt man von Mitarbeitern aber, dass sie eine hohe Kreativität und Eigenmotivation mitbringen, um an einem herausragenden Produkt mitzuarbeiten, wird es schwierig.

Selbst wenn der Laden unter normalem Betrieb halbwegs gut läuft, Features nach Schema F implementiert und Bugs abgearbeitet werden, kommt dieses System ins Wanken, wenn etwas mehr Freigeist gefordert wird. Größere Veränderungen wie beispielsweise das angestrebte Transformationsprojekt stellen hier eine Art Sollbruchstelle für diese Art der Teamzusammenstellung dar.

Gründe dafür, warum Mitarbeiter in diesen Umbruchzeiten besonders verunsichert sind, sind zahlreich vorhanden. In einem sich selbst überlassenen Entwicklerteam (oft beobachtet in mittelständischen Betrieben, die noch zu klein für eine eigene Managementebene sind und deren Geschäftsführer ►

„Diese Abwehrhaltung ist so mächtig, dass Betroffene sich ihrer meist überhaupt nicht bewusst sind.“

primär mit Kundenmanagement blockiert ist) tauchen immer wieder dieselben Muster auf, die Entwickler in ihrer Schaffenskraft blockieren:

- **Fehlende Priorisierung:** Entwickler sollen eigenständig über die Priorität von Arbeitsaufträgen entscheiden. Bei zu viel Arbeit entscheidet sich der normale Menschenverstand üblicherweise für die Arbeitspakete, in denen er routiniert ist. So bleiben Modernisierungsarbeiten oft viel länger liegen als geplant.
- **Fehlender Aufgabenfilter:** Der Druck des Kunden landet direkt beim Entwickler. Dadurch kann keine objektive Priorisierung stattfinden, und wieder bleibt das Transformationssticket liegen.
- **Sprunghafte Priorisierung:** Kurzfristige Kundenanforderungen gehen den direkten Weg zum Entwickler, weshalb Arbeiten sehr schwer zu planen sind.
- **Umgang mit Fehlschlägen:** Gemachte Fehler landen unvermittelt wieder auf dem Schreibtisch des Entwicklers, wodurch noch mehr Arbeit entsteht. Der Mitarbeiter verliert seine Experimentierfreude.

So ungefähr würden die Mitarbeiter das Problem beschreiben, fragte man sie nach den Dingen, die ihnen Stress verursachen. Alle Punkte klingen vordergründig so, als würden hier ein paar Prozessoptimierungen helfen – also doch nur Handlungshinweise? Sicher lohnt es sich, auf entsprechende Prozesse ein Auge zu werfen. Was den Mitarbeiter aber eigentlich belastet, haben alle vermeintlichen Gründe gemeinsam: Der Mitarbeiter fühlt sich mit seiner Verantwortung alleingelassen, nicht integriert in eine Gemeinschaft, die die Last gemeinsam trägt.

Und damit haben wir des Pudels Kern auch schon gefunden: Das Grundbedürfnis, dem eigenen Schaffen einen Sinn zu geben, und das in einer Gemeinschaft aus Gleichgesinnten zu erarbeiten und auch zu feiern. Eine vertrauensvolle Gemeinschaft gibt Sicherheit und Halt. Und nur in der Sicherheit entfaltet der Mitarbeiter sein volles kreatives Potenzial. Behalten Sie das immer im Hinterkopf, wenn es um Ihre Mitarbeiter und auch Kollegen geht. Wir sind, ob wir es glauben oder nicht, soziale Wesen, die ohne Zugehörigkeit und Sinnhaftigkeit krank werden.

Denken Sie im Konkreten darüber nach, wie Sie in folgenden Punkten Verbesserungen herbeiführen können, sodass sich das Sicherheitsgefühl Ihrer Kollegen und Mitarbeiter steigern lässt. Im Kern ist es aber eine Frage der Haltung zu all diesen Dingen und der gelebten Kultur. Tun Sie die Dinge, die sich richtig für Sie und auch für das Team anfühlen.

- **Vertrauen durch Transparenz:** Kommunizieren Sie offen über Entscheidungswege, Probleme und Prozesse und auch darüber, wie Sie sich fühlen. Gehen Sie in den Dialog, um Meinungen, Bedenken und Sorgen anzuhören und ernst zu nehmen.
- **Gemeinsame Vision:** Nichts ist demotivierender als ein Hamschiff. Sich von Ticket zu Ticket zu schleppen, ohne das große Ganze und das Ziel zu sehen. Schaffen Sie eine Perspektive, indem Sie gemeinsame Teamziele herausarbeiten, die bei Erreichen auch gefeiert werden dürfen.

- **Fehler- und Lernkultur:** Fehler sind menschlich und treten vermehrt in kreativen Prozessen wie der Softwareentwicklung auf. Ermutigen Sie zum Experimentieren, Lernen und auch zum Scheitern und sorgen Sie dafür, dass das Team gemeinschaftlich gestärkt daraus hervorgeht.
- **Offener Dialog:** Unterschiedliche Meinungen, Konflikte und Streit sind eine wertvolle Triebfeder kreativer Prozesse und führen zu guten Kompromissen in einer diversen Gesellschaft. Unterstützen Sie dabei, Konflikte konstruktiv zu lösen.
- **Empathie:** In einer Gruppe ist es für das gegenseitige Vertrauen unerlässlich, dass man als Einzelner mit all seinen Stärken und Schwächen gesehen und geschätzt wird. Stellen Sie einen sicheren Rahmen her, in dem sich jeder ohne Sorge öffnen kann. Hier können Instanzen von außen helfen, etwa Coaches, Berater, Mediatoren und Psychologen.
- **Anerkennung und Wertschätzung:** Wertschätzung ist das, was uns als Mensch motiviert, uns in einer Gemeinschaft einzubringen. Unterm Strich ist es genau das, was uns in der Gruppe über uns selbst hinauswachsen lässt.

Erschütterung des Selbst- und Fremdbildes

Wo auf der jeweiligen Skala der Ausprägungen wichtiger menschlicher Grundbedürfnisse wie Sicherheit und Selbstakzeptanz sich ein Mensch verortet, wird unter anderem durch frühkindliche Erfahrungen mitbestimmt.

SICHERHEIT

Bedrohung vs. ich bin sicher

KONTROLLE | WAHL

Ohnmacht/Hilflosigkeit vs. meine Welt ist kontrollierbar

SELBSTAKZEPTANZ

Selbstabwertung vs. ich fühle mich wertvoll

VERANTWORTUNG

Schuld/Scham vs. ich kann Menschen vertrauen

Am Ende sind es Wohlwollen, Verständnis für die Umstände des jeweils anderen, gegenseitige Akzeptanz und der Wille, sich zu unterstützen, die ein stabiles Team ausmachen. Wir sprechen also eher über eine Kultur als Basis eines vertrauensvollen Miteinanders auf Augenhöhe.

Zurück zum Transformationsprojekt. Denn laufen diese Dinge gut im Team, sind alle Teammitglieder über den Prozess, dessen Entscheidungsfindung und den Zeitplan informiert. Alle kennen die Risiken und Probleme, die eventuell auf dem Weg liegen. Es besteht das Vertrauen darin, dass alles nach bestem Wissen und Gewissen passiert ist und dass

man als Team den richtigen Weg eingeschlagen hat. Es folgt das Commitment der Entwicklergruppe, dass jeder sein Bestes gibt, um das gemeinsame Ziel zu erreichen. Und los gehts!

Einen Großteil der Entwickler sollten Sie damit abgeholt haben – möglicherweise sogar einen ordentlichen Teil aus Gruppe drei, die durch die engere Gemeinschaft und das Verständnis des Teams ihren Weg gefunden haben. Bleiben noch die Hardliner übrig. Es besteht nämlich die Gefahr, dass hier noch jemand auf der Strecke bleibt. Nämlich derjenige, der exakt mit den Maßnahmen zur Förderung von Vertrauen, Gemeinschaft und Beziehung seine Schmerzpunkte erreicht.

Versetzen wir uns in eine Person, die in jungen Jahren die Erfahrung gemacht hat, dass selbst eng vertraute Menschen wie beispielsweise die Familie keinen zuverlässigen Zufluchtsort darstellen. Vielleicht noch schlimmer: Dass Verrat und Gefahr von jenen Personen ausgingen, auf die man angewiesen war. Noch mal zur Erinnerung: Was aus Erwachsenensicht möglicherweise nicht ganz so schlimm klingen mag, ist aus Sicht eines Kindes, dessen physisches Überleben von Personen abhängt, die sich nicht als vertrauenswürdig erwiesen haben, fatal. Jene Person entwickelt möglicherweise eine Überlebensstrategie, die ohne zwischenmenschliche Beziehung auskommen will.

So eine Person wird eher misstrauisch, wenn Sie ein Gespräch über emotionale Probleme anbieten, weil das innere Kind hier schon auf den nächsten Verrat wartet. Zumal Sie als Kollege oder Chef ja auch noch Teil des Systems sind, das den aktuellen Stressor darstellt.

Diese Abwehrhaltung ist so mächtig, dass der Betroffene sich ihrer meist überhaupt nicht bewusst ist. Selbst in einem Gespräch mit einer vertrauten Person würden entsprechende Menschen steif und fest behaupten, dass zwischenmenschliche Beziehungen für sie keine Relevanz haben und im Leben nicht wichtig sind. Das geschieht mit einer solchen Überzeugung, dass ein Argumentieren vollkommen zwecklos ist.

Daran knüpft ein Merksatz der Coaches und Therapeuten: Man kann einen anderen Menschen nicht ändern. Das klingt jetzt endgültig, soll aber nur verdeutlichen, dass Sie nicht erzwingen können, dass jemand anderes sich nach Ihrem Willen ändert – und erscheint Ihnen dessen Verhaltensweise auch noch so abwegig und selbstzerstörerisch. Gewaltsam oder mit Druck erreichen Sie exakt das Gegenteil.

Zu Hilfe kommt ein anderer Merksatz, der besagt, dass man sich seinem inneren Schmerz erst dann widmet, wenn der äußere Schmerz größer ist. Im Grunde bedeutet es, dass dieser Mensch zunächst erkennen muss, wie unglücklich und fremdgesteuert er durch seine inneren Dämonen ist, um den Willen zur Veränderung zu entwickeln.

Sie könnten, sollte sich dieser Mitarbeiter als schwierig erweisen, den einfachen Weg gehen und sich von ihm trennen. Sie könnten aber auch im Dialog bleiben, Verständnis und Geduld aufbringen. In festgefahrenen Fällen, wenn es zu offenen Konflikten im Arbeitsumfeld kommt, lohnt sich auch das Hinzuziehen eines Moderators, also einer neutralen Person von außen, die Erfahrung darin hat, von der vordergründigen Konfliktlösung in eine Richtung zu lenken, die dem Betroffenen eine größere Einsicht ermöglicht. Oft reicht ein klei-

ner Funke, um ein Umdenken zu erwirken. Letztendlich muss der Mitarbeiter die Veränderung aber selbst wirklich wollen, wenn sie erfolgreich vollzogen werden soll.

Was kann ich selbst tun?

Vermutlich haben Sie bei der Lektüre des Artikels die eine oder andere Stelle entdeckt, die Sie möglicherweise an das eigene Verhalten erinnert. Kein Wunder, denn Trigger, die uns Automatismen abspielen lassen, haben wir alle. Und das ist auch okay so. Die meisten dieser Trigger wollen wir ja auch genau so haben. Oder etwa nicht? Immerhin gibt der Verstand hier kurzzeitig die Kontrolle ab.

Verrückterweise sagen wir zwar oft: „Oh, das triggert mich“. Aber die meisten unserer Trigger sind uns überhaupt nicht bewusst. Erst wenn uns andere Personen darauf ansprechen, dass wir in einem Meeting oder beim Gespräch sonderbar oder unpassend reagiert haben, zeigt uns das, wie untermischlich all dies passiert.

Zusammengefasst sagt uns das, dass unser Unterbewusstsein es für nötig hält, hier schnell zu reagieren – schneller und vermutlich anders, als es unser Bewusstsein getan hätte. Das deutet darauf hin, dass sich hier ein Muster eingebrannt hat, das irgendwann in unserer Vergangenheit sehr nützlich war und deshalb seine Daseinsberechtigung hat. Im Hier und Heute aber ändern sich in unserem Leben immer wieder die Umstände, weshalb solche Muster gerne hinterfragt werden dürfen.

Es lohnt sich also, gerade wenn man schon eine Ahnung hat, in welchen Situationen wieder ein Trigger zuschlagen könnte, mit erhöhter Aufmerksamkeit in eine solche Situation zu ►

● Zurück ins Hier und Jetzt: Die 5-4-3-2-1-Übung

Befinden Sie sich gerade in einem getriggerten Zustand und brauchen eine schnelle Abhilfe? Dann gibt es eine gute Fingerübung, um sich wieder in die Gegenwart zu holen und sich auf sich selbst zu konzentrieren. Die Idee dahinter ist, sich mit allen Sinnen auf Dinge in unmittelbarer Nähe zu konzentrieren und damit einen neuen Fokus zu bekommen.

Beginnen Sie damit, fünf Dinge in unmittelbarer Nähe zu benennen, die Sie sehen können. Das können beliebige Dinge sein, auf die Sie sich einen Moment lang konzentrieren. Anschließend versuchen Sie, fünf Dinge zu benennen, die Sie hören können. Im letzten Schritt versuchen Sie fünf Dinge zu benennen, die Sie spüren können. Beispielsweise die Kleidung auf der Haut.

Haben Sie dies geschafft, beginnen Sie erneut – diesmal mit vier wahrgenommenen Dingen. Jeweils sehen, hören, spüren. Dann noch mal von vorn, mit nur noch drei Dingen. Ab zwei kommt ein neuer Sinn hinzu: das Riechen. In der letzten Runde nehmen Sie dann noch den Geschmackssinn hinzu.

Diese Methode schafft nicht nur in Triggersituationen Abhilfe, sondern ist auch förderlich beim Einschlafen und wird sogar bei Panikattacken eingesetzt.

gehen. Bringt uns vielleicht ein spezieller Arbeitskollege in Meetings immer wieder auf die Palme? Dann beobachten Sie doch bitte einmal ganz genau, was in Ihrem Körper vor sich geht, wenn es wieder passiert.

Zum einen bringt Ihnen das Erkenntnisse darüber, welche Reize und welches Muster zum Trigger führen. Zum anderen erlaubt es Ihnen, die Reiz-Reaktions-Kette genau zu beobachten. Diese beiden Dinge ermöglichen es Ihnen, die Kontrolle über die Situation zurückzugewinnen! Das erfordert zwar ein bisschen Training, aber bemühen Sie sich, diese Kette bewusst zu unterbrechen. Im Extremfall dadurch, dass Sie die Situation verlassen und sich gegebenenfalls später erklären. Im Grunde ist dies das Äquivalent zu „erst mal tief durchatmen, bevor man etwas sagt“.

Spannender wird es, wenn Sie die Ursache weiter ergründen wollen. Möglicherweise haben Sie aufkommende Gefühle wie Wut identifiziert, bevor der Trigger zuschlägt. Wut als Gefühl hilft uns Menschen, uns im geeigneten Zeitpunkt zur Wehr zu setzen. Der Trigger war also vermutlich früher einmal dazu da, sich gegen eine erlebte Ungerechtigkeit zur Wehr zu setzen.

Oft hilft es dann schon, sich in einer ruhigen Minute in Gedanken noch einmal in die Triggersituation zu versetzen und alles in Ruhe zu beobachten. Fragen Sie sich, woher Sie diese Situation kennen, was Sie damals empfanden und warum die hinterlegte Reaktion zu jenem Zeitpunkt notwendig war. Bei einer solchen Reflexion kann es helfen, die eigenen Gedanken vor einer vertrauten Person laut auszusprechen. Suchen Sie sich eine Person, die Ihnen nicht nur gut zuredet und Sie darin bestärkt, dass der entsprechende Arbeitskollege definitiv ein Idiot ist, sondern möglicherweise jemanden, dem Sie gerne vertrauensvoll Ihren emotionalen Zustand schildern wollen und der mit Ihnen auf eine Expedition geht.

Sollten Sie hierbei auf innere Widerstände stoßen oder nur unbefriedigende Fortschritte machen, lohnt sich der Weg zur professionellen Hilfe. Vor allem dann, wenn Sie bemerken, dass Sie im Leben sehr darunter leiden oder dass Ihnen die Verhaltensweisen im Weg stehen, Sie sich fremdgesteuert fühlen und aktiv etwas daran ändern wollen. Hier reicht oft schon jemand, der routiniert die richtigen Fragen stellt und hilfreiche Werkzeuge an der Hand hat.

Ich habe in diesem Zusammenhang gute Erfahrungen mit Coaches und Psychotherapeuten gemacht, die den systemischen Ansatz mit einbeziehen. Die Erstgespräche sind in der Regel kostenfrei, und die Sitzungskosten bleiben meist im erschwinglichen Rahmen, sodass Sie das gefürchtete Psycho-Stigma der Krankenkasse nicht befürchten müssen. Alles läuft diskret ab. Geben Sie der Sache eine Chance.

Fazit

Eine wilde Reise liegt hinter uns. Das seichte Einstiegsthema eines Transformationsprojekts hat uns schnell in die Tiefen des menschlichen Unterbewusstseins geführt, um dort die Eigenheiten eines jeden Menschen zu ergründen. Wir alle sammeln im Lauf unseres Lebens – besonders prägend in der Kindheit – Erfahrungen, die unser Handeln bestimmen, im Bewussten wie im Unbewussten. Einige haben hier schwere-

Weiterführende Literatur

Wer sich tiefer mit der Thematik des menschlichen Gehirns auseinandersetzen, mehr über sich selbst erfahren oder gar die ersten Schritte der eigenen Selbstreflexion gehen möchte, dem seien die folgenden Bücher empfohlen:

- „Grundformen der Angst“ von Fritz Riemann [1]. Die verschiedenen Formen der Angst werden hier kategorisiert und verständlich eingeordnet.
- „Vom Mythos des Normalen“ von Gabor und Daniel Maté [2]. Eine kritische Auseinandersetzung mit der aktuellen Lebenskultur und möglichen Zusammenhängen moderner Krankheitsbilder.
- „Bedienungsanleitung für ein menschliches Gehirn“ von Gerald Hüther [3]. Eine leicht verständliche Abhandlung zur Funktionsweise des menschlichen Gehirns.
- „Trauma und Beziehungen“ von Verena König [4]. Hier geht es um Ursachen und Behandlung von Traumata und wie sie unsere Bindungen beeinflussen.

re Päckchen zu tragen als andere. Und so zeigen sich die Folgen dieser Erfahrungen mal im Privatleben, mal im Arbeitsumfeld, wie beispielsweise beim angeführten Transformationsprojekt, wenn der Mensch aus der selbst geschaffenen Komfortzone getrieben wird.

Heilung und Schutz verschafft die Gemeinschaft – egal ob in der Familie, in Freundschaften oder im Arbeitsumfeld. Gegenseitiges Vertrauen, Wertschätzung und Akzeptanz mit all den individuellen Ecken und Eigenheiten machen eine diverse und kreative Gesellschaft aus. Für viele reicht das, um Glück und Liebe zu finden. Andere brauchen eventuell professionelle Hilfe. Verurteilen Sie bitte niemanden deswegen. Alles hat Gründe. ■

[1] Fritz Riemann, *Grundformen der Angst*, Ernst Reinhardt, 2021, ISBN 978-3-497-01749-2

[2] Gabor Maté, Daniel Maté, *Vom Mythos des Normalen*, Penguin, 2015, ISBN 978-3-328-11193-1

[3] Gerald Hüther, *Bedienungsanleitung für ein menschliches Gehirn*, Vandenhoeck & Ruprecht, 2016, ISBN 978-3-525-01464-6

[4] Verena König, *Trauma und Beziehungen*, Arkana, 2024, ISBN 978-3-442-34310-2



Tom Wendel

ist freier Softwareentwickler, Coach und spezialisiert auf IT-Krisenberatung. Er unterstützt Unternehmen dabei, technische und organisatorische Herausforderungen zu lösen und ihre Entwicklerteams zu stärken.

tom.wendel@wendelpunkt.de

dnpCode

A2412Reflexion

PROGRAMM

DDC



.NET DEVELOPER CONFERENCE '24

DIE KONFERENZ FÜR .NET ENTWICKLER
25. - 28.11.2024 | KÖLN

dotnet-developer-conference.de | [#DDC24](https://twitter.com/DDC24) | Find us on [f](#) [i](#) [X](#) [in](#)

Präsentiert von:



MONTAG, 25. NOVEMBER 2024: WORKSHOPS

UHRZEIT	WORKSHOP 1	WORKSHOP 2	WORKSHOP 3
09.00 - 17.00	 Die perfekte .NET-Architektur David Tielke Die Welt der Architekturen für die Softwareentwicklung ist groß, der Überblick geht dabei schnell verloren. Architektur-experte David Tielke zeigt Ihnen in diesem Workshop, wie Sie mit wenig Aufwand die perfekte Architektur für Ihr .NET Projekt entwickeln, kommunizieren und kontrollieren.	 ASP.NET von 0 auf 100 Christian Giesswein ASP.NET Core ist ein leistungsstarkes, plattformübergreifendes Framework zur Erstellung moderner Webanwendungen. In diesem Workshop lernen Sie die Grundlagen und fortgeschrittenen Techniken von ASP.NET Core kennen. Schritt für Schritt führt Sie Christian Giesswein durch den gesamten Entwicklungsprozess - von der Einrichtung Ihrer Umgebung bis zur Bereitstellung einer voll funktionsfähigen Webanwendung.	 Dein Weg zum Cross-Plattform-App-Entwickler mit .NET MAUI und .NET 9 André Krämer André Krämer zeigt in diesem Workshop, wie .NET-Entwickler zu Cross-Plattform-Entwicklern werden können. Ausgehend von einer einfachen Bleistiftskizze entwickelt er live eine App für Android, iOS, macOS und Windows mit .NET MAUI. Sie erhalten eine detaillierte Einführung in die Entwicklung von Dashboards, die Gestaltung von Eingabefeldern, die lokale Datenhaltung, die Validierung von Eingaben und den Zugriff auf Gerätefunktionen. Dabei bietet der Workshop nicht nur tiefgehende Einblicke, sondern auch praktische Tipps, um die Herausforderungen der Cross-Plattform-Entwicklung zu meistern und beeindruckende Apps zu erstellen.
	 Single-Page-Apps mit Blazor 9.0 und C# 13.0 Dr. Holger Schwichtenberg Microsoft hat die Einsatzszenarien von Blazor durch Blazor Static-Server-Rendering und den Auto-Modus mit Umschaltung zwischen Blazor Server und Blazor WebAssembly erheblich erweitert. In diesem DDC-Workshop präsentiert Dotnet-Doktor Holger Schwichtenberg zunächst den Stand von Blazor in Version 9.0 und stellt die verschiedenen Blazor-Architekturmodelle mit ihren Vor- und Nachteilen gegenüber und leitet die Teilnehmer*innen anschließend an, selbst eine Blazor 9.0-Anwendung zu entwickeln, die serverseitig vorrendert, dann Blazor Server nutzt und im Hintergrund automatisch die Dateien für Blazor WebAssembly herunterlädt (Progressive Enhancements).	 AI-driven development mit GPT und Co. Jörg Neumann LLMs eröffnen eine völlig neue Welt. Mussten wir bisher prozeduralen Code schreiben, um Software etwas beizubringen, reicht nun die menschliche Sprache. Doch auch für die klassische Entwicklung bieten LLMs wie ChatGPT, GitHub Copilot & Co. Erleichterungen. Fest steht schon jetzt: LLMs werden die Anwendungsentwicklung grundlegend verändern. Jörg Neumann zeigt, wie LLMs funktionieren und wie man sie in Projekte integriert. Das Spektrum reicht von einfachen Anpassungen der Modelle per Few-Shot-Learning, dem Trainieren tiefen Wissens über Fine Tuning bis hin zu semantischer Suche mittels Retrieval Augmented Generation. Darüber zeigt er, wie sich LLMs in Microsoft Teams integrieren sowie Datenbanken und APIs innerhalb einer Konversation abfrage lassen.	 Cloud aspirations: building cloud-native applications with .NET Aspire Loek Duys, Alex Thissen  Starting by examining typical distributed applications and their architecture styles in the cloud, leveraging PaaS services from cloud providers with their resilience and scalability characteristics, we will introduce common concerns for distributed applications, such as observability, service discovery and configuration, and how .NET Aspire can help. You learn how to build your apps with .NET Aspire, by making proper use of the orchestration, components and toolings. We also cover the .NET Aspire service discovery and connection management, discover the programming model and the developer workflow. In the hands-on labs you experience and practice all learnings, enhancing an application by composing it using the .NET Aspire primitives of projects, containers and cloud resources.

DIENSTAG, 26. NOVEMBER 2024: KONFERENZ

UHRZEIT	SOFTWARE ARCHITECTURE, TESTING, AND QUALITY <i>David Tielke</i>	FRONTEND DEVELOPMENT <i>Fabian Gosebrink</i>	ARTIFICIAL INTELLIGENCE <i>Jörg Neumann</i>	DATA & DATABASES <i>Thorsten Kansy</i>
09.00 - 10.00	Nicht mehr das .NET Deiner Kindheit <i>Christian Giesswein</i>			
10.00 - 10.30	Kaffeepause @ Text Control			
10.30 - 11.30	Architekturen testen mit NDepend <i>David Tielke</i>	Let's turn your website into a hybrid .NET MAUI app <i>Mark Allibone</i>	AI-driven Software Development mit Azure AI <i>Jörg Neumann</i>	Die versteckten EF Core Features <i>Christian Giesswein</i>
RAUMWECHSEL				
11.45 - 12.45	Im Notfall Scheibe ein- schlagen: Strategien zum richtigen Behandeln von Fehlern <i>Florian Bader</i>	Migration: WinForms und WPF goes Web <i>Dr. Veikko Krypczyk</i>	Azure OpenAI und Semantic Kernel für Softwareentwickler <i>Dr. Damir Dobric</i>	Deep Dive: Entity Framework Core-Performance <i>Dr. Holger Schwichtenberg</i>
12.45 - 14.15	Mittagspause und Besuch der Fachmesse			
14.15 - 15.15	Stop starting with Micro- services - Der modulare Monolith in .NET <i>Florian Lenz</i>	Migrating from Xamarin.Forms to .NET MAUI <i>Iris Classon</i>	DDC-Panel: Chancen und Risiken der KI in der .NET- Entwicklung <i>Dr. Damir Dobric,</i> <i>Ina Einemann, Jörg Neumann,</i> <i>Julia Kordick</i> (Host: Tilman Börner)	GraphQL on .NET with steroids! <i>Mauro Bennici</i>
RAUMWECHSEL				
15.30 - 16.30	Taktisches Refactoring von .NET Legacy Code <i>Gregor Biswanger</i>	Enterprise Apps mit Blazor Server und EF Core entwickeln <i>Thomas Claudius Huber</i>	Big Brother is watching you - Lokales GenAI in Anwendungen <i>Thomas Tomow</i>	Azure Datenanalyse: Wie Licht ins Dunkel kommt <i>Florian Bader</i>
16.30 - 17.00	Kaffeepause @ Xebia Xpirit Germany			
17.00 - 18.00	Beyond Clean Code: Zeitgemäße Ansätze für hochqualitatives Software- engineering <i>Alexander Rampf</i>	Webanwendungen und Cross-Platform-Apps mit Blazor: Soll ich's machen, oder lass ich's lieber sein? <i>Dr. Holger Schwichtenberg</i>	„Talk to your systems“: Strukturierte Daten & Tool- Integration <i>Christian Weyer</i>	Visual Studio als Entwicklungstool für SQL Server basierte Datenbanken <i>Thorsten Kansy</i>
RAUMWECHSEL				
18.15 - 19.15	Wir testen. Aber testen wir auch gut genug? <i>Patrick Drechsler</i>	Echtes Cross-Plattform mit Blazor, WPF und MAUI - ein Praxisbericht <i>André Krämer</i>	Geschüttelt, nicht gerührt: KI-Agenten im Dev-Team <i>Alex Netsch,</i> <i>Markus Weiss</i>	SQL Server - eine sichere Sache <i>Thorsten Kansy</i>
ab 19.15	ABENDVERANSTALTUNG			
19.30 - 20.30	AI Ninja Moves: Wie kann ich AI in der Entwicklung konkret nutzen? <i>Jörg Neumann und Oliver Sturm</i>			
20.30 - 23.00	GET TOGETHER			

MITTWOCH, 27. NOVEMBER 2024: KONFERENZ

UHRZEIT	.NET LANGUAGES, FUNDAMENTALS, BEST PRACTICES <i>Christian Giesswein</i>	APPROACHES <i>Michael Kaufmann</i>	BACKEND TECHNOLOGIES <i>Thomas Claudius Huber</i>	CLOUD DEVELOPMENT <i>Marc Müller</i>
09.00 - 10.00	(How to destroy) Happiness at work <i>Kathleen Janz und Stefan Mintert</i>			
10.00 - 10.30	Kaffeepause @ Xebia Xpirit Germany			
10.30 - 11.30	.NET-Whatsup: Was gibt es Neues in .NET 9? <i>Christian Giesswein</i>	DevOps 3 Ways - Die Philosophie hinter DevOps <i>Thomas Schissler</i>	API FOMO: REST, gRPC or GraphQL? <i>Roland Guijt</i>	Azure Container Apps: The Future of Microservices in Azure <i>Geert van der Cruisen</i>
	RAUMWECHSEL			
11.45 - 12.45	Be Sharp with New Features in C# <i>Roland Guijt</i>	Event Storming: Kreativität trifft auf Struktur <i>Ina Einemann</i>	Entwicklung von ASP.NET Core Microservices mit Dapr: Eine praktische Anleitung <i>Marc Müller</i>	Azure Functions Deep Dive: Building Serverless Solutions for Scalable Applications <i>Michiel Hamers</i>
12.45 - 14.15	Mittagspause und Besuch der Fachmesse			
14.15 - 15.15	Writing (in)efficient C# <i>Callum Whyte</i>	Breaking Free: Kubernetes Unleashed for Small .NET Teams <i>Marius Boden</i>	Aus der Praxis: ASP.NET Core 8.0 WebAPI <i>Thorsten Kansy</i>	Typsicheres, polyglottes Messaging und Eventing: Stream/Queue/Topic of T <i>Clemens Vasters</i>
	RAUMWECHSEL			
15.30 - 16.30	Say goodbye to IConfiguration and embrace the power of the IOption pattern <i>Pieter Samyn</i>	Agil, aber richtig: Kundenzentrierung als Erfolgsfaktor für Teams <i>Miriam Hunger</i>	I followed the Microsoft docs, so my app is secure right? <i>Bart Lannoeye</i>	Versioniert, automatisiert und sicher - Optimale Bereitstellung von Azure-Ressourcen <i>Marc Müller</i>
16.30 - 17.00	Kaffeepause @ Text Control			
17.00 - 18.00	From C# to Rust <i>Sebastian Nilsson</i>	Pull Requests und Commit-Hygiene: Vom Branching bis zum Conventional Commit <i>Florian Bader</i>	To the Core: Migrating to ASP.NET Core Microservices <i>Iris Classon</i>	Container und WebAssembly: Verteilte Applikationen mit .NET Aspire orchestrieren <i>Thorsten Hans</i>
	RAUMWECHSEL			
18.15 - 19.15	Von Blazor zu WASI: Praktische Einblicke in WebAssembly mit .NET und C# <i>Oliver Sturm</i>	Maximale Effizienz: Automatisierte Releases in einer Multi-Team-Umgebung <i>Michael Kaufmann</i>	Containerisierung von .NET-Applikationen <i>Roland König</i>	Scaling your .NET app with Azure <i>Callum Whyte</i>
19.30 - 20.30	Being a Bad(ass) Developer <i>Iris Classon</i>			

DONNERSTAG, 28. NOVEMBER 2024: DEVSESSIONS

UHRZEIT	DEVSESSION 1	DEVSESSION 2	DEVSESSION 3	DEVSESSION 4
09.00 - 13.00	Architecting and Development of Custom Copilots <i>Dr. Damir Dobric</i> Die DevSession richtet sich an professionelle Softwarearchitekten oder -entwickler, die gefordert sind, Anwendungen zu entwickeln, welche Large-Language-Modelle (LLM) nutzen. Dazu erfahren sie, wie LLMs und GenAI unter der Haube funktionieren. Wir beginnen mit den Grundlagen und befassen uns mit Token-Generierung, Vector-Similarity, Embeddings, verschiedene Modelltypen, Plugins, Agents usw. Sie lernen Microsofts Semantic Kernel kennen und wie Sie AI-Agents erstellen können, die intelligenten Code in jeder Anwendung über zahlreiche Demos ausführen, sowie Anwendungen und mächtige Backends, die AI in die Azure-Plattform integrieren.	Testautomatisierung mit Playwright <i>Nico Orschel, Harald Binkle</i> „Testautomatisierung mit Playwright“ bietet eine umfassende Einführung in die Verwendung des Playwright-Frameworks zur Testautomatisierung, das die Entwicklung zuverlässiger und skalierbarer Testskripte für Webanwendungen ermöglicht. Die Teilnehmer lernen das Open-Source-Tool zu installieren, konfigurieren und Testskripte zu schreiben und auszuführen. Außerdem werden Best Practices für die Organisation und Pflege von Testfällen diskutiert. Am Ende werden die Teilnehmer in der Lage sein, Playwright effektiv einzusetzen, um automatisierte Tests für Webanwendungen zu erstellen und die Qualität ihrer Software zu verbessern.	Mit .NET Aspire verteilte Anwendungen entwickeln <i>Christian Giesswein</i> In der modernen Softwareentwicklung ist die Entwicklung verteilter Anwendungen ein entscheidender Erfolgsfaktor. Daher tauchen wir in dieser DevSession tief in die Möglichkeiten ein, die .NET Aspire bietet, um robuste, skalierbare und performante verteilte Anwendungen zu erstellen. Sie erhalten Einblick in die hierfür notwendigen wichtigsten Konzepte und Werkzeuge. Dazu gehören unter anderem Architektur und Design verteilter Anwendungen, Kommunikationsprotokolle und -methoden sowie Datenmanagement und -synchronisation. Durch praktische Beispiele und Demos wird das theoretische Wissen direkt anwendbar gemacht.	Vom Monolith zum Modulith - Eine Fallstudie aus der Halbleiterindustrie <i>Hendrik Lösch</i> Monolithische Architekturansätze sind nicht sehr beliebt, da sie als komplex und träge gelten. Wir nehmen in der DevSession daher ein reales Projekt unter die Lupe, um zu sehen, welche Herausforderungen lauern, wenn ein solcher Monolith umstrukturiert wird, und wie das Architekturmuster des Moduliths helfen kann. Dazu werden wir uns das Team-Setup, die Migrationsstrategie und wichtige Detailentscheidungen ansehen. Hierbei klären wir das Vorgehen bei der Konzeptionierung, Grundlagen aus dem Domain Driven Design und die Besonderheiten des Productline Software Engineerings, das dabei hilft, gleichartige Softwareprodukte plattformgestützt weiter zu entwickeln. Ziel ist es, Muster und Verfahren kennenzulernen und zu bewerten und sich darüber auszutauschen.
14.00 - 18.00	Kleinteilig & Großartig - Micro Frontends mit Blazor <i>Dr. Florian Rapp</i> In den letzten Jahren hat sich das Architekturmuster der Micro Frontends zur Skalierung von Webprojekten etabliert. In dieser DevSession wollen wir gemeinsam eine Micro Frontend Lösung exklusiv in Blazor aufbauen - mit serverseitiger Zusammenstellung der Micro Frontends. Gezielt werden wir uns im Detail anschauen wie die Lösung aufgebaut ist und welche Entscheidungen dafür notwendig waren. Dabei wird der vom Monolith gewohnte Entwicklungskomfort (DX) einfach in die verteilte Welt übertragen - ohne die UX z. B. im Bereich der Performance zu verringern. Das Ziel der DevSession ist, dass einzelne Teams in eigenen Repositories verteilt an der vorgestellten Lösung weiterarbeiten und jegliche Entscheidung bzw. Deployments auch autonom vorgenommen werden können.	Hochqualitative Softwareentwicklung <i>David Tielke</i> Eine Software zu entwickeln, ist nicht besonders schwer. Eine hochqualitative Software zu entwickeln, erfordert allerdings einiges mehr. Wir schauen uns in dieser DevSession an, worauf Sie in Ihrer Softwareentwicklung achten müssen, was es in den Bereichen Anforderungen, Architektur, Entwicklung, Testen und Bereitstellung zu beachten gilt und wie auch aus Ihrer Entwicklung eine hochqualitative Software schmiede werden kann.	Azure Messaging - Streams, Queues und Pub Sub <i>Clemens Vasters</i> Azure Messaging Services - einschließlich Azure Service Bus, Event Grid, Event Hubs, Relay, Stream Analytics und Fabric Event Streams - bieten einige Tools für eine zuverlässige Nachrichtenübermittlung und ereignisgesteuerte Architektur. Die DevSession vermittelt ein klares Verständnis der Fähigkeiten der Dienste und wie sie integriert werden, um skalierbare, robuste Anwendungen zu erstellen. Dazu diskutieren wir den Auswahlprozess für das richtige Messaging-Tool auf Grundlage der Anforderungen Ihrer Anwendung anhand von Beispielen aus der Praxis. Der Fokus liegt auf praktischen Anleitungen und Best Practices für die Implementierung einer kohärenten Messaging-Strategie auf Azure. Zudem werden wir uns mit den neuesten Entwicklungen der einzelnen Produkte befassen.	Deep dive in git <i>Michael Kaufmann</i> In dieser DevSession behandeln wir alle fortgeschrittenen Themen von Git, u.a.: Wie funktioniert Git? Was sind DAG, SHA, Patch und Commit? Was versteht man unter Zweigen und Tags? Wie löst man Merge-Konflikte? Wie manipulieren Sie Ihre Historie mit Commit amend, reset, rebase und cherry-pick? Wie arbeiten Sie mit dem Stash? Wie sucht man in Git? Zahlreiche praktische Tipps helfen bei der Arbeit mit Git: die richtige Konfiguration, Autokorrektur von Befehlen, Aufzeichnung von Merges und vieles anderes mehr. Der Workshop eignet sich für alle, die bereits über Grundkenntnisse in Git verfügen, und diejenigen, die neu in Git sind und nur Erfahrung mit anderen Versionskontrollsystemen haben. Voraussetzung sind eine lokale Git-Installation, ein Texteditor und ein kostenloses GitHub-Konto.

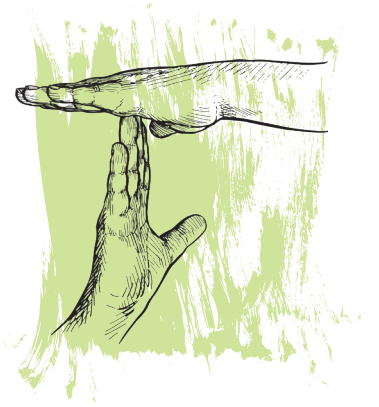
TIME-OUT

Just don't blame

Schulduweisungen helfen niemandem, wenn etwas schiefgegangen ist. Wie wir uns von unproduktiven Verhaltensmustern trennen.



Time-out Start



Da ist eine Aufgabe mal wieder nicht fertig geworden. Ich hatte es dem Kunden explizit versprochen, und nun hat mich der zuständige Mitarbeiter im Stich gelassen. Mir schwillt der Kamm. Können die nicht einmal das tun, was man ihnen sagt? Ist das denn so schwer zu verstehen? Sind wir hier beim Wunschkonzert? Gleich nach dem Anruf des Kunden renne ich zum Arbeitsplatz des betreffenden Mitarbeiters und lasse meinem Frust freien Lauf.

So mag es hin und wieder im Alltag aussehen. Ich nehme mich da nicht aus, dass mir das auch gelegentlich einmal passiert. Die Emotionen kochen hoch, und man sucht sich ein Ventil: den Mitarbeiter.

Und natürlich, es ist immer so eindeutig: Es ist genau dieser eine Mitarbeiter, der versagt hat. Er ist der Schuldige. Wenn wir andere Leute hätten, wäre das alles nicht so, wie es ist.

Kommt es zu einem Konflikt, zu einem Problem, ist der natürliche Instinkt, den oder die Schuldigen zu finden, und in der Regel ist der Schuldige der andere. Danach wird er für alles verantwortlich gemacht. Die Haltung des „Schuldigen“ ist dann ebenso nachvollziehbar wie verständlich: Es ist ein Angriff auf seine Person, und jeder Angriff – so das Gesetz der Natur – kennt nur zwei Reaktionen: Flucht oder Gegenangriff. Die Flucht kann dann dazu führen, dass der Mitarbeiter innerlich kündigt, nur Dienst nach Vorschrift macht und sich schlechte Stimmung im Unternehmen verbreitet.

Weg von Schuld ...

Was kann man dem entgegensetzen? Wie entkommt man dieser Falle? Denn am Ende schadet man mehr, als dass man etwas Gutes tut.

● In fünf Schritten zu nachhaltigeren Lösungen kommen

1. Passt es grundsätzlich?

Wenn Probleme immer mit dem gleichen Mitarbeiter auftreten oder es immer um das gleiche Thema geht, dann kann es sein, dass etwas Grundsätzliches nicht passt. Vielleicht ist diese Person nicht für die Aufgabe geeignet oder passt nicht zu Ihrer Firma. Wenn aber diese Frage geklärt ist, erwartet die Person 100 Prozent Commitment und Unterstützung von Ihnen.

2. „Don't blame“-Filter installieren

In der Situation selbst sind wir überfordert. Hier hilft nur Erinnerung, kein logisches Argument. Installieren Sie sich diese an einem Anker, etwa „Immer dann, wenn ich wütend aus dem Büro renne“ oder „Immer dann, wenn ich bei einer E-Mail auf Senden klicke“. In diesem Moment muss der installierte Filter greifen: „Wie ist die letzte Blame-Aktion gelaufen, und was kam effektiv raus?“ Sie können sich auch an Ihre eigene Abmachung halten, die Sie nach der Lektüre des Artikels getroffen haben. Oder Sie bauen eine Zeitverzögerung ein, um später noch mal darüber nachzudenken.

3. Perspektive wechseln

Versuchen Sie sich vorzustellen, wie Sie bisher wahrgenommen wurden beim Briefing, bei der Beschreibung der Aufgabe. Nicht das, was Sie im Kopf haben, zählt, sondern das, was angekommen ist. Wie könnte man die Aufgabe verstanden haben? Was weiß der Mitarbeiter alles? Und was kann er vielleicht gar nicht wissen, weil Sie es nicht kommuniziert haben?

4. In den Lösungsraum kommen

Was passiert ist, ist passiert. Es geht darum, Lösungen zu schaffen, wie Sie selbst es besser machen können. Was müssten Sie ändern? Was braucht der Mitarbeiter, damit das nicht noch mal passiert?

5. Das System ändern

Manchmal kann es sein, dass es organisatorische oder strukturelle Veränderungen braucht. Welche wären sinnvoll? Achtung: Setzen Sie sie nicht sofort um! Schreiben Sie sie stattdessen auf und schauen Sie sie sich nach einem Monat noch einmal an. Wenn sie dann immer noch sinnvoll erscheinen, dann setzen Sie sie um.

Im Grunde ist es ganz einfach: „Stop blaming!“ Ich habe noch nie erlebt, dass eine Schuldzuweisung irgendetwas besser gemacht hätte. Sie mag für die Juristen interessant sein, die am Ende immer ein Urteil und eine Strafe anstreben. Hilfreich zur Bewältigung eines Konflikts ist sie aber niemals. Wir gehen auf Angriff, der Angeklagte auf Abwehr oder Gegenangriff. Das Problem selbst ist damit aber nicht gelöst, im Gegenteil: Es entsteht ein Schaden, der größer ist als das eigentliche Problem.

Immer dann, wenn es einen Konflikt gibt, wenn ein Mitarbeiter nicht das getan hat, was Sie erwartet hatten, würde ich als Erstes dazu raten innezuhalten – und den natürlichen Drang des Blamings zu unterdrücken. Das ist zugegebenermaßen sehr, sehr schwer. Die Muster von Schuld und Sühne sind tief in uns eingebrannt, das Fingerzeigen ist ein ganz natürlicher Instinkt.

Die Lösung: Sie müssen einen mentalen Blocker für sich einbauen. Versuchen Sie, den Moment zu erkennen, in dem Blaming beginnt. Das Aufstehen vom Arbeitsplatz voller Wut, das Verfassen der Mail, was immer es ist – diesen Zeitpunkt sollten Sie erkennen. Im Moment selbst werden Sie zu schwach sein, um sich argumentativ selbst einzubremsen. Deshalb vertrauen Sie hier auf eine Entscheidung, die Sie jetzt und auf der Basis von Argumenten treffen. Nach dem Lesen des Artikels – und sollten Sie meiner Meinung sein – treffen Sie für sich die Entscheidung: Ich werde niemals anklagen. Wenn Sie das tun, und wenn Sie sich das in Ihren Kopf „einbrennen“, dann schaffen Sie es, „Entfernen“ zu drücken, wenn die Schimpftirade in der Mail gerade ihren Höhepunkt erreicht hat.

Und dann? Spannend ist es, sich selbst in diesem Moment eine Frage zu stellen: „(Was) habe ich zum Problem beigetragen?“ Eine schmerzhafteste Frage, denn sie sieht die Schuld immer auch ein Stück weit bei sich selbst.

Aber: Es ist die einzig sinnvolle Frage. Denn eines ist sicher: Sie werden es niemals schaffen, einen anderen Menschen zu ändern. Nicht ohne sein Einverständnis, und ganz sicher nicht durch Anklage. Das Einzige, was Sie schaffen, ist – sich selbst zu ändern. Und das ist schon schwer genug.

Also: Was habe ich zum Problem beigetragen? Im eingangs beschriebenen Fall weit mehr, als auf den ersten Blick sichtbar war. Deshalb sollte man sich immer fragen: Habe ich wirklich unmissverständlich klargemacht, wie wichtig diese Aufgabe ist? Habe ich den nötigen Freiraum geschaffen, sie zu erledigen? Oder war es einfach eine Zusatzaufgabe, die nicht zu schaffen war?

Es gibt keinen Konflikt mit anderen, bei dem Sie nicht Teil des Problems sind. Oder bei dem Sie nicht Teil der Lösung sein können. Vielleicht ist den anderen nicht klar genug, was Sie erwarten. Vielleicht gibt es Hindernisse, die Sie noch nicht sehen (wollen). Auf jeden Fall sind Sie immer auch Beteiligter und damit immer Teil des Konflikts.

Ist also der erste Impuls, dem anderen die Schuld zu geben und sich über unfähige Mitarbeiter oder über die Jugend von heute im Allgemeinen zu ärgern, im Keim erstickt, dann kommt der zweite, diskursive und schmerzhafteste Schritt – nämlich zu überlegen, welchen Teil man selbst dazu beigetragen hat.

... und hin zur Lösung

Sind diese Punkte identifiziert, muss man in den Lösungsraum kommen. Wie kann ich signalisieren, was mir wirklich wichtig ist? Sollte ich nicht besser die Aufgabe überwachen und eine Erinnerung senden, statt jemanden ins offene Messer laufen zu lassen? Wie wichtig ist mir, wer einen Fehler gemacht hat, und wie wichtig, dass die Arbeit erledigt ist? Was kann ich tun, um die Motivation zu steigern?

„Just don't blame“ kann gelingen, wenn wir ein anderes Verständnis für unsere Kollegen aufbringen, ein anderes Bild von ihnen haben. Wir sollten anerkennen, dass sie genauso wie wir den Erfolg der Firma wollen, das Beste tun und mit ihren Mitteln einen wertvollen Beitrag leisten wollen. Und wenn es denn zum Fehler kommt, dann hat das Ursachen, zu denen ich selbst auch beigetragen habe.

Manuel Neuer, Torhüter des FC Bayern München und bis vor Kurzem auch der deutschen Fußballnationalmannschaft, wurde einmal gefragt, wie er bei einem Fehler seiner Abwehr mit den Spielern umgehe. Ob da auch mal deutliche Worte angebracht seien. Seine Antwort hat mich auf den ersten Blick verblüfft, zeigt aber, dass Profisportler sehr gut verstanden haben, wie man gemeinsam Leistung erzielt: „Die wissen selbst, was sie falsch gemacht haben, und ärgern sich darüber weit mehr als ich. Warum sollte ich das noch mal aufwärmen? Ich will, dass sie motiviert weitermachen.“

So sieht echter Teamgeist aus. Schau, welchen Beitrag du zum Problem geleistet hast, und arbeite an einer Lösung, damit so etwas nicht noch einmal geschieht. Just don't blame! Damit verhindern wir die innere Emigration von Mitarbeitern und darüber hinaus weiteres „Versagen“, das zu noch mehr Konflikten führt.

Wie schwierig es ist, diesen inneren Schweinehund, diese eingeübten Rollen zu verlassen, ist uns allen bewusst. Wie viel es bringt, leider erst, wenn wir es mal ausprobiert haben. Tun Sie das! Und leben Sie von der positiven Energie, die Sie daraus gezogen haben.

In diesem Sinne: Happy Coding! ■



Time-out End



Dipl.-Theol. Bernhard Pichler

war 20 Jahre lang Geschäftsführer der Softwarefirma informare, bevor er 2020 zum Leiter Entwicklung und Produktmanagement der DPS BS ernannt wurde. Die DPS BS ist Deutschlands größter Softwarepartner für betriebswirtschaftliche Software für den Mittelstand.

dnpCode

A2412Timeout

SLEEPY-HOLLOW-ARCHITEKTUR, TEIL 2

Ich packe meinen Koffer ...

Von den Anforderungen zu einer testbaren, modularen, wandelbaren Implementation:
Wie geht das?

Dieser Artikel möchte Ihnen zeigen, wie Sie über eine Anforderungsanalyse mit Slicing (siehe die Serie von [1] bis [2]) und das Sleepy-Hollow-Architekturmuster [3] einem sehr geraden Weg von den Anforderungen zu einer testbaren, modularen, wandelbaren Implementation folgen können.

Lassen Sie uns gleich in die Praxis einsteigen. Das Handwerkszeug habe ich in den bisherigen Artikeln ausführlich vorgestellt. Jetzt soll es endlich in einem etwas größeren Beispiel zum Einsatz kommen.

Das Szenario

Es soll um ein Softwaresystem gehen, mit dem ein Team gemeinsam Ordnungen herstellen kann. Jedes Teammitglied soll die Positionen in einer Ordnung allerdings nicht absolut bewerten, sondern relativ.

- **Beispiel Priorisierung:** Aufgaben sind in eine Ordnung in Bezug auf die zeitliche Abfolge ihrer Erledigung zu bringen. In welcher Reihenfolge sollen die Aufgaben A1, A2, A3 bearbeitet werden? Teammitglied Paul findet, die Reihenfolge sollte A2, A3, A1 sein; Teammitglied Maria hingegen denkt, die Prioritäten sollten A3, A2, A1 sein; und Peter schließlich hält auch A2, A3, A1 für die beste Reihenfolge.
- **Beispiel Komplexitätsschätzung:** Issues sind in eine Ordnung in Bezug auf den Aufwand, der für ihre Umsetzung nötig ist, zu bringen. Welches der Issues I1, I2, I3, I4 ist komplexer als welches andere und wird deshalb wahrscheinlich mehr Zeit brauchen? Peter meint, das komplexeste Issue sei I3, dem I1, I4, I2 in absteigender Komplexität folgen; Paul hingegen sieht es so: I1, I3, I2, I4; und Maria schätzt die Komplexität so: I3, I4, I1, I2.

In Ermangelung von Kriterien, aus denen sich absolute Werte für Priorität oder Komplexität ableiten ließen, ist eine solche subjektive, relative Schätzung ehrlicher und einfacher als eine, die künstlich absolute Werte erzwingt (zum Beispiel Story Points). Teammitglieder müssen nur „ein Gefühl haben“, dass eine Position im Hinblick auf das jeweilige Ordnungsthema vor einer anderen stehen sollte.

Eine gemeinschaftliche resultierende Ordnung des Teams kann aus den verschiedenen individuellen Ordnungen der Teammitglieder leicht berechnet werden. Für jede Position werden die Indizes der Stellen

summiert, an denen sie steht. Je geringer die Summe, desto weiter vorne steht die Position in der Ordnung des gesamten Teams.

Bild 1 zeigt das für die beiden Beispiele. Zur Erläuterung die Bestimmung der Prioritätsordnung:

- Aufgabe A1 steht bei Peter und allen anderen an dritter Stelle. Sie hat immer den Index 3, die Summe ihrer Indizes ist also $3+3+3=9$. Aufgabe A2 steht bei Peter und Paul an erster Stelle (Index 1) und bei Maria an zweiter Stelle (Index 2); daraus ergibt sich die Summe 4. A3 steht bei Peter und Paul an zweiter Stelle (Index 2) und bei Maria an erster Stelle (Index 1); die Summe ist 5.
- Die Positionen mit ihren Summen: (A1,9), (A2,4), (A3,5).
- Sortiert nach aufsteigenden Summen ergibt sich als kalkulierte Ordnung für das Team: A2, A3, A1. Diese Ordnung ist sozusagen das Ergebnis der „wisdom of the crowd“.

Bei diesem Vorgehen wird (zunächst) bewusst nicht berücksichtigt, dass zum Beispiel der Abstand zwischen A1 und A2 viel größer ist ($9-4=5$) als der zwischen A3 und A2 ($5-4=1$). Es geht um eine schnelle, grobe Einschätzung der Ordnung „aus dem Bauch heraus“.

Das Softwaresystem soll eine Möglichkeit bieten, Positionen zu sammeln, die dann von Teammitgliedern in individuelle Ordnungen gebracht werden. Diese individuellen Ordnungen werden zu einer gemeinschaftlichen Ordnung verrechnet und angezeigt.

Teammitglieder sollen für die Bestimmung einer gemeinschaftlichen Ordnung nicht co-located sein müssen. Sie sollen auch nicht zeitlich beschränkt sein; die gemeinschaftliche Ordnung soll also asynchron stattfinden können.

	A	B	C	D	E	F	G	H	I	J				
1	Priorisierung													
2	Peter		Paul		Maria		A1		A2		A3			
3	1	A2	A2		A3		Peter Paul Maria	3	1	2				
4	2	A3	A3		A2			3	1	2				
5	3	A1	A1		A1			3	2	1				
6							9		4		5			
7							Ergebnis		A2, A3, A1					
8														
9	Komplexitätsschätzung													
10	Peter		Paul		Maria		I1		I2		I3		I4	
11	1	I3	I1		I3		Peter Paul Maria	2	4	1	3			
12	2	I1	I3		I4			1	3	2	4			
13	3	I4	I2		I1			3	4	1	2			
14	4	I2	I4		I2		6		11		4		9	
15							Ergebnis		I3, I1, I4, I2					
16														

Berechnung der Ordnung für das Team (Bild 1)

Erste Einschätzung

Wie ist Ihre Einschätzung dieses Projekts nach dieser Anforderungspräsentation? Das ist natürlich nur eine Skizze, und Sie stehen in keinem Dialog mit einem Auftraggeber. Das ist für Sie in dieser Größenordnung vielleicht ungewohnt. Aber nehmen Sie es als Übung und lassen Sie sich darauf ein, selbst wenn ich im Folgenden andere Annahmen machen sollte als Sie. Es werden auch Lücken bleiben, die Sie gern schließen würden, ich aber offen lasse. Das ist dem Monolog geschuldet, den dieser Artikel darstellt, und auch seinem notwendig begrenzten Umfang.

Es geht allerdings nicht wirklich darum, diese Idee marktreif umzusetzen (so sehr mir das immer mal wieder in den Fingern juckt), sondern um die Demonstration der Anwendung von Slicing und Sleepy Hollow. Das kann auch mit divergierenden Annahmen und Lücken geschehen.

Damit im Hinterkopf: Wie ist dieses Projekt einzuschätzen? Machbar? Wie ist der Aufwand? Was braucht es dafür? Wie würden Sie vorgehen? Was ist Ihr Reflex?

So viel kann ich sagen: Das ist gut machbar. Der Aufwand ist überschaubar. Ich habe es auch schon mal umgesetzt [4], die Implementation ist allerdings im Web nicht mehr lauffähig. Das hat Spaß gemacht; ich fand das Tool sehr nützlich und „mal etwas anderes“ als das ewige Planning Poker.

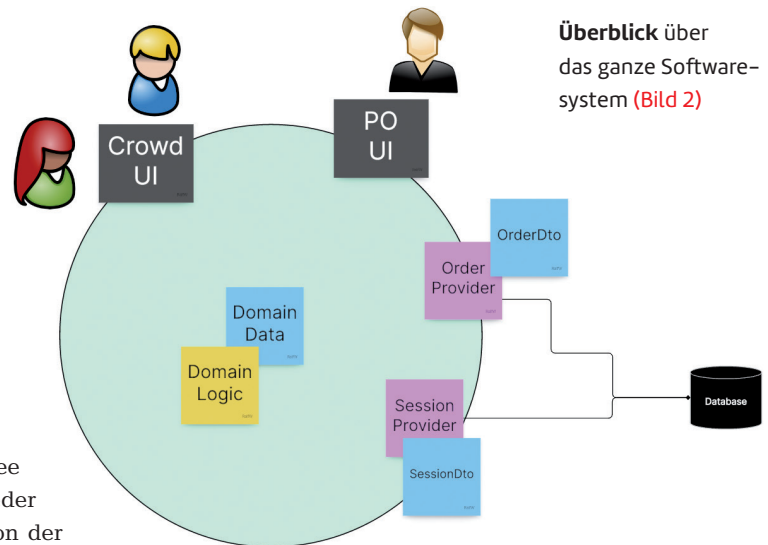
Was braucht es dafür?

- Entscheidungen für Technologien: Frontend, Backend, Persistenz und so weiter.
- Das ist sicher ein verteiltes System, da die Teammitglieder nicht co-located sein müssen.
- Die Usability bei der individuellen Ordnung sollte dem relativen, vergleichenden Schätzen gerecht werden. Drag-and-drop kommt Ihnen da vielleicht in den Sinn, um die Positionen leicht in eine Wunschordnung zu bringen.
- Wie steht es mit der Security? Wie viel Aufwand muss dafür getrieben werden? Meine Meinung: Es gibt kein Security-Problem. Die Daten, um die es geht, sind sehr flüchtig, nicht persönlich und enthalten keine Geheimnisse.
- Gibt es ein Skalierungsproblem? Meine Meinung: Nein, denn die Datenmengen sind marginal, die Zugriffsraten sind minimal, und die entscheidende Berechnung ist trivial und kann auch noch auf einem Client stattfinden.
- Die Domäne scheint einfach. Im Wesentlichen besteht sie aus der Logik, die individuelle Ordnungen der Teammitglieder zu einer gemeinsamen verrechnet.

Ist das eine Aufgabe, die etwas Systematik trägt, weil ich sie nicht „einfach runtercodieren“ kann? Aber sicher! Also los ...

Das ganze Softwaresystem aus 10 000 Meter Flughöhe

Die formellere Anforderungsanalyse mit Slicing beginnt in großer Flughöhe mit einem Blick auf das gesamte Softwaresystem. Was ist von dort aus mindestens zu sehen? Bild 2 zeigt den Überblick:

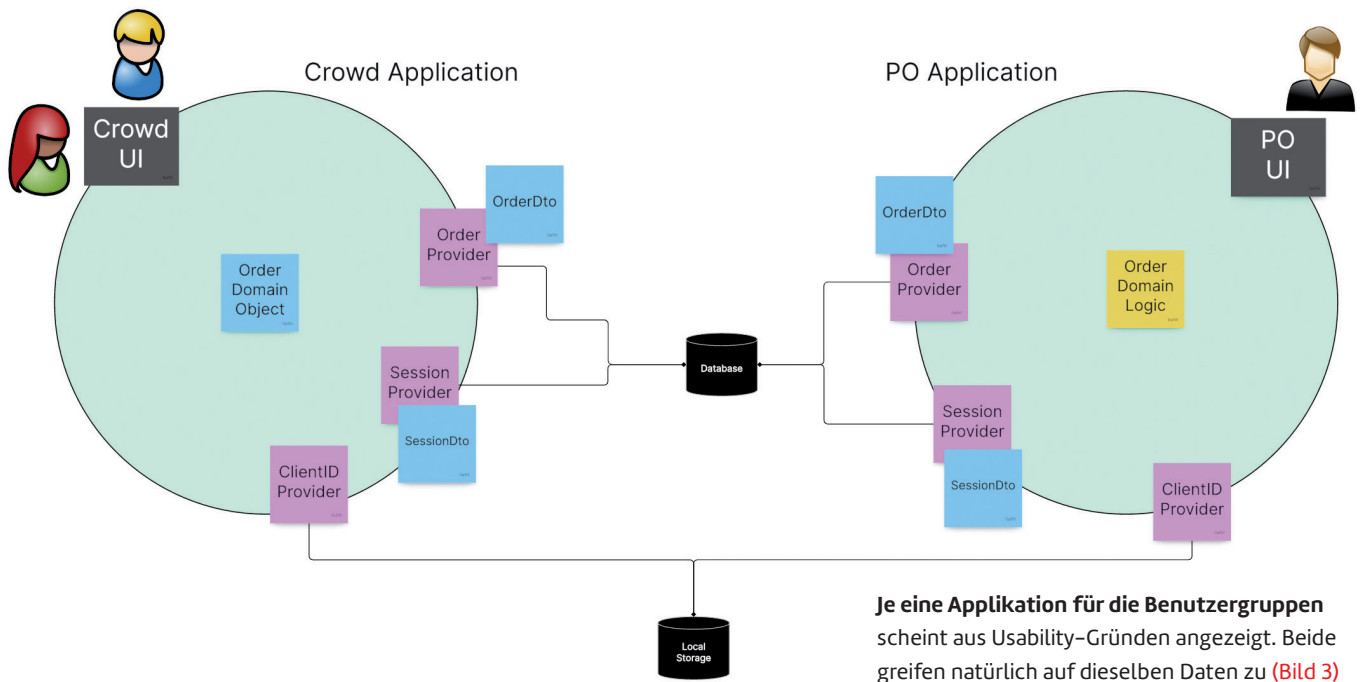


Überblick über das ganze Softwaresystem (Bild 2)

- Ich identifiziere zwei Benutzergruppen (Rollen): Da ist der PO, der die zu ordnenden Positionen festlegt; er leitet die „Ordnungssitzung“. Und da sind Teammitglieder (Crowd), die die Positionen in individuelle Ordnungen bringen. Jede Benutzergruppe verdient eine eigene Benutzerschnittstelle. Die Benutzerschnittstellen bilden den Rahmen für Interaktionen von Benutzern mit dem Softwaresystem.
- Andererseits braucht das Softwaresystem eine Ressource, eine Datenbank, um die Sitzungen mit ihren Positionen (oder Einträgen) beziehungsweise den verschiedenen Ordnungen der Teammitglieder zu speichern. Dafür scheinen mir zwei getrennte Provider nötig, die die persistenten Daten in interne Datenstrukturen transformieren.
- Außerdem gibt es noch nicht näher bestimmte Domänenlogik, die mit den Daten arbeitet, und vielleicht auch noch ausgewiesene Domänendatenstrukturen auf einem höheren Abstraktionsniveau als die DTOs der Provider.

Das sind alles keine tiefen Einsichten. Mich überrascht hier nichts. Das zu erkennen hat mich fünf Minuten gekostet. Dennoch ist dieser Überblick nützlich, denn er bietet mir eine erste Checkliste für Entscheidungen:

- Welche Technologie soll für die Benutzerschnittstellen zum Einsatz kommen? Ich entscheide mich zunächst der Einfachheit halber für ein Console-UI. Einem Web-UI steht später nichts im Wege, doch zu Anfang würde es mir mehr Kopfschmerzen in puncto Gestaltung und Implementation bereiten. Beides ist für das, worum es mir im Artikel geht, jedoch unwesentlich. Welche UIs es gibt, ist wichtig, nicht aber, wie genau sie ausgeprägt sind.
- Welche Technologie soll für die Datenbank zum Einsatz kommen? Auch wenn ich mir die Interaktionen noch nicht näher angeschaut habe, scheint mir nahezuliegen, dass die Zugriffsmuster sehr, sehr simpel sind. Im Grunde dreht sich alles um ein „Dokument“: eine Sitzung (Session) mit zu ordnenden Einträgen (Items). Zu einer Sitzung gehören verschiedene Ordnungen der Teammitglieder. Alles könnte in einem JSON-Objekt gespeichert werden. Aber irgendwie fühlt es sich nicht so gut an, mehrere Teammitglieder ►



potenziell gleichzeitig ein Sitzungsdocument mit ihren Ordnungen verändern zu lassen. Deshalb sehe ich die individuellen Ordnungen getrennt von den Sitzungen. Und wie sollen die Daten persistiert werden? Ich würde mal ganz einfach anfangen mit ... einem Dateisystem. Wenn ich die Daten in meiner Dropbox ablege, können mehrere Prozesse sehr einfach darauf zugreifen. Die Details sind in den Providern gekapselt. Wenn ich später meine, eine MongoDB einsetzen zu müssen, implementiere ich nur die Provider neu. Auch das Persistenzmedium ist für das, worum es mir hier geht, unwesentlich. Dass es eines gibt und in welcher Granularität es angesprochen wird, ist entscheidend; das drückt sich in den Providern aus.

- Soll das Softwaresystem verteilt laufen? Sicher. Das ist nötig, um den Teammitgliedern Freiheit in Bezug auf Ort und Zeit für ihre Ordnungen zu geben.

Näheres zu Domäne oder Datenstrukturen will ich an dieser Stelle noch nicht entscheiden, dazu möchte ich näher heranzoomen ...

Applikationen für jede Rolle

Die Verteilung des Codes für das Softwaresystem wird von nichtfunktionalen Anforderungen getrieben. Wenn Logik eine nichtfunktionale Anforderung nicht erfüllen kann, dann muss verteilt werden. Das ist offensichtlich hier der Fall: Die Rollen haben sehr unterschiedliche Anforderungen an die Benutzerschnittstelle. PO und Crowd gehen mit den Sitzungen und ihren Einträgen jeweils ganz anders um. Das scheint mir konsequent ausgedrückt durch spezialisierte Applikationen für die Rollen (Bild 3).

Bei dieser Spaltung des Softwaresystems frage ich mich, wie Benutzer sich ihren Applikationen gegenüber identifizieren (Authentifizierung). Ist eine Identifikation überhaupt nötig? Ja, ich denke, eine Identifikation ist aus zwei Gründen nötig:

- Ein PO soll nur seine Sitzungen bearbeiten können. Sitzungen haben zwar eine ID, doch wenn die außerhalb des Teams bekannt würde, könnte ein anderer PO die Sitzung einsehen. Das ist zwar kein großes Problem, glaube ich, aber das zu unterbinden ist einfach mit einer Identifikation des PO.
- Ein Teammitglied soll den Einträgen einer Sitzung nicht mehrere Ordnungen geben können. Veränderung ist zwar möglich, doch dadurch wird keine weitere Ordnung angelegt. Also muss erkannt werden, wer gerade die Einträge einer Sitzung „in Ordnung bringen will“.

Damit bei der Identifikation keine persönlichen Daten notwendig werden, deren Validität und Zugehörigkeit auch noch geprüft werden müssten, will ich es mir einfach machen: Jeder Client, den PO oder Teammitglieder nutzen, generiert eine ID bei der ersten Nutzung und merkt sie sich. Fertig. Dass innerhalb einer Sitzung die Clients gewechselt werden, ist unwahrscheinlich. Dass das (bei Teammitgliedern) „zum Mogeln“ bewusst gemacht wird, ist möglich, aber ich will es nicht mit mehr Aufwand berücksichtigen.

Also reicht es, bei den Applikationen noch einen weiteren Provider für die Client-ID hinzuzufügen, die in irgendeiner Form von lokalem Speicher (auf der Festplatte oder im Browser) gehalten wird.

Die Domäne habe ich nach ein wenig Nachdenken jetzt ebenfalls differenziert:

- In der PO-Applikation ist Domänenlogik nötig, um all die individuellen Ordnungen der Teammitglieder zu einer gemeinsamen zu verrechnen. Dort gibt es zu Recht den Überblick über alle eingereichten Ordnungen.
- In der Crowd-Applikation für die Teammitglieder ist wahrscheinlich nur eine Datenstruktur nötig, die es einfach macht, Positionen einer Sitzung neu anzuordnen. Logik für „Berechnungen“ ist dort nicht gefragt.

So weit der erste Zoomschritt in das Softwaresystem hinein. Sind damit alle nichtfunktionalen Anforderungen bedacht? Nein. Also noch mal hineinzoomen ...

Worker für die Sicherheit

Die Persistenz findet auf einem anderen Rechner statt als den Client-Rechnern für PO und Teammitglieder. Von Verteilung würde ich deshalb aber noch nicht sprechen. Wenn ein PO-Client auf eine Festplatte auf einem Server zugreift, tut er das mit eigenem Code. Das Persistenzmedium erscheint ihm als „dumm“; in der Nutzung der APIs des Persistenzmediums steckt die Intelligenz, die liegt beim Client.

Dafür braucht ein Client jedoch das Wissen über die APIs und auch noch unmittelbare Zugriffsrechte auf das Persistenzmedium. Bei einer Inhouse-Desktop-Applikation würde ich darüber nicht weiter nachdenken. Wenn die Applikationen jedoch potenziell im Web laufen und dann auch noch Code im Browser am Start ist ... würde ich die APIs und Credentials nicht in den Clients haben wollen.

Diese minimale Sicherheitsanforderung veranlasst mich dazu, ein Backend für die Datenbank auszuprägen. Das Softwaresystem zerfällt damit in drei Worker (Bild 4). Wie die Worker zu ihren Zugriffsrechten auf den Datenbankserver kommen (Autorisierung), ist eine andere Sache. Die lasse ich hier aus, weil ich schon mit der bisherigen Sicherheitsanforderung zu einem Backend-Worker gekommen bin. Mehr wollte ich nicht.

Im Worker für die Datenbank erwarte ich keine Domänenlogik. Eine solche könnte dort zentral angesiedelt werden, falls sie von der Nähe zu den Daten profitieren würde, doch in diesem Szenario ist das nicht der Fall. Das Backend ist insofern im Grunde nur ein Gateway zu den konkreten Funktionen für den Umgang mit dem Persistenzmedium.

Die Frontends sind Clients, die in Terminalfenstern laufen. Für meine Zwecke hier ist das der kleinstmögliche Einstieg.

Aber wie setze ich das Backend um? Es soll HTTP-Endpunkte für den Aufruf der Frontends bieten und auf meine Dropbox zugreifen. Ich könnte es mit C# oder Typescript unter Nutzung des Dropbox-API realisieren. Da ich jedoch Fan der Automatisierungsplattform Make.com bin, sehe ich natürlich überall Nägel für diesen Hammer: Das Backend realisiere ich in Form von Make.com-Szenarien. In Make.com ist der Zugriff auf die Dropbox trivial und schön visuell.

In Bild 4 unterscheidet sich das Backend in der grundsätzlichen Darstellung nicht von den Frontends. Das ist korrekt: Egal wie die Implementation aussieht, es wird Portale, Provider und Domäne geben. Die werden in immer gleicher Weise dargestellt. Die Ausprägungen hingegen mögen stark variieren:

- Wenn ich mich entscheiden sollte, das Crowd-Frontend mit C# und das PO-Frontend mit Python zu implementieren, ist der Code für die gleich benannten Provider natürlich unterschiedlich.
- Ein Provider in C# sieht sicher anders aus als einer, der in Make.com den Zugriff auf die Dropbox tatsächlich durchführt.

Die „Kärtchen“ in den Abbildungen sind nur Platzhalter. Sie zeigen an, was ich mindestens brauche und nicht vergessen darf. Implementationsdetails sind auf dieser Flughöhe nicht zu sehen.

Bisher hat die Flughöhe auch immer noch einen Überblick über das Gesamtsystem gestattet. Ich bin breadth-first vorgegangen. Mit dem nächsten Schritt näher ans System heran wird sich das ändern. Ich muss mich konzentrieren, um genauer hinschauen zu können.

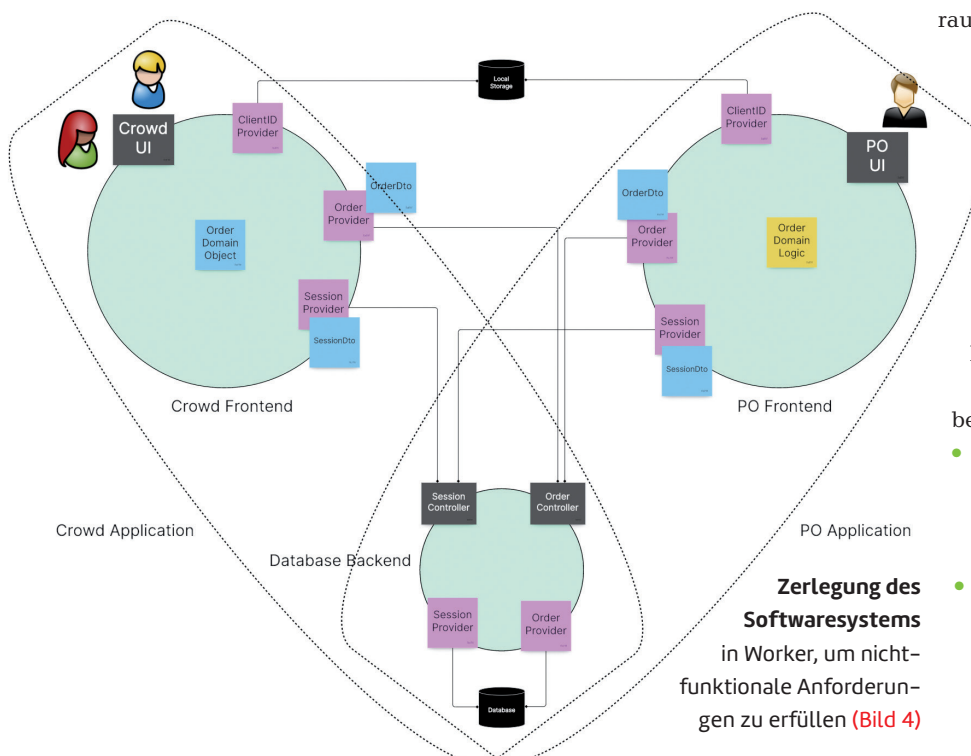
Weiter geht es mit dem konkreten Umgang der Rollen mit den Frontends. Wie soll die Bedienung aussehen? Daraus leiten sich Schnittstellen und Datenstrukturen ab.

Das Frontend für den PO

Der PO stößt die Ordnung von Positionen für einen gewissen Zweck an. Er definiert eine Sitzung mit ihren Positionen, die anschließend von den Teammitgliedern individuell geordnet werden sollen – um am Ende zu einer gemeinschaftlichen Ordnung verrechnet zu werden.

Ich sehe ganz grob folgende Aktivitäten beim PO:

- Der PO legt eine Sitzung mit einer kurzen Beschreibung an, zum Beispiel „Sprint KW 19: Ordne die User Stories in absteigender Komplexität an.“
- Der PO füllt die Sitzung mit Einträgen, das heißt den zu ordnenden Positionen, zum Beispiel „User können sich mit E-Mail und Passwort anmelden“, „User können ihr Passwort zurücksetzen“, „User können ►



Zerlegung des Softwaresystems
in Worker, um nicht-funktionale Anforderungen zu erfüllen (Bild 4)

sich mit E-Mail und Passwort registrieren“, „User bekommen eine Registrierungsbenachrichtigung per E-Mail und können sie bestätigen“.

- Der PO veröffentlicht die Sitzung zur Ordnung durch die Teammitglieder.
- Der PO verfolgt die Entwicklung der gemeinschaftlichen Ordnung.

Das ist das Mindeste. Weitere Aktivitäten könnten sein: Der PO schaut sich ältere Sitzungen an, der PO löscht Sitzungen, der PO verändert eine Sitzung, der PO leitet eine neue Sitzung von einer vorherigen ab, Veränderungen an der gemeinschaftlichen Ordnung werden automatisch dargestellt ...

Im Slicing können Sie den Umgang mit einem Softwaresystem in drei Schritten konkretisieren:

- Perspektiven
- Dialoge
- Interaktionen

Wie bei Applikationen und Workern ist kein Slice zwingend zu schneiden. Sie greifen dort zu, wo es nützlich für Sie ist, um Klarheit zu gewinnen. Ziel sind die Interaktionen. Sie repräsentieren testbare Eintrittspunkte im Code.

Die Frage ist: Was soll automatisiert testbar werden, und was soll andererseits ein Detail der Benutzerschnittstelle bleiben und nicht automatisiert getestet werden?

Aus meiner obigen Beschreibung der PO-Aktivitäten leite ich schon mal zwei testwürdige Interaktionen ab:

- Veröffentlichung einer Sitzung: Die Sitzung mit ihren Einträgen wird im Backend gespeichert. Anschließend können die Teammitglieder mit ihrem Crowd-Frontend darauf zugreifen.
- Anzeige des aktuellen Stands der gemeinschaftlichen Ordnung für die Reflexion mit dem Team: Die von den Teammitgliedern eingereichten individuellen Ordnungen werden vom Backend geladen und zu einer gemeinschaftlichen Ordnung verrechnet.

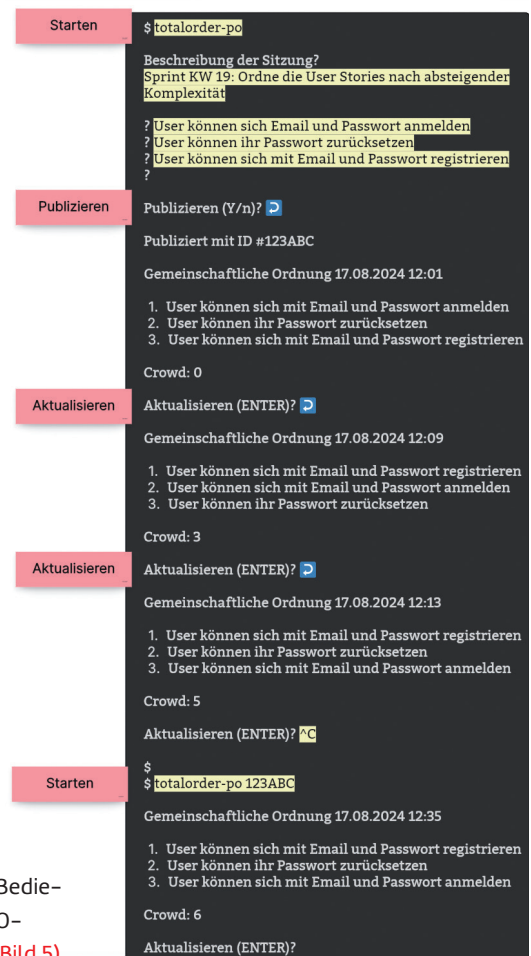
Wie wird eine Sitzung zusammengestellt? Das ist ein Detail der Benutzerschnittstelle. Im allereinfachsten Fall könnte ein PO eine JSON-Struktur in ein Textfeld eintragen. Aber das ist natürlich nicht benutzerfreundlich, also wird das UI es dem PO etwas bequemer machen, die Daten zu sammeln.

Wenn das ebenfalls testwürdig sein sollte, kann ich weitere Interaktionen explizieren, zum Beispiel:

- Sitzung anlegen mit einer kurzen Beschreibung.
- Sitzung um einen Eintrag erweitern.
- Einen Eintrag aus einer Sitzung löschen.

Der Einfachheit halber möchte ich mich jedoch auf die beiden obigen Interaktionen konzentrieren: Veröffentlichung, Reflexion.

Jetzt bin ich aber schon gleich auf der Ebene der Interaktionen gelandet. Was ist mit Perspektiven und Dialogen? Wie gesagt: Sie sind nicht zwingend für das Ziel von Slicing. In diesem Fall brauche ich sie nicht, glaube ich. Die Benutzerschnittstelle ist nicht weitläufig, also sind Perspektiven über-



Skizze der Bedienung des PO-Frontends (Bild 5)

flüssig. Und ich denke, ich komme mit zwei Dialogen aus:

- Publizieren
- Reflektieren

Oder beide Interaktionen könnten sogar in einem Dialog zusammengefasst sein. Bild 5 skizziert, wie ich mir den Umgang mit dem PO-Frontend in einem Terminalfenster vorstelle. Hervorgehoben habe ich Benutzereingaben und Interaktionen. Hinzugekommen ist noch eine Interaktion fürs Starten, um auch bei erneutem Programmaufruf auf eine vorherige Sitzung zugreifen zu können.

Aber ich könnte natürlich auch eine stärker visuell ausgezeichnete Benutzerschnittstelle entwerfen (Bild 6). An den Interaktionen ändert das nichts. Auch hinter einem GUI oder einem Web-UI stehen dieselben Entry-Point-Funktionen.

Sie sehen, aus Sicht des Entwicklers ist das UI zunächst nicht so wichtig. Es trägt nichts zur „eigentlichen Verarbeitung“ bei. Das Verhalten der Software insgesamt, das heißt die Transformation von Input in Output, findet hinter dieser Fassade statt.

Das macht die Fassade nicht unwichtig. Ohne gute Usability wird auch die schönste Transformation nicht genutzt. Doch ohne saubere Transformation, das heißt korrekte, verständliche, wandelbare Softwarestrukturen, lohnt sich kein Aufwand für Usability. Deshalb ist das UI im Detail für mich zweitrangig. Ich will mir (zunächst) nur so viel davon klarmachen, wie

ich brauche, um die Interaktion zu erkennen, die mir die Slicing-Entry-Points auf den Sleepy-Hollow-Processor-Schnittstellen beschreiben.

Wenn ich sie nach Bild 2 schon hätte listen können, hätte ich mir einigen Aufwand sparen können. Doch auf dieser Flughöhe habe ich sie noch nicht gesehen. Also habe ich hineingezoomt. Die Slicing-„Zoomstufen“-Perspektive konnte ich allerdings überspringen. Und bei Interaktionen in Dialogen bin ich nicht streng „von oben nach unten“ vorgegangen, weil der Scope des Frontends so überschaubar ist.

Das Frontend für die Crowd

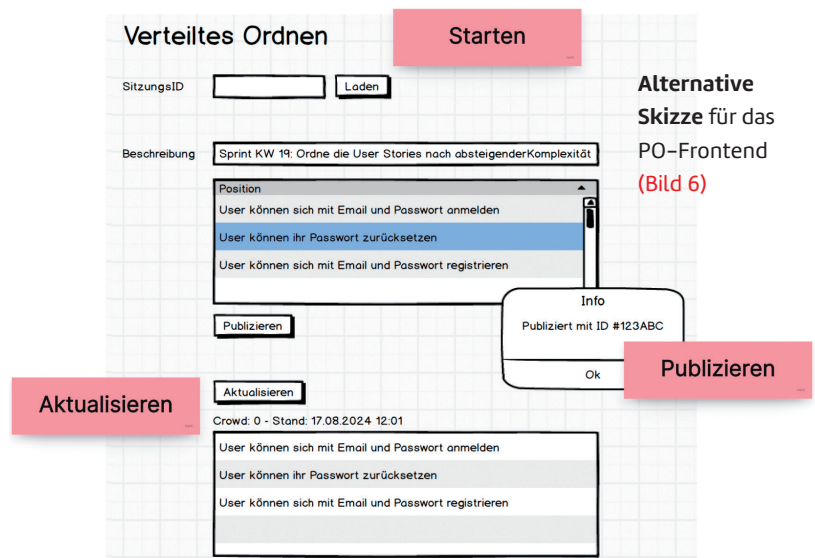
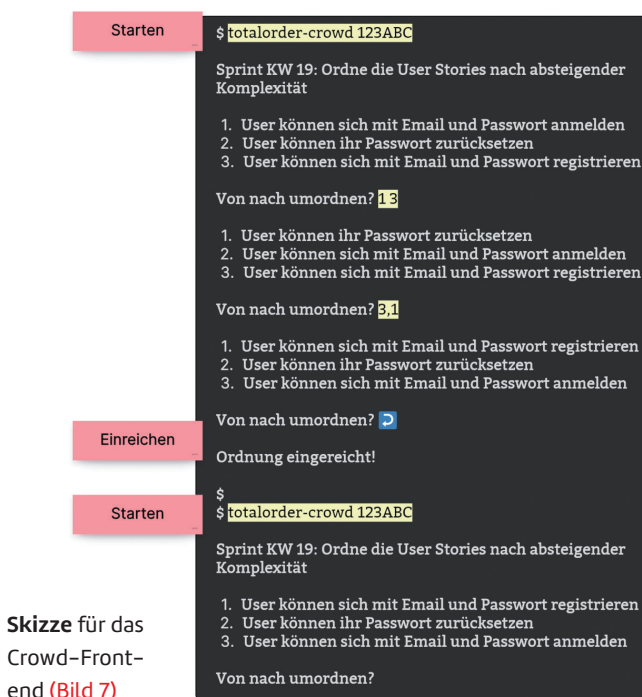
Für die Teammitglieder sieht der Client noch einfacher aus. Was ist ihre Aufgabe? Ordnung in die Positionen einer Sitzung bringen: Sobald eine Sitzung publiziert ist, können Teammitglieder sie öffnen, sehen die Positionen in ihrer Ordnung (beziehungsweise zu Anfang in der erfassten Reihenfolge) und ordnen sie um. Wenn sie mit ihrer Ordnung zufrieden sind, reichen sie sie ein zur Verrechnung mit allen anderen eingereichten, individuellen Ordnungen.

Hören Sie schon aus dieser Beschreibung die Interaktionen heraus? Oder wäre es Ihnen lieber, sich an sie schrittweise über Perspektiven und Dialoge heranzuarbeiten?

Ich denke, wir können einen Sprung wagen. Die Interaktionen für den Crowd-Client sind:

- Starten mit Öffnung einer Sitzung.
- Einreichen der eigenen Ordnung.

Wie die eigene Ordnung hergestellt wird, ist ein Detail der Benutzerschnittstelle. In einem GUI- oder Web-Frontend geschieht das anders als in einem Console-Frontend (Bild 7).



Theoretisch könnte ein solches verteiltes Ordnen auch ohne Benutzerschnittstelle ablaufen:

- Der PO legt in einem Dropbox-Verzeichnis eine Datei (*session.txt*) mit den zu ordnenden Positionen ab. In jeder Zeile steht eine Position mit einer ID, zum Beispiel einem Index 1..n.
- Die Teammitglieder öffnen diese Datei und speichern ihre individuelle Ordnung in einer neuen Datei (*peter.txt*, *paul.txt*, *maria.txt*) als Folge von Position-IDs, zum Beispiel „2, 1, 3“.
- Ein automatischer Prozess (oder ein von Hand gestarteter) erzeugt eine Ergebnisdatei, die die Positionen mit IDs und Bezeichnungen in einer weiteren Datei ablegt (*result.txt*).

Fertig. In dieser Variante gäbe es nur eine einzige Interaktion: den Start des Verrechnungsprozesses.

Was ich damit sagen will: Wir lassen uns oft durch Benutzerschnittstellen ins Bockshorn jagen. Und mit grafischen Benutzerschnittstellen, die hübsch und animiert und ganz CI-konform sein sollen, gleich doppelt. Die eigentlich interessante Verhaltensweise von Software hat oft nichts damit zu tun.

Darum finde ich es beim Slicing wichtig, die Benutzerschnittstelle so abstrakt wie möglich, so konkret wie nötig zu betrachten. Verlieren Sie nicht das Ziel aus den Augen: Es geht (zunächst) nur darum, die Entry Points zu finden. Denn sind die Entry Points definiert, kann die Arbeit am zentralen Verhalten der Software beginnen. Die Domäne liegt hinter der Schnittstelle der Processor-Integrationen von Sleepy Hollow.

Indem ich hier zwei Applikationen mit eigenen Frontends „herausanalysiert“ habe, bin ich also schon über einen Proof of Concept (PoC) weit hinausgegangen. Ob das Verfahren „relative Ordnung“ (oder konkreter: relative, vergleichende Schätzung) überhaupt interessant ist, könnte einfacher getestet werden.

Ein allererster PoC für die Machbarkeit könnte mit Google Sheet realisiert werden. Der PO trägt in einer Google-Sheet-Datei die Positionen ein, und die Teammitglieder „nummerieren“ diese in je eigenen Spalten nach ihren Vorstellun- ►

gen. Ständig wird eine gemeinschaftliche Ordnung auf einem zweiten Tabellenblatt angezeigt. Das sollte ein No-brainer sein.

So wäre die Herangehensweise effizient und effektiv. Ob es überhaupt mehr braucht als ein Google Sheet, würde sich herausstellen. Allerdings bliebe dabei der Programmierspaß auf der Strecke, und ich müsste mir ein anderes Szenario für die Demonstration von Slicing und Sleepy Hollow ausdenken.

Aber sehen wir einmal darüber hinweg. Wir sind hier in einem Sandkasten, in dem vieles nicht ganz real(istisch) ist. Zumindest einmal wollte ich jedoch so weit zurücktreten und reflektieren. Nicht, dass Slicing oder Sleepy Hollow in den Verruf kommen, Projektaufwände in die Höhe zu treiben. Nein, im Gegenteil: Es geht mir stets um „das rechte Maß“ (das oft kleiner ist als das, wohin ein Projektteam tendiert).

Die Entry Points für die Interaktionen

Für mein Gefühl ist genügend Klarheit an der Oberfläche hergestellt. Die grobe Struktur des Softwaresystems ist definiert, die Interaktionen sind herausgearbeitet. Damit bin ich bereit, unter die Oberfläche zu tauchen.

PO Frontend		Ressourcen		Signatur	
	Entry Point	Datenbank	Client ID	Request	Response
	start	read	read/write	sessionID:string	description:string, items:string[]
	publish	write		description:string, items:string[]	sessionID:string
	update	read			items:string[]
Crowd Frontend					
	start	read	read/write	sessionID:string	description:string, items:{index:number, title:string}[]
	submit	write		indexes:number[]	

Signaturen der Entry Points (Bild 8)

Unter der Oberfläche der Interaktionen, die in Dialogen von Workern in Applikationen stattfinden, liegt Code. Das Pendant von Interaktionen sind Entry Points, das heißt Funktionen, die im Rahmen der Interaktionen getriggert werden. Sie sind Ansatzpunkte für automatisierte Tests. Durch sie kann ein Softwaresystem „headless“ betrieben werden – oder eben auch mit verschiedenen Heads. Deshalb ist mir das Sleepy-Hollow-Architekturmuster so wichtig.

Bei geklärten Interaktionen ist also jetzt die Frage, wie dazu die Entry Points aussehen sollten. Das Command-Query-Separation-Prinzip (CQS) kann hier auch helfen, um noch weiter zu differenzieren und Entry Points nicht zu überladen.

Zu Entry Points gehört als Erstes eine Signatur mit typisierten Parametern und Rückgabewert. Für mich ist das in erster Linie ein „Wunschkonzert“: Wie stelle ich mir die Entry Points idealerweise vor? Wie vermeide ich, dass an ihnen (enge) Kopplung zwischen Head und Body entsteht?

Ich versuche es mit einem ersten Pass über die Interaktionen mit Blick durch die CQS-Brille:

PO-Frontend

- **Starten:** Eine Query, die gegebenenfalls eine Sitzung für eine gegebene ID herausucht.
- **Publizieren:** Ein Command, das eine Sitzung speichert, also den Datenbankzustand verändert.
- **Aktualisieren:** Eine Query, die individuelle Ordnungen liest und zu einer gemeinschaftlichen verrechnet (die nicht in der Datenbank gespeichert wird).

Crowd-Frontend

- **Starten:** Eine Query, die eine Sitzung und gegebenenfalls die schon vorhandene individuelle Ordnung eines Users herausucht.
- **Einreichen:** Ein Command, das eine individuelle Ordnung in der Datenbank speichert.

Aus CQS-Sicht sind die Interaktionen also sauber. Das ist ein guter Anfang. Jetzt weiter ins Detail: die Signaturen.

- Was soll an einen Entry Point übergeben werden? Was wurde in der Benutzerschnittstelle gesammelt?
- Was soll ein Entry Point liefern? Was soll in der Benutzerschnittstelle projiziert werden?
- Welche Seiteneffekte (Zustandsänderungen) treten auf? Welche Daten(strukturen) zum Beispiel in einer Datenbank sind wie beteiligt?

Bild 8 zeigt, wie ich mir die Signaturen und die groben Seiteneffekte vorstelle. Wie Sie sehen, ist aus der reinen Query *Starten* nun ein Hybrid geworden. Es wird nicht nur aus der Da-

```

1  type SessionID = string;
2  type Description = string;
3  type Item = string;
4
5  0 implementations
6  interface IP0Processor {
7      // Start method
8      start(sessionID: SessionID): Promise<{
9          description: Description;
10         items: Item[];
11     }>;
12
13     // Publish method
14     publish(description:string, items:Item[]): Promise<{
15         sessionID: SessionID;
16     }>;
17
18     // Update method
19     update(): Promise<{
20         items: Item[];
21     }>;
22
23  0 implementations
24  interface ICrowdProcessor {
25      // Start method
26      start(sessionID: SessionID): Promise<{
27          description: Description;
28          items: Array<{
29              index: number;
30              title: string;
31          }>;
32      }>;
33
34      // Submit method
35      submit(indexes: number[]): Promise<void>;
36  }

```

Die Schnittstellen für die Entry Points (Bild 9)

tenbankressource und auch der ClientID-Ressource gelesen. Falls die Client-Instanz noch keine ID hat, wird auch eine geschrieben. Das halte ich aber für einen vernachlässigbaren Seiteneffekt, der auch nur einmal pro Instanz auftritt. Für mich bleibt das Starten also eine Query.

Eine KI wie ChatGPT oder Claude hilft mir, aus der Tabelle zwei Interfaces abzuleiten (Bild 9). Durch das Sleepy-Hollow-Architekturmuster weiß ich, dass und wo ich sie brauche (siehe unten).

Woher weiß ich, dass die Entry Points „gut und richtig“ sind? Ich weiß es nicht. Ich muss es auch nicht zu 100 Prozent wissen. Dieser erste aus der Anforderungsanalyse abgeleitete Entwurf für die Schnittstelle des Bodys muss nur gut genug sein, um mir Gedanken über Akzeptanztests machen zu können. Während ich die durchgehe, merke ich, ob die Entry Points wirklich plausibel sind. Test-first-Vorgehen hilft bei der Ausgestaltung von Schnittstellen. Eine Form von „eating your own dog food“.

Die Tests stellen sehr reduzierte Benutzerschnittstellen dar. Ich merke also schon mit ihnen, ob meine Idee der Zustandshaltung in den Prozessoren taugt. Haben Sie bemerkt, dass ich die Interfaces so gestaltet habe, dass ihre Implementationen zustandsbehaftet sein müssen? Sie sehen es zum Beispiel daran, dass bei `IPOProcessor.update()` keine `sessionID` übergeben wird. Woher weiß die Implementation, um welche Session es geht? Sie hat sich die Session vorher bei `start()` oder `publish()` gemerkt.

Prozessoren mit Zustand ziehe ich vor, weil damit mehr Logik testbar wird. Ein Head über dem Sleepy-Hollow-Body soll so wenig wie nur irgend möglich leisten. Es kann daher sogar dazu kommen, dass ich die Prozessoren nochmal mit einer weiteren Schicht umwickle, die dem UI Arbeit abnimmt und sie testbar macht. Mal sehen ...

Die Sleepy-Hollow-Architektur

Jetzt rede ich immer von Prozessoren, auf denen die Entry Points sitzen, aber wo sind die in der Architektur, wie ist deren Verhältnis zu den Strukturelementen, die ich schon herausgearbeitet habe?

Betrachten wir zur Erinnerung noch einmal das Sleepy-Hollow-Architekturmuster [3] in Bild 10. Zu sehen sind die Arbeitspferde einer Anwendung: Portale, Provider und Domäne. Darin steckt die verhaltenserzeugende Logik.

Ins Auge stechen jedoch mehr die gelben Integratoren. Ihre Aufgabe ist es, ganz ohne Logik die Arbeitspferde zu einem leistungsfähigen Gespann zusammenzuschirren. Diese Integratoren, die in einer Übersicht wie Bild 3 ganz bewusst nicht zu sehen sind, machen das Besondere der IODA-Architektur [6] und des darauf aufbauenden Sleepy-Hollow-Architekturmusters aus.

Als Body, der die eigentliche Leistung einer Anwendung erbringt, bezeichne ich

vor allem das, was von der Processor-Integration verbunden wird. Aber auch der Interactor kann noch dazugezählt werden, weil er ebenfalls dafür sorgt, dass Logik von den technologischen Feinheiten eines UI getrennt ist. Aber der Interactor ist optional. Der Body ist vergleichsweise leicht zu testen (auch wenn darin Ressourcenabhängigkeiten stecken).

Der Head ist das UI in der einen oder anderen Ausprägung. Es ist technologiespezifisch und schwer zu testen. Deshalb wird es vom Body getrennt, isoliert.

Wie dieses Muster auf das hiesige Softwaresystem angewendet werden kann, zeigt Bild 11. Die Strukturelemente der bisherigen „Zoomstufen“ finden sich darin – und auch die neuen Integrationsbausteine: App, Interactor und Processor.

Die Schnittstellen aus Bild 9 sind die für die Integratoren in der Mitte, die mit dem „You are here“-Symbol gekennzeichnet sind. Die Anforderungsanalyse hatte also vor allem das Ziel, genau diese Integrationen zu bestimmen und auszuprägen. Sie sind die Ansatzpunkte für Akzeptanztests; von ihnen aus werden die Schnittstellen der darunter hängenden schon sichtbaren und weiterer noch zu findender Bausteine getrieben.

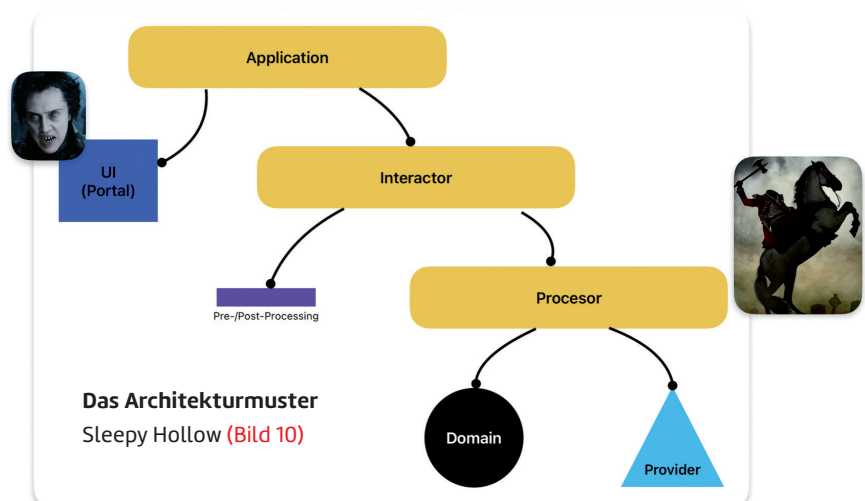
Die Verteilung auf Worker habe ich in Bild 11 bewusst übergangen. Sie trägt nichts zum Verständnis des grundlegenden Architekturmusters bei. Am Ende läuft der Durchstich je Interaktion mit oder ohne Verteilung vom Prozessor bis zum Provider mit Ressourcenzugriff.

Akzeptanztests für die Entry Points

Da stehe ich nun mit zwei Schnittstellen für Integrationen „irgendwo in der Mitte“ der Architektur. Was soll das? Ist das nicht ein bisschen wenig?

Nein, das ist der natürliche Endpunkt der Anforderungsanalyse. Der ist sogar schon viel detaillierter als alles, was üblicherweise bei der Anforderungsanalyse herauskommt. Überlegen Sie einmal, womit Sie Ihr PO sonst so zurücklässt, wenn er meint, die User Stories seien besprochen.

Mit dem Sleepy-Hollow-Architekturmuster ist von vornherein klar, dass es Prozessoren gibt, die „ein API“ mit Funktionen anbieten, die quasi eins zu eins den Interaktionen der Benutzerschnittstelle entsprechen. Sie wissen sogar, was ►



grundsätzlich darüber und darunter passiert. Genaue Kenntnis ist jedoch noch nicht nötig. Für den nächsten Schritt, den wahren Abschluss der Anforderungsanalyse, reicht es, dass Sie die Prozessoren mit ihren Schnittstellen kennen. Denn mit diesen Schnittstellen legen Sie den PO final auf den Grill: Sie fordern von ihm Testszenarien, die Sie dann mit den Entry Points umsetzen. Ohne solche Akzeptanztests sollten Sie den PO nicht ziehen lassen. Es droht Unterspezifikation mit bitteren Folgen: unnötige Kommunikation, Konflikte, Unzuverlässigkeit, Nachbesserungen.

Akzeptanztests, die für den PO Sinn ergeben, sind Teil des Full Kitting [5], das Sie betreiben sollten, bevor Sie sich zu weiterem Entwurf und Umsetzung zurückziehen. Sie sollten alles bei der Hand haben, was Sie brauchen, um gut passenden Code „in einem Rutsch“ liefern zu können. Das ist vor allem ... Klarheit. So viel Klarheit, wie Sie nur kriegen können. Klare Schnittstellen und klare Erwartungen an Output für Input sind das Mindeste.

Akzeptanztests sind natürlicherweise Integrationstests. Sie setzen an Bausteinen an, die auf die eine oder andere Weise nur „die Spitze des Eisbergs“ der Verarbeitung darstellen. Es stellt sich daher die Frage, ob Sie im Akzeptanztest hinunter bis auf die eigentlichen Ressourcen testen (hier: Dropbox-Dateien) oder ob Sie einfachere, billigere, besser kontrollierbare Ressourcen einschalten (Attrappen der einen oder anderen Art).

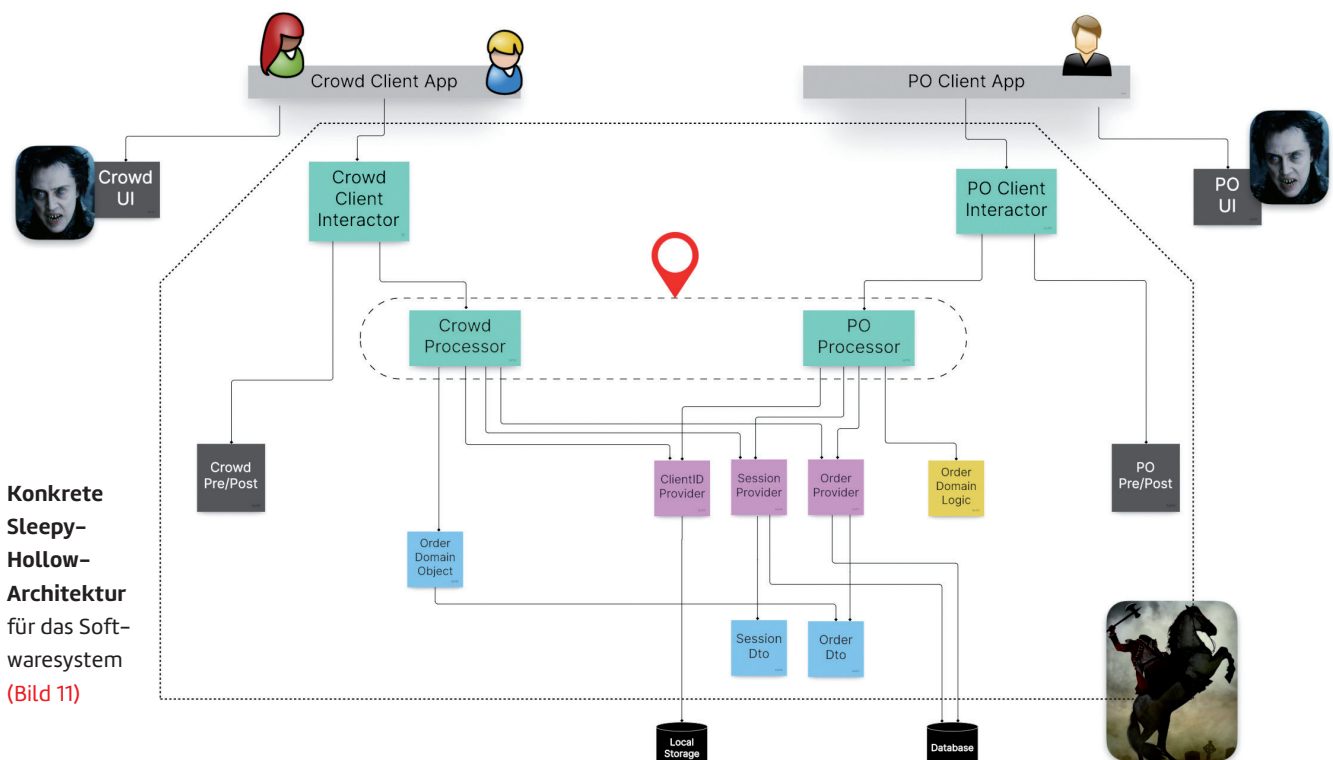
Diese Entscheidung möchte ich hier nicht weiter diskutieren. Nur so viel sei gesagt: Wenn Sie sich für Attrappen entscheiden, müssen Sie für Akzeptanztests auch die Schnittstellen der Provider für zu ersetzende Ressourcen kennen, um dafür Surrogate bauen und Prozessoren unterstecken zu können. Das erfordert zu diesem Zeitpunkt weiteren Entwurfsaufwand und verzögert das Aufsetzen von Akzeptanztests.

Die Formulierung der Testszenarien selbst ist davon jedoch unabhängig. Darin spielen nur die unmittelbar zu testenden Bausteine (hier: die Prozessoren) eine Rolle.

Aufgabe von Akzeptanztests ist nicht, die komplette Logik eines Bodys durchzuexerzieren. Es geht um das Big Picture, die wichtigsten Zusammenhänge, ein Gefühl dafür, ob typische Use Cases abgedeckt sind.

Für das vergleichende Ordnen mögen dies repräsentative Akzeptanztests sein:

- Neue Sitzung publizieren.
 - Starten ohne SessionID.
 - Publizieren einer Session mit der Beschreibung „B“ und den Einträgen „X“, „Y“, „Z“. Es wird eine SessionID geliefert.
 - Aktualisieren der Session in der Erwartung, dass die Einträge „X“, „Y“, „Z“ in der Reihenfolge geliefert werden.
- Sitzung individuell das erste Mal ordnen.
 - Starten mit einer SessionID. Es werden Beschreibung „B“ und die Einträge [{0, „X“}, {1, „Y“}, {2, „Z“}] geliefert.
 - Einreichen der individuellen Ordnung [2,1,0].
- Sitzung individuell das erste Mal durch einen weiteren Client ordnen lassen.
 - Starten mit einer SessionID. Es werden Beschreibung „B“ und die Einträge [{0, „X“}, {1, „Y“}, {2, „Z“}] geliefert.
 - Einreichen der individuellen Ordnung [0, 2, 1].
- Starten einer existierenden Sitzung und Aktualisieren nach individuellen Ordnungen.
 - Starten mit SessionID; es werden Beschreibung „B“ und die Einträge „X“, „Y“, „Z“ geliefert.



Konkrete Sleepy-Hollow-Architektur für das Softwaresystem (Bild 11)

- Aktualisieren der Session. Die Erwartung ist, dass sich die Ordnung gegenüber der ursprünglichen verändert hat. Sie soll nun „Z“, „X“, „Y“ lauten.

Ich habe diese Tests zwar separat formuliert, doch für mich spannen sie eigentlich einen Bogen. Ich würde sie zusammen laufen lassen. Das nenne ich ein Szenario. In diesem Szenario wird eine Sitzung definiert und publiziert, von zwei Teammitgliedern individuell geordnet und am Ende das Ergebnis betrachtet. Da ist alles Wesentliche dabei, das in der Praxis über zwei Clients verteilt passiert, nur ist es hier für besseren Überblick und bessere Testbarkeit in einem Test zusammengefasst.

Auf diese Weise muss ich mir weniger Gedanken über die Überprüfung von Zuständen machen. Wird die Einreichung einer individuellen Ordnung wirklich gespeichert? Das merke ich am Ende, wenn das Resultat stimmt oder nicht.

Allerdings fühle ich jetzt schon, dass hier und da die Prozessor-Schnittstellen nicht ausreichen. Wie unterscheide ich zum Beispiel die beiden Crowd-Clients? Sie laufen während des Tests ja im selben Prozess, dürfen jedoch nicht dieselbe ClientID haben. Brauche ich dafür eine Attrappe für den ClientID-Provider? Das ist ein Detail, das ich bei der Implementation des Tests entscheide.

Wie steht es mit der Abdeckung? Nicht der Code-Pfadabdeckung, sondern der Feature-Abdeckung. Ist in dem Akzeptanztest alles Interessante getestet? Nein. Es fehlt die Veränderung einer Sitzung, wie in **Bild 5** als Möglichkeit gezeigt. Das wäre einen zweiten Szenariotest wert:

- Neue Sitzung publizieren wie oben.
- Sitzung individuell ordnen wie oben.
- Existierende Sitzung verändern.
 - Starten der gerade publizierten Sitzung.
 - Publizieren mit den Einträgen „X“, „Y“, „Z“, „AA“; es wird eine neue Sitzungsnummer zurückgeliefert.
- Sitzung individuell ordnen.
 - Starten mit der neuen Sitzungsnummer; es sollten die Beschreibung „B“ und die Einträge [{2, „Z“}, {1, „Y“}, {0, „X“}, {3, „AA“}] zurückgeliefert werden.
 - Einreichen der individuellen Ordnung [2, 3, 0, 1].
- Starten der neuen Sitzung und Aktualisieren.
 - Starten mit neuer SessionID; es sollte „B“ und [„X“, „Y“, „Z“, „AA“] geliefert werden.
 - Aktualisieren; das Ergebnis sollte [„Z“, „AA“, „X“, „Y“] sein, weil nur ein Teammitglied eine Ordnung eingereicht hat.

Das sind für einen PO greifbare Formulierungen, würde ich sagen. Sie sind frei von technischen Feinheiten, und trotzdem sind sie präzise für mich als Entwickler. Ich weiß genau, welche Funktion ich wie aufzurufen habe – während der PO nur in Interaktionen denkt. So begegnen wir uns auf Augenhöhe:

- Der PO bekommt Klarheit darüber, was er will, was seine Erwartungen sind. Es ist zu wenig, wenn er nur mit den Händen wedelt und sagt: „Na ja, du weißt schon ...“ oder „Du bist doch der Experte“. Klar, ich bin der Experte, Sie sind der Experte. Für Codierung von Transformationen, die

in Anforderungen versteckt sind. Wir sind keine Experten für Anforderungen. Die muss der PO schon selbst definieren. Präzise!

- Ich bekomme Klarheit darüber, wie meine grobe Architektur aussieht und wo die Startpunkte liegen, mit denen ich in Entwurf und Codierung einsteige.

Zwischenstand

Ausgehend von einer Anforderungsskizze habe ich mit Slicing eine Liste von Funktionen bestimmt, mit denen sich die Anforderungen verhaltensmäßig abdecken lassen. Für zwei Verhaltensszenarien habe ich konkrete Vorstellungen entwickelt, die ich als Akzeptanztests gegen die Funktionen laufen lassen kann. Damit habe ich eine Ziellinie gezogen. Die ist erreicht, sobald diese Tests grün sind.

Das Sleepy-Hollow-Architekturmuster hat mir ein Terrain geliefert, in dem ich genau verortet bin. Ich weiß, wo ich stehe, was ich habe – und ich weiß, was noch fehlt, was nun näher zu entwerfen und dann zu codieren ist.

Jetzt habe ich alles beieinander, um mit möglichst wenig Kommunikation mit einem PO meinen eigentlichen Job zu machen. Ich habe Full Kitting so weit wie möglich betrieben. Sicher gibt es noch eine Grauzone, nicht alles lässt sich vor der Codierung haarklein bestimmen. Ich will ja auch nicht in einen Wasserfall zurückfallen. Doch ich empfinde, dass ich mich zumindest gut präpariert habe. Ohne Vorbereitung in das Abenteuer Codierung einzusteigen ist zu wenig.

Jetzt ist mein Rucksack gepackt, und ich habe eine Karte in der Hand. Damit geht es nächstes Mal los in die Implementation. ■

- [1] Ralf Westphal, *Strukturiert zerlegen*, dotnetpro 4/2024, Seite 42 ff., www.dotnetpro.de/A2404Slicing
- [2] Ralf Westphal, *Inkrementale innerhalb von Interaktionen finden*, dotnetpro 9/2024, Seite 32 ff., www.dotnetpro.de/A2409Slicing
- [3] Ralf Westphal, *Den Kopf vom Körper trennen*, dotnetpro 11/2024, Seite 46 ff., www.dotnetpro.de/A2411SleepyHollow
- [4] Ralf Westphal, *TotalOrder: Konsequent vergleichend schätzen*, www.dotnetpro.de/SL2412SleepyHollow1
- [5] Goldratt UK, *A Simple but Highly Effective Way to Increase Output: Full-Kitting*, www.dotnetpro.de/SL2412SleepyHollow2
- [6] Ralf Westphal, *IODA Architecture*, www.dotnetpro.de/SL2412SleepyHollow3



Ralf Westphal

ist Trainer, Berater und Mitgründer der Clean Code Developer Initiative (<https://clean-code-developer.de>). Seine Schwerpunkte sind dauerhaft hohe Produktivität für die Softwareentwicklung und zukunftsfähige Teamorganisation.

<https://ralfw.de>

dnpCode

A2412SleepyHollow

DESIGNSYSTEME UND .NET MAUI

Gestaltung mit System

Designsysteme sind hilfreich, wenn es gilt, die Qualität der Benutzerschnittstelle zu steigern und Richtlinien für die Entwicklung von digitalen Produkten bereitzustellen.

Unter einem Designsystem versteht man eine strukturierte Sammlung von Gestaltungsrichtlinien, Komponenten und Ressourcen, die dazu dienen, konsistente und effiziente Designs in einem Unternehmen oder Projekt zu fördern. Diese Systeme spielen eine entscheidende Rolle bei der Sicherstellung von Einheitlichkeit, Wiedererkennbarkeit und Benutzerfreundlichkeit in verschiedenen digitalen Produkten und Plattformen. Designsysteme sind eng mit dem Konzept des menschenzentrierten Designs (siehe Kasten **Human-Centered Design**) verbunden und können dazu beitragen, die Prinzipien und Ziele dieses Ansatzes zu unterstützen.

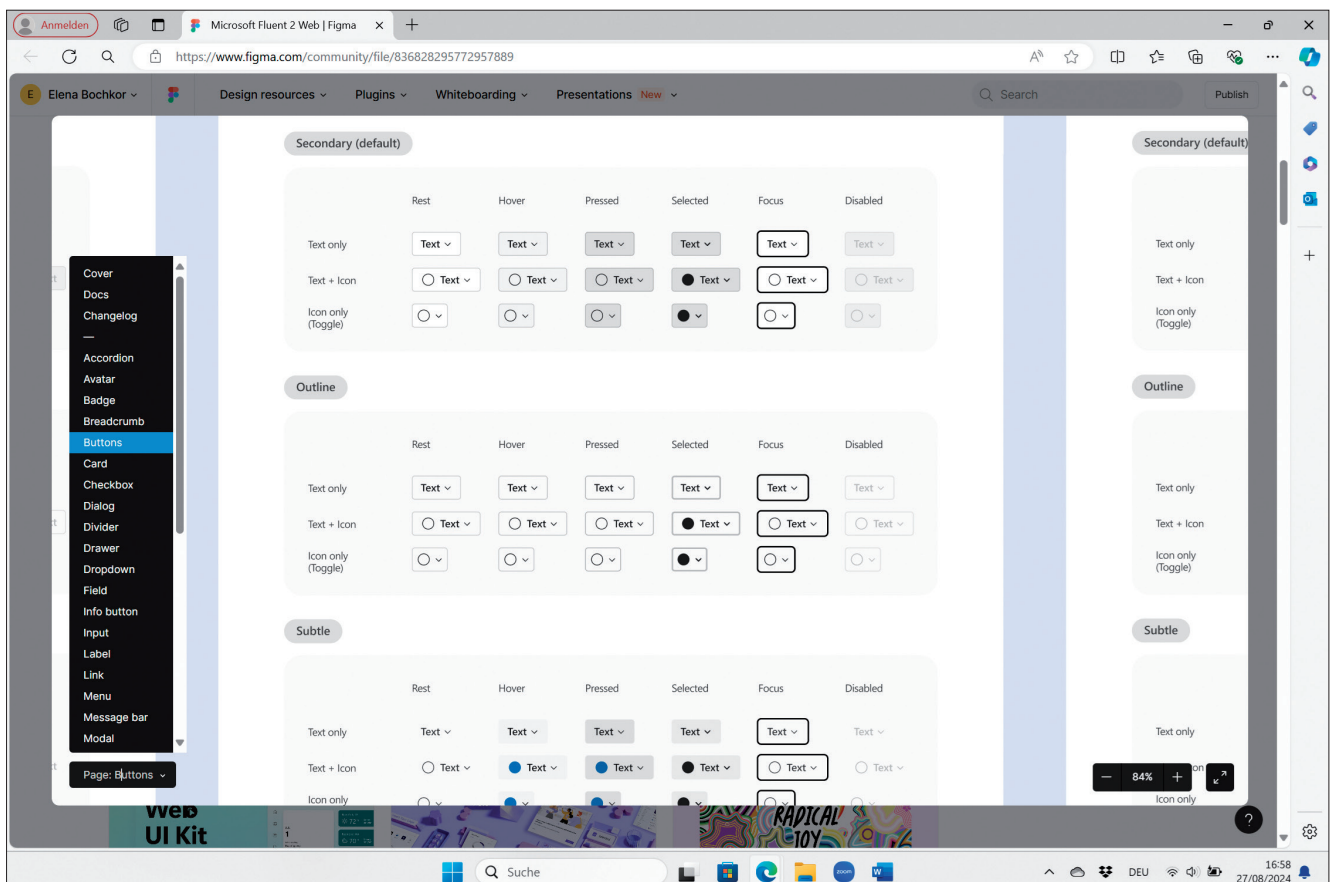
Nach einem Überblick über das Thema will dieser Artikel anhand eines konkreten Beispiels zeigen, wie sich unterschiedliche Ansätze im Design bei der Entwicklung von Apps mit .NET MAUI verwenden lassen. Beispielsweise kann man die Designvorgaben von Microsoft anwenden oder auf den

Ansatz des Material Designs von Google setzen. Entwickler und Designer können damit erkennen, wie die theoretischen Ansätze in der Praxis angewendet werden, und auf dieser Grundlage Apps mit einem konsistenten und effizienten Design gestalten.

Überblick über Designsysteme

Es gibt unterschiedliche Designsysteme, die von verschiedenen Unternehmen, Organisationen und Communities entwickelt wurden. Verschaffen wir uns in einem ersten Schritt einen Überblick über die bekanntesten Designsysteme, die in der Praxis eingesetzt werden.

- **Material Design** wurde von Google entwickelt. Es ist eines der bekanntesten und umfassendsten Designsysteme. Es bietet eine konsistente Sprache für Design und UI-Komponenten, die auf natürlichen Materialien und Bewegung basieren.



Das Fluent-Design-System von Microsoft (Bild 1)

- Die **Human Interface Guidelines** von Apple sind eine Sammlung von Designprinzipien, Richtlinien und Best Practices, die Entwicklern und Designern helfen sollen, konsistente und benutzerfreundliche Apps für Apple-Plattformen wie iOS, iPadOS, macOS, watchOS und tvOS zu erstellen.
- **Fluent** ist das Designsystem von Microsoft, das in Windows, Office und anderen Microsoft-Produkten zum Einsatz kommt. Es zielt darauf ab, eine einheitliche und ansprechende Benutzeroberfläche über alle Geräte hinweg zu bieten (Bild 1).
- **Polaris** ist das Designsystem von Shopify, das speziell für die Erstellung von E-Commerce-Erlebnissen entwickelt wurde. Es enthält Richtlinien und UI-Komponenten, die auf die Bedürfnisse von Online-Shops zugeschnitten sind.
- **Ant Design** wurde von Alibaba entwickelt und ist besonders in Asien sehr beliebt. Es bietet eine Vielzahl von UI-Komponenten und Tools zur Unterstützung der Webentwicklung.
- Das **Atlassian Design System** ist ein sehr leistungsfähiges Werkzeug, das speziell für die Entwicklung konsistenter, effizienter und benutzerfreundlicher Anwendungen innerhalb des Atlassian-Ökosystems wie zum Beispiel Jira oder Trello entwickelt wurde. Es unterstützt Entwickler und Designer dabei, die Kernprinzipien von Atlassian – wie Agilität, Flexibilität und Effizienz – nahtlos in ihre Produkte zu integrieren.
- Mit **Lightning** hat Salesforce ein Designsystem entwickelt, das besonders hilfreich für die Erstellung von Business-Apps ist. Salesforce bietet Cloud-Lösungen in den Bereichen CRM, Kundenservice, Automatisierung, Analyse und Anwendungsentwicklung an. Das Designsystem umfasst eine Vielzahl von Elementen, darunter detaillierte Richtlinien, Design Patterns und Design Tokens, die die Entwicklung erheblich erleichtern.

Die Bestandteile eines Designsystems im Überblick

Ein Designsystem besteht aus wiederverwendbaren Komponenten, Designprinzipien und Richtlinien und dient dazu, Konsistenz und Effizienz bei den digitalen Produkten eines Unternehmens zu gewährleisten. Ein typisches Designsystem enthält die folgenden Bestandteile:

- Vordefinierte Elemente für Benutzeroberflächen, darunter Textfelder, Drop-down-Menüs und Navigationsstrukturen.
- Eine Palette genehmigter Markenfarben und -schriften sowie Richtlinien für deren Anwendung.
- Einheitliche Schaltflächen und andere interaktive Bedienelemente.
- Visuelle Anweisungen für die Platzierung bestimmter Handlungsaufforderungs-Schaltflächen (Call to Action).
- Ein anpassbares Rastersystem für eine einheitliche Anordnung der Bildschirm Inhalte.
- Richtlinien zur Positionierung bestimmter Elemente, wie zum Beispiel ein stets oben rechts zu platzierender Anmelde-/Abmelde-Button oder das Profil-Symbol auf dem Desktop.
- Vorgaben für Interaktionen, das heißt beispielsweise, dass das System bestimmt, ob ein seitlich einblendbares Panel

Human-Centered Design

Als Human-Centered Design (HCD) [11] oder als menschen-orientierte Gestaltung bezeichnet man nach ISO 9241-210:2019 einen Ansatz für die Entwicklung interaktiver Systeme, die darauf abzielen, Produkte, Systeme oder Dienstleistungen benutzbar und nützlich zu machen, indem man sich auf die Nutzer, ihre Bedürfnisse und Anforderungen konzentriert und Kenntnisse und Techniken aus den Bereichen Human Factors/Ergonomie und Usability anwendet. Die vier wichtigsten Prinzipien, auf denen HCD basiert, sind:

- Die Entwickler sollen nicht nur die Nutzer, sondern auch den Verwendungszweck und die Umgebung, in der das Produkt zum Einsatz kommen wird, verstehen.
- Die Anwender sind Teil des gesamten Entwicklungs- und Designprozesses.
- Multidisziplinäre Fähigkeiten und Perspektiven sind feste Bestandteile des HCD-Prozesses.
- Das Design berücksichtigt die gesamte User-Experience.

Grundsätzlich bringt Human-Centered Design Produktivitätssteigerung und Wettbewerbsvorteile mit sich. Jedoch sind einige Entwickler kritisch und bemängeln, dass eine Fokussierung auf die Nutzerbedürfnisse die Perspektive echter technologischer Innovation behindert.

verwendet wird und ob Bilder durch Zoomen oder Pinch-Gesten vergrößert werden können.

- Eine Definition von Symbolen und deren Bedeutungen.
- Richtlinien für den Einsatz und die Formulierung von Inhalten, zum Beispiel, ob Menüeinträge als Verben oder Substantive gestaltet sein sollen und ob die Ansprache des Nutzers in der ersten oder zweiten Person erfolgen soll.
- Vorgaben dafür, wann grafische Symbole und wann Text als Auszeichnungen für Interaktionsflächen genutzt werden sollen.

Designsysteme können für Teams jeder Größe und mit unterschiedlichen Erfahrungsniveaus nützlich sein. Wichtig ist: Nicht jeder benötigt ein komplexes, unternehmensweites Designsystem. Ein einfaches System mit einigen wiederverwendbaren Vorlagen und Komponenten reicht in vielen Fällen schon aus, um den Arbeitsablauf zu optimieren und Einheitlichkeit zu gewährleisten. Das erstellte System kann sich mit der Zeit weiterentwickeln, um mit den wachsenden Anforderungen des Teams und der Projekte Schritt zu halten. Wichtig ist es, ein System zu entwickeln, das auf die spezifischen Bedürfnisse zugeschnitten ist.

Da der Aufbau eines Designsystems eine gemeinsame Aufgabe ist, sollte man frühzeitig Unterstützer ins Boot holen und nach Personen suchen, die sich für konsistentes Design begeistern oder bereits Erfahrung mit Designsystemen haben. Dabei geht es nicht nur um Designer, sondern auch um Entwickler, Produktmanager, eventuell Mitarbeitende des ►

Kundenservice und anderer Bereiche des Unternehmens. Unterschiedliche Perspektiven tragen dazu bei, dass ein Designsystem die Bedürfnisse des gesamten Produkts und der Organisation berücksichtigt, anstatt sich nur auf eine bestimmte Gruppe zu konzentrieren.

Es gibt zwei Hauptansätze, um ein Designsystem aufzubauen: Entweder man entwickelt es komplett neu und passt es exakt an die Anforderungen an, oder man übernimmt ein bestehendes Framework und adaptiert es an die Bedürfnisse der Zielgruppe. Beide Ansätze haben ihre Vor- und Nachteile. Ein eigenes System bietet den Vorteil, dass es speziell auf die eigenen Bedürfnisse zugeschnitten ist, erfordert aber anfänglich mehr Zeit und Ressourcen. Die Nutzung eines vorhandenen Frameworks oder eines Open-Source-Systems ermöglicht einen schnelleren Start, kann jedoch mehr Anpassungen erfordern, um genau den vorgegebenen Anforderungen gerecht zu werden. Welche Methode die richtige ist, hängt von den Fähigkeiten des Teams, den verfügbaren Ressourcen und den langfristigen Zielen ab. Es kann auch sinnvoll sein, „klein“ anzufangen und das System im Lauf der Zeit weiterzuentwickeln. In dieser frühen Phase ist es entscheidend, dass alle im Team ein gemeinsames Verständnis darüber haben, was ein Designsystem bedeutet.

In der Regel beginnt die Erstellung eines Designsystems mit einer Bestandsaufnahme: Alle Elemente, die man verwendet, werden gesammelt und kategorisiert: Farben, Typografie, Icons, Komponenten, Muster – alles zählt. Dabei sollte man Beispiele des Produkts auf verschiedenen Plattformen und Geräten berücksichtigen. Eventuell erkennt man schon die ersten Muster.

Ein Designsystem ist mehr als nur eine Sammlung im Sinne eines UI-Kits. Es umfasst auch Richtlinien, Prinzipien und wiederverwendbare Komponenten, die zusammen eine konsistente und effiziente Gestaltung digitaler Produkte ermöglichen. Es dient als umfassender Leitfaden, der nicht nur das visuelle Erscheinungsbild, sondern auch das Nutzererlebnis, die Interaktion und die technische Umsetzung prägt. Die Musterbibliothek ist ein Teil innerhalb des Designsystems; sie wird plattformübergreifend eingesetzt, um eine konsistente Markenbotschaft und einen einheitlichen Stil sicherzustellen. Der Style Guide, ein weiterer Bestandteil des Designsystems, stellt die Dokumentation und Beschreibung des Systems selbst dar. Er definiert, wie Produkte aussehen und sich anfühlen sollen, legt Anwendungsfälle für UI-Muster fest und gibt die korrekten typografischen Skalen vor. Das UI-Kit hingegen ist eine Sammlung von Grafikdateien und Ressourcen, die Designer bei der Erstellung von Benutzeroberflächen für verschiedene Anwendungen unterstützen. Es umfasst in der Regel UI-Komponenten wie Schaltflächen, Kontrollkästchen, Widgets und Fortschrittsbalken, die sowohl Bedeutung vermitteln als auch den Benutzern Funktionalität bieten.

Vorteile von Designsystemen

Ein präzise zugeschnittenes und angepasstes Designsystem bringt mehrere Vorteile für den Design- und Entwicklungsprozess mit sich. An der ersten Stelle geht es um eine verbesserte Konsistenz. Der Einsatz von Designsystemen gewähr-

leistet Einheitlichkeit der UI-Elemente, Typografie, Abständen et cetera. Diese Konsistenz erstreckt sich über alle Plattformen und Geräte hinweg. Die einheitliche Verwendung von Designelementen trägt zur Stärkung der Markenwiedererkennung bei. Das führt zu einem vereinfachten Designprozess und erhöht die Effizienz.

Ein weiterer Vorteil ist die verbesserte Kommunikation zwischen Designer und Entwickler. Eine gemeinsame „Sprache“ erleichtert den Design- ebenso wie den Entwicklungsprozess. Das Designsystem dient außerdem als Orientierung und hilft, eine gleichbleibend hohe Designqualität sicherzustellen. Wichtig sind jedoch regelmäßige Aktualisierungen und Wartungsarbeiten, um das System auf dem neuesten Stand zu halten. Bei zunehmendem Projektumfang hilft das Designsystem, die gestalterische Konsistenz zu bewahren. Ein Designsystem mit Richtlinien für Barrierefreiheit und inklusives Design gewährleistet, dass diese Aspekte im Endprodukt berücksichtigt werden und eine breite Nutzbarkeit für möglichst viele Personengruppen ermöglichen.

Die Bedeutung eines Designsystems wächst mit der zunehmenden Größe eines Unternehmens. In seiner grundlegendsten Form bietet ein richtig implementiertes Designsystem eine Sammlung bewährter Designpraktiken, Prozesse, Muster und Komponenten, die vom Management und ihren Teams genutzt werden, um Konsistenz über alle Produkte und Plattformen hinweg zu gewährleisten. Während Produkte durch verschiedene Phasen im Unternehmen gehen, können wesentliche Designkomponenten bei der Umsetzung verloren gehen. Ein solides Designsystem ist die beste Möglichkeit, diese Konsistenz zu sichern. Ein gut gestaltetes Designsystem spart den Teams unzählige Stunden an Arbeit, da es nicht notwendig ist, detaillierte Designspezifikationen jedes Mal neu zu erstellen – einschließlich aller variablen Maße, die Entwickler benötigen, um neue Bildschirme, Seiten oder vollständige UX-Flows zu gestalten. Wenn ein Unternehmen mehrere Produkte entwickelt, ist ein Designsystem eine der effizientesten Methoden, um die Produktivität des Teams zu skalieren, da es Entwicklern und Designern ermöglicht, schneller, effektiver und konsistenter Prototypen zu erstellen, während gleichzeitig Designwissen unternehmensweit geteilt wird.

Barrierefreiheit als wichtiger Aspekt

Ein wichtiger Aspekt im Rahmen von UI/UX-Design – unter anderem auch bei der Erstellung eines Designsystems – ist die Barrierefreiheit. Der Begriff der Barrierefreiheit bezieht sich in diesem Fall auf die Gestaltung von digitalen Produkten, die für alle Benutzer zugänglich und nutzbar sind, unabhängig von ihren individuellen Fähigkeiten oder Einschränkungen. Ziel ist es, ein Produkt inklusive einer positiven Benutzererfahrung zu erstellen, das alle Personen, einschließlich solcher mit permanenten oder situativen Einschränkungen, problemlos verwenden können. Dabei geht es um folgende Aspekte:

- **Textgestaltung:** Schriftgrößen sollten vom Benutzer anpassbar sein, und es sollten gut lesbare Schriftarten verwendet werden. Der Text selbst sollte verständlich und in einer kla-

ren Sprache formuliert werden. Bei der Textgestaltung für ein Designsystem beginnt man mit dem Festlegen von Schriftfamilien und -gewichten sowie Fallbacks. Anschließend legt man fest, wie eine Hierarchie durch Größe, Farbe und zusätzliche Details wie Zeilenhöhe und Zeilendurchschuss sowie Reaktionsfähigkeit aufgebaut werden kann. Die erstellten Modelle werden dann auf die Komponente aus der Bibliothek des Systems angewendet.

- **Kontrast:** Farben sollten so gewählt werden, dass sie einen hohen Kontrast aufweisen, um die Lesbarkeit zu verbessern, insbesondere für Menschen mit Sehschwächen oder Farbblindheit. Farbe spielt ebenfalls eine entscheidende Rolle in der typografischen Hierarchie einer Benutzeroberfläche, häufig durch festgelegte Typen. So verwendet man die Primärfarbe für den Großteil des Interface-Textes, sei es Fließtext oder Überschrift. Die Sekundärfarbe verwendet man für ergänzende Informationen mit geringerem Kontrast (oft als „grauer Text“). Interaktive Farben nimmt man nicht nur für Links, sondern auch für flache Schaltflächen und Tab-Beschriftungen. Eine deaktivierte Farbe ist eine Farbe mit besonders niedrigem Kontrast. Die Fehlerfarbe, in der Regel Rot, sollte den höchsten Kontrast zu ihrer Umgebung aufweisen.
- **Alternative Texte für Bilder:** Bilder, Grafiken und andere visuelle Inhalte sollten mit alternativen Texten (Alt-Tags) versehen werden, damit Screenreader die Inhalte für blinde oder sehbehinderte Nutzer beschreiben können.
- **Struktur und Navigation:** Diese sollen intuitiv und leicht handhabbar sein. Alle interaktiven Elemente müssen auch ohne Mausbedienung zugänglich sein, zum Beispiel durch Tastatur- oder Sprachnavigation. Überschriften sind für die Hierarchie einer Anwendung entscheidend. Die meisten Designsysteme bieten mindestens vier Ebenen von Überschriften an, andere stellen sogar noch weitere Ebenen bereit. In der Regel entspricht der Seitentitel der obersten Ebene. Die anderen Ebenen finden sich dann in verschiedenen Komponenten wieder, wie zum Beispiel einem Titel in einer Karte, einer Überschrift in einer Warnmeldung oder einem zweistufigen Titel in einem modalen Dialogfeld.

Komplexe Interaktionen wie Drag-and-drop oder animierte Effekte sollten vermieden oder alternative Optionen bereitgestellt werden, damit Menschen mit motorischen Einschränkungen oder kognitiven Beeinträchtigungen das Produkt ebenso problemlos nutzen können. Die gesamte Benutzeroberfläche sollte so gestaltet sein, dass sie leicht verständlich ist. Komplexe Informationen sind einfach aufzuschlüsseln und zu präsentieren.

Designsysteme und .NET MAUI

In .NET MAUI (Multiplatform App UI) können wir ebenfalls vom Einsatz eines Designsystems profitieren, um konsistente, skalierbare und wiederverwendbare UI-Komponenten zu erstellen. Zunächst ist jedoch festzustellen, dass es von Haus aus kein vorgefertigtes Designsystem für .NET MAUI gibt. Das wäre bei einem Werkzeug für die plattformübergreifende Entwicklung auch kaum möglich. Mit .NET MAUI können

wir Apps für die Systeme iOS, Android, Windows und macOS und, über den Support von Samsung, auch für Tizen (Smart-TV-Geräte) erstellen. Jedes dieser Zielsysteme hat seine eigenen Designrichtlinien:

- **iOS (Human Interface Guidelines):** Die Human Interface Guidelines von Apple geben Entwicklern und Designern Empfehlungen, wie Apps für iOS-Geräte gestaltet werden sollten. Diese Richtlinien umfassen Themen wie Navigation, Layout, Typografie und Touch-Interaktionen, um konsistente und intuitive Benutzeroberflächen zu erstellen [1].
- **Android (Material Design):** Material Design ist die offizielle Designrichtlinie von Google für Android und andere Plattformen. Es basiert auf einer „physischen“ Metapher und legt großen Wert auf klare Hierarchien, Übergänge und die konsistente Verwendung von Farben, Typografie und Formen. Material Design wird regelmäßig aktualisiert, um moderne Designtrends zu berücksichtigen [2].
- **Windows (Fluent Design System):** Das Fluent Design System von Microsoft [3] ist die Designrichtlinie für Windows-Anwendungen. Es betont Einfachheit, Licht, Tiefe, Bewegung und Materialinteraktionen, um ein modernes und visuell ansprechendes Erlebnis zu schaffen. Fluent Design wird für alle Windows-Anwendungen empfohlen, um eine kohärente Benutzererfahrung über verschiedene Geräte hinweg zu gewährleisten [4].
- **macOS (Human Interface Guidelines):** Ähnlich wie für iOS bietet Apple auch für macOS detaillierte Human Interface Guidelines. Diese Richtlinien zielen darauf ab, die Mac-Erfahrung zu vereinheitlichen, indem sie Entwickler und Designer bei der Gestaltung von Oberflächen unterstützen, die die Plattformstandards für Fenster, Menüs, Interaktionen und visuelle Ästhetik erfüllen [5].
- **Design Guidelines für Tizen TV:** Diese sind speziell darauf ausgelegt, die Benutzererfahrung auf Samsung-Fernsehgeräten zu optimieren, da diese sich von mobilen oder Desktop-Umgebungen stark unterscheiden [6].

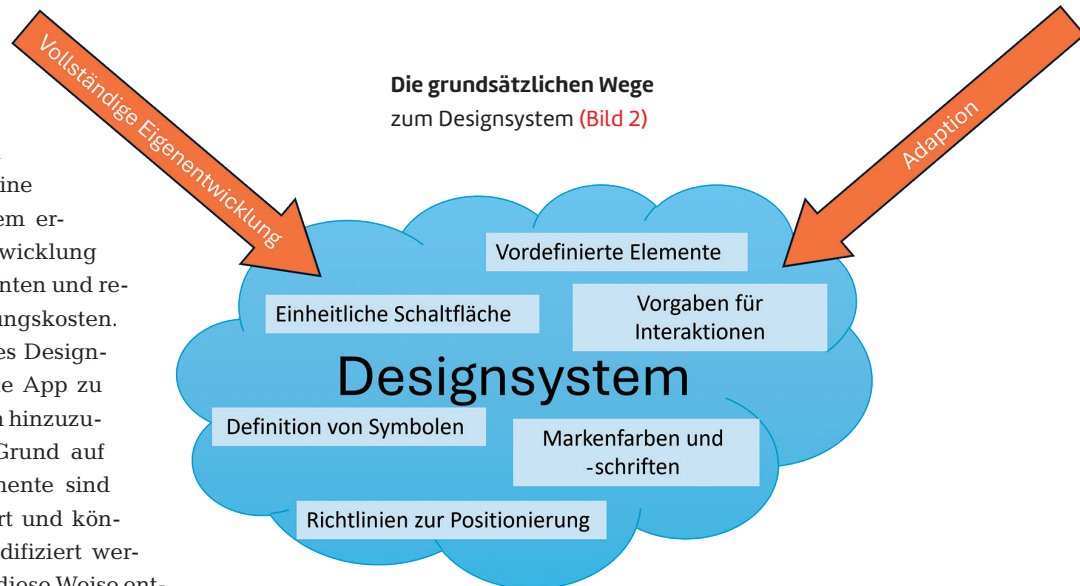
Fassen wir das Zwischenergebnis zusammen und leiten daraus die folgende Schlussfolgerung ab: Jede Plattform hat ihre eigenen Designrichtlinien, die spezifische Prinzipien und Designmuster für die jeweilige Benutzererfahrung festlegen. Grundsätzlich ist es wichtig, die jeweiligen Designstandards zu befolgen, um ein konsistentes und intuitives Erlebnis zu schaffen, das den Erwartungen der Nutzer der Plattform entspricht. Dennoch kann es sinnvoll sein, ein Designsystem für .NET-MAUI-Apps zu erstellen. Die Gründe dafür sind beispielsweise:

- **Plattformübergreifende Konsistenz:** Ein eigenes Designsystem stellt sicher, dass zentrale Designprinzipien und UI-Komponenten über alle Plattformen hinweg konsistent bleiben, was die Benutzererfahrung verbessert, insbesondere für Apps, die auf mehreren Plattformen genutzt werden. Der Vorteil ist Einheitlichkeit in Erscheinungsbild und Benutzerführung (Wiedererkennung).
- **Effizienz und Wiederverwendbarkeit:** Ein Designsystem ermöglicht die Wiederverwendung von UI-Komponenten, Layouts, Stilen und Interaktionen über mehrere Projekte ►

hinweg. Dies reduziert den Aufwand, jedes Mal plattformspezifische Anpassungen vorzunehmen, und sorgt für eine effizientere Entwicklung. Zudem ermöglicht es eine schnellere Entwicklung durch standardisierte Komponenten und reduzierte Design- und Entwicklungskosten.

- **Skalierbarkeit:** Ein gut definiertes Designsystem macht es einfacher, eine App zu erweitern oder neue Funktionen hinzuzufügen, ohne das Design von Grund auf neu zu erstellen. Alle UI-Elemente sind bereits im System standardisiert und können einfach erweitert oder modifiziert werden, wenn die App wächst. Auf diese Weise entsteht ein flexibles und skalierbares System, das leicht aktualisiert werden kann.
- **Kohärente Markenidentität:** Ein eigenes Designsystem ermöglicht es, plattformspezifische Richtlinien mit den Branding-Richtlinien eines Unternehmens zu kombinieren. Das hilft dabei, die visuelle Identität der Marke auf jeder Plattform zu wahren, ohne die Nutzererwartungen auf iOS, Android, Windows oder macOS zu verletzen. Die Marke bleibt über alle Plattformen hinweg erkennbar und konsistent, ohne an Benutzerfreundlichkeit einzubüßen.
- **Anpassung an spezifische Anwendungsfälle:** Plattformrichtlinien bieten allgemeine Designprinzipien, berücksichtigen aber nicht immer spezielle Anforderungen oder Besonderheiten der jeweiligen App. Ein eigenes Designsystem erlaubt es, Designs und Interaktionen an spezifische Bedürfnisse und Anwendungsfälle der App anzupassen. Wir erreichen dadurch eine hohe Flexibilität, um auf spezielle App-Anforderungen und komplexe Benutzerinteraktionen einzugehen.
- **Optimierung für plattformübergreifend arbeitende Teams:** In einem plattformübergreifenden Entwicklerteam kann ein Designsystem als gemeinsame Sprache zwischen Designer und Entwickler fungieren. Dies hilft, Missverständnisse zu vermeiden und sicherzustellen, dass alle Teammitglieder dieselben Designrichtlinien befolgen. Das führt zu einer

Die grundsätzlichen Wege
zum Designsystem (Bild 2)



verbesserten Zusammenarbeit und Kommunikation im Team sowie einer nahtlosen Übergabe von Designs in den Entwicklungsprozess.

Sehen wir uns jetzt Ansatzpunkte in .NET MAUI an, um ein Designsystem aufzubauen.

Designsystem aufbauen

In .NET MAUI (Multiplatform App UI) können wir Designsysteme anwenden, um konsistente, skalierbare und wiederverwendbare UI-Komponenten zu erstellen. Während es kein vorgefertigtes Designsystem für .NET MAUI gibt, können wir ein eigenes Designsystem umsetzen oder bestehende Richtlinien aus anderen Systemen adaptieren (Bild 2). Technische Ansatzpunkte sind beispielsweise:

- **Erstellung eines Stylesheets (XAML Styles):** In .NET MAUI kann man XAML verwenden, um Styles für die UI-Komponenten zu definieren. Das ermöglicht es, wiederverwendbare Designelemente wie Farben, Schriftarten, Abstände und Layouts in einer zentralen Ressourcendatei zu speichern. Das kann beispielsweise wie in Listing 1 dargestellt aussehen. Solche zentralen Ressourcen stellen sicher, dass die gleichen Farben und Stile über die gesamte App hinweg verwendet werden, was die Konsistenz fördert.

Listing 1: Definition von Styles als Ressourcen für die zentrale Nutzung in der App

```
<ResourceDictionary>
  <!-- Farben -->
  <Color x:Key="PrimaryColor">#3498db</Color>
  <Color x:Key="SecondaryColor">#2ecc71</Color>

  <!-- Schriften -->
  <Style TargetType="Label">
    <Setter Property="FontSize" Value="18" />
    <Setter Property="TextColor"
      Value="{StaticResource PrimaryColor}" />
  </Style>

  <!-- Buttons -->
  <Style TargetType="Button">
    <Setter Property="BackgroundColor"
      Value="{StaticResource PrimaryColor}" />
    <Setter Property="TextColor" Value="White" />
    <Setter Property="CornerRadius" Value="5" />
  </Style>
</ResourceDictionary>
```

● Listing 2: Themes in der .NET MAUI-App

```

<!-- LightTheme.xaml -->
<?xml version="1.0" encoding="UTF-8" ?>
<ResourceDictionary xmlns=
    "http://schemas.microsoft.com/dotnet/2021/maui">
    <Color x:Key="PrimaryColor">#3498db</Color>
</ResourceDictionary>

<!-- DarkTheme.xaml -->
<?xml version="1.0" encoding="UTF-8" ?>
<ResourceDictionary xmlns=
    "http://schemas.microsoft.com/dotnet/2021/maui">
    <Color x:Key="PrimaryColor">#2980b9</Color>
</ResourceDictionary>

<!-- Laden der Ressourcen in der `App.xaml`-->
<Application xmlns=
    "http://schemas.microsoft.com/dotnet/2021/maui"
    x:Class="YourNamespace.App">
    <Application.Resources>
        <!-- Hier wird das Light-Theme als Standard
            gesetzt -->
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source=
                    "LightTheme.xaml" />
            </ResourceDictionary.MergedDictionaries>
        </ResourceDictionary>
    </Application.Resources>
</Application>

// Umschalten des Themas zur Laufzeit (C#-Code)
public partial class App : Application
{
    public App()
    {
        InitializeComponent();

        // Standardmäßig wird das Light-Theme gesetzt
        SetTheme("Light");
    }

    // Methode zum Umschalten des Themas
    public void SetTheme(string theme)
    {
        ResourceDictionary newTheme;

        if (theme == "Dark")
        {
            newTheme = new ResourceDictionary {
                Source = new Uri("DarkTheme.xaml",
                    UriKind.RelativeOrAbsolute) };
        }
        else
        {
            newTheme = new ResourceDictionary {
                Source = new Uri("LightTheme.xaml",
                    UriKind.RelativeOrAbsolute) };
        }

        // Entferne vorhandene MergedDictionaries und
        // füge das neue Thema hinzu
        Application.Current.Resources.MergedDictionaries.
            Clear();
        Application.Current.Resources.MergedDictionaries.
            Add(newTheme);
    }

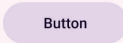
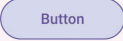
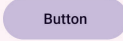
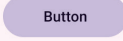
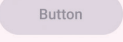
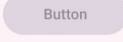
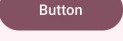
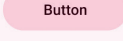
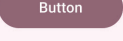
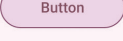
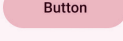
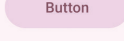
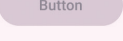
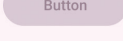
    // Wechsel des Themas aus dem UI für einen Button
    private void OnThemeToggleButtonClicked(
        object sender, EventArgs e)
    {
        if (App.Current.Resources.MergedDictionaries[0].
            Source.ToString().Contains("LightTheme"))
        {
            ((App)Application.Current).SetTheme("Dark");
        }
        else
        {
            ((App)Application.Current).SetTheme("Light");
        }
    }
}

```

- **Theme-Unterstützung:** Wir können in der App sowohl ein helles als auch ein dunkles Design unterstützen. Man kann separate Ressourcen für verschiedene Themes definieren und diese zur Laufzeit ändern. Die App kann dann dynamisch zwischen den Themes wechseln, abhängig von den Benutzereinstellungen (Listing 2).
- **Komponentenkatalog:** Ein wichtiger Teil eines Designsystems ist ein Komponentenkatalog, der standardisierte UI-Komponenten wie Buttons, Textfelder und Navigationselemente umfasst. In .NET MAUI kann man benutzerdefinierte

Controls erstellen und diese als wiederverwendbare Komponenten in verschiedenen Bereichen der App einsetzen.

- **Responsive Layouts:** Da .NET MAUI plattformübergreifend funktioniert, sollte das Designsystem responsive Layouts beinhalten. Dies bedeutet, dass UI-Komponenten auf verschiedenen Bildschirmgrößen und Plattformen (iOS, Android, Windows, macOS) gut aussehen sollten. So lassen sich hier etwa *Grid* oder *StackLayout* in Kombination mit adaptiven *Margins* und *Paddings* nutzen, um das Layout flexibel zu gestalten. ▶

	ACCENT	OUTLINED	FILLED	TEXT BUTTON
DEFAULT				
FOCUSED				
PRESSED				
DISABLED				
SELECTED				
SELECTED PRESSED				
SELECTED DISABLED				

Material Design mit der .NET-MAUI-Suite von DevExpress (Bild 3)

Hinweise zum Erstellen von Designs in .NET MAUI-Apps findet man ebenfalls in der offiziellen Dokumentation des Herstellers [7]. Mithilfe von XAML können wir schrittweise individuelle (in Bezug auf das Projekt oder das Unternehmen) Designressourcen definieren und einsetzen.

Designsysteme von Drittanbietern

Es gibt einige Drittanbieter, die UI-Komponentenbibliotheken oder Designsysteme für .NET MAUI bereitstellen. Beispielsweise bieten Syncfusion, Telerik oder DevExpress UI-Komponenten an, die man in die eigenen MAUI-Projekte integrieren kann. Dazu die folgenden Beispiele:

- **Material Design in .NET-MAUI-Apps mit DevExpress [8]:** Die DevExpress .NET MAUI Suite umfasst zehn integrierte Material-Design-3-Farbthemen – verfügbar in dunklen und hellen Varianten. Mithilfe der ThemeColor-Markup-Erweiterung und Projektvorlagen können Sie Material-Design-3-Designs problemlos in Anwendungen unterstützen, die DevExpress oder standardmäßige .NET-MAUI-Steuerelemente verwenden (Bild 3).
- **Material Design mit MDC-MAUI [9]:** Bereit steht eine umfangreiche Palette an Komponenten, die das Material Design in .NET-MAUI-Apps umsetzen.
- **Icons für .NET MAUI [10]:** Interessante Bild- und Icon-Ressourcen im Fluent-Design-Style findet man beispielsweise von Grial.

Diese externen Bibliotheken können ein Ausgangspunkt sein, um ein eigenes Designsystem aufzubauen. In kleineren Projekten kann es auch bereits genügen, bestehende Design-Ressourcen, beispielsweise von einem Drittanbieter, konsequent einzusetzen.

Fazit

Es ergibt Sinn, Designsysteme einzusetzen, weil sie Konsistenz, Effizienz und Skalierbarkeit in der Entwicklung von digitalen Produkten fördern. Sie sorgen dafür, dass alle Mitglieder des Teams einheitliche Standards und wiederverwendbare Komponenten nutzen, was die Zusammenarbeit erleichtert und die Qualität des Endprodukts verbessert. Zudem beschleunigen Designsysteme den Entwicklungsprozess, indem sie wiederholbare Muster und klare Richtlinien bereitstellen. Dies führt zu einer besseren Benutzererfahrung und reduziert gleichzeitig den Aufwand für Design- und Entwicklungsarbeiten. ■

[1] Apple Developer, Human Interface Guidelines, www.dotnetpro.de/SL2412MAUIDesign1

[2] Google Material Design, <https://material.io/design>

[3] Fluent 2, Start Designing, www.dotnetpro.de/SL2412MAUIDesign2

[4] Microsoft Learn, Design and code Windows apps, www.dotnetpro.de/SL2412MAUIDesign3

[5] Apple Developer, Designing for macOS, www.dotnetpro.de/SL2412MAUIDesign4

[6] Tizen Developers, Design Principles, www.dotnetpro.de/SL2412MAUIDesign5

[7] Microsoft Learn, Designen einer App, www.dotnetpro.de/SL2412MAUIDesign6

[8] DevExpress, How to Implement Material Design 3 in a .NET MAUI Application, www.dotnetpro.de/SL2412MAUIDesign7

[9] MDC-MAUI, Introduction, <https://mdc-maui.github.io/>

[10] Grial, Emojis in .NET MAUI, www.dotnetpro.de/SL2412MAUIDesign8

[11] Human-Centered Design bei Wikipedia, www.dotnetpro.de/SL2412MAUIDesign9



Dr. Veikko Krypczyk
Elena Bochkor

Dr. Veikko Krypczyk ist seit Jahren begeisterter Softwareentwickler. Elena Bochkor beschäftigt sich mit dem Entwurf und Design mobiler Anwendungen, Webseiten und digitaler Produkte. Ihr technologischer Fokus liegt dabei auf C#, WPF, WinUI 3 und .NET MAUI. Beide bieten entsprechende Workshops für Entwickler an.

<https://larinet.com>

<https://www.tech-punkt.com>





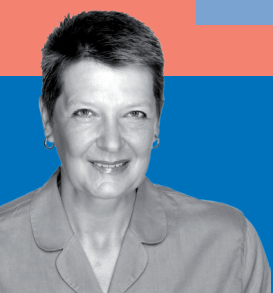
HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN



Hands-on Workshops .NET / C#

C#-Grundlagen	(3 Tage)
C# für Fortgeschrittene	(3 Tage)
ASP.NET Core und Web API	(3 Tage)
Datenbankentwicklung mit dem Entity Framework Core	(3 Tage)
Clean Code und Software Design	(3 Tage)
Async & Await in der Praxis	(2 Tage)
C#, .NET & SQL-Server-Update	(1 Tag)
Fix von .NET Framework zu .NET (CORE)	(1 Tag)
Einstieg in test- und behaviour-driven development (TDD/BDD) mit .NET Core	(2 Tage)
Testen von .NET-Anwendungen	(3 Tage)
Einstieg in MVVM mit WPF	(2 Tage)
Modulare WPF-Anwendungen mit PRISM	(3 Tage)
Einstieg in NoSQL mit MongoDB für .NET-Entwickler	(3 Tage)
ASP.NET Blazor - SPA-Anwendungen mit C# und .NET	(3 Tage)

**REMOTE
ODER
INHOUSE**



Weitere Informationen auf developer-academy.de
Preise auf Anfrage!

Ihre Ansprechpartnerin: Susanne Herl • +49 731 88005-8835 • susanne.herl@ebnermedia.de

QUESTPDF

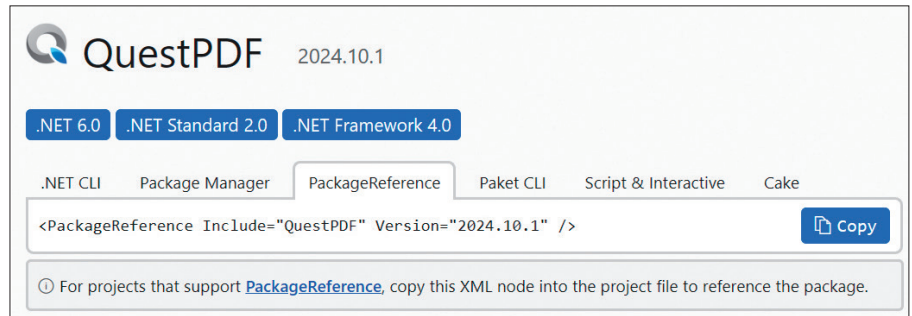
PDFs generieren



Wie Sie PDF-Dokumente mit C# erzeugen.

Ein Thema, das .NET-Entwicklern immer wieder Kopfzerbrechen bereitet, ist das Generieren von Berichten und PDF-Dokumenten. In der Vergangenheit war das nie einfach, und es standen auch nur wenige Tools dafür zur Verfügung. Besonders zu Zeiten des klassischen .NET Frameworks gab es nur eine Handvoll kostenpflichtiger Lösungen zur PDF-Generierung. Mit .NET Core und der Open-Source-Entwicklung hat sich in den letzten Jahren jedoch viel getan – eine besonders beliebte Bibliothek ist QuestPDF. Es ist an der Zeit, einen Blick auf diese Lösung zu werfen.

Die Open-Source-Lösung QuestPDF [1] hat in jüngster Zeit einen beträchtlichen Aufschwung erlebt und erlaubt das Erzeugen von PDFs mit einer schlanken und sehr performanten Bibliothek. Die Lizenzbedingungen wurden vor geraumer Zeit



Das NuGet-Paket QuestPDF (Bild 1)

geändert; seither gilt, dass die Nutzung von QuestPDF für kleinere Projekte oder Unternehmen kostenlos ist. Für größere Unternehmen ist hingegen eine Lizenz für das Entwicklungsteam erforderlich, Details dazu finden Sie unter [2]. Doch auch diese Kosten sind im Vergleich zu anderen Lösungen überschaubar. Wichtiger ist aber, dass der Funktionsumfang von QuestPDF mittlerweile recht groß ist.

Lets get started!

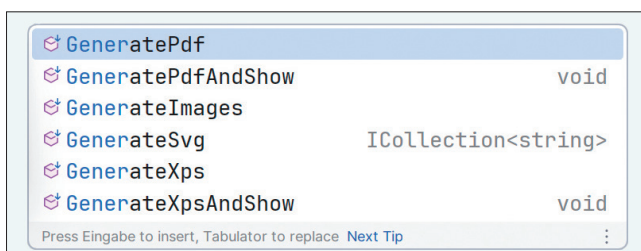
Ein kleines Beispiel soll demonstrieren, wie man mit der Bibliothek PDFs generieren kann und was QuestPDF so besonders macht. Die Entwickler von QuestPDF haben für das Erzeugen von PDFs einen Developer-First-Ansatz gewählt. Man startet also nicht mit einem Designer, einem WYSIWYG-Editor oder ähnlichen Tools, sondern mit purem C#-Code. Die Lösung ist damit für Entwickler maßgeschneidert. Für die Umsetzung reicht eine einfache Konsolenanwendung, die vom NuGet-Paket QuestPDF [3] unterstützt wird (Bild 1). Für das Beispiel kommt .NET 8 zum Einsatz.

Direkt nach der Installation des NuGet-Pakets steht die Factory-Methode `Document`.

```
var document = Document.Create(container =>
{
    container.Page(page =>
    {
        page.Size(PageSizes.A4);
        page.PageColor(Colors.White);
        page.DefaultTextStyle(x:TextStyle => x.Bold().FontSize(20));
    });
});

var pdf:byte[] = document.GeneratePdf();
document.GeneratePdfAndShow();
```

Wenige Zeilen Code genügen für ein einfaches PDF-Dokument im DIN-A4-Format (Bild 2)



Die Ausgabeformate von QuestPDF (Bild 3)

```
QuestPDF.Settings.License = LicenseType.Community;
```

Über `Settings.License` wird die Lizenz festgelegt (Bild 4)

`Create` zur Verfügung, deren Aufruf mit einigen wenigen Vorgaben ein erstes PDF baut, siehe Bild 2.

QuestPDF schlägt dabei den Weg des Fluent API ein: Als Vorgaben für das zu erzeugende PDF-Dokument werden mehrere

Lambda-Ausdrücke aneinandergereiht und weitere Parameter gegebenenfalls über Method Chaining hintereinander gesetzt (etwa `x.Bold().FontSize(20)`).

Im ersten Beispiel wird ein Dokument angefordert, das in DIN-A4-Größe gerendert werden soll, Weiß als Hintergrundfarbe hat und – sofern nicht abweichend bestimmt – mit fetter Schrift in der Schriftgröße 20 pt mit Text befüllt wird.

Hat man ein Dokument auf diese Weise konfiguriert, kann es mithilfe unterschiedlicher Methoden in das gewünschte Ausgabeformat konvertiert werden. Dabei ist PDF in der Regel das Zielformat, aber QuestPDF erlaubt auch den Export in den Formaten XPS und SVG oder in Form von Bildern (Bild 3).

Die Methode `GeneratePDF` liefert das gewünschte Ergebnis als Byte-Array. Wer möchte, kann auch einen Pfad angeben, unter dem die Daten gespeichert werden sollen. Der in Bild 2 genutzte Aufruf `GeneratePdfAndShow` dient dazu, das PDF in einem temporären Ordner zu erzeugen, um es anschließend mit dem Standard-PDF-Viewer zu öffnen.

Damit Sie das Beispiel nachvollziehen können, muss zuvor noch einmalig die Lizenz für den Prozess festgelegt werden. Dies erfolgt – wie in Bild 4 zu sehen ist – über das Setzen der Property `License` in der `Settings`-Klasse. Anschließend zeigt die Bibliothek das PDF-Dokument zum Startbeispiel bereits an, siehe Bild 5.

Lasset die Spiele beginnen

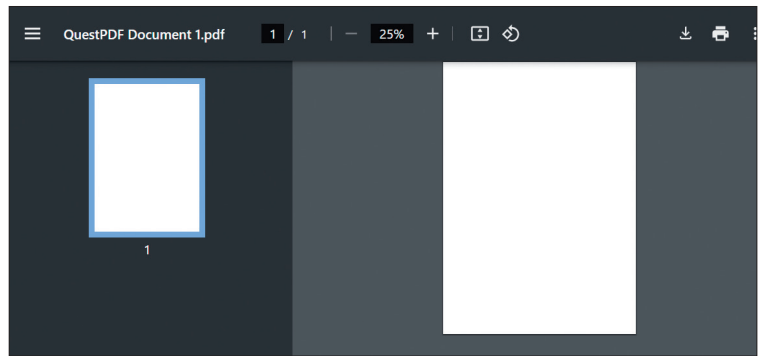
Das leere weiße Blatt macht kaum Aufwand. Wie sieht es aber aus, wenn mit QuestPDF komplexe Dokumente erzeugt werden sollen? Zunächst kommen Kopf- und Fußzeilen dazu. Diese lassen sich in den `Page(...)`-Aufruf einbauen, und dazwischen wird der Content der Seite platziert:

```
Page.Header()
...
Page.Content()
...
Page.Footer()
```

In Bild 6 sehen Sie ein Beispiel für ein Dokument mit Kopf- und Fußzeile sowie etwas Text. Die Fußzeile wird dabei angewiesen, die aktuelle Seitennummer anzuzeigen. Direkt nach dem Start des Programms erscheint die aktuelle Fassung auf dem Monitor (Bild 7).

Wie Sie erkennen können, stellt die Bibliothek auch kleine „Helferlein“ für das schnelle Prototyping zur Verfügung; so wurden hier ein Platzhalter für Blindtext (`Placeholders.LoremIpsum()`) und einer für ein Bild genutzt (`Placeholders.Image(width: 200, height: 100)`).

Auf der Webseite von QuestPDF finden Sie einige komplexere Beispiele [4] sowie eine sehr gute Dokumentation. Zu den Beispielen gehören Doku-



Die Vorschau des Startbeispiels (Bild 5)

```
page.Header() // IContainer
    .Text("DOTNETPRO ROCKS!");

page.Content() // IContainer
    .Column(x:ColumnDescriptor =>
    {
        x.Spacing(20);

        x.Item().Text(Placeholders.LoremIpsum());
        x.Item().Image(Placeholders.Image(width: 200, height: 100));
    });

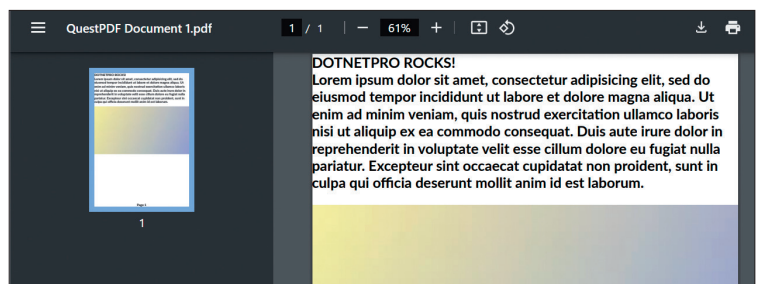
page.Footer()
    .AlignCenter() // IContainer
    .Text(content: x:TextDescriptor =>
    {
        x.Span(text: "Page ");
        x.CurrentPageNumber();
    });
```

Code für ein Dokument mit Kopf- und Fußzeile und Inhalt (Bild 6)

mente mit Tabellen, Karten und Bildern sowie Rechnungen (siehe Bild 8 und Bild 9) – alle mit Code, den man für die eigenen Anforderungen anpassen kann.

Weitere Funktionen

Nachdem QuestPDF stark auf C# fokussiert ist, integriert es sich sehr gut in die Programmiersprache: Grundsätzlich ist zum Beispiel ein `Document` immer ein `IDocument` und ein ►



Vorschau des PDF-Dokuments mit Kopf- und Fußzeile und Text (Bild 7)



Beispiel eines komplexen Dokuments mit Tabellen und Bildern von der Getting-Started-Webseite (Bild 8)

Container ein *IContainer*. Dies führt dazu, dass man Vorlagen beziehungsweise Templates für eigene Dokumente anlegen kann, siehe Bild 10 und Bild 11.

Es ist möglich, beliebige Komponenten oder Vorlagen zu definieren und mit der jeweiligen Datenquelle zu verbinden. Dies erleichtert die Wiederverwendbarkeit und sorgt für einen einheitlichen Look in verschiedenen Dokumenten. Diese Funktionalität lässt sich von *IComponent* bis hin zu ganzen Dokumenten übertragen. Mittels *IDocument* kann das Erzeugen eines Berichts oder Dokuments vollständig in eine Klasse gekapselt werden. Da es sich um einfachen C#-Code handelt, lässt sich dieser mit beliebigen Daten verknüpfen.

Invoice #5597

Issue date: 25.10.2020
Due date: 08.11.2020

From

Perspicatis Veritatis
Ipsa tempore
Illum sed minima, Ratione quia illum
consequuntur81@voluptatem.com
216-974-7107

For

Similique Accusamus
Voluptates ipsa
Quod necessitatibus, Sint adipisci iure
consectetur91@voluptates.com
319-637-2443

#	Product	Unit price	Quantity	Total
1	Provident temporibus	10,67\$	5	53,35\$
2	Dolor est ut	15,34\$	3	46,02\$
3	Nostrum id aspernatur	97,38\$	5	486,90\$
4	Modi est	17,35\$	1	17,35\$
5	Nulla non porro	59,63\$	2	119,26\$
6	Sint porro	38,35\$	9	345,15\$
7	Quae enim blanditiis	96,33\$	4	385,32\$
8	Inventore numquam cum	62,42\$	3	187,26\$
Grand total:				1640,61\$

Comments

Nulla rem hic voluptatum placeat explicabo. Sint pariatur doloremque corrupti enim commodi queraat fugiat suscipit id ea. Queraat incidunt quae similique commodi neque corporis vero blanditiis queraat quo. Debitis aspernatur labore ad atque sunt quae. Distinctio totam nesciunt queraat sit sint inventore.

Ein Rechnungsdokument mit Tabelle und Summenfunktion (Bild 9)

```
page.Content()
    .Container() // IContainer
    .Column(x:ColumnDescriptor =>
    {
        x.Item().Component<MyComponent>();
    });
```

Komponentenorientiertes Programmieren mit QuestPDF (Bild 10)

```
public class MyComponent : IComponent
{
    public void Compose(IContainer container)
    {
        container.Text("Hallo Welt")
            .Bold();
    }
}
```

Wiederverwendbare Komponente für das Rendern von Dokumenten (Bild 11)

Egal ob REST-Schnittstelle, ADO.NET, EF Core oder In-Memory-Daten – alles kann programmatisch mit dem Dokument verknüpft werden.

Fazit

Dieses Mal gewährt NETirol keinen Blick in die Welt von Microsoft, sondern betrachtet ein – meines Erachtens – besonders nützliches Werkzeug im .NET-Umfeld. Für die Qualitäten von QuestPDF spricht auch, dass der Entwicklerwerkzeug-Hersteller JetBrains die Bibliothek QuestPDF als Beispiel für Performance-Optimierung herangezogen hat [5]. Wer also gerade eine nervige PDF-Aufgabe vor sich hat oder davon ausgeht, dass das in absehbarer Zeit der Fall sein wird, für den ist QuestPDF einen näheren Blick wert. ■

[1] QuestPDF, www.questpdf.com

[2] Lizenz von QuestPDF, www.questpdf.com/license/

[3] NuGet-Paket QuestPDF, www.nuget.org/packages/QuestPDF

[4] QuestPDF, Getting Started, www.questpdf.com/getting-started.html

[5] JetBrains, Profiling QuestPDF: In Pursuit of Performance, www.dotnetpro.de/SL2412NETirol1



Christian Giesswein

studierte Wirtschaftsinformatik in Wien und entwickelt von klein auf Software mit .NET und C#. In Tirol hat er das Unternehmen Giesswein Software-Solutions gegründet, das sich auf Individualsoftware und Consulting spezialisiert.

christian@software.tirol

dnpCode

A24012NETirol

.NET ASPIRE IN LEGACY-APPLIKATIONEN EINSETZEN

Abseits des Trampelpfads

Aspire verspricht, dem Entwickler durch die Übernahme von Standardaufgaben viel Arbeit abzunehmen – wenn die Applikation modern ist. Klappt das auch bei betagterer Software?

Aspire, Microsofts Starthilfe für die Erstellung von Cloud-Native-Applikationen, haben wir bereits anhand des von Microsoft gelieferten Beispiels unter die Lupe genommen [1]. Und auf der grünen Wiese sah das auch alles ganz wunderbar aus. Aber wie schlägt sich Aspire, wenn man versucht, dessen Verlockungen in einer älteren Applikation wahr werden zu lassen?

Keine Legacy-Applikation ist wie die andere, deshalb kann die Antwort auf diese Frage unterschiedlich ausfallen. Wir ziehen als Studienobjekt die Applikation „Rate My Beer“ heran. Da uns diese Applikation schon seit langer Zeit begleitet, hat sie bereits viele .NET-Versionen kommen und gehen sehen – was dem Code auch anzumerken ist. Die Tatsache, dass es sich „nur“ um eine Demoapplikation handelt, ist dem historischen Sammelsurium auch nicht zuträglich. Alles in allem also eine sehr überzeugende und realitätsnahe Legacy-Applikation. Die Architektur der Anwendung ist in **Bild 1** dargestellt.

Die Applikation umfasst drei Dienste – je einen für Checks, für Kommentare und die Bewertung eines Biers. Jeder Dienst besteht aus einem HTTP-API und einem NServiceBus-Endpunkt. Jeder Dienst verfügt über eine eigene Datenbank und ist zur Kommunikation an einen gemeinsamen Azure Service Bus angeschlossen. Dazu kommen noch ein Frontend und ein Backend for Frontend. Das gesamte Projekt basiert noch auf .NET 6.

Aspire zu einer bestehenden .NET-Applikation hinzufügen

Microsoft hat selbst einen Beitrag zum Thema veröffentlicht [2]. Wenn dort jedoch von einer bestehenden Applikation die Rede ist, dann sicher nicht von einer ernst zu nehmenden Legacy-Applikation. Denn wie bereits in den Vorbedingungen erwähnt wird, ist dort .NET 8.0 gefordert. Diese Anforderung wird von der Demo-Applikation, die Microsoft anbietet, dann auch erfüllt. Aber wir wollten ja auch runter vom Trampelpfad – also stürzen wir uns ins Gestrüpp.

Zuerst kann man sich natürlich fragen, warum man die Demoapplikation nicht einfach auf .NET 8 migriert. Schließlich macht Microsoft keinen Hehl daraus, dass Aspire eigentlich .NET 8 voraussetzt. Da ist es unfair, das Ganze an einer .NET-6-Applikation zu verproben. Zum einen werden wir das bei der Bewertung wohlwollend betrachten, zum anderen besteht die Welt nun mal nicht nur aus .NET-8-Applikationen. In einem Beitrag auf Stack Overflow unter [3] gibt uns David Fowler ein bisschen Hoffnung – hier lernen wir, dass man im

Grunde auch .NET-6/7/8-Applikationen mit Aspire orchestrieren kann, man aber .NET 8 für den vollen Umfang an Features benötigt.

Dann wird uns noch mitgeteilt, dass ein Upgrade von .NET 6 auf .NET 8 gradlinig und einfach sein sollte. Dennoch kann es durchaus Gründe geben, aus denen heraus das Ganze nicht ganz so einfach ist, wie es sich vielleicht anhört. Wir nehmen also jetzt einfach mal an, dass wir einen solchen Fall vorliegen haben und uns das Leben trotzdem mit Aspire leichter machen wollen.

Also kein Update auf .NET 8 – dafür aber trotzdem Aspire. Immerhin gibt uns Microsoft in [4] einen Leitfaden an die Hand, wie wir Aspire in unsere bestehende Applikation einbauen sollen. Auch hier ist die Demo-Applikation bereits auf dem Stand von .NET 8, aber die vorgeschlagene Reihenfolge können wir uns zunutze machen. Es wird empfohlen, zuerst die Service Defaults einzubauen und sich danach dem AppHost – also der eigentlichen Orchestrierung – zu widmen.

Wie war das noch mal?

Ein kurzer Recap: Aspire fügt zu einer bestehenden Solution zwei Projekte hinzu: die Service Defaults und den AppHost. Damit man die notwendigen Templates zur Verfügung hat, muss man zuerst den Aspire-Workload installieren. Dies gelingt folgendermaßen:

```
dotnet workload update
dotnet workload install aspire
dotnet workload list
```

Wenn man diese Schritte ausgeführt hat, kann man das Service-Defaults-Projekt wie folgt anlegen:

```
dotnet new aspire-servicedefaults -n ServiceDefaults
```

Dieses Projekt fügt man dann zu jedem anderen Projekt in der Solution als Referenz hinzu und ruft auf dem Application-Builder die Methode `AddServiceDefaults()` auf. Im zweiten Schritt kann man dann mittels

```
dotnet new aspire-apphost -n AppHost
```

den AppHost hinzufügen, über den später das Projekt mit samt allen Abhängigkeiten gestartet wird. So einfach ist das zumindest, wenn man bereits eine .NET-8-Applikation hat – für uns wird das Leben leider ein bisschen schwieriger. ►

Die Service Defaults auf .NET 6 portieren

Denn leider stellt sich die erste Ernüchterung bereits ein, wenn wir das Service-Defaults-Projekt einem Projekt der Beispielapplikation hinzufügen möchten, denn die Service Defaults kommen als .NET-8-Projekt daher.

Also stellen wir dieses Projekt auf .NET 6 um, was gleich das nächste Problem mit sich bringt: Das NuGet-Paket für die Service Discovery ist nicht mit .NET 6.0 kompatibel. Dies muss man also mit einem bedingten Import lösen:

```
<PackageReference Include="
  "Microsoft.Extensions.ServiceDiscovery"
  Condition="'$(TargetFramework)' ==
    'net8.0'" Version="8.2.0" />
```

Damit ahnen wir auch leider schon, dass wir ohne tiefgreifendere Umbauten bereits ein nettes Feature nicht bekommen werden – aber wie wir später sehen werden, gibt es durchaus noch Hoffnung.

Wenn man das Projekt nun so weit umgebaut hat, kann man versuchen, die magischen Zeilen in den eigenen Boot-Code einzubauen:

```
builder.AddServiceDefaults();
```

und

```
app.MapDefaultEndpoints();
```

Hier wartet gleich die nächste Schwierigkeit auf uns.

Zeig mir deinen Boot-Code, und ich sage dir, wie alt du bist

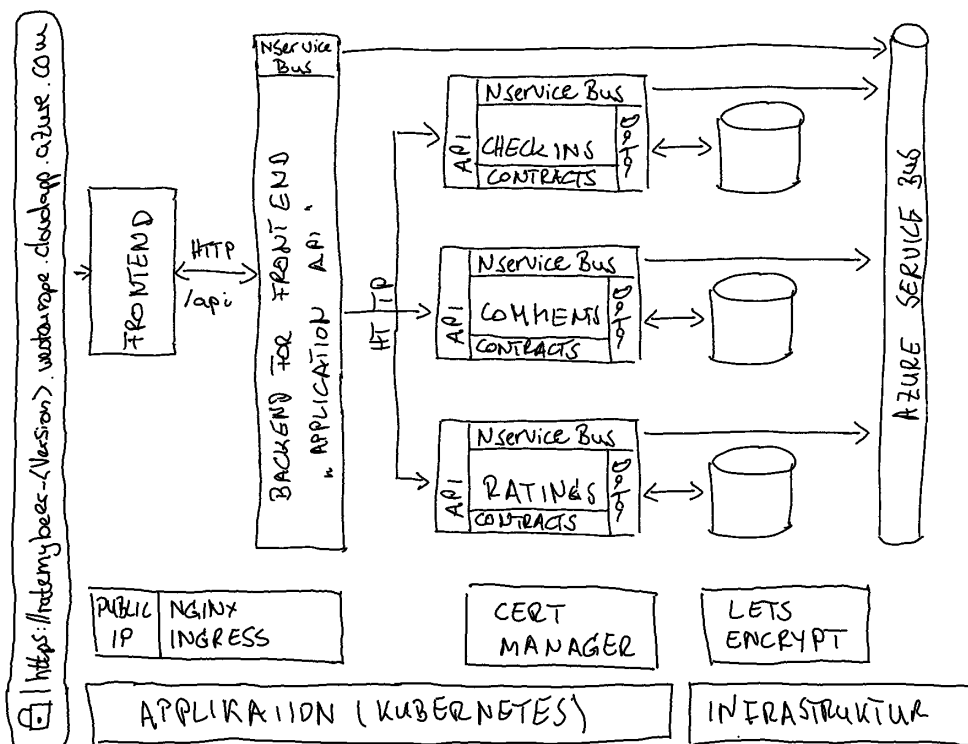
Die Erweiterungsmethode `AddServiceDefaults()` basiert auf dem `IHostApplicationBuilder` – und der kam erst mit .NET 8. Es mag bei einem .NET-Update zunächst wie ein Segen erscheinen, dass die Builder zum Hosten eines API abwärtskompatibel sind. Es ist jedoch erstaunlich, wie viele Varianten dazu sich in einer Codebasis ansammeln können.

Seit .NET Core gibt es verschiedene Varianten – eine gute Übersicht finden Sie unter [5]. Die Reise beginnt mit .NET Core 2. Zu dieser Zeit war es noch üblich, eine Klasse `Program` (zur Konfiguration des Hosts) und eine Klasse `Startup` (zur Konfiguration der Applikation) zu definieren. Kern der Host-Konfiguration war etwa dieser Code:

```
public static IWebHost BuildWebHost(string[] args) =>
    WebHost.CreateDefaultBuilder(args)
        .UseStartup<Startup>()
        .Build();
```

Dabei liefert die Methode `WebHost.CreateDefaultBuilder()` einen `IWebHostBuilder`, der dann zu einem `IWebHost` gebaut wurde.

In der folgenden Version .NET Core 3 und in .NET 5 wurde dieser Ansatz durch einen generischen Host abgelöst. Während der vorherige Ansatz nur für Web-Hosts funktioniert hat, sollten der gleiche Ansatz in der neuen Version auch



Die Architektur der Beispielanwendung „RateMyBeer“ (Bild 1)

Des Weiteren muss nun der Code in der `Extensions.cs` ebenfalls angepasst werden:

```
#if NET8_0_OR_GREATER
builder.Services.AddServiceDiscovery();
#endif

builder.Services.ConfigureHttpClientDefaults(http =>
{
    http.AddStandardResilienceHandler();

#if NET8_0_OR_GREATER
    http.AddServiceDiscovery();
#endif
});
```

für lang laufende Prozesse (Worker) und gRPC-Services funktionieren.

Für Web-Hosts gab es dann eine Erweiterungsmethode, um die Migration zu ermöglichen:

```
public static IHostBuilder CreateHostBuilder(string[]
    args) =>
    Host.CreateDefaultBuilder(args)
        .ConfigureWebHostDefaults(webBuilder =>
        {
            webBuilder.UseStartup<Startup>();
        });
```

Hier hat man über die Methode `Host.CreateDefaultBuilder()` zunächst einen generischen `IHostBuilder` produziert; die Methode `ConfigureWebHostDefaults` hat dann den alten `IWebHostBuilder` zur Verfügung gestellt. Nach dem Bauen hatte man einen ebenfalls generischen `IHost` vor sich.

Mit .NET 6 hat sich der Boot-Code noch einmal deutlich verändert: Die Trennung in `Program` und `Startup` wurde aufgehoben, und die Konfiguration für Web-Hosts erfolgte nicht mehr über den generischen Builder, sondern sah jetzt so aus:

```
var builder = WebApplication.CreateBuilder(args);
// Konfiguration des Hosts
var app = builder.Build();
// Konfiguration der Application
app.Run();
```

Die Methode `WebApplication.CreateBuilder` liefert uns hier einen `WebApplicationBuilder`, der ausnahmsweise mal gar kein eigenes Interface implementierte. Die Applikation wurde in dieser Version durch eine Instanz der Klasse `WebApplication` dargestellt, die dann das generische `IHost`-Interface implementiert. Um die Kompatibilität zu gewährleisten, kann man aber über `builder.WebHost` auf eine Instanz des altbekannten Interfaces `IWebHostBuilder` zugreifen.

In .NET 8 sieht das Ganze erst einmal ziemlich ähnlich aus, doch leider trägt der Schein.

```
var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();
app.Run();
```

Denn hier liefert uns die Methode `WebApplication.CreateBuilder()` eine Klasse zurück, die das Interface `IHostApplicationBuilder` implementiert, was in .NET 6 noch nicht der Fall war. Hierauf basieren auch die Erweiterungsmethoden, die Aspire verwendet.

Der Haken an der Kompatibilität

Kompatibilität hört sich also erst mal gut an. Die Kehrseite dieser Medaille ist aber, dass man unter .NET 6 noch Code betreiben kann, der aus der .NET-Core-Ära stammt.

Konkret bedeutet das, dass dort, wo für Web-Hosts noch der generische Host verwendet wird, ein Update auf das WebApplication-API erforderlich ist. Leider fehlt hier die Me-

thode `AddWebHostDefaults`, und damit kann man die eigene `Startup`-Klasse nicht ohne Weiteres verwenden.

Wenn man nicht den gesamten Boot-Code umstricken möchte, kann man sich mit dem in [Listing 1](#) gezeigten Umweg behelfen. Hier verwenden wir den `WebApplicationBuilder` und instanzieren unsere `Startup`-Klasse selbst. Dabei übergibt man die Konfiguration per `builder.Configuration`. Danach kann man die Methode `ConfigureServices` aufrufen und übergibt die `IServiceCollection`, auf die man über `builder.Configuration` zugreifen kann. Danach lässt sich der Builder weiter konfigurieren, wie es in .NET 6 üblich ist – vor allem kann man über die Eigenschaft `builder.WebHost` auf das Interface `IWebHostBuilder` zugreifen. Und dieses Interface wird zum Ziel der Implementierung der Methode `AddServiceDefaults()` für .NET 6. Nachdem man die Applikation über `builder.Build()` gebaut hat, kann man die `Configure`-Methode der `Startup`-Klasse aufrufen und die Applikation und das Environment übergeben. Auf der Applikation selbst ruft man dann `app.MapDefaultEndpoints` auf und fügt so den zweiten Teil der Service-Defaults-Magie hinzu. Diese Methode ist zwischen .NET 6 und .NET 8 kompatibel.

Service Defaults und der IWebHostBuilder

Damit das alles zusammen so wie in [Listing 1](#) gezeigt funktioniert, muss man die Methode `AddServiceDefaults` also vom `IHostApplicationBuilder` auf den `IWebHostBuilder` umbauen. Der Code dazu befindet sich im GitHub-Repository der Demo-Applikation unter [6]. Im Kern geht es darum, die Methode

```
public static IHostApplicationBuilder AddServiceDefaults(
    this IHostApplicationBuilder builder)
```

zu folgender Signatur umzubauen:

```
public static IWebHostBuilder AddServiceDefaults(
    this IWebHostBuilder builder) ▶
```

● Listing 1: WebApplicationBuilder mit Startup-Klasse

```
public static void Main(string[] args)
{
    var builder = WebApplication.CreateBuilder();
    var startup = new Startup(builder.Configuration);
    startup.ConfigureServices(builder.Services);

    builder
        .WebHost // IWebHostBuilder
        .AddServiceDefaults();

    var app = builder.Build();
    startup.Configure(app, app.Environment);
    app.MapDefaultEndpoints();
    app.Run();
}
```

● Listing 2: Der AppHost für die Beispielapplikation

```
var builder = DistributedApplication.
    CreateBuilder(args);

var checkinsApi = builder.AddProject("checkinsApi",
    @"..\..\..\checkins\src\Lab.RateMyBeer.Checkins.Api\
    Lab.RateMyBeer.Checkins.Api.csproj");
var checkins = builder.AddProject("checkins",
    @"..\..\..\checkins\src\Lab.RateMyBeer.Checkins\
    Lab.RateMyBeer.Checkins.csproj");

var commentsApi = builder.AddProject("commentsApi",
    @"..\..\..\comments\src\Lab.RateMyBeer.Comments.Api\
    Lab.RateMyBeer.Comments.Api.csproj");
var comments = builder.AddProject("comments",
    @"..\..\..\comments\src\Lab.RateMyBeer.Comments\
    Lab.RateMyBeer.Comments.csproj");

var ratingsApi = builder.AddProject("ratingsApi",
    @"..\..\..\ratings\src\Lab.RateMyBeer.Ratings.Api\
    Lab.RateMyBeer.Ratings.Api.csproj");
var ratings = builder.AddProject("ratings",
    @"..\..\..\ratings\src\Lab.RateMyBeer.Ratings\
    Lab.RateMyBeer.Ratings.csproj");

var frontendApi = builder.AddProject("frontendApi",
    @"..\..\..\frontend\src\Lab.RateMyBeer.Frontend.Api\
    Lab.RateMyBeer.Frontend.Api.csproj")
    .WithReference(checkinsApi)
    .WithReference(commentsApi)
    .WithReference(ratingsApi);

var frontend = builder.AddProject("frontend",
    @"..\..\..\frontend\src\Lab.RateMyBeer.Frontend\
    Lab.RateMyBeer.Frontend.csproj")
    .WithReference(frontendApi);

builder.Build().Run();
```

Das läuft im Grund relativ gradlinig ab, lediglich die Methoden *ConfigureOpenTelemetry*, *AddOpenTelemetryExporters* und *AddDefaultHealthChecks* müssen ebenfalls noch umgebaut werden, weil diese von der Methode *AddServiceDefaults* aufgerufen werden. Auch hier muss jeweils *IApplicationHostBuilder* durch den *IWebHostBuilder* ersetzt werden.

Der Code innerhalb der Methoden lässt sich fast eins zu eins übertragen. Der wesentliche Unterschied besteht darin, dass sich das API für die Konfiguration von Callback-Methoden aus .NET 6 zu einer etwas schnörkelloseren Implementierung über Properties verändert hat. Anstelle von

```
builder.Services.AddOpenTelemetry()
```

muss man in .NET 6 schreiben:

```
builder.ConfigureServices(services =>
{
    services
        .AddOpenTelemetry()
});
```

Wenn man diese Fleißarbeit hinter sich gebracht hat, kann man die Service Defaults zumindest in den Web-Host-Projekten auf die in [Listing 1](#) gezeigte Weise integrieren.

Weniger Glück hat man mit den NServiceBus-Endpunkten. Da diese den generischen Host verwenden und keine Web-Hosts sind, kann man hier die Methode *ConfigureWebHostDefaults*() nicht aufrufen – das wird mit einer Exception quittiert. Somit kommt man hier ebenfalls nicht an einen *IWebHostBuilder*. Entsprechend gelangt man auch nicht in den Genuss der Service Defaults.

Dieses war der erste Streich, doch der AppHost folgt sogleich

Die Service Defaults bereiten die APIs für die Nutzung vieler Vorteile von Aspire vor – und auch sonst haben sie einiges zu bieten: die Integration von Polly, um den HTTP-APIs eine bessere Resilienz zu verleihen, standardmäßiges Health Checking und die Anbindung von OpenTelemetry. Und wenn man die Service Defaults schon umgeschrieben hat, kann man natürlich noch weitere Standards für das eigene Team mit hinzufügen, wie zum Beispiel Logging über Seq oder Ähnliches. Wenn man das Ganze dann für sich als NuGet bereitstellt, das man in die eigenen Projekte einbindet, kann man den Entwicklern schon einiges an Boilerplate-Code abnehmen.

Viel besser wird das aber alles noch, wenn man den AppHost mit in Betracht zieht. Dieser bietet die Möglichkeit, den strukturellen Aufbau der Applikation in C#-Code zu beschreiben. Damit kann man leicht veranschaulichen, welche Projekte zu der gesamten Applikation gehören und wie diese voneinander abhängen. Das ist für sich allein genommen schon eine gute Sache, aber noch nicht alles. Ein weiterer Benefit ist, dass man durch das Starten des AppHosts alle dort aufgelisteten Projekte mitstarten kann. Wer schon einmal versucht hat, eine Applikation zu starten, die aus vielen verschiedenen Services besteht, weiß, was das für ein Gamechanger ist: Anstatt sechs Instanzen von Visual Studio zu öffnen und überall auf *Play* zu klicken, startet man einfach ein einzelnes Projekt, und die gesamte Applikation läuft.

Will ich haben!

Vergleicht man die Integration des AppHosts mit dem Einbinden der Service Defaults, ist das wirklich wohlthuend einfach. Man erstellt den AppHost einfach mittels

```
dotnet new aspire-apphost -o AppHost
```

und fügt für die Beispiel-Applikation den Code aus [Listing 2](#) hinzu. Hier findet man für jedes Element aus [Bild 1](#) eine entsprechende Code-Zeile wieder. Anders als es in den gängigen Beispielen gezeigt wird, werden die Projekte dem *DistributedApplicationBuilder* aber nicht über die automatisch generierte Klasse, sondern über den Pfad zur Projektdatei hinzugefügt. Damit die Klasse generiert wird, muss der AppHost eine Referenz auf das Zielprojekt haben. Das ist aber dann hinderlich, wenn der Code der Applikation auf mehrere Repositories verteilt ist. Aus diesem Grund hat Microsoft die Möglichkeit eingebaut, den Pfad zur Projektdatei anzugeben. Ein Nachteil dieser Variante ist allerdings, dass beim Starten des AppHosts die Projekte nicht automatisch neu kompiliert werden. Änderungen bekommt man also unter Umständen nicht sofort mit.

Die Angabe der Referenzen über *WithReference* wird normalerweise für die Service Discovery verwendet. Diese können wir aufgrund der Einschränkung auf .NET 6 jedoch leider nicht verwenden. Die Verweise wurden hier trotzdem aufgenommen – auf diese Weise kommuniziert der Code besser die tatsächliche Struktur der Applikation. Da die Beispielapplikation ja zuvor ohne Aspire betrieben wurde, sind bereits Mechanismen vorhanden, um Connection Strings und Ähnliches zu laden – zum Beispiel über eine Azure App Configuration oder einfach über die *appsettings.json*. Dementsprechend braucht es gar nicht viel mehr – der vorliegende Code reicht aus, um mit einem Druck auf [F5] alle Projekte im Debug-Modus zu starten. Bis dann auch alles auf einen Klick läuft, ist noch ein bisschen Fummelei nötig gewesen. Hier war mal eine *launchSettings.json* fehlerhaft – Aspire stört sich an Trailing Commas –, dort war ein Projekt nicht kompiliert, oder es hatten sich Pfade geändert. Aber nachdem es dann lief: echte Quality of Life.

Obwohl wir auf die Service Discovery verzichten müssen, kann man sich aber ein Feature von Aspire zunutze machen, um die Anwendung über den AppHost für den lokalen Betrieb zu konfigurieren. Nehmen wir zum Beispiel den Azure Service Bus – vielleicht wollen wir für das lokale Testen einen anderen Namespace verwenden. Für diesen Fall können wir im AppHost einen Parameter definieren:

```
var serviceBus = builder.AddParameter("serviceBus",
    secret: true);
```

Den Wert des Parameters legt man zum Beispiel in der *appsettings.json* in einem Objekt namens *Parameters* fest:

```
{
  "Parameters": {
    "serviceBus": "Hier der ConnectionString"
  }
}
```

Oder man definiert ihn, wenn der Connection String einen Access Key enthält, in den User Secrets. Den Parameter kann

man dann zum Beispiel an eine Umgebungsvariable binden. Die API-Projekte erwarten den Connection String zum Service Bus unter folgendem Konfigurationsschlüssel:

```
Dependencies:NServiceBus:TransportConnectionString
```

Dieser lässt sich wie folgt über eine Umgebungsvariable mit dem Wert des Parameters belegen:

```
var checkinsApi = builder
    .AddProject("checkinsApi", "...")
    .WithEnvironment("Dependencies__NServiceBus__
        TransportConnectionString", serviceBus);
```

Auf diese Weise muss man Konfigurationswerte nicht in allen Projekten ändern und kann diese leicht für die lokale Ausführung anpassen. Das gleiche Prinzip lässt sich auch auf Connection Strings für Datenbanken ausweiten.

Fazit

Abseits des Trampelpfades macht Aspire erwartungsgemäß etwas mehr Arbeit als bei Happy-Path-Demos. Den reduzierten Funktionsumfang darf man nicht negativ auslegen – schließlich war das Einsatzfeld ja außerhalb dessen, was offiziell unterstützt wird. Trotzdem bietet Aspire auch für ein .NET-6-Projekt schon viele Vorteile – hauptsächlich das einfache Starten der Anwendung. Aber auch schnell einen Docker-Container mit einzubinden geht leicht von der Hand.

Wer den vollen Funktionsumfang genießen möchte, muss dann die Migration auf .NET 8 oder höher in Angriff nehmen – was allerdings in Anbetracht der Tatsache, dass der Support für .NET 6 gegen Ende dieses Jahres abläuft, ohnehin eine gute Idee ist. ■

- [1] Tobias Richling, *Das Streben nach der Cloud*, *dotnetpro* 6/2024, Seite 14 ff., www.dotnetpro.de/A2406Aspire
- [2] Microsoft Learn, *Tutorial: Add .NET Aspire to an existing .NET app*, www.dotnetpro.de/SL2412Aspire1
- [3] Stackoverflow, *Is it possible to use .NET Aspire with a .NET 6 app?*, www.dotnetpro.de/SL2412Aspire2
- [4] Jon Galloway, *Adding .NET Aspire to your existing .NET apps*, www.dotnetpro.de/SL2412Aspire3
- [5] Andrew Lock, *Comparing WebApplicationBuilder to the Generic Host*, www.dotnetpro.de/SL2412Aspire4
- [6] *Beispielanwendung zum Artikel auf GitHub*, www.dotnetpro.de/SL2412Aspire5



Tobias Richling

arbeitet als Softwareentwickler und betreut die B2B-Plattformen der apetito AG. Er sucht dabei stets nach Möglichkeiten zur Verbesserung von Funktionalität und Performance. Sie erreichen ihn unter tobias.richling@apetito.de oder per [@trichling](https://twitter.com/trichling).

dnpCode

A2412Aspire

REAKTIV PROGRAMMIEREN, TEIL 3

Rx.NET Deep Dive



Threading, Scheduling und Testing.

Dieser dritte und abschließende Teil der Artikel-Trilogie zu Rx.NET betrachtet technische Details zum Threading, Scheduling und dem Testing. Kurzum alles, was man für einen fortgeschrittenen Umgang mit der Technologie wissen muss. Im ersten Teil der Serie [1] wurde die Open-Source-Bibliothek Rx.NET, die Teil der .NET-Foundation ist, als Implementierung der Reactive Extensions (Rx) vorgestellt. Im zwei-

ten Teil [2] wurde eine Übersicht der wichtigsten Operatoren erläutert und der Unterschied zwischen Hot- und Cold-Observables deutlich gemacht.

Dabei wurde bislang stets das Standardverhalten angenommen und ein genauer Blick auf das Verhalten von Threads und Scheduling vermieden. Dennoch ist es gerade in Multi-Threaded-Anwendungen essenziell zu verstehen, welche Annahmen man treffen kann und wann es notwendig ist, das Verhalten genauer zu steuern.

Um diesem abschließenden Artikel folgen zu können, empfiehlt es sich, die beiden vorangegangenen Teile der Serie zu lesen oder bereits Erfahrungen im Bereich Rx.NET zu haben. Zusätzlich sind Kenntnisse im Bereich Multi-Threading mit .NET empfehlenswert.

Listing 1: ManagedThreadId

```
var subject = new Subject<string>();
var subscription = subject.Select(name => {
    Console.WriteLine(
        $"{name}: Select() in Thread
        {CurrentThread.ManagedThreadId}");
    return name;
});
.Where(name => {
    Console.WriteLine(
        $"{name}: Where() in Thread
        {CurrentThread.ManagedThreadId}");
    return true;
}).Subscribe(name => {
    Console.WriteLine(
        $"{name}: Subscribe() in Thread
        {CurrentThread.ManagedThreadId}");
});

var name = "main";
Console.WriteLine(
    $"{name}: start in Thread
    {CurrentThread.ManagedThreadId}");
subject.OnNext(name);
Console.WriteLine(
    $"{name}: fertig in Thread
    {CurrentThread.ManagedThreadId}");
Console.ReadLine();

// Ausgabe
main: start in Thread 1
main: Select() in Thread 1
main: Where() in Thread 1
main: Subscribe() in Thread 1
main: fertig in Thread 1
```

Eine zweite Spielregel

Ein *IObservable<T>* bietet drei Methoden an: *OnNext*, *OnError* und *OnCompleted*. *OnNext* informiert über neue Daten und kann beliebig oft hintereinander aufgerufen werden. Abgeschlossen wird ein Observable entweder mit *OnCompleted* oder mit *OnError*.

Die erste Spielregel zu der Reihenfolge der Aufrufe besagt, dass nach *OnCompleted* oder *OnError* keine andere Methode mehr aufgerufen werden darf: weder *OnNext* noch *OnCompleted* oder *OnError*.

Bezüglich Threading kommt lediglich eine zweite Spielregel hinzu: Für keine Subscription dürfen parallele Aufrufe getätigt werden. Das bedeutet: Solange *OnNext* von einer Subscription verarbeitet wird, darf *OnNext* nicht noch einmal auf dieser Instanz aufgerufen werden. Das gilt auch für *OnError* oder *OnCompleted*. Der Aufrufer muss also so lange warten, bis die *OnNext*-Methode der Subscription fertig ist.

Diese Spielregel vereinfacht die Implementierung von Observables extrem. Denn solange ein Observable lediglich bei einer einzigen Quelle registriert ist, braucht sich dieses über parallele Aufrufe der Methoden *OnNext*, *OnError* und *OnCompleted* keine Sorgen zu machen. Die Aufrufe der Methoden sind immer strikt sequenziell.

Diese Spielregel gilt auch, wenn man sich bei Operatoren registriert, die mehrere Observables kombinieren, wie beispielsweise *Merge*. Kurz gesagt: Das Ergebnis des *Merge*-Operators ist immer ein Observable, bei dem alle Daten via *OnNext* strikt sequenziell emittiert werden.

Der Thread ist nicht garantiert

Die zweite Spielregel besagt zwar, dass die Methoden des Observables immer strikt sequenziell aufgerufen werden

müssen, aber das impliziert noch nicht, dass die Methoden von einem speziellen Thread aufgerufen werden oder dass es immer der gleiche Thread sein wird. Darauf kann und darf man sich nicht verlassen, wenn man eigene Observables implementiert oder abonniert.

Man liest im Zusammenhang mit Rx.NET bisweilen vom „free threaded model“ [3]. Dieses besagt im Grunde nur, dass man sich keine Gedanken darüber machen muss, in welchem Thread ein Observable aufgerufen wird. Die Thread-Safety-Eigenschaft wird von der Spielregel „immer strikt sequenziell“ und, falls notwendig, von internen Synchronisierungsmaßnahmen innerhalb des Operators gewährleistet.

Die meisten Operatoren sind „pragmatisch“

Die meisten Operatoren (*Select*, *Where*, *Take*, *Buffer* et cetera) sind pragmatisch programmiert. Sie werden von einem beliebigen Thread aufgerufen und rufen den nachfolgenden Operator per Function-Call in demselben Thread auf. Ein Function-Call ist deutlich effizienter als das Auslagern in einen anderen Thread. Betrachten Sie dazu das Beispiel einer

halten sich analog dazu und führen die Berechnung im Aufrufer-Thread durch.

OnNext in unterschiedlichen Threads aufrufen

Interessant ist, was passiert, wenn man eine Subscription sowie eine Verarbeitungskette [*Select()*, *Where()*] erzeugt und diese aus zwei verschiedenen Threads mit *OnNext* befüllt. In Listing 2 wird dieselbe Verarbeitungskette wie in Listing 1 genutzt. Allerdings wird diese Subscription aus mehreren Threads heraus aufgerufen: einmal aus dem Haupt-Thread 1 und zweimal aus den neuen Threads 12 und 13. Wie Sie an der Ausgabe erkennen können, verarbeitet der Code nun drei Threads. Die Verarbeitungsketten laufen unabhängig voneinander gleichzeitig, aber jeweils vollständig in ihrem eigenen Thread.

Zeitbasierte Operatoren

Es gibt Operatoren, die einen Zeitbezug haben, wie beispielsweise *Delay*. Der *Delay*-Operator verzögert die Ausführung um eine gewisse Zeit. Nehmen wir folgendes Szenario an:

● Listing 2: Aufruf in unterschiedlichen Threads

```
void logic(string name)
{
    Console.WriteLine(
        $"{name}: start in Thread
        {CurrentThread.ManagedThreadId}");
    subject.OnNext(name);
    Console.WriteLine(
        $"{name}: fertig in Thread
        {CurrentThread.ManagedThreadId}");
};

new Thread(() => logic("Worker1")).Start();
new Thread(() => logic("Worker2")).Start();
logic("Main "); // wichtig nach Worker2
                // aufrufen, da dieser Call
                // blockend ist

// Ausgabe
Main   : start in Thread 1
Worker1: start in Thread 12
Worker2: start in Thread 13
Worker1: Select() in Thread 12
Main   : Select() in Thread 1
Worker2: Select() in Thread 13
Main   : Where() in Thread 1
Worker2: Where() in Thread 13
Worker1: Where() in Thread 12
Worker1: Subscribe() in Thread 12
Main   : Subscribe() in Thread 1
Main   : fertig in Thread 1
Worker1: fertig in Thread 12
Worker2: Subscribe() in Thread 13
Worker2: fertig in Thread 13
```

komplexen Verarbeitungskette in Listing 1. Anhand der Ausgabe des Codes erkennen Sie, dass alle Operatoren im gleichen Thread aufgerufen werden.

Thread 1 ist der Haupt-Thread von Konsolenanwendungen. Die gesamte Verarbeitungskette wird durchlaufen, bevor *fertig in Thread 1* ausgegeben wird.

Gemäß Spielregel 2 darf man sich zwar nicht darauf verlassen, dass alle Operatoren und die Subscription in demselben Thread aufgerufen werden, aber aus Performancegründen und auch für eine bessere Nachvollziehbarkeit verhält es sich in der Praxis in der Regel so. Viele weitere, nicht zeitabhängige Operatoren (*Select*, *Where*, *Take*, *Buffer* et cetera.) ver-

```
Subscription = subject.Select(...)
    .Delay(5s)
    .Where(...)
    .Subscribe(...);
```

Der *Delay*-Operator ist so implementiert, dass er nicht blockiert. Das heißt, dass der aufrufende Thread in unserem Beispiel nicht fünf Sekunden lang warten muss, bis er mit dem *Where*-Operator fortfährt. Dies wäre in den allermeisten Applikationen inakzeptabel. Man stelle sich vor, das UI würde fünf Sekunden lang nicht reagieren, weil ein *Delay*-Operator wartet, bis ein Time-out erreicht ist. ►

Betrachten Sie nun das Beispiel in [Listing 3](#). Darin werden die beiden Threads *Worker1* und *Worker2* gestartet, welche die Verarbeitungskette durchlaufen. Zusätzlich wird diese aus dem Haupt-Thread heraus aufgerufen. Die Ausgabe des Codes zeigt interessante Aspekte auf:

- *Worker1* startet in Thread 11, *Worker2* startet in Thread 12, der *Main*-Aufruf ist im Haupt-Thread 1.
- Die Methode *logic* kehrt zurück und gibt *fertig in Thread [11/12/1]* aus, bevor der *Where*-Operator und *Subscribe* aufgerufen wurden.
- Nach rund fünf Sekunden werden der *Where*-Operator und die Subscription ausgeführt und zwar alle im Thread 10.

Hier wird sehr deutlich, dass man sich nicht darauf verlassen kann, dass eine Operation in einem speziellen Thread ausgeführt wird. *Delay* lagert die Fortsetzung der Ausführung in einen separaten Thread (hier 10) aus. Nachfolgende Operatoren werden „pragmatisch“ im selben Thread 10 ausgeführt.

Gehen wir noch einen Schritt tiefer. Wie funktioniert das Auslagern von Thread 1, 11 und 12 in den Thread 10, und wer hat Thread 10 erzeugt und verwaltet diesen? Die Antwort liegt im Zauberwort „Scheduler“.

Scheduler

Scheduler erzeugen einen Kontext für die Ausführung von Observables und entkoppeln die Aufrufe zwischen Operatoren. Zum Kontext gehören unter anderem der Thread der Ausführung sowie zeitliche Konfigurationen. Scheduler gehen zwei wesentlichen Aufgaben nach:

- Sie entscheiden, wann eine bestimmte Aufgabe erledigt wird, und bestimmen folglich die Reihenfolge mit.
- Sie bringen zeitliche Aspekte ein, wie wir es bei *Delay* gesehen haben, um Ausführungen in die Zukunft zu verlagern.

SelectMany

In den nachfolgenden Beispielen werden Scheduler mit dem Operator *SelectMany* demonstriert. *SelectMany* konsumiert ein Observable und erzeugt für jeden Eintrag der Eingabesequenz ein neues Observable-Objekt. Anschließend werden die Events dieser Observables in eine einzige Ausgabe-sequenz „geflattet“. In anderen Programmiersprachen heißt *SelectMany* daher auch *FlatMap*.

SelectMany lässt sich in .NET auch dafür nutzen, um für jeden Eintrag der Sequenz einen asynchronen Task aufzurufen

● Listing 3: Multithreading mit Delay

```
var subscription = subject.Select(name =>
{
    Console.WriteLine(
        $"{name}: Select() in Thread
        {CurrentThread.ManagedThreadId}");
    return name;
})
.Delay(TimeSpan.FromSeconds(5))
.Where(name =>
{
    Console.WriteLine(
        $"{name}: Where() in Thread
        {CurrentThread.ManagedThreadId}");
    return true;
}).Subscribe(name =>
{
    Console.WriteLine(
        $"{name}: Subscribe() in Thread
        {CurrentThread.ManagedThreadId}");
});

void logic(string name)
{
    Console.WriteLine(
        $"{name}: start in Thread
        {CurrentThread.ManagedThreadId}");
    subject.OnNext(name);
    Console.WriteLine(
        $"{name}: fertig in Thread
        {CurrentThread.ManagedThreadId}");
};

new Thread(() => { logic("Worker1"); }).Start();
new Thread(() => { logic("Worker2"); }).Start();
logic("Main  ");

Console.ReadLine();

// Ausgabe
Worker2: start in Thread 12
Worker1: start in Thread 11
Main   : start in Thread 1
Worker1: Select() in Thread 11
Worker2: Select() in Thread 12
Main   : Select() in Thread 1
Worker1: fertig in Thread 11
Worker2: fertig in Thread 12
Main   : fertig in Thread 1

// ...ca. 5 Sekunden später ...
Worker1: Where() in Thread 10
Worker1: Subscribe() in Thread 10
Worker2: Where() in Thread 10
Worker2: Subscribe() in Thread 10
Main   : Where() in Thread 10
Main   : Subscribe() in Thread 10
```

Listing 4: Ausgabe von Koordinaten

```
Observable.Range(0, 4)
    .SelectMany(x =>
    {
        Console.WriteLine(
            $"Creating a new Observable for x={x} in
            Thread {CurrentThread.ManagedThreadId}");
        return Observable.Range(0, 3).Select(
            y => new { x = x, y = y });
    })
    .Select(x => $"({x.x}, {x.y})")
    .Subscribe(x =>
    {
        Console.WriteLine(
            $"({x}) @ THREAD
            {CurrentThread.ManagedThreadId}");
    });
Console.WriteLine("finish");
```

```
// Ausgabe mit seltsamer Reihenfolge
Creating a new Observable for x=0 in Thread 1
(0, 0) @ THREAD 1
Creating a new Observable for x=1 in Thread 1
(0, 1) @ THREAD 1
(1, 0) @ THREAD 1
Creating a new Observable for x=2 in Thread 1
(0, 2) @ THREAD 1
(1, 1) @ THREAD 1
(2, 0) @ THREAD 1
Creating a new Observable for x=3 in Thread 1
(1, 2) @ THREAD 1
(2, 1) @ THREAD 1
(3, 0) @ THREAD 1
(2, 2) @ THREAD 1
(3, 1) @ THREAD 1
(3, 2) @ THREAD 1
finish
```

und die Ergebnisse zusammenzuführen. **Bild 1** zeigt die Funktionsweise von *FlatMap* respektive *SelectMany*. Für jeden Eintrag des Input-Observables (rote Kugel, grüne Kugel, blaue Kugel) wird eine Observable-Sequenz von zwei Rechtecken erzeugt, die anschließend terminiert. Diese Sequenzen werden zu einem einzigen Ausgabe-Observable zusammengefasst. Dadurch entsteht ein Ausgabe-Observable mit insgesamt sechs Elementen (zwei rote Rechtecke, zwei grüne Rechtecke und zwei blaue Rechtecke).

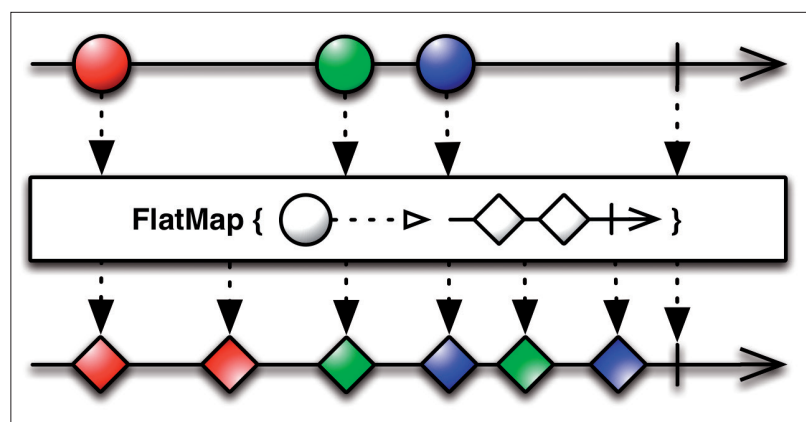
Interessant ist die Reihenfolge der Ausgabe-sequenz. Grüne und blaue Rechtecke sind hier ineinander verschachtelt. Dies ist akzeptabel, da beim Flatten der Sequenzen keine eindeutige Reihenfolge existiert.

Scheduler und SelectMany

Im folgenden Beispiel werden x- und y-Koordinaten verwendet. Die x-Koordinaten reichen von 0 bis 3 und haben vier Einträge, während die y-Koordinaten von 0 bis 2 reichen und drei Einträge haben. Alle Koordinaten sollen in der Form (x, y) in der Konsole ausgegeben werden. *SelectMany* wird genutzt, um für jede x-Koordinate eine Abfolge von y-Koordinaten zu erzeugen.

Die Ausgabe des Codes aus **Listing 4** zeigt alle Koordinaten an, und man sieht, dass alle Daten im Main-Thread 1 ausgegeben werden. Allerdings ist die Reihenfolge der Ausgabe seltsam, denn beispielsweise erscheint die Koordinate (1, 0) vor der Koordinate (0, 2). Das ist bereits ein starker Hinweis darauf, dass *Observable.Range(0, 3)* nicht einfach nur eine naive *for*-Schleife ist, die *OnNext(0)* bis *OnNext(2)* aufruft und mit *OnComplete* abschließt, sondern dass hier Schedu-

ler am Werk sind. Der Standard-Scheduler für *Observable.Range* ist der *CurrentThreadScheduler* (mehr zu den einzelnen Schemulern später). Stellt man den Scheduler von *SelectMany* auf den *ImmediateScheduler* um, bekommt man die Ausgabe, die man von einer naiven *for*-Schleife erwartet hätte (**Listing 5**).



SelectMany oder FlatMap in der Observable Notation (Bild 1)

Dadurch, dass anstelle des implizit verwendeten *CurrentThreadSchedulers* nun der *ImmediateScheduler* zum Einsatz kam, ändert sich das Ausführungsverhalten.

Neben *ImmediateScheduler* und *CurrentThreadScheduler* gibt es noch weitere Scheduler, deren Eigenschaften nachfolgend skizziert werden.

- **ImmediateScheduler:** Der einfachste Scheduler. Er führt die Arbeitspakete unmittelbar im aktuellen Thread aus. Wird der Scheduler mit einem Time-Overload von weniger als ►

● Listing 5: Observable.Range mit ImmediateScheduler

```
Observable.Range(0, 4)
    .SelectMany(x =>
    {
        Console.WriteLine(
            $"Creating a new Observable for x={x} in
            Thread {CurrentThread.ManagedThreadId}");
        return Observable.Range(
            0, 3, ImmediateScheduler.Instance)
            .Select(
                y => new { x = x, y = y });
    })
    .Select(x => $"({x.x}, {x.y})")
    .Subscribe(x =>
    {
        Console.WriteLine(
            $"({x.x}, {x.y}) @ THREAD {CurrentThread.ManagedThreadId}");
    });
Console.WriteLine("finish");
```

```
// Ausgabe
Creating a new Observable for x=0 in Thread 1
(0, 0) @ THREAD 1
(0, 1) @ THREAD 1
(0, 2) @ THREAD 1
Creating a new Observable for x=1 in Thread 1
(1, 0) @ THREAD 1
(1, 1) @ THREAD 1
(1, 2) @ THREAD 1
Creating a new Observable for x=2 in Thread 1
(2, 0) @ THREAD 1
(2, 1) @ THREAD 1
(2, 2) @ THREAD 1
Creating a new Observable for x=3 in Thread 1
(3, 0) @ THREAD 1
(3, 1) @ THREAD 1
(3, 2) @ THREAD 1
finish
```

zehn Sekunden ausgeführt, blockiert er den Thread. Andernfalls wird ein Timer verwendet.

- **DefaultScheduler:** Wird als Standard für alle zeitbasierten Operatoren genutzt (etwa *Delay*, *Observable.ToAsync*) und lagert die Arbeitspakete in einen CLR-ThreadPool-Thread aus, um den Aufrufer-Thread nicht zu blockieren.
- **NewThreadScheduler:** Führt die Arbeitspakete in einem eigenen Thread aus, der nicht Teil des Thread-Pools ist. Dies ist selten eine gute Idee, außer die nachfolgenden Arbeitspakete laufen so lange, dass kein Thread-Pool-Thread blockiert wird.
- **ThreadPoolScheduler:** Ist veraltet und stammt aus einer Zeit, als es in der Universal Windows Platform (UWP) noch keinen Zugriff auf den Thread-Pool gab. Aktuell sollte besser der *DefaultScheduler* verwendet werden, der einen Thread-Pool-Thread nutzt.
- **CurrentThreadScheduler:** Führt die Arbeitspakete im aktuellen Thread aus, ähnlich wie der *ImmediateScheduler*. Allerdings gibt es einen Unterschied bei der Verarbeitung verschachtelter Observables, wie in Listing 4 zu sehen ist. Der *CurrentThreadScheduler* führt die Arbeitspakete nicht unmittelbar aus, sondern fügt sie zunächst in eine Queue ein. Dies erlaubt dem Scheduler, flexibel zwischen verschachtelten Observables zu wechseln. Das erklärt auch die ungewöhnliche Reihenfolge in der Ausgabe.
- **EventLoopScheduler:** Verhält sich wie der *CurrentThreadScheduler*, bietet jedoch die Möglichkeit, dass ein Event-Loop in einem eigens dafür erzeugten Thread startet. Der *EventLoopScheduler* implementiert *IDisposable*, womit der erzeugte Thread beendet werden kann. Er ist dann praktisch, wenn eine genaue Kontrolle darüber gewünscht ist, dass Arbeitspakete in einem spezifischen Thread abgear-

beitet werden. Diese Anforderung tritt beispielsweise auf, wenn mit einem einzigen, wohldefinierten Long-Running-Thread auf ein sequenzielles API, wie beispielsweise Peripheriegeräte, zugegriffen werden soll.

- **SynchronizationContextScheduler:** Greift automatisch auf den UI-Thread bei verbreiteten .NET UI-Frameworks zu, zum Beispiel auf den *Dispatcher* bei WPF-Anwendungen oder den *ControlScheduler* bei Windows-Forms-Anwendungen. Der Nachteil bei der Verwendung von *SynchronizationContextScheduler* ist, dass der Zugriff auf den Scheduler im UI-Thread erfolgen muss. Dies kann mit den nachfolgend vorgestellten, spezialisierten UI-Framework-Schedulern umgangen werden.
- **UI Framework Scheduler (ControlScheduler, DispatcherScheduler und CoreDispatcherScheduler):** Diese spezialisierten Scheduler verhalten sich im Grunde genauso wie der *SynchronizationContextScheduler*, jedoch muss der Zugriff auf diese Scheduler nicht im UI-Thread erfolgen. Sowohl die spezialisierten UI-Framework-Scheduler als auch der *SynchronizationContextScheduler* sind immer dann zu empfehlen, wenn auf UI-Elemente zugegriffen werden soll, ohne dass man sich jedes Mal Gedanken darüber machen möchte, ob sich der Zugriff im UI-Thread befindet oder nicht.

Testen reaktiver Anwendungen

In Listing 6 wird zweimal das Gleiche getestet: Mit *Observable.Interval* wird jede Sekunde ein aufsteigender Wert produziert *[0, 1, 2, 3, ...]*.

Der Sinn des Unit-Tests besteht darin zu prüfen, ob die Verarbeitungskette nach einer gewissen Zeit eine *InvalidOperationException* produziert oder nicht. Dies geschieht beim Wert 3, also nach rund vier Sekunden.

SlowTest() geht naiv vor. Dieser Test wartet zuerst 3,5 Sekunden und prüft, ob die *exceptionThrown*-Variable *null* ist, also keine Exception vorkam. Nach weiteren zwei Sekunden Wartezeit wird die Prüfung wiederholt. Der Unit-Test verbraucht also mindestens 5,5 Sekunden!

Diese Art der Vorgehensweise ist problematisch. Zum einen, weil sich bei steigender Anzahl an Tests die Zeit fürs Testen der Anwendung enorm vergrößert, und zum anderen, weil das Ganze wenig deterministisch ist. Denn *Thread.Sleep* wartet zwar mindestens die vorgegebene Zeit, aber es garantiert nicht, dass es sofort weitergeht.

Rx.NET bietet mit dem *TestScheduler*, dem *HistoricalScheduler* sowie dem Konzept der virtuellen Zeit eine clevere Lösung an: Der *TestScheduler* ist ein spezieller Scheduler, der extern gestartet (*Start*), gestoppt (*Stop*), pausiert (*Sleep*) und zudem für „Zeitreisen in die Zukunft“ genutzt werden kann. Mit *AdvanceBy* wird die Zeit um eine gewisse Anzahl Ticks

vorgestellt, und mithilfe von *AdvanceTo* wird ein bestimmter Zeitpunkt angegeben.

Der *HistoricalScheduler* wird zusätzlich mit einem beliebig festgelegten Zeitpunkt initialisiert.

Der *FastTest* aus Listing 6 nutzt diese Möglichkeiten. Er verwendet den *TestScheduler* und initialisiert damit das *Observable.Interval*. Dadurch ist es später möglich, die virtuelle Zeit mit *AdvanceBy* zu erhöhen, anstatt *Thread.Sleep* zu verwenden und unnötig lange zu warten.

Wird die Zeit mit *AdvanceBy* in die Zukunft verlegt, werden alle Werte von *Observable.Interval* unmittelbar ausgegeben. Es muss also nicht erst physisch Zeit vergehen. Zusätzlich kann die Zeit präzise festgelegt werden, wodurch kein nicht-deterministisches Zeitverhalten mehr befürchtet werden muss.

Der zweite Unit-Test ist binnen wenigen Millisekunden abgeschlossen und erzeugt genau das zu erwartende Ergebnis. ►

Listing 6: Naiver Test versus TestScheduler

```
using System.Reactive.Linq;
using Microsoft.Reactive.Testing;
// befindet sich im gleichnamigen NuGet-Paket.

namespace Test;
public class OneFastAndOneSlowTestTests
{
    [Fact]
    public void SlowTest()
    {
        Exception? exceptionThrown = null;
        Observable
            .Interval(TimeSpan.FromSeconds(1))
            .Select(x =>
            {
                if (x == 3)
                {
                    throw new InvalidOperationException(
                        "this operation is invalid");
                }
                return x;
            })
            .Subscribe(onNext: x => { },
                onError: (exception) => {
                    exceptionThrown = exception; },
                onCompleted: () =>
                    exceptionThrown = null);

        Thread.Sleep(3500);
        Assert.Null(exceptionThrown);

        Thread.Sleep(2000);
        Assert.IsType<InvalidOperationException>(
            exceptionThrown);
    }
}
```

```
[Fact]
public void FastTest()
{
    Exception? exceptionThrown = null;
    TestScheduler testScheduler =
        new TestScheduler();

    Observable
        .Interval(TimeSpan.FromSeconds(1),
            testScheduler)
        .Select(x =>
        {
            if (x == 3)
            {
                throw new InvalidOperationException(
                    "this operation is invalid");
            }
            return x;
        })
        .Subscribe(onNext: x => { },
            onError: (exception) => {
                exceptionThrown = exception; },
            onCompleted: () => exceptionThrown = null);

    testScheduler.AdvanceBy(
        TimeSpan.FromSeconds(3.5).Ticks);
    Assert.Null(exceptionThrown);

    testScheduler.AdvanceBy(
        TimeSpan.FromSeconds(100).Ticks);
    Assert.IsType<InvalidOperationException>(
        exceptionThrown);
}
```

nis. *TestScheduler* und *HistoricalScheduler* sind somit ideal für das Testen von Time-outs, Delays, Intervallen und ähnlichen Szenarien.

Zusammengefasst: Unterschiedliche Scheduler in Rx.NET bieten spezialisierte Eigenschaften, die bestimmen, in welchem Thread und in welcher Reihenfolge Code-Pakete bearbeitet werden. Das Konzept der virtuellen Zeit ermöglicht es, das Zeitverhalten zu simulieren, ohne dass die tatsächliche Zeit verstreichen muss. Dadurch können Tests effizienter und präziser durchgeführt werden.

SubscribeOn and ObserveOn

Die Erweiterungsmethoden *SubscribeOn* und *ObserveOn* für *IObservable* sind trickreich, und nicht wenige Entwickler stolpern hier. Aber im Grunde tun die Methoden exakt das, wonach sie benannt sind.

```
public static IObservable<TSource> SubscribeOn<TSource>(
    this IObservable<TSource> source,
    IScheduler scheduler)
```

SubscribeOn ist ein selten verwendeter Operator, der festlegt, welcher Scheduler genutzt wird, während ein Observable abonniert wird. An dieser Stelle ist es wichtig zu verstehen, dass *SubscribeOn* ausschließlich während des Subscribe-Vorgangs relevant ist.

Warum sollte der Scheduler für das Subscribe bestimmt werden? Dies ist nur dann von Bedeutung, wenn das Observable als Standard den *ImmediateScheduler* oder *CurrentThreadScheduler* verwendet. Mit *SubscribeOn* wird die Subscription unter Anwendung des angegebenen Schedulers erzeugt. Nutzt ein Operator den *ImmediateScheduler* oder *CurrentThreadScheduler*, wird implizit auch der Thread des angegebenen Schedulers verwendet.

Verwendet das Observable dagegen einen *NewThreadScheduler*, dann ist es überflüssig zu bestimmen, mit welchem Scheduler beziehungsweise welchem Thread der Subscription-Vorgang stattfinden soll, da in jedem Fall ein neuer Thread genutzt wird.

```
public static IObservable<TSource> ObserveOn<TSource>(
    this IObservable<TSource> source,
    IScheduler scheduler)
```

Mit *ObserveOn* kann festgelegt werden, mit welchem Scheduler *OnNext*, *OnError* und *OnComplete* verarbeitet wird. So kann man beispielsweise mehrere Observables auf einen *EventLoopScheduler* umleiten oder im Fall von UI-Anwendungen den jeweiligen UI-Framework-Scheduler verwenden. In WPF-Anwendungen sieht ein typisches Pattern mit *SubscribeOn* und *ObserveOn* wie folgt aus:

```
Observable
    .Defer(FetchAndProcessArticles)
    .SubscribeOn(TaskPoolScheduler.Default)
    .ObserveOn(DispatcherScheduler.Instance)
    .Subscribe(AddArticles);
```

Zunächst wird durch *Defer* ein Observable zum Zeitpunkt der Subscription registriert und die Logik getriggert. *SubscribeOn* stellt sicher, dass die Initialisierungslogik nicht im UI-Thread ausgeführt wird – selbst dann, wenn *.Subscribe* im UI-Thread aufgerufen wird – und die Operatoren beispielsweise den *ImmediateScheduler* oder den *CurrentThreadScheduler* verwenden. *ObserveOn* sorgt dafür, dass die Ergebnisse in jedem Fall per *Subscribe* wieder im UI-Thread zur Verfügung gestellt werden, sodass in der *AddArticles*-Methode sicher auf UI-Elemente zugegriffen werden kann.

Fazit

Rx.NET kann die Produktivität erheblich steigern. Allerdings ist es in der .NET-Community eine Nischentechnologie. Soll Rx.NET in einem .NET-Projekt genutzt werden, muss das Team vorab entsprechend geschult werden. Folgende Aspekte dürfen dabei nicht fehlen:

- **Was ist Rx.NET**, und wo liegen die Stärken gegenüber .NET-Events?
- **Nomenklatur**: Subject, Operator, Observable, Subscription.
- **Eine Übersicht und Hilfestellung** bei der Suche der richtigen Operatoren.
- **Die verschiedenen Arten von Subjects** (BehaviorSubject, Subject).
- **Grundlegendes Wissen** über Scheduler und Multi-Threading mit den UI-Schedulern, die Auswirkungen in ASP.NET-Anwendungen mit parallelen Controllern und das Serialisieren mithilfe des *EventLoopScheduler*.

Diese Artikelreihe entstand nach Abschluss eines Projekts, in dem Rx.NET eingeführt wurde, um den Datenaustausch in einer Layer-Architektur von niedrigeren zu höheren Layern zu ermöglichen. Die drei Teile haben die oben genannten Punkte und das in der Praxis relevanteste Wissen kompakt zusammengefasst. ■

- [1] Tim Borowski, *Einführung in Rx.NET, Reaktiv programmieren, Teil 1*, dotnetpro 10/2024, Seite 51 ff., www.dotnetpro.de/A2410RxNET
- [2] Tim Borowski, *LINQ für Events, Reaktiv programmieren, Teil 2*, dotnetpro 11/2024, Seite 71 ff., www.dotnetpro.de/A2411RxNET
- [3] Rx.NET, *Scheduling and Threading*, www.dotnetpro.de/SL2412RxNET1
- [4] *SelectMany/FlatMap in RX*, www.dotnetpro.de/SL2412RxNET2



Tim Borowski

ist freiberuflicher Softwareentwickler und -berater. Seine Schwerpunkte sind die .NET/C#-Entwicklung, Angular sowie AWS, K8s und Istio Service Mesh. Zudem ist er Fachartikelautor und Konferenzsprecher. Sie erreichen ihn unter tim@borowski-software.de.

dnpCode

A2412RxNET

DEV ACADEMY

HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN



REMOTE
ODER
INHOUSE

Hands-on Workshops Azure

Einführung in Azure OpenAI - Von der Datenintegration bis zur Entwicklung eigener KI-Agenten (3 Tage)

Cloud-native basierte Entwicklung mit Azure-Diensten (2 Tage)

Message-basierte Azure-Dienste (2 Tage)

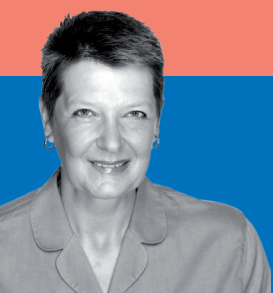
Azure: Architektur und Dienstausswahl (2 Tage)

Verteilte Anwendungen bei Azure mit Docker und Kubernetes (3 Tage)

Azure Pipelines - Multistage YAML Pipelines für Kubernetes (3 Tage)

Azure DevOps - CI/CD mit Azure Pipelines und Git (3 Tage)

Microsoft Azure DB (2 Tage)



Weitere Informationen auf developer-academy.de
Preise auf Anfrage!

Ihre Ansprechpartnerin: Susanne Herl • +49 731 88005-8835 • susanne.herl@ebnermedia.de

MESSAGING MIT WOLVERINE UND .NET, TEIL 6

Fehlersuche

Bisher lief mit Wolverine alles glatt. Doch was tun, wenn die Anwendung bockt?

Wolverine präsentiert sich im praktischen Einsatz als eine sehr angenehm zu verwendende Bibliothek. Dennoch können sich aufgrund des Umfangs und der vielen Konfigurations- und Erweiterungsmöglichkeiten Probleme bei der Entwicklung einschleichen.

Deshalb werden wir im Folgenden einige Möglichkeiten aufzeigen, wie man etwaige Fehler in einer Wolverine-Anwendung identifizieren und somit auch beheben kann.

Lesen, prüfen, fragen ...

Treten Probleme mit Wolverine auf, empfiehlt es sich natürlich, zuerst die Dokumentation unter [1] zu konsultieren. Hilft dies nicht weiter, gibt es kostenlosen Support in den GitHub-Issues zum Projekt [2] oder in einem Discord-Chat [3]. Die Präferenz für den Erstkontakt seitens des Entwicklerteams von Wolverine liegt klar auf Discord. GitHub-Issues werden in der Regel erstellt, wenn sich in einem Chat herausstellt, dass es sich tatsächlich um einen Bug, ein fehlendes Feature in Wolverine oder schlicht fehlende oder missverständliche Dokumentation handelt. Die in diesem Artikel vorgestellten Methoden zur Selbsthilfe unterstützen dabei, entweder das Problem direkt selbst zu lösen oder zumindest mit qualifizierten Informationen eine Diskussion im Discord-Chat zu starten. Diese Informationen bei Beginn der Unterhaltung parat zu haben und zur Verfügung zu stellen verkürzt das übliche Frage- und Antwort-Prozedere immens und macht die Fehlersuche für alle Beteiligten einfacher. Dies wiederum erhöht die Chance, dass sich jemand des Problems annimmt.

... und mitmachen

Bei Wolverine handelt es sich um ein Open-Source-Projekt. Zielgruppe des Projekts sind Entwickler von Anwendungen, die eine gewisse Größe und Komplexität erreichen. Außer zu Lern- oder Schulungszwecken wird praktisch niemand eine Anwendung mit Wolverine „just for fun“ entwickeln. Anwendungen dieser Größe und Komplexität schaffen Nutzen, mit dem Geld verdient wird.

Identifiziert man, wie oben geschildert, fehlende oder missverständliche Dokumentation, sollte man sich die Zeit nehmen, einen Pull Request mit den notwendigen Änderungen zu erstellen. Damit wird Wolverine stetig verbessert und alle profitieren. Der Aufwand für den Einzelnen bleibt dennoch überschaubar, da die Last auf viele Schultern verteilt wird.

```
dotnet run
Successfully found Ponger.PingHandler during scanning.
Methods:

Method: Handle(Ping, ILogger<PingHandler>, IMessageContext)
HIT -- Method name is 'Handle' (case sensitive)
MISS -- Method name is 'HandleAsync' (case sensitive)
MISS -- Method name is 'Handles' (case sensitive)
MISS -- Method name is 'HandlesAsync' (case sensitive)
MISS -- Method name is 'Consume' (case sensitive)
MISS -- Method name is 'ConsumeAsync' (case sensitive)
MISS -- Method name is 'Consumes' (case sensitive)
MISS -- Method name is 'ConsumesAsync' (case sensitive)
MISS -- Method name is 'Orchestrate' (case sensitive)
MISS -- Method name is 'OrchestrateAsync' (case sensitive)
MISS -- Method name is 'Orchestrates' (case sensitive)
MISS -- Method name is 'OrchestratesAsync' (case sensitive)
MISS -- Method name is 'Start' (case sensitive)
MISS -- Method name is 'StartAsync' (case sensitive)
MISS -- Method name is 'Starts' (case sensitive)
MISS -- Method name is 'StartsAsync' (case sensitive)
MISS -- Method name is 'StartOrHandle' (case sensitive)
MISS -- Method name is 'StartOrHandleAsync' (case sensitive)
MISS -- Method name is 'StartsOrHandles' (case sensitive)
MISS -- Method name is 'StartsOrHandlesAsync' (case sensitive)
MISS -- Method name is 'NotFound' (case sensitive)
MISS -- Method name is 'NotFoundAsync' (case sensitive)
MISS -- Has attribute [WolverineHandler]
OK -- Method is declared by object
OK -- IDisposable.Dispose()
OK -- IAsyncDisposable.DisposeAsync()
OK -- Contains Generic Parameters
OK -- Special Name
OK -- Has attribute [WolverineIgnore]
OK -- Has no arguments
OK -- Cannot determine a valid message type
OK -- Returns a primitive type
```

Handler Match für die Klasse PingHandler (Bild 1)

Ein Pull Request für die Änderung der Dokumentation dauert je nach eigener Erfahrung mit GitHub nur wenige Minuten, und die Änderungen werden in der Regel schon nach wenigen Tagen – manchmal sogar am gleichen Tag – integriert.

Natürlich sind auch alle aufgerufen, Verbesserungen am Quellcode von Wolverine beizutragen.

Wolverine wird seit mehreren Jahren entwickelt und ist seitdem kostenlos. Ein Projekt dieses Umfangs bindet enorme Kapazitäten.

Wie erwähnt, setzt man eine Bibliothek wie Wolverine nicht ohne Grund ein. Es handelt sich dabei um einen zentralen Baustein einer Anwendung, dessen Austausch mit erheblichem Aufwand – und damit Kosten – verbunden ist.

Deshalb ist es im Interesse aller Beteiligten – Entwickler wie Anwender von Wolverine –, das Projekt langfristig am Leben zu halten. Setzt man Wolverine gewinnbringend ein, sollte man sich daher auch ernsthaft mit dem Gedanken befassen,

sen, finanzielle Unterstützung in Form von GitHub-Sponsoring zu leisten [4]. Damit hat das Team die Chance, weitere Kapazitäten für Entwicklung und Support aufzubauen.

Betreibt man Wolverine produktiv in einem professionellen Umfeld, empfiehlt sich natürlich wie bei jedem Produkt dieser Kategorie der Abschluss eines Support-Vertrags [5] mit klar geregelten Bedingungen und Ansprüchen.

Hilfe zur Selbsthilfe

Nach diesem kurzen Exkurs in den nachhaltigen Umgang mit Open-Source-Bibliotheken sowie Community- und kommerziellem Support soll nun die Fehlersuche im Code im Vordergrund stehen.

Hat man sich mit der Dokumentation auseinandergesetzt, und funktioniert die Anwendung dennoch nicht wie gewünscht, gibt es mehrere generelle Schritte, die man durchführen kann, um gängige Fehlerquellen auszuschließen.

Auf der Suche nach dem Handler

Ein Problem, das immer wieder auftritt, ist, dass Message-Handler wider Erwarten von Wolverine nicht ausgeführt werden. Wenn Handler nicht ausgeführt werden, liegt dies im Normalfall daran, dass die sogenannte Handler Discovery diese nicht finden kann beziehungsweise gefundene Handler ausschließt.

Die Handler Discovery findet Handler beim Start der Applikation, wenn sie folgenden zwingenden Regeln entsprechen:

- Der Handler ist als konkrete Klasse implementiert.
- Die Klasse ist als *public* deklariert.

Die Handler Discovery findet Handler, die einer der folgenden Konventionen entsprechen:

- Die Klasse implementiert das Marker-Interface *Wolverine.IWolverineHandler*.
- Die Klasse wurde mit dem *[Wolverine.WolverineHandler]*-Attribut dekoriert.
- Der Name der Klasse endet mit *Handler* oder *Consumer*.

Die Methoden zur Behandlung von Nachrichten müssen folgenden Regeln entsprechen:

- Die Methode ist eine Instanz-Methode.
- Die Methode ist als *public* deklariert.
- Der erste Parameter der Methode ist die eingehende Nachricht.

Die Methoden der Handler-Klasse müssen einer der folgenden Konvention entsprechen:

- Der Name lautet *Handle* oder *HandleAsync*.
- Der Name lautet *Handles* oder *HandlesAsync*.
- Der Name lautet *Consume* oder *ConsumeAsync*.
- Der Name lautet *Consumes* oder *ConsumesAsync*.
- Die Methode ist mit dem *[WolverineHandler]* dekoriert.

Bevor man sich auf die Suche nach der Verletzung der genannten Konventionen macht, kann man mit Wolverine-Bordmitteln prüfen, ob die Handler Discovery beim Start der Anwendung einen Handler findet. Wir verwenden dazu wieder unser Ping-Pong-Beispiel aus Teil 1 dieser Serie [6]. Der Einfachheit halber sind sowohl Ping- als auch Pong-Handler sowie die Nachrichten-Typen im gleichen Projekt angesiedelt.

Die Suche der Handler Discovery nach einem Handler wird als Handler Match bezeichnet. Wolverine bietet die Möglichkeit, Handler Matches beim Start zu beschreiben. Der Code hierfür sieht wie folgt aus:

```
var host = Host.CreateDefaultBuilder(args)
    .UseWolverine(
        options =>
        {
            options.ListenAtPort(5580);
            options
                .PublishMessage<Ping>()
                .ToPort(5581);

            options.Services.AddHostedService<Worker>();

            options.ListenAtPort(5581);

            Console.WriteLine(options.DescribeHandlerMatch(
                typeof(PingHandler)));
```

```
Successfully found HandlerDiscoveryDemo.SumHandler during scanning.
Methods:
Method: Handle(Addition)
HIT -- Method name is 'Handle' (case sensitive)
MISS -- Method name is 'HandleAsync' (case sensitive)
MISS -- Method name is 'Handles' (case sensitive)
MISS -- Method name is 'HandlesAsync' (case sensitive)
MISS -- Method name is 'Consume' (case sensitive)
MISS -- Method name is 'ConsumeAsync' (case sensitive)
MISS -- Method name is 'Consumes' (case sensitive)
MISS -- Method name is 'ConsumesAsync' (case sensitive)
MISS -- Method name is 'Orchestrate' (case sensitive)
MISS -- Method name is 'OrchestrateAsync' (case sensitive)
MISS -- Method name is 'Orchestrates' (case sensitive)
MISS -- Method name is 'OrchestratesAsync' (case sensitive)
MISS -- Method name is 'Start' (case sensitive)
MISS -- Method name is 'StartAsync' (case sensitive)
MISS -- Method name is 'Starts' (case sensitive)
MISS -- Method name is 'StartsAsync' (case sensitive)
MISS -- Method name is 'StartOrHandle' (case sensitive)
MISS -- Method name is 'StartOrHandleAsync' (case sensitive)
MISS -- Method name is 'StartsOrHandles' (case sensitive)
MISS -- Method name is 'StartsOrHandlesAsync' (case sensitive)
MISS -- Method name is 'NotFound' (case sensitive)
MISS -- Method name is 'NotFoundAsync' (case sensitive)
MISS -- Has attribute [WolverineHandler]
OK -- Method is declared by object
OK -- IDisposable.Dispose()
OK -- IAsyncDisposable.DisposeAsync()
OK -- Contains Generic Parameters
OK -- Special Name
OK -- Has attribute [WolverineIgnore]
OK -- Has no arguments
OK -- Cannot determine a valid message type
EXCLUDED -- Returns a primitive type
```

Handler Match für eine invalide Handler-Methode (Bild 2)

Message Name	Message Type namespace	Handler.Method() namespace	Generated Type Name
PingPongMessages.Ping	Ping PingPongMessages	PingHandler.Handle(ping, logger, context) Ponger	PingHandler850181254
PingPongMessages.Pong	Pong PingPongMessages	PongHandler.Handle(pong, logger) Pinger	PongHandler850181196

Aktivierte Handler
der Applikation
(Bild 3)

```

        Console.WriteLine(options.DescribeHandlerMatch(
            typeof(PongHandler)));
    }
}
)
.Build();
await host
.RunAsync();

```

Relevant sind hier die Aufrufe von *options.DescribeHandlerMatch*.

Die Ausgabe für den Handler Match von *PingHandler* ist in **Bild 1** dargestellt. Im ersten Schritt gibt der Handler Match die Information aus, ob er den Handler finden konnte. Danach wird ausführlich beschrieben, welche der Methoden-Konventionen erfüllt beziehungsweise nicht erfüllt wurden – die Konventionen, die im Bild nach *ConsumesAsync* folgen, waren in der bisherigen Liste der Konventionen bewusst nicht aufgeführt; diese werden für uns erst in einem späteren Artikel der Serie relevant. Nach dem letzten *MISS*-Eintrag für die Namenskonventionen der Methode folgt die Beschreibung der Methoden-Matches für die übrigen Konventionen der Methoden. Lauten alle diese Einträge *OK*, wird die Methode zur Laufzeit ausgeführt.

Wie bereits in Teil 4 dieser Serie [7] beschrieben, sollte ein Handler nie primitive Typen wie *int* oder *string* zurückgeben, damit der Rückgabewert immer serialisierbar ist. Diese Konvention würde mit dem nachfolgenden *SumHandler* verletzt werden:

```

public record Addition(int A, int B);

public class SumHandler

```

```

{
    public int Handle(
        Addition addition
    ) =>
        addition.A + addition.B;
}

```

Der Handler Match für diesen Handler sähe dann wie **Bild 2** dargestellt aus.

Auch hier wird der Handler zunächst als „erfolgreich gefunden“ deklariert. Allerdings wurde die letzte Konvention verletzt und die gefundene Methode *Handle* als *EXCLUDED* markiert. Damit wird sie zur Laufzeit ignoriert, und Nachrichten vom Typ *Addition* würden von *SumHandler* nicht behandelt werden.

Die Ausgabe des Handler Matches muss also vollständig gelesen werden, da die Angaben der ersten Zeile nur einen Teil der Wahrheit beinhalten und das Finden des Handlers nicht gleichbedeutend ist mit dessen Ausführung zur Laufzeit.

Infos auf Kommando

Um die Match-Informationen für die drei Handler zu erhalten, mussten wir jeweils einen *DescribeHandlerMatch*-Aufruf durchführen. Mit einer steigenden Anzahl an Handlern steigt also auch der Aufwand, um die Handler zu beschreiben – außerdem wird der Code für die Konfiguration von Wolverine unübersichtlicher.

Unter anderem aus diesem Grund stellt Wolverine eine weitere Möglichkeit bereit, um Auskünfte über die gestartete Anwendung zu erhalten.

Startet man den Host oder die Anwendung statt mit *host.RunAsync()* mit der Methode *await host.RunOaktonCommands(args)*, wird die Kommandozeilen-Unterstützung der Oakton-Bibliothek in der Anwendung aktiviert. Oakton ist eine Open-Source-Bibliothek, die Kommandozeilen-Tools für .NET-Anwendungen bereitstellt und ebenfalls von den Machern von Wolverine stammt.

Nachdem die Oakton-Kommandozeilen-Unterstützung über den gezeigten Aufruf aktiviert wurde, können beim Start der Anwendung verschiedene Kommandos für Wolverine ausgeführt werden.

Gibt man keines der nachfolgend beschriebenen Kommandos an, startet die Anwendung wie mit dem bisherigen *host.RunAsync()*-Aufruf. Startet man dagegen die Anwendung mit *dotnet run -- describe*, erhält

```

— Wolverine Rabbit MQ Usage —
HostName      localhost
Port          -1
VirtualHost    /
AutoProvision  True
AutoPurgeAllQueues False

— Queues —

```

QueueName	AutoDelete	IsDurable	IsExclusive	Arguments
pings	False	True	False	x-message-ttl = 15000

```

— Exchanges —

```

Name	Type	AutoDelete	IsDurable	Arguments	Bindings
pings	Fanout	False	True	-	To queue(s) pings

RabbitMQ-Konfiguration mittels *describe* (Bild 4)

man die Handler Matches für alle Handler, die gefunden wurden und nicht ausgeschlossen sind. Die Darstellung erfolgt in tabellarischer Form, wie in **Bild 3** zu sehen ist.

Die Darstellung ist etwas kompakter als der explizite *DescribeHandlerMatch*-Aufruf, das heißt, es fehlen die Informationen bezüglich der Einhaltung der Regeln. Möchte man den Grund für den Ausschluss von *SumHandler* finden, muss man also danach gezielt dessen Handler Match ausführen.

Zusätzlich zu den Handler Matches gibt das *describe*-Kommando noch etliche weitere Informationen aus, deren Screenshots allerdings den Rahmen dieses Artikels sprengen würden.

Folgende Informationen werden von *describe* ergänzend zu den Handler Matches generiert:

- Wolverine-Optionen,
- Handler-Discovery-Regeln,
- Wolverine-Listener,
- Message-Routing-Informationen,
- Sending-Endpoints,
- Error-Handling-Regeln,
- Informationen zur Assembly der Anwendung,
- Informationen zur *IConfiguration*, die der Anwendung zur Verfügung steht, sowie
- Informationen zu referenzierten Assemblies.

Darüber hinaus werden Informationen der jeweiligen Wolverine-Erweiterungen dargestellt, zum Beispiel wenn RabbitMQ als Message-Broker verwendet wird.

Starten wir unseren Pong-Service aus Teil 3 dieser Serie [8], in dem wir uns mit Wolverine und RabbitMQ befassen haben, mit dem *describe*-Kommando, erhalten wir beispielsweise die Informationen zur RabbitMQ-Konfiguration, wie in **Bild 4** dargestellt.

Damit sind die Möglichkeiten der Oakton-Kommandos aber noch nicht erschöpft. Wie wir aus Teil 1 dieser Serie [6] wissen, werden die Handler per Code-Generierung erzeugt. Vermuten wir einen Fehler in dieser Generierung, können wir zunächst die Handler erzeugen lassen, indem wir das Kommando *dotnet run -- codegen write* ausführen. Die generierten C#-Dateien befinden sich dann im Ordner *Internal/Generated/WolverineHandlers*.

Alternativ kann der generierte Quellcode auch nur im Terminal ausgeben werden. Hierzu dient das Kommando *dotnet run -- codegen preview*.

Schließlich kann mit dem Kommando *dotnet run -- check-env* die Laufzeitumgebung geprüft werden. Die Prüfung beinhaltet folgende Tests:

- Kann die Anwendung auf die Datenbank (mehr dazu in einem späteren Teil der Serie), die für Wolverine konfiguriert wurde, zugreifen?
- Kann die Anwendung auf Message-Broker wie RabbitMQ oder AWS SQS zugreifen?
- Kann auf die IoC-Container-Registrierungen zugegriffen werden?

Wenn die Checks erfolgreich sind, schaut die Ausgabe wie in **Bild 5** aus.

```
dotnet run -- check-env
Searching 'JasperFx.CodeGeneration.Commands,
Version=0.0.0.0, Culture=neutral, PublicKeyToken=null'
for commands
Searching 'Wolverine, Version=2.17.2.0, Culture=neutral,
PublicKeyToken=null' for commands

Oakton

Info: Wolverine.Runtime.WolverineRuntime[0]
Exporting Open Telemetry metrics from Wolverine with
name Wolverine:HandlerDiscoveryDemo, version 2.17.2.0
1.) Success: Resource Envelope Storage (Wolverine)

Running Environment Checks 100%
[green]All environment checks are good![/]
```

Die Prüfung der Laufzeitumgebung war erfolgreich (**Bild 5**)

Fazit

Ähnlich wie bei der Unterstützung für Unit- und Integrations-tests steht uns Wolverine auch bei der Fehlersuche zur Seite. Damit bleibt der Frust-Level bei den ersten Schritten mit Wolverine niedrig.

Sollte die Behebung der Fehler nicht mit den Bordmitteln möglich sein, steht auch die Wolverine-Community für Fragen zur Verfügung.

Derart ausgerüstet, können wir uns im nächsten Teil der Integration von Wolverine mit Marten (bekannt aus einer vorangegangenen Serie) widmen. ■

[1] Wolverine-Dokumentation, <https://wolverinefx.net>

[2] GitHub-Issues zum Wolverine-Projekt, www.dotnetpro.de/SL2412Wolverine1

[3] Discord-Chat zu Wolverine, www.dotnetpro.de/SL2412Wolverine2

[4] GitHub-Sponsoring für Wolverine, www.dotnetpro.de/SL2412Wolverine3

[5] Support-Vertrag zu Wolverine, <https://jasperfx.net/support-plans/>

[6] Alexander Zeitler, Einsteigen, bitte!, dotnetpro 7/2024, Seite 66 ff., www.dotnetpro.de/A2407Wolverine

[7] Alexander Zeitler, Red/Green mit Wolverine, dotnetpro 10/2024, Seite 66 ff., www.dotnetpro.de/A2410Wolverine

[8] Alexander Zeitler, Hattu Nachricht?, dotnetpro 9/2024, Seite 82 ff., www.dotnetpro.de/A2409Wolverine



Alexander Zeitler

entwickelt als Cloud Solutions Architect B2B-Plattformen im industriellen Umfeld mit Schwerpunkt AWS, .NET und Node.js. Er unterstützt Unternehmen mit Architekturberatung und Entwicklung für Event-basierte Systeme.

alexander.zeitler@pdmilab.com

dnpCode

A2412Wolverine

WEBANWENDUNGEN MIT CODEBEHIND

Der Code im Hintergrund

Wenig Code nötig und dank mehrerer Pattern flexibel einsetzbar: Das sind die Vorzüge des Web-Backend-Frameworks CodeBehind.

Die Wahl des richtigen Backend-Frameworks spielt bei der Entwicklung einer modernen Webanwendung eine entscheidende Rolle: Es muss nicht nur effizient und flexibel sein, sondern auch eine einfache Integration neuer Technologien ermöglichen.

Eine der größten Herausforderungen in diesem Bereich ist die Balance zwischen Leistung und Entwicklerfreundlichkeit. Viele Frameworks bieten entweder hohe Leistung auf Kosten der Benutzerfreundlichkeit oder umgekehrt. Zudem müssen Entwickler oft zwischen verschiedenen Architekturen wie MVC, MVVM oder Microservices wählen, was die Komplexität erhöht und die Lernkurve steiler macht.

Was ist das CodeBehind-Framework?

CodeBehind ist ein Backend-Framework für ASP.NET Core, das von Elanat [1] geschaffen wurde. Es versteht sich als moderne Alternative zu den etablierten Microsoft-Frameworks wie Razor Pages, MVC und Blazor, die auf der .NET-Core-Plattform aufbauen. Bei Elanat handelt es sich um ein Team von Softwareentwicklern, die sich auf die Verbesserung und Optimierung von Webtechnologien im .NET-Ökosystem spezialisiert haben.

Der Hauptzweck von CodeBehind ist es, eine effizientere und flexiblere Alternative zu den bestehenden ASP.NET-Core-Frameworks zu bieten.

● Listing 1: Eine Standard-Konfiguration von CodeBehind in einem ASP.NET-Core-Projekt

```
using CodeBehind;
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.Hosting;

var builder = WebApplication.CreateBuilder(args);
var app = builder.Build();

SetCodeBehind.CodeBehindCompiler.Initialization();

if (app.Environment.IsDevelopment())
{
    app.UseDeveloperExceptionPage();
}
else
{
    app.UseExceptionHandler("/Error");
    app.UseHsts();
}

app.UseHttpsRedirection();
app.UseStaticFiles();
app.UseRouting();

app.UseCodeBehind();

app.Run();
```

Eine weitere Schwierigkeit ist die Skalierbarkeit. Mit dem Wachstum von Anwendungen steigt oft auch die Komplexität des Codes, was die Wartung erschwert sowie die Entwicklungsgeschwindigkeit verlangsamt. Darüber hinaus stellen Sicherheitsaspekte und die nahtlose Integration von Frontend-Technologien eine weitere Hürde dar, die es zu überwinden gilt. In dieser Episode der Kolumne „Frameworks und mehr“ liegt der Fokus auf dem Framework CodeBehind.

Das Projekt möchte die Entwicklung von Webanwendungen verbessern, indem es als Backend-Framework unterschiedliche Architekturstile und eine ganze Palette von Features mitbringt – unter anderem hohe Modularität, Integrationsmöglichkeiten und Sicherheitsfunktionen.

Außerdem zielt es darauf ab, eine intuitivere Struktur für die Trennung von Präsentationslogik und Geschäftslogik zu schaffen und dabei die Vorteile der modernen .NET-Core-Plattform auszunutzen.

Das Framework ist recht jung. Die erste stabile Version wurde im Jahr 2023 veröffentlicht, die erste öffentliche Version des Frameworks mit der Version 1.9.2 datiert vom 10. Dezember 2023 [2].

Seitdem hat CodeBehind mehrere Aktualisierungen erfahren, wobei die Version 2.5 einen bedeutenden Meilenstein in der Entwicklung des Frameworks darstellt.

Diese Version führte wesentliche Verbesserungen in Bezug auf die Konfiguration und Integration in ASP.NET-Core-Pro-

jekte ein und hat einen deutlichen Geschwindigkeitszuwachs erfahren.

Was macht das Framework aus?

Der Hauptunterschied zwischen CodeBehind und den Microsoft-Frameworks wie ASP.NET Core MVC und Razor Pages liegt in der Flexibilität und Einfachheit der Entwicklung. So bietet CodeBehind eine View-zentrierte Architektur. Anders als bei Microsoft-Frameworks, wo der Controller oft den Einstiegspunkt bildet, beginnt in CodeBehind der Anfragezyklus bei der View. Dies ermöglicht eine intuitivere Strukturierung des Codes.

CodeBehind unterstützt verschiedene Entwicklungsmuster wie MVC, Model-View, nur View und sogar einen Web-Forms-ähnlichen Ansatz.

Diese Flexibilität erlaubt es, den für ein Projekt geeigneten Ansatz zu wählen, ohne auf die Vorteile des Frameworks zu verzichten.

Ein weiterer wichtiger Unterschied ist die Leistung. CodeBehind ist laut Benchmarks schneller als die Standard-Microsoft-Frameworks, was es besonders für leistungskritische Anwendungen attraktiv macht. Zudem bietet CodeBehind eine einfachere Integration von Webkomponenten und unterstützt modulare Projektstrukturen.

Das erleichtert die Entwicklung und Wartung großer, komplexer Anwendungen. Insgesamt bietet CodeBehind eine Alternative, die die Stärken von ASP.NET Core mit einer flexibleren und entwicklerfreundlicheren Architektur kombiniert.

Architektur und Struktur

Das Framework baut auf dem bewährten MVC (Model-View-Controller)-Muster auf, interpretiert dieses jedoch auf eine moderne und effiziente Weise neu. Diese Struktur ermöglicht es, Webanwendungen flexibel und effizient zu erstellen, während diese gleichzeitig von den Vorteilen einer klaren Trennung der Verantwortlichkeiten profitieren.

Eine der markantesten Eigenschaften von CodeBehind ist die Art und Weise, wie Controller und Model mit der View verknüpft werden. Sie werden direkt in den Attributen der View-Seite spezifiziert.

Dies geschieht typischerweise in der `.aspx`-Datei, die die View repräsentiert. Der folgende Code zeigt ein Beispiel für die Definition von Controller und Model in einer View-Datei:

```
<%@ Page Controller="MyController" Model="MyModel" %>
```

Diese Herangehensweise reduziert die Komplexität der Projektstruktur und macht die Beziehungen zwischen den verschiedenen Komponenten auf einen Blick ersichtlich.

Das Feature hat zudem das Potenzial, die Navigation im Code und damit auch den Entwicklungsprozess spürbar zu beschleunigen.

CodeBehind zeichnet sich zudem durch die Unterstützung verschiedener Entwicklungsmuster aus:

MVC (Model-View-Controller): Das vollständige MVC-Muster lässt sich durch die Definition aller drei Komponenten implementieren.

Listing 2: Code für die Datei Home.aspx

```
<%@ Page Controller="HomeController"
    Model="HomeModel" %>

<!DOCTYPE html>
<html>
<head>
    <title>CodeBehind Example</title>
</head>
<body>
    <h1>Welcome to CodeBehind!</h1>
    <p>Current time: <%= DateTime.Now %></p>
    <p>Message from Model: <%= Model.Message %></p>
</body>
</html>
```

Model-View: Für einfachere Anwendungen kann das Controller-Element entfallen, sodass sich direkt mit Model und View arbeiten lässt.

Nur View: Bei statischen oder sehr einfachen dynamischen Seiten ist es möglich, ausschließlich mit der View zu arbeiten, ohne ein Model oder einen Controller zu definieren.

Web-Forms: CodeBehind unterstützt auch ein Web-Forms-ähnliches Muster, das allen, die mit älteren ASP.NET-Technologien vertraut sind, einen sanften Übergang ermöglicht.

Diese Vielfalt an Entwicklungsmustern macht CodeBehind zu einem äußerst anpassungsfähigen Framework, das sich an verschiedene Projektanforderungen und Entwicklerstile anpassen kann. Es erlaubt, das für spezifische Bedürfnisse am besten geeignete Muster zu wählen, ohne dabei die Vorteile des Frameworks zu verlieren. Die Architektur von CodeBehind fördert somit nicht nur eine klare Trennung der Verantwortlichkeiten, sondern bietet auch die Flexibilität, die Komplexität der Anwendung je nach Bedarf zu skalieren.

Installation, Konfiguration und Einrichtung

Die Integration von CodeBehind in ASP.NET-Core-Projekte ist ein unkomplizierter Prozess, der mit minimaler Konfiguration auskommt. Die Installation des Frameworks erfolgt primär über einen Paketmanager wie NuGet [3]. Dieser Ansatz ermöglicht eine nahtlose Integration in sowohl neue als auch bestehende ASP.NET-Core-Projekte, beispielsweise über den folgenden Befehl:

```
dotnet add package CodeBehind --version 3.6.0
```

Nach der Installation des Pakets ist die Datei `Program.cs` im Projekt anzupassen, um CodeBehind zu initialisieren und zu konfigurieren. Listing 1 zeigt eine minimale Konfiguration – eine der Stärken des CodeBehind-Frameworks. Neben dem Import von Bibliotheken initialisiert das Beispiel den CodeBehind-Compiler mit `SetCodeBehind.CodeBehindCompi-` ►

ler.Initialization()). Zudem wird CodeBehind zur Middleware-Pipeline hinzugefügt, was über den Aufruf von *app.UseCodeBehind()* geschieht.

Das Framework steht aktuell in Version 3.6.0 von Mitte September 2024 zur Verfügung. Der Code liegt auf GitHub [4] und steht unter der GPL-3.0-Lizenz. Aktuell ist das Projekt noch nicht sonderlich bekannt. Das Entwicklerteam arbeitet an einer neuen Webseite, die während der Entstehung dieses Artikels zahlreiche Updates erhält.

Erstellung einer einfachen View mit CodeBehind

Nachdem CodeBehind im Projekt konfiguriert ist, lässt sich eine einfache View erstellen, um das Framework zu testen. Listing 2 zeigt ein Beispiel für eine Datei namens *Home.aspx*. In diesem Beispiel sind der Controller und das Model direkt in den Seitenattributen definiert. Den zugehörigen Controller und das zugehörige Model zeigt Listing 3.

Der Controller füllt das Model mit Daten und gibt es an die View zurück. Diese Struktur ermöglicht eine klare Trennung von Präsentation und Logik, während sie gleichzeitig eine intuitive Entwicklung fördert.

Statische Informationen anzeigen

Das folgende Beispiel demonstriert, wie eine Webseite für Informationen aufgebaut ist, die Daten aus einer XML-Datei

Der SeriesController ist dabei das Herzstück des Beispiels. Er liest die XML-Datei, verarbeitet die URL-Parameter und bereitet die entsprechenden Modelle für die Views vor. Listing 4 zeigt einen Teil des Controllers. Der Code verdeutlicht, wie CodeBehind URL-Parameter verarbeitet und darauf basierend entweder eine einzelne Serie oder eine Liste von Serien anzeigt. Es demonstriert auch die Fehlerbehandlung für den Fall, dass eine angeforderte Serie nicht existiert.

Layouts

Layouts sind in der Webentwicklung insbesondere dann wichtig, wenn es darum geht, konsistente und wartbare Websites zu erstellen. Sie bieten eine Möglichkeit, gemeinsame Elemente wie Header, Footer und Navigationsbereiche zentral zu definieren und über mehrere Seiten hinweg wiederzuverwenden. In der Praxis haben viele Websites eine ähnliche Grundstruktur auf allen Seiten. Anstatt diese Struktur auf jeder einzelnen Seite zu wiederholen, lässt sich ein Layout erstellen, das als Vorlage dient. Dies verbessert nicht nur die Codeorganisation, sondern erleichtert auch Änderungen am gesamten Seitendesign, da Aktualisierungen nur an einer Stelle vorgenommen werden müssen.

Im CodeBehind-Framework ist die Implementierung von Layouts einfach und intuitiv. Ein Layout wird als spezielle View-Datei erstellt und mit dem Attribut *@islayout* gekennzeichnet.

● Listing 3: Der Code des Controllers und des zugehörigen Models

```
using CodeBehind;

public class HomeController : CodeBehindController
{
    public override void Load(CodeBehindModel model)
    {
        var homeModel = (HomeModel)model;
        homeModel.Message = "Hello from CodeBehind!";
    }
}

public class HomeModel : CodeBehindModel
{
    public string Message { get; set; }
}
```

liest und basierend auf URL-Parametern anzeigt. Ein wichtiger erster Schritt ist die Aktivierung der Unterstützung für statische Dateien in ASP.NET Core. Dies wird durch Hinzufügen von *app.UseStaticFiles()* zur Middleware-Pipeline vor *app.UseCodeBehind()* erreicht. Diese Konfiguration ermöglicht es der Anwendung, statische Ressourcen wie CSS-Dateien und Bilder auszuliefern.

Für die Datenmodellierung kommen zwei Klassen zum Einsatz: *SeriesModel* für einzelne Serien und *SeriesModelList* für eine Liste von Serien. Diese Modelle werden verwendet, um Daten zwischen dem Controller und den Views zu übertragen. Zudem werden zwei Views implementiert: eine für die Anzeige der Serienliste und eine für die Detailansicht einer einzelnen Serie. Die Views verwenden eine Mischung aus HTML und CodeBehind-spezifischer Syntax, um Daten aus den Modellen anzuzeigen.

zeichnet. Innerhalb des Layouts können Sie Platzhalter für den spezifischen Inhalt jeder Seite definieren, typischerweise mit der Variablen *@PageReturnValue*. Listing 5 zeigt ein Beispiel für ein einfaches Layout.

Um dieses Layout in einer spezifischen Seite zu verwenden, fügen Sie einfach das *@layout*-Attribut hinzu und geben den Pfad zur Layout-Datei an. Der Inhalt der Seite wird dann automatisch an der Stelle des *@PageReturnValue* im Layout eingefügt.

Ein weiterer Vorteil von Layouts ist die Möglichkeit, dynamische Titel für jede Seite zu setzen. Dies geschieht über die *ViewData*-Sammlung, die es erlaubt, Daten zwischen dem Controller und der View zu übertragen. Im Controller können wir den Titel für jede Seite individuell festlegen, und das Layout greift dann darauf zu, um einen passenden Titel im *<title>*-Tag zu setzen.

● Listing 4: Der Code des SeriesController, um statische Daten anzuzeigen

```
public partial class SeriesController
    : CodeBehindController
{
    public void PageLoad(HttpContext context)
    {
        XmlDocument doc = new XmlDocument();
        doc.Load("data/series_list.xml");

        if (Section.Count() == 1)
        {
            string Section0 = System.Web.HttpUtility
                .UrlEncode(Section.GetValue(0));
            XmlNode SeriesNode = doc
                .SelectSingleNode(
                    "series_lis/series[url_value='" + Section0
                    + "']");

            if (SeriesNode == null)
            {
                IgnoreViewAndModel = true;

                Write("Series not exist!");
                return;
            }

            SeriesModel model = new SeriesModel();
            model.SeriesTitle =
                SeriesNode["title"].InnerText;
            model.SeriesUrlValue =
                SeriesNode["url_value"].InnerText;
            // ... weitere Eigenschaften werden gesetzt

            View("/series_page/content.aspx", model);
        }
        else if (Section.Count() == 0)
        {
            SeriesModelList model = new SeriesModelList();
            // ... Code zum Füllen der Liste
        }
    }
}
```

● Listing 5: Beispielstruktur für ein Layout

```
@page
@islayout
<!DOCTYPE html>
<html>
<head>
    <title>@ViewData.GetValue("title")</title>
    <link rel="stylesheet" type="text/css"
        href="/styles/main.css" />
</head>
<body>
    <header>
        <h1>Meine Website</h1>
        <nav>
            <!-- Navigationsmenü hier -->
        </nav>
    </header>

    <main>
        @PageReturnValue
    </main>

    <footer>
        <p>&copy; 2024 Meine Website</p>
    </footer>
</body>
</html>
```

Die Verwendung von Layouts in CodeBehind fördert nicht nur die Wiederverwendung von Code, sondern verbessert auch die Wartbarkeit und Konsistenz der Website. Entwickler können sich auf den spezifischen Inhalt jeder Seite konzentrieren, während das allgemeine Struktur- und Designgerüst zentral verwaltet wird.

Formulare

Das CodeBehind-Framework bietet zudem eine intuitive und effiziente Methode zur Handhabung von Formularen in Webanwendungen. Die Formularverarbeitung ist nahtlos in die Architektur des Frameworks integriert und ermöglicht eine unkomplizierte Datenerfassung und -verarbeitung. In CodeBehind werden Formulare typischerweise in den View-Dateien (.aspx) definiert.

Die Formularfelder können mit Standard-HTML-Elementen erstellt werden, was eine große Flexibilität bei der Gestaltung erlaubt. Die Verarbeitung der Formulardaten erfolgt im zugehörigen Controller.

Ein besonderes Merkmal von CodeBehind ist die automatische Bindung von Formulardaten an Modelleigenschaften. Dies vereinfacht die Datenextraktion und -validierung erheblich. Entwickler können auf die übermittelten Formulardaten entweder über die *Request.Form*-Sammlung zugreifen oder die automatische Modellbindung nutzen.

Die Formularvalidierung kann sowohl clientseitig als auch serverseitig implementiert sein. CodeBehind unterstützt die Integration von Validierungsbibliotheken und bietet gleichzeitig die Möglichkeit, benutzerdefinierte Validierungslogik im Controller zu implementieren. ►

● Listing 6: HTML-Struktur in CodeBehind für ein Formular

```

<%@ Page Controller="ContactController"
Model="ContactModel" %>

<!DOCTYPE html>
<html>
<head>
    <title>Kontaktformular</title>
</head>
<body>
    <h1>Kontaktieren Sie uns</h1>
    <form method="post">
        <div>
            <label for="Name">Name:</label>
            <input type="text" id="Name" name="Name"
                required />
        </div>
        <div>
            <label for="Email">E-Mail:</label>
            <input type="email" id="Email"
                name="Email" required />
        </div>
        <div>
            <label for="Message">Nachricht:</label>
            <textarea id="Message" name="Message"
                required></textarea>
        </div>
        <button type="submit">Absenden</button>
    </form>

    <% if (!string.IsNullOrEmpty(Model
        .SubmissionMessage)) { %>
        <p><%= Model.SubmissionMessage %></p>
    <% } %>
</body>
</html>

```

Ein weiteres Beispiel zeigt, wie die Formularverarbeitung in CodeBehind funktioniert. Listing 6 enthält den Code für die View-Datei. Anschließend lässt sich das Model definieren:

```

public class ContactModel : CodeBehindModel
{
    public string Name { get; set; }
    public string Email { get; set; }
    public string Message { get; set; }
    public string SubmissionMessage { get; set; }
}

```

Schließlich kommt noch der Controller an der Reihe, den Sie in Listing 7 sehen. Die View definiert in diesem Beispiel das HTML-Formular, das Model enthält die Datenstruktur, und der Controller verarbeitet die Formulareingaben. Wenn Sie das Formular absenden, validiert der Controller die Eingaben und setzt eine entsprechende Bestätigungsnachricht, die in der View angezeigt wird.

Fazit

CodeBehind erweist sich als ein vielversprechendes Framework für die Entwicklung moderner Webanwendungen im

● Listing 7: Der Code des Controllers zum Validieren der Daten aus dem Formular

```

public class ContactController : CodeBehindController
{
    public override void Load(CodeBehindModel model)
    {
        var contactModel = (ContactModel)model;

        if (IsPostBack)
        {
            // Formulardaten verarbeiten
            contactModel.Name = Request.Form["Name"];
            contactModel.Email = Request.Form["Email"];
            contactModel.Message = Request.Form["Message"];

            // Einfache Validierung
            if (string.IsNullOrEmpty(contactModel.Name) ||
                string.IsNullOrEmpty(contactModel.Email) ||
                string.IsNullOrEmpty(contactModel.Message))
            {
                contactModel.SubmissionMessage =
                    "Bitte füllen Sie alle Felder aus.";
            }
            else
            {
                // Hier würde normalerweise die Logik zum
                // Speichern oder Versenden der
                // Nachricht stehen
                contactModel.SubmissionMessage =
                    "Vielen Dank für Ihre Nachricht!";
            }
        }
    }
}

```

ASP.NET-Core-Ökosystem. Mit seiner View-zentrierten Architektur und der Flexibilität, verschiedene Entwicklungsmuster zu unterstützen, bietet es eine attraktive Alternative zu etablierten Microsoft-Frameworks.

Die Stärken von CodeBehind liegen in seiner Effizienz, Flexibilität und Entwicklerfreundlichkeit. Die vereinfachte Struktur und intuitive Handhabung von Benutzeranfragen können den Entwicklungsprozess beschleunigen und die Wartbarkeit von Anwendungen verbessern. Besonders hervorzuheben sind:

- die Unterstützung verschiedener Entwicklungsmuster (MVC, Model-View, nur View, Web-Forms-ähnlich),
- eine unkomplizierte Installation und Konfiguration, und
- die Möglichkeit, Layouts effizient zu implementieren und wiederzuverwenden.

Trotz seiner kurzen Historie – man beachte, dass die erste stabile Version erst 2023 veröffentlicht wurde – zeigt CodeBehind ein wirklich großes Potenzial.

Es adressiert wichtige Herausforderungen moderner Webentwicklung, wie beispielsweise die Balance zwischen Leistung und Benutzerfreundlichkeit sowie die Skalierbarkeit von Anwendungen.

Alle, die das Framework in Betracht ziehen, sollten die Community-Unterstützung und die langfristige Wartung im Auge behalten.

Zum jetzigen Stand verdient sich das Projekt ein „Gut“ und eine Empfehlung, einen Blick darauf zu werfen. Zum „Sehr gut“ fehlen noch ein paar Beispiele mehr und eine umfangreichere Dokumentation, die aber in der Entstehung ist.

Geben wir dem Framework noch etwas Zeit in der Öffentlichkeit, um zu reifen. ■

[1] Die Website von Elanat, dem Team hinter CodeBehind, <https://elanat.net>

[2] Release der Version 1.9.2 von CodeBehind, www.dotnetpro.de/SL2412Frameworks1

[3] Das Projekt auf NuGet, www.dotnetpro.de/SL2412Frameworks2

[4] CodeBehind auf GitHub, www.dotnetpro.de/SL2412Frameworks3



Dr. Fabian Deitelhoff

arbeitet als IT-Leiter, Head of Software Development und Head of Product bei der Education Partners GmbH. Darüber hinaus ist er als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig.

@FDeitelhoff

dnpCode

A2412Frameworks

DEV ACADEMY

3 TAGE
REMOTE/MÜNCHEN

HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN



ASP.NET Blazor - SPA-Anwendungen mit C# und .NET

TRAINING!

Was wird behandelt

- Blazor-Grundlagen
- Routing und Binding mit Blazor
- Formularanbindung mit Blazor
- Validierung von Daten mit Blazor
- Datenbereitstellung mit REST und Integration in Blazor



Christian Giesswein
Trainer

KI UND DATA, TEIL 2

Über Sprache lernen

Wohl keine Entwicklung im Bereich der künstlichen Intelligenz steht aktuell so stark im Rampenlicht wie Sprachmodelle. Ein Überblick.

Die Verbindung von Daten und künstlicher Intelligenz (KI) hat eine rasante Entwicklung erfahren. Der erste Teil dieser Artikelreihe [1] hat gezeigt, wie KI und Daten als Treiber der Wertschöpfung agieren und viele Lebensbereiche, einschließlich der Softwareentwicklung, maßgeblich beeinflussen.

KI hat das Potenzial der „nächsten industriellen Revolution“, weil sie die Fähigkeit hat, große Datenmengen effizient zu verarbeiten und daraus wertvolle Muster zu erkennen. Diese Artikelserie zielt darauf ab, die verschiedenen Aspekte und Anwendungen von KI und Daten zu beleuchten, in diesem zweiten Teil mit einem besonderen Fokus auf Sprachmodelle. Der Artikel betrachtet, wie diese Technologien in der Praxis ankommen und welchen messbaren Nutzen sie heute schon bieten.

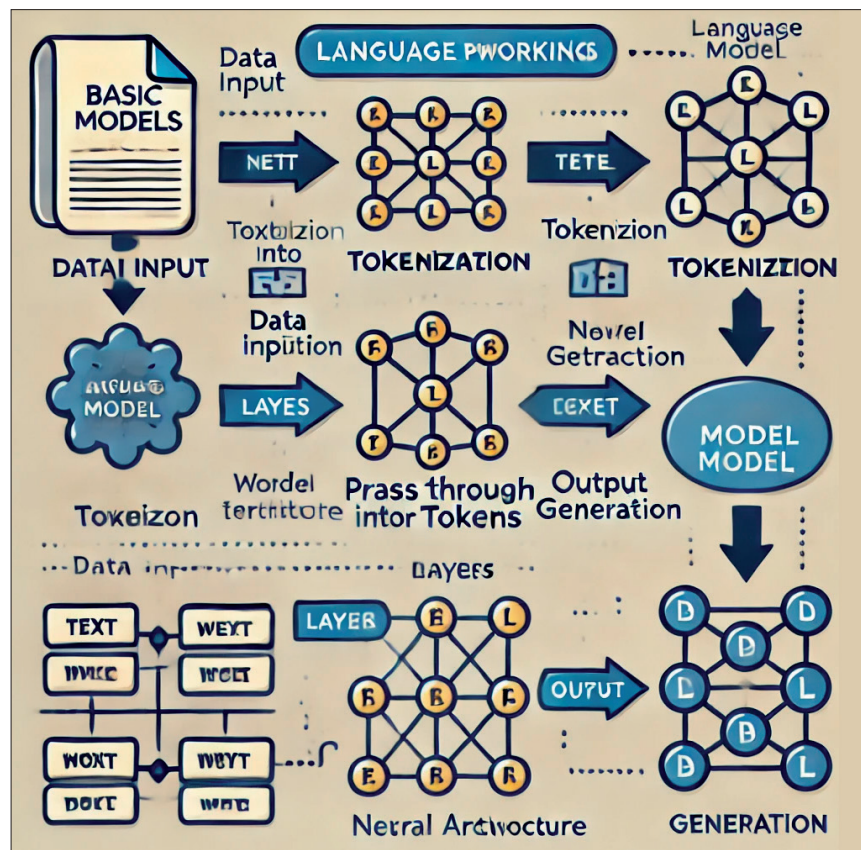
In den kommenden Beiträgen wird es dann primär um die praktische Umsetzung und Integration von KI-Modellen und -Technologien gehen. Der Fokus liegt dabei auf Modellen, Tools und Programmiersprachen im und rund um das .NET-Universum.

Grundlegende Funktionsweise von Sprachmodellen

Sprachmodelle sind ein fundamentaler Bestandteil der modernen künstlichen Intelligenz und haben in den letzten Jahren eine beeindruckende Entwicklung durchlaufen. Diese Modelle bilden die Grundlage für zahlreiche Anwendungen, die natürliche Sprache verarbeiten und generieren können. Dabei sind Sprachmodelle komplexe mathematische Systeme, die darauf trainiert sind, die Strukturen und Muster natürlicher Sprache zu erfassen und zu reproduzieren.

Die Funktionsweise von Sprachmodellen basiert auf dem Prinzip des maschinellen Lernens. Die Modelle sind mit enormen Mengen an Textdaten trainiert, aus denen sie statistische Muster und Zusammenhänge extrahieren. Moderne Sprachmodelle verwenden meist die Transformer-Architektur, die es ihnen ermöglicht, den Kontext über lange Textpassagen hinweg zu berücksichtigen und so ein tieferes Verständnis für die Bedeutung und Struktur von Sprache zu entwickeln. Auf die Transformer-Architektur geht dieser Artikel im weiteren Verlauf noch näher ein.

Ein entscheidender Aspekt der Funktionsweise von Sprachmodellen ist ihre Fähigkeit zur Vorhersage. Ist eine Sequenz von Wörtern vorgegeben, kann ein Sprachmodell die Wahr-



Da geht noch was – noch nicht ganz optimaler Versuch, via ChatGPT 4o und DALL-E darzustellen, wie Sprachmodelle funktionieren (Bild 1)

scheinlichkeit für das nächste Wort berechnen. Diese Fähigkeit bildet die Grundlage für Aufgaben wie Textgenerierung, Übersetzung und Zusammenfassung.

Historische Entwicklung

Die Geschichte der Sprachmodelle reicht bis in die frühen Tage der Computerlinguistik zurück. Frühe Ansätze setzten auf regelbasierte Systeme und einfache statistische Modelle. Diese Modelle waren jedoch in ihrer Fähigkeit, die Komplexität natürlicher Sprache zu erfassen, stark eingeschränkt.

Ein bedeutender Meilenstein war die Einführung von N-Gramm-Modellen in den 1980er-Jahren. Diese Modelle betrachteten die Wahrscheinlichkeit eines Wortes basierend auf den n-1 vorhergehenden Wörtern. Obwohl einfach, waren sie für ihre Zeit bemerkenswert effektiv und fanden breite Anwendung in der Sprachverarbeitung.

Mit dem Aufkommen neuronaler Netze in den 2000er-Jahren begann eine neue Ära für Sprachmodelle. Rekurrente neuronale Netze (RNNs) und insbesondere LSTM-Netzwerke (Long Short-Term Memory) ermöglichten es, längere Kontexte zu berücksichtigen und komplexere sprachliche Strukturen zu erfassen.

Der wohl bedeutendste Durchbruch in der jüngeren Geschichte der Sprachmodelle war die Einführung der Transformer-Architektur im Jahr 2017, die auf dem Konzept der Selbstaufmerksamkeit basiert und die Verarbeitung natürlicher Sprache revolutionierte. Sie ermöglichte die Entwicklung von Modellen wie BERT und GPT, die in der Lage sind, ein tiefes Verständnis für Kontext und Bedeutung zu entwickeln.

Die Entwicklung großer Sprachmodelle wie GPT-3 und seiner Nachfolger markierte schließlich einen weiteren Meilenstein. Diese Modelle, die auf riesigen Datenmengen trainiert wurden, zeigten beeindruckende Fähigkeiten in einer Vielzahl von Sprachaufgaben, von der Textgenerierung bis hin zur Beantwortung komplexer Fragen.

In jüngster Zeit hat sich der Fokus auf die Entwicklung effizienterer und ethisch verantwortungsvollerer Modelle verlagert [2]. Forscher arbeiten an Methoden, um die Größe der Modelle zu reduzieren, ohne ihre Leistungsfähigkeit zu beeinträchtigen, und an Wegen, um Voreingenommenheit [3] und ethische Probleme in Sprachmodellen zu adressieren, eine Thematik, die im Verlauf dieses Artikels noch mehrmals angesprochen wird.

Große Sprachmodelle

Die Welt der Sprachmodelle umfasst verschiedene Typen, die jeweils für spezifische Anwendungsfälle und Anforderungen optimiert sind. Von großen, allgemeinen Modellen bis hin zu kleinen, spezialisierten Varianten bietet jede Art von Sprachmodell einzigartige Vor- und Nachteile.

Große Sprachmodelle (Large Language Models, LLMs) wie GPT-3, GPT-4 und PaLM repräsentieren den Höhepunkt der aktuellen KI-Technologie im Bereich der natürlichen Sprachverarbeitung. Diese Modelle zeichnen sich durch ihre Fähigkeit aus, Texte zu generieren, die von Menschen erzeugten ähneln, und komplexe Sprachaufgaben zu bewältigen.

GPT-3, entwickelt von OpenAI, war ein Meilenstein in der Entwicklung von LLMs. Mit 175 Milliarden Parametern zeigte es eine bis dahin unerreichte Vielseitigkeit und Leistungsfähigkeit. Bei seinem Nachfolger GPT-4 wurden diese Fähigkeiten weiter verbessert und multimodale Funktionen eingeführt, die es dem Modell ermöglichen, sowohl Text als auch Bilder zu verarbeiten.

Neben GPT ist auch Googles PaLM (Pathways Language Model) ein Beispiel für ein fortschrittliches LLM. Es zeichnet sich durch seine Fähigkeit zur Multitask-Verarbeitung und effizientes Training aus.

Als kleine Demonstration zeigt **Bild 1**, was das multimodale Modell ChatGPT-4o, das DALL-E im Hintergrund nutzt, aus folgender, sehr simpel gehaltener Anfrage des Autors macht:

„Bitte erzeuge mir ein Bild für einen Fachartikel, um die grundlegende Funktionsweise von Sprachmodellen zu visualisieren.“

Da ist noch viel Luft nach oben, vor allem im für das Bild erzeugten Text. Mit etwas Fingerspitzengefühl beim Prompting lässt sich das aber verbessern. Probieren Sie es aus: Vielleicht haben Sie ja Lust, sich selbst daran zu versuchen, die Abfrage zu optimieren?

Zusammenfassend lässt sich sagen: Die Vorteile dieser großen Sprachmodelle sind vielfältig. Sie können komplexe Aufgaben wie Textgenerierung, Übersetzung, Zusammenfassung und sogar Programmierung mit beeindruckender Genauigkeit bewältigen. Ihre Fähigkeit, Kontext zu „verstehen“ und auf subtile sprachliche Nuancen zu reagieren, macht sie zu leistungsfähigen Werkzeugen für eine Vielzahl von Anwendungen.

Allerdings bringen diese Vorteile auch Nachteile mit sich. Der enorme Ressourcenbedarf für Training und Betrieb macht diese Modelle kostspielig und energieintensiv. Datenschutzbedenken und ethische Fragen bezüglich der Verwendung und eines potenziellen Missbrauchs dieser Technologie sind ebenfalls wichtige Diskussionspunkte.

Kleine und lokale Sprachmodelle

Im Gegensatz zu den ressourcenintensiven LLMs bieten kleine und lokale Sprachmodelle (Small Language Models, SLMs) eine Reihe von Vorteilen, insbesondere in Bezug auf Effizienz und Privatsphäre. Diese Modelle sind darauf ausgelegt, mit begrenzten Ressourcen zu arbeiten, und lassen sich vielfach direkt auf Endgeräten einsetzen.

Aufgrund ihrer Kompaktheit benötigen diese Modelle weniger Speicherplatz und Rechenleistung, was sie ideal für den Einsatz auf Mobilgeräten oder in Umgebungen mit begrenzten Ressourcen macht. Zudem ermöglichen sie schnellere Inferenzzeiten, was für Anwendungen mit Echtzeitanforderungen entscheidend sein kann.

Ein wichtiger Anwendungsbereich für kleine Sprachmodelle sind Embedded-Systeme, insbesondere in IoT-Geräten. Hier können sie Aufgaben wie Spracherkennung, einfache Textklassifikation oder grundlegende Dialogfunktionen direkt auf dem Gerät ausführen, ohne auf eine Cloud-Verbindung angewiesen zu sein. Dies verbessert nicht nur die Reaktionszeit, sondern auch den Datenschutz, da sensible Informationen das Gerät nicht verlassen müssen. Zudem ermöglichen sie eine Offline-Nutzung und können Kosten einsparen, da keine Cloud-Gebühren anfallen.

Zu den lokalen Möglichkeiten gehören auch WebLLM [4], das die Ausführung von Sprachmodellen direkt im Browser ermöglicht, LLM in der Kommandozeile für eine einfache Installation und Nutzung von Open-Source-Modellen [5] sowie Desktop-Anwendungen wie „Jan“ [6], die benutzerfreundliche Oberflächen für lokale KI-Modelle bieten. Hervorzuheben sind außerdem kleine Sprachmodelle wie Microsofts Phi-3-Familie, die sich für ressourcenbeschränkte Umgebungen und On-Device-Szenarien eignen.

Besonders erwähnenswerte kompakte Modelle sind BERT-Tiny und DistilBERT. BERT-Tiny ist eine stark reduzierte Version des ursprünglichen BERT-Modells, die trotz ihrer geringen Größe beachtliche Leistungen in verschiedenen NLP-Aufgaben erbringt. DistilBERT [7], entwickelt durch ►

Wissenstransfer von einem größeren BERT-Modell, bietet eine gute Balance zwischen Größe und Leistung und ist besonders für mobile Anwendungen geeignet.

Cloud-Modelle

Cloud-Modelle bieten Zugang zu höherer Rechenleistung, regelmäßige Updates und eine flexible Skalierbarkeit. Sie sind über API-Zugriff nutzbar, beispielsweise OpenAIs GPT-Modelle oder Googles PaLM. Zudem bieten Cloud-Anbieter wie Microsoft Azure oder Amazon AWS verwaltete KI-Dienste an. Ein weiterer Vorteil ist schließlich die Möglichkeit des Fine-Tunings, also der Anpassung vortrainierter Modelle an spezifische Aufgaben in der Cloud.

Es gibt auch hybride Ansätze, die lokale und cloudbasierte Modelle kombinieren und somit mehr Flexibilität in der Nutzung bieten. Die Wahl zwischen lokalen und Cloud-Modellen hängt von den spezifischen Anforderungen ab. Während lokale Modelle sich besonders für datenschutz sensible Anwendungen und Offline-Szenarien eignen, sind Cloud-Modelle oft leistungsfähiger und einfacher zu skalieren. Mit der fortschreitenden Entwicklung kleiner, effizienter Modelle wie Phi-3 werden lokale Lösungen insbesondere für ressourcenbeschränkte Umgebungen zunehmend attraktiver.

Domänenspezifische Sprachmodelle

Während allgemeine Sprachmodelle ein breites Spektrum an Aufgaben bewältigen können, gibt es Situationen, in denen spezialisierte Kenntnisse erforderlich sind. Hier kommen domänenspezifische Sprachmodelle ins Spiel, die auf bestimmte Fachgebiete zugeschnitten sind. Diese Modelle sind auf spezifischen Datensätzen trainiert, die für ihr jeweiliges Anwendungsgebiet relevant sind. Dadurch entwickeln sie ein tiefes Verständnis für die Fachsprache, Konzepte und Zusammenhänge in ihrem Spezialgebiet. Dies macht sie besonders wertvoll in Bereichen, wo Präzision und fachliche Tiefe entscheidend sind.

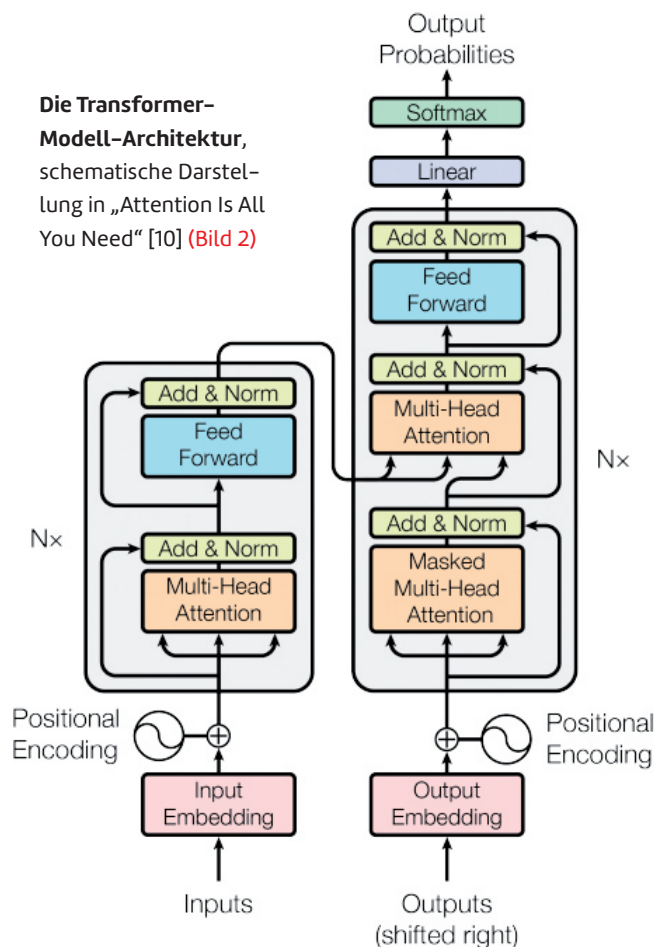
In der Medizin beispielsweise können solche Modelle bei der Analyse von Patientenakten, der Erstellung von Diagnosen oder der Zusammenfassung medizinischer Literatur unterstützen. Sie „verstehen“ medizinische Terminologie und können komplexe medizinische Konzepte verarbeiten.

Open-Source-Modelle als Alternativen

Eine weitere wichtige Entwicklung in der Welt der Sprachmodelle ist der Aufstieg von Open-Source-Alternativen zu proprietären Modellen. Diese Modelle bieten nicht nur kostenlose Zugänglichkeit, sondern auch Transparenz und die Möglichkeit zur Anpassung und Weiterentwicklung durch die Community. Forscher und Entwickler können diese Modelle frei nutzen, modifizieren und für ihre spezifischen Bedürfnisse optimieren. Dies fördert Innovation und ermöglicht es auch kleineren Organisationen und Einzelpersonen, fortschrittliche KI-Technologien zu nutzen.

Ein bemerkenswertes Beispiel für ein Open-Source-Sprachmodell ist GPT-Neo. Entwickelt von EleutherAI [8], einem dezentralen Kollektiv von Forschern, zielt GPT-Neo darauf ab, die Fähigkeiten von GPT-3 in einem offenen Format

Die Transformer-Modell-Architektur, schematische Darstellung in „Attention Is All You Need“ [10] (Bild 2)



nachzubilden. Es erreicht zwar nicht ganz die Größe und Leistungsfähigkeit von GPT-3, bietet aber dennoch beeindruckende Ergebnisse in verschiedenen NLP-Aufgaben.

Ein weiteres Beispiel ist das Open-Source-Projekt BLOOM (BigScience Large Open-Science Open-access Multilingual Language Model) [9]. An BLOOM arbeitet ein internationales Konsortium von Forschern. Das Modell zeichnet sich vornehmlich durch seine multilinguale Ausrichtung aus. Es unterstützt 46 Sprachen und 13 Programmiersprachen, was es zu einem wertvollen Werkzeug für globale und mehrsprachige Anwendungen macht.

Technologische Grundlagen

Die eingangs bereits angesprochene Transformer-Architektur, eingeführt in der Veröffentlichung „Attention Is All You Need“ [10] von Vaswani et al. im Jahr 2017, revolutionierte die Verarbeitung natürlicher Sprache. Im Gegensatz zu früheren Ansätzen, die auf rekurrenten oder faltenden Netzwerken basierten, setzt die Transformer-Architektur vollständig auf den Aufmerksamkeitsmechanismus (Attention).

Die Transformer-Modell-Architektur ist in Bild 2 zu sehen, Quelle ist das oben genannte Paper von Vaswani et al. Der Kern der Transformer-Architektur besteht aus mehreren Schlüsselkomponenten:

- **Multi-Head Attention:** Dieser Mechanismus ermöglicht es dem Modell, gleichzeitig auf verschiedene Aspekte der Eingabe zu achten. Jeder „Kopf“ kann sich auf unterschiedliche Teile der Sequenz konzentrieren, was zu einem umfassenderen Verständnis des Kontexts führt.

- **Positionscodierung:** Da Transformer keine inhärente Reihenfolge der Eingabedaten haben, werden Positionsinformationen durch additive oder gelernte Codierungen hinzugefügt.
- **Feed-Forward-Netzwerke:** Jede Transformerschicht enthält ein vollständig verbundenes Feed-Forward-Netzwerk, das auf jedes Token einzeln angewendet wird.
- **Residuale Verbindungen und Normalisierung:** Diese Techniken verbessern den Informationsfluss durch das Netzwerk und stabilisieren das Training.

Die Bedeutung der Transformer-Architektur für moderne Sprachmodelle kann gar nicht hoch genug geschätzt werden. Sie ermöglicht eine effiziente Verarbeitung langer Sequenzen und erfasst komplexe Abhängigkeiten in den Daten. Dies führte zu einer deutlichen Verbesserung der Leistung bei verschiedenen NLP-Aufgaben wie Übersetzung, Zusammenfassung und Textgenerierung. Modelle wie BERT, GPT und T5, die alle auf der Transformer-Architektur basieren, haben neue Maßstäbe in der Sprachverarbeitung gesetzt.

Das Training moderner Sprachmodelle erfordert eine Kombination aus großen Datenmengen, massiver Rechenleistung und fortschrittlichen Algorithmen. Typische Trainingsmethoden umfassen vortrainiertes Lernen (Pre-Training) gefolgt von Feinabstimmung (Fine-Tuning) für spezifische Aufgaben. Neuere Ansätze wie Few-Shot Learning und Zero-Shot Learning ermöglichen es Modellen, neue Aufgaben mit minimaler oder keiner zusätzlichen Anpassung zu bewältigen. Zu den Voraussetzungen für erfolgreiches Training gehören:

- **Daten:** Große, diverse Textkorpora sind entscheidend. Diese können Bücher, Websites, wissenschaftliche Artikel und mehr umfassen.
- **Rechenleistung:** Hochleistungs-GPUs oder spezialisierte Hardware wie TPUs sind notwendig, um die komplexen Berechnungen zu bewältigen.
- **Algorithmen:** Fortschrittliche Optimierungsalgorithmen wie Adam oder Adafactor kommen zum Einsatz, um das Training zu steuern und zu beschleunigen.

• Listing 1: Extraktion von Entitäten mittels spaCy

```
import spacy

# Laden des englischen Sprachmodells
nlp = spacy.load("en_core_web_sm")

text = "Apple Inc. was founded by Steve Jobs in
        Cupertino, California on April 1, 1976."

# Verarbeitung des Textes
doc = nlp(text)

# Extraktion der Entitäten
for ent in doc.ents:
    print(f"Entität: {ent.text}, Typ: {ent.label_}")
```

Um die Leistungsfähigkeit von Sprachmodellen weiter zu verbessern und gleichzeitig deren Ressourcenverbrauch zu optimieren, existieren verschiedene innovative Ansätze:

- **Sparse Attention:** Traditionelle Transformer-Modelle verwenden „Dense Attention“, bei der jedes Token mit jedem anderen Token interagiert. Dies führt zu quadratischer Komplexität in Bezug auf die Sequenzlänge. Sparse-Attention-Mechanismen reduzieren diese Komplexität, indem sie selektiv nur mit einer Teilmenge der Tokens interagieren. Ansätze wie Longformer oder Reformer nutzen verschiedene Strategien, um die Aufmerksamkeit zu reduzieren, was längere Kontexte bei geringerem Rechenaufwand ermöglicht.
- **Distillation von Modellen:** Knowledge Distillation ist eine Technik, bei der ein großes, komplexes Modell (der „Lehrer“) verwendet wird, um ein kleineres Modell (den „Schüler“) zu trainieren. Der Schüler lernt nicht nur aus den harten Labels der Trainingsdaten, sondern auch aus den weichen Vorhersagen des Lehrers. Dies ermöglicht es, viel von der Leistungsfähigkeit großer Modelle in kleinere, effizientere Modelle zu übertragen. Ein bekanntes Beispiel ist das bereits erwähnte DistilBERT [7], das 40 Prozent kleiner ist als BERT, aber 95 Prozent seiner Leistung beibehält und zudem 60 Prozent schneller ist.

Sprachmodelle als Basis für Innovationen

Sprachmodelle haben sich zu leistungsfähigen Werkzeugen entwickelt, die die Grundlage bilden für verschiedene fortgeschrittene Anwendungen im Bereich der Datenverarbeitung und -analyse. In diesem Teil der Artikelserie betrachten wir zwei beispielhafte Anwendungsbereiche: die Extraktion von Entitäten aus unstrukturierten Daten sowie die Zuordnung von Inhalten zwischen verschiedenen Datenbanken durch semantische Analyse.

Die Fähigkeit, relevante Informationen wie Namen, Orte und Ereignisse aus unstrukturierten Textdaten zu extrahieren, ist von unschätzbarem Wert für viele Anwendungsbereiche. Sprachmodelle können diese Aufgabe mit hoher Genauigkeit bewältigen. Listing 1 zeigt dazu ein Beispiel in Python unter Verwendung der spaCy-Bibliothek [11]. In diesem Beispiel extrahiert das Sprachmodell automatisch Entitäten wie Unternehmen (*ORG*), Personen (*PERSON*), geopolitische Entitäten (*GPE*) und Datumsangaben (*DATE*) aus dem unstrukturierten Text. Eine Ausgabe könnte wie folgt aussehen:

```
Entität: Apple Inc., Typ: ORG
Entität: Steve Jobs, Typ: PERSON
Entität: Cupertino, Typ: GPE
Entität: California, Typ: GPE
Entität: April 1, 1976, Typ: DATE
```

Sprachmodelle sind zudem nützlich, um Inhalte verschiedener Datenbanken semantisch zu analysieren und zuzuordnen, etwa bei der Integration verschiedener Datensysteme oder Migrationen. Listing 2 zeigt dazu ein Beispiel in Python. Darin wird ein vortrainiertes Sprachmodell verwendet, um die semantische Ähnlichkeit zwischen Spalten aus zwei ►

● Listing 2: Semantische Analyse mittels Pandas

```
import pandas as pd
from sentence_transformers import
    SentenceTransformer
from sklearn.metrics.pairwise import
    cosine_similarity

# Laden des Sprachmodells
model = SentenceTransformer(
    'distilbert-base-nli-mean-tokens')

# Beispieldaten aus zwei verschiedenen Datenbanken
db1 = pd.DataFrame({
    'customer_name': ['John Doe', 'Jane Smith'],
    'customer_address': ['123 Main St', '456 Elm St']
})

db2 = pd.DataFrame({
    'client_fullname': ['John Doe', 'Jane Smith'],
    'client_residence': ['123 Main Street', '456 Elm
        Street']
})

# Funktion zur Berechnung der semantischen
# Ähnlichkeit
def get_semantic_similarity(col1, col2):
    embeddings1 = model.encode(col1)
    embeddings2 = model.encode(col2)
    return cosine_similarity(embeddings1, embeddings2)

# Vergleich der Spalten
for col1 in db1.columns:
    for col2 in db2.columns:
        similarity = get_semantic_similarity(db1[col1],
            db2[col2])
        avg_similarity = similarity.mean()
        print(f"Ähnlichkeit zwischen {col1} und {col2}:
            {avg_similarity:.4f}")
```

verschiedenen Datenbanken zu berechnen. Das Modell könnte beispielsweise erkennen, dass *customer_name* in der ersten Datenbank semantisch äquivalent zu *client_fullname* in der zweiten Datenbank ist, auch wenn die Spaltennamen unterschiedlich sind.

Verbesserungen bei der und für die Integration

In den letzten Jahren gab es deutliche Verbesserungen hinsichtlich der Integration von Sprachmodellen in andere Systeme. Drei Bereiche stechen dabei besonders hervor: die Entwicklung effektiver Methoden für Few-Shot und Zero-Shot Learning, vereinfachte APIs und Schnittstellen sowie die verbesserte Integration von Text-, Bild- und Audioanalyse in einem Modell.

Few-Shot und Zero-Shot Learning haben die Art und Weise, wie Modelle neue Aufgaben erlernen und bewältigen, stark verbessert. Diese Techniken ermöglichen es Modellen, mit minimalen oder sogar ohne spezifische Trainingsbeispiele neue Aufgaben zu meistern. Bei Few-Shot Learning wird das Modell mit nur wenigen Beispielen (typischerweise einem bis fünf) trainiert, um eine neue Aufgabe zu erlernen. Dies ist besonders nützlich in Situationen, in denen große Mengen an markierten Daten nicht verfügbar oder zu teuer zu beschaffen sind [12]. Ein Beispiel für die Anwendung von Few-Shot Learning zeigt [Listing 3](#).

Zero-Shot Learning geht noch einen Schritt weiter und ermöglicht es Modellen, Aufgaben zu bewältigen, für die sie überhaupt nicht explizit trainiert wurden [12]. Ein Beispiel hierfür ist die Verwendung eines vortrainierten Modells für eine völlig neue Klassifikationsaufgabe, wie [Listing 4](#) zeigt.

Die Integration von KI-Modellen in bestehende Systeme hat sich durch die Entwicklung vereinfachter APIs und Schnittstellen verbessert. Diese Verbesserungen ermöglichen es, KI-Funktionen nahtlos in Anwendungen einzubinden, ohne tiefgreifende Kenntnisse der zugrunde liegenden KI-Modelle zu benötigen. Dazu gehört der Aufruf von Modellen über simple APIs, die Definition von Ausgabeformaten, an die sich die Modelle zu 100 Prozent halten, und vorgefertigte Plattformen, um Daten zu generieren, Modelle zu trainieren und generell Modelle zur Verfügung zu stellen.

OpenAI hat beispielsweise die erzwungenen, strukturierten Ausgaben eingeführt [13]. Wie man auf GPT-4 zugreift und eine JSON-Ausgabe mit „Safe Structured Outputs“ erzwingt, soll ein Beispiel zeigen. Dabei wird das OpenAI-API mit der neuen strukturierten Ausgabefunktion verwendet, die sicherstellt, dass die Antwort einem bestimmten JSON-Schema entspricht. [Listing 5](#) zeigt das Beispiel in C#. Der *response_format*-Parameter kommt zum Einsatz, um ein JSON-Schema anzugeben, dem die Ausgabe des Modells entsprechen muss. Dies gewährleistet konsistente und gültige JSON-Antworten.

Ein weiterer bedeutender Fortschritt ist die Entwicklung multimodaler Modelle, die Text-, Bild- und Audioanalyse in einem einzigen System integrieren. Diese Modelle können komplexe Aufgaben bewältigen, die mehrere Modalitäten umfassen, wie beispielsweise die Beschreibung von Bildinhalten oder die Transkription und Analyse von Audiodaten. Ein Beispiel für die Verwendung eines solchen multimodalen Modells zeigt [Listing 6](#).

Bias und Datenschutz

Mit der zunehmenden Verbreitung und Leistungsfähigkeit von KI-Systemen, insbesondere von Sprachmodellen, rücken ethische Fragen und Verantwortung in den Vordergrund. Die Bedenken betreffen dabei drei Hauptbereiche: Bias in Sprachmodellen, Datenschutz und Sicherheit sowie KI-Transparenz und -Erklärbarkeit.

Bias in Sprachmodellen ist ein ernsthaftes Problem, das weitreichende Auswirkungen haben kann. Die Basis der Trainings von Sprachmodellen sind große Textkorpora, die oft gesellschaftliche Vorurteile und Ungleichheiten widerspiegeln.

● Listing 3: Anwendung des Few-Shot-Lernens

```
from transformers import GPT2LMHeadModel,
    GPT2Tokenizer

model = GPT2LMHeadModel.from_pretrained("gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("gpt2")

# Few-Shot Beispiele
examples = [
    ("Der Film war großartig.", "Positiv"),
    ("Das Essen schmeckte schrecklich.", "Negativ"),
    ("Ich liebe dieses Buch.", "Positiv")
]

# Neue Bewertung
new_review = "Die Vorstellung war atemberaubend."

# Few-Shot Prompt erstellen
prompt = "Sentiment-Analyse Beispiele:\n"
for example, sentiment in examples:
    prompt += f"Text: {example}\nSentiment: {sentiment}\n\n"
prompt += f"Text: {new_review}\nSentiment:"

# Modellvorhersage
input_ids = tokenizer.encode(prompt,
    return_tensors="pt")
output = model.generate(input_ids, max_length=len(
    input_ids[0])+5, num_return_sequences=1)
prediction = tokenizer.decode(output[0])

print(prediction)
```

Infolgedessen können die Modelle diese Voreingenommenheit in ihren Ausgaben reproduzieren oder sogar verstärken. Dies kann zu unfairen oder diskriminierenden Ergebnissen führen, wenn sie in Bereichen wie Personalauswahl, Kreditvergabe oder Strafverfolgung eingesetzt werden.

Forscher arbeiten daher an Methoden zur Erkennung und Minderung von Bias in KI-Systemen. Dies umfasst die sorgfältige Auswahl und Aufbereitung von Trainingsdaten, die Implementierung von Fairness-Metriken während des Trainings und die Entwicklung von Post-Processing-Techniken zur Reduzierung von Bias in den Modellergebnissen.

Datenschutz und Sicherheit sind weitere kritische Aspekte bei der Verwendung großer Sprachmodelle. Diese Modelle sind oft mit sensiblen persönlichen Daten trainiert und können, wenn sie nicht sorgfältig gehandhabt werden, zur unbeabsichtigten Offenlegung privater Informationen führen. Darüber hinaus können böswillige Akteure versuchen, diese Modelle zu manipulieren oder auszunutzen, um Fehlinformationen zu verbreiten oder andere schädliche Aktivitäten durchzuführen.

Um diese Risiken zu mindern, ist es wichtig, robuste Sicherheitsmaßnahmen zu implementieren, einschließlich Datenverschlüsselung, strenger Zugriffskontrollen und regelmäßiger Sicherheitsaudits. Zudem müssen Unternehmen und Organisationen, die KI-Systeme einsetzen, klare Richtlinien und Verfahren für den verantwortungsvollen Umgang mit Daten und KI-generierten Inhalten entwickeln. Anbieter von Modellen wie beispielsweise OpenAI veröffentlichen sogenannte Model Cards [14], um Transparenz und Verantwortlichkeit in der KI-Entwicklung zu fördern.

Diese Karten bieten eine standardisierte Dokumentation zu Architektur, Trainingsdaten und Leistung eines Modells sowie ethischen Überlegungen. Diese Informationen helfen, die Ergebnisse eines Modells zu verstehen und mit anderen Modellen zu vergleichen, während sie gleichzeitig die Auswirkungen auf Datenschutz und Bias beleuchten. Model Cards dienen als Spezifikationsblätter für Organisationen, um fundierte Entscheidungen über den Einsatz neuer KI-Tools zu treffen. Sie sind ein wichtiges Instrument zur Förderung von Zusammenarbeit und Innovation in der KI-Community und unterstützen eine verantwortungsvolle Nutzung von KI-Technologien [15].

Federated Learning als Trainingsansatz

Darüber hinaus sind KI-Transparenz und -Erklärbarkeit entscheidend, um das Vertrauen in KI-Systeme zu stärken und ihre Entscheidungsprozesse nachvollziehbar zu machen. Dies ist besonders wichtig in Bereichen, in denen KI-Entscheidungen erhebliche Auswirkungen auf das Leben von Menschen haben können, wie etwa im Gesundheitswesen oder im Rechtssystem.

Um Transparenz und Erklärbarkeit von Sprachmodellen zu verbessern, werden verschiedene Ansätze verfolgt. Dazu gehören die Entwicklung von Visualisierungstechniken, die die internen Prozesse des Modells veranschaulichen, die Implementierung von Erklärungsmechanismen, die die Gründe für bestimmte Ausgaben darlegen, sowie die Förderung von Open-Source-Initiativen, die eine gründliche Überprüfung der Modellarchitektur ebenso wie des Trainingsverfahrens ermöglichen. ►

● Listing 4: Anwendung des Zero-Shot-Lernens

```
from transformers import pipeline

classifier = pipeline("zero-shot-classification")

text = "Der neue Elektroauto-Prototyp hat eine
    Reichweite von 500 km."
candidate_labels = ["Technologie", "Sport",
    "Politik", "Unterhaltung"]

result = classifier(text, candidate_labels)
print(result)
```


Ein innovativer Ansatz zum dezentralen Training von Modellen ist das Federated Learning, der sowohl die Leistungsfähigkeit von KI-Systemen als auch den Schutz der Privatsphäre verbessert [16]. Hierzu werden lokale Modelle auf den Geräten der Benutzer trainiert, wobei nur die Modell-Updates, nicht aber die zugrunde liegenden Daten an einen zentralen Server übermittelt werden. Der Server aggregiert diese Updates, um ein globales Modell zu verbessern, das dann wieder an die lokalen Geräte verteilt wird. Dieser Prozess wird iterativ wiederholt, um das Modell kontinuierlich zu verbessern. Der Ansatz bietet mehrere Vorteile:

- **Datenschutz:** Sensible persönliche Daten verbleiben auf den Geräten der Benutzer und werden nicht zentral gespeichert oder übertragen.
- **Reduzierte Datenübertragung:** Es sind nur Modell-Updates, nicht aber die gesamten Datensätze zu übertragen, was die Netzwerkbelastung reduziert.
- **Anpassungsfähigkeit:** Lokale Modelle können sich an Nutzungsmuster einzelner Benutzer anpassen, während das globale Modell von den kollektiven Erkenntnissen profitiert.

Federated Learning findet Anwendung in verschiedenen Bereichen, darunter mobile Tastaturvorhersage, Gesundheitswesen und Finanzdienstleistungen. Es ermöglicht Organisationen, von den Vorteilen des maschinellen Lernens zu profitieren, ohne dabei die Privatsphäre ihrer Nutzer zu gefährden. Die Implementierung von Federated Learning bringt jedoch auch Herausforderungen mit sich, wie die Notwendigkeit effizienter Kommunikationsprotokolle, den Umgang mit heterogenen Daten und Geräten sowie die Sicherstellung der Modellintegrität in einem verteilten Umfeld.

Soziale und ökonomische Implikationen

Sprachmodelle und KI-Technologien haben weitreichende gesellschaftliche und wirtschaftliche Auswirkungen. Eine der bedeutendsten Veränderungen betrifft den Arbeitsmarkt. Automatisierung und KI können Aufgaben effizienter erledigen, was zu einer Verschiebung der Arbeitsanforderungen führt. Während einige Berufe durch Technologie ersetzt werden, entstehen neue Rollen, die spezialisierte Fähigkeiten im Umgang mit KI erfordern.

Listing 5: Sichere, weil erzwungene Ausgabestruktur bei einem GPT-Modell

```
private static readonly HttpClient client =
    new HttpClient();
private const string apiKey = "YOUR_API_KEY";
private const string endpoint = "https://
    api.openai.com/v1/chat/completions";

static async Task Main(string[] args)
{
    var response = await GetStructuredOutputAsync(
        "Explain the benefits of JSON.");
    Console.WriteLine(response);
}

public static async Task<string>
GetStructuredOutputAsync(string prompt)
{
    var requestBody = new
    {
        model = "gpt-4o-2024-08-06",
        messages = new[]
        {
            new { role = "system", content = "You are an
                assistant and you only reply with JSON." },
            new { role = "user", content = prompt }
        },
        response_format = new
        {
            type = "json_schema",
            json_schema = new
            {
                strict = true,
```

```
schema = new
{
    type = "object",
    properties = new
    {
        explanation = new { type = "string" },
        benefits = new { type = "array", items =
            new { type = "string" } }
    },
    required = new[] { "explanation",
        "benefits" }
    }
}

var jsonRequestBody = JsonConvert.SerializeObject(
    requestBody);
client.DefaultRequestHeaders.Authorization =
    new System.Net.Http.Headers.
    AuthenticationHeaderValue("Bearer", apiKey);

var response = await client.PostAsync(endpoint, new
    StringContent(jsonRequestBody, Encoding.UTF8,
        "application/json"));
var responseContent =
    await response.Content.ReadAsStringAsync();

return responseContent;
}
```

● Listing 6: Nutzung eines multimodalen Modells in Python

```
from transformers import VisionEncoderDecoderModel,
    ViTImageProcessor, AutoTokenizer
import torch
from PIL import Image

model = VisionEncoderDecoderModel.from_pretrained(
    "nlpconnect/vit-gpt2-image-captioning")
feature_extractor = ViTImageProcessor.from_pretrained(
    "nlpconnect/vit-gpt2-image-captioning")
tokenizer = AutoTokenizer.from_pretrained("nlpconnect/
    vit-gpt2-image-captioning")

image = Image.open("example_image.jpg")
pixel_values = feature_extractor(images=image,
    return_tensors="pt").pixel_values

generated_ids = model.generate(pixel_values,
    max_length=30)
generated_caption = tokenizer.batch_decode(
    generated_ids, skip_special_tokens=True)[0]

print(generated_caption)
```

Die Chancen und Risiken dieser Technologien sind vielfältig. Einerseits bieten sie Möglichkeiten zur Effizienzsteigerung, Innovation und Problemlösung in Bereichen wie Medizin, Bildung und Wirtschaft. Andererseits bergen sie Risiken wie Datenschutzverletzungen, Abhängigkeit von Technologie und potenziellen Missbrauch. Insgesamt erfordert der verantwortungsvolle Einsatz von Sprachmodellen eine sorgfältige Abwägung zwischen Nutzen und möglichen negativen Konsequenzen, um positive gesellschaftliche und wirtschaftliche Entwicklungen zu fördern.

Zwischenfazit und Ausblick

Sprachmodelle spielen eine zentrale Rolle in der KI und bieten immenses Potenzial für zahlreiche Branchen. Die rasante Entwicklung von KI-Technologien, insbesondere durch generative Modelle wie GPT, ermöglicht revolutionäre Anwendungen, von der Textverarbeitung über die kreative Unterstützung bis hin zur wissenschaftlichen Forschung. Große Modelle bergen jedoch auch Herausforderungen wie hohen Ressourcenbedarf und ethische Bedenken. Kleinere, spezialisierte und Open-Source-Modelle können hier effiziente Alternativen darstellen.

Die Weiterentwicklung von Sprachmodellen und KI-Systemen eröffnet faszinierende Möglichkeiten in verschiedenen Bereichen. In der personalisierten Bildung können KI-gestützte Tutoren Lernende individuell unterstützen, indem sie sich an deren Tempo, Lernstil und Interessen anpassen. Dies ermöglicht maßgeschneiderte Lernpfade und fördert effektiveres Lernen. Im Bereich der kreativen Unterstützung fungieren KI-Systeme als Ideengeneratoren und Kooperationspartner für Künstler, Schriftsteller und Designer. Sie können neue Perspektiven eröffnen, unkonventionelle Kombinationen vorschlagen und den kreativen Prozess beschleunigen.

Im kommenden dritten Teil dieser Serie liegt der Fokus auf der praktischen Umsetzung zweier Szenarien. Diese lassen sich auf verschiedene Arten umsetzen, etwa mit klassischen Algorithmen oder Sprachmodellen sowie lokal oder in der Cloud. Wir werden daher einen Blick auf verschiedene Implementierungen und Entscheidungen werfen, um konkrete Anforderungen aus der Praxis umzusetzen. ■

- [1] Fabian Deitelhoff, *Vom Hype zum Mehrwert*, dotnetpro 11/2024, Seite 94 ff., www.dotnetpro.de/A2411KIundData
- [2] *Identifying and Mitigating the Security Risks of Generative AI*, <https://arxiv.org/abs/2308.14840>
- [3] *The Google Health Digital Well-Being Study: Protocol for a Digital Device Use and Well-Being Study*, www.dotnetpro.de/SL2412KIundData1
- [4] WebLLM, <https://webllm.mlc.ai/>
- [5] LLM Python command-line utility, <https://llm.datasette.io/en/stable/>
- [6] Jan, <https://jan.ai>
- [7] DistilBERT auf Hugging Face, www.dotnetpro.de/SL2412KIundData2
- [8] EleutherAI, <https://www.eleuther.ai>
- [9] BLOOM, <https://bigscience.huggingface.co/blog/bloom>
- [10] Vaswani et al., *Attention Is All You Need*, www.dotnetpro.de/SL2412KIundData3
- [11] spaCy, <https://spacy.io>
- [12] *Guide zu Zero-Shot und Few-Shot*, www.dotnetpro.de/SL2412KIundData4
- [13] *Blogpost zu strukturierten Ausgaben bei OpenAI*, www.dotnetpro.de/SL2412KIundData5
- [14] *Die „System Card“ von OpenAI zu GPT 4o*, www.dotnetpro.de/SL2412KIundData6
- [15] *Informationen und Einschätzungen zu Model Cards*, www.dotnetpro.de/SL2412KIundData7
- [16] *Einschätzung zu lokalen Sprachmodellen*, www.dotnetpro.de/SL2412KIundData8



Dr. Fabian Deitelhoff

arbeitet als IT-Leiter, Head of Software Development und Head of Product bei der Education Partners GmbH. Darüber hinaus ist er als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig.

@FDeitelhoff

dnpCode

A2412 KIundData



DATA ACCESS

Die Datenzugriffsberatung mit DR. HOLGER SCHWICHTENBERG

NEUERUNGEN IN SYSTEM.TEXT.JSON 9.0

Verlangsamte Fahrt

Microsofts JSON-Serialisierer, der hinsichtlich neuer Features in den letzten Versionen immer schnell voranschritt, bietet in der Version 9.0 lediglich vier Neuerungen.

Der JSON-Serializer `System.Text.Json` (NuGet-Paket `System.Text.Json`, siehe [1]) wird zusammen mit dem modernen .NET entwickelt und mit den aktuellen .NET-SDKs ausgeliefert, auch wenn er nicht daran gebunden ist. Im Unterschied zu anderen Teilen des modernen .NET (zum Beispiel Entity Framework Core oder ASP.NET Core) läuft `System.Text.Json` nicht nur auf modernen .NET-Versionen, sondern auch auf deutlich älteren: `System.Text.Json` 9.0 funktioniert mit dem klassischen .NET Framework ab Version 4.6.2 und dem modernen .NET ab Version 2.0.

.NET libraries

`System.Text.Json` has new options that let you customize the indentation character and size of written JSON. It also includes a new `JsonSerializerOptions.Web` singleton that makes it easier to serialize using web defaults.

Bild: Microsoft [3]

Spärliche Informationen über `System.Text.Json` 9.0 (Bild 1)

Stand von Code und Dokumentation

Die bei Erstellung dieses Beitrags aktuelle Version von `System.Text.Json` 9.0 ist „Release Candidate 1“ vom 10. September 2024. Bis zum Erscheinen von .NET 9.0 am 12. November 2024 sind noch Änderungen und Ergänzungen möglich.

● Listing 1: Setzen von `IndentCharacter` und `IndentSize` für die JSON-Serialisierung mit `System.Text.Json`

```
public void JSON_Indent()
{
    // Erstelle Objekt
    var consultant = new Consultant()
    {
        ID = 42,
        FullName = "Holger Schwichtenberg",
        Salutation = "Dr.",
        PersonalWebsite = "www.dotnet-doktor.de"
    };
    consultant.Languages = ["C#", "Visual Basic .NET",
        "JavaScript", "TypeScript"];

    // Serialisierungsoptionen festlegen
    var options = new JsonSerializerOptions
    {
        WriteIndented = true,

        IndentCharacter = ' ', // ACHTUNG: Als
        // IndentCharacter werden nur ein Leerzeichen
        // ' ' oder ein Tabulator '\t' unterstützt!!!
        IndentSize = 4,
    };

    CUI.Yellow("IndentCharacter = "
        + options.IndentCharacter + " | IndentSize = "
        + options.IndentSize);

    // Serialisieren in JSON
    var json1 = JsonSerializer.Serialize(
        consultant, options);

    // Ausgabe
    Console.WriteLine(json1);
}
```



Die Dokumentation zu `System.Text.Json` [2] ist zu diesem Zeitpunkt noch nicht aktualisiert worden, sondern entspricht dem Stand für .NET 8.0. Das Dokument „What's new in .NET 9“ [3] enthält lediglich zwei Sätze zu `System.Text.Json 9.0`, die nicht alle neuen Features erwähnen (Bild 1).

Einrückungen anpassen

Die Version 9.0 der JSON-Bibliothek `System.Text.Json` ermöglicht es, die Einrückung in der JSON-Zeichenkette über die `JsonSerializerOptions` anzupassen. Hierfür wurden zwei neue Einstellungen eingeführt: `IndentCharacter` und `IndentSize`. Listing 1 zeigt ein Beispiel; seine Ausgabe ist in Bild 2 zu sehen. Als `IndentCharacter` werden aber nur ein Leerzeichen (' ') oder ein Tabulator ('\t') unterstützt.

Einstellungen wie bei Web-APIs mit `JsonSerializerOptions.Web`

Durch die Verwendung der Eigenschaft `JsonSerializerOptions.Web` (siehe Listing 2) können Entwickler nun auf einfache Weise dieselben Einstellungen für die JSON-Serialisierung und -Deserialisierung wählen, die standardmäßig in ASP.NET-Core-basierten Web-APIs verwendet werden, vergleiche Bild 3 und die Ausgabe des Beispiels in Bild 4.

JSON-Schema exportieren

`System.Text.Json 9.0` besitzt eine neue Klasse `System.Text.Json.Schema.JsonSchemaExporter`, mit der man die Metadaten zu einem .NET-Typ in JSON-Form erhalten kann – in

```

IndentCharacter und IndentSize
{
    "Languages": [
        "C#",
        "Visual Basic .NET",
        "JavaScript",
        "TypeScript"
    ],
    "PersonalWebsite": "www.dotnet-doktor.de",
    "ID": 42,
    "FullName": "Holger Schwichtenberg",
    "Salutation": "Dr.",
    "Address": null
}

```

Ausgabe des Beispiels in Listing 1 (Bild 2)

der gleichen Form, wie diese auch bei der OpenAPI-Spezifikation eines Web-API zum Einsatz kommen.

Gegeben sei die Klasse in Listing 3. Die folgenden Codezeilen liefern die Metadaten für die Klasse `PersonWithAutoID`:

```

JsonNode schema1 = JsonSchemaExporter.GetJsonSchemaAsNode(
    JsonSerializerOptions.Default,
    typeof(PersonWithAutoID));
Console.WriteLine(schema1);

```

Dies liefert folgendes JSON-Dokument: ▶

● Listing 2: Serialisierungsoptionen aus ASP.NET-Core-Web-APIs übernehmen mit `JsonSerializerOptions.Web`

```

public void JSON_SerializerOptionsWeb()
{
    // Ausgabe der Serialisierungsoptionen, die
    // ASP.NET Core WebAPI verwendet
    CUI.PrintYellow(
        JsonSerializerOptions.Web.ToNameValueString());
    CUI.HR();

    // Erstelle Objekt
    var consultant = new Consultant()
    {
        ID = 42,
        FullName = "Holger Schwichtenberg",
        Salutation = "Dr.",
        PersonalWebsite = "www.dotnet-doktor.de"
    };
    consultant.Languages = ["C#", "Visual Basic .NET",
        "JavaScript", "TypeScript"];

    // Serialisieren
    var json = JsonSerializer.Serialize(
        consultant, JsonSerializerOptions.Web);
    // Ausgabe
    Console.WriteLine(json);
}

```



```
{
  "type": [
    "object",
    "null"
  ],
  "properties": {
    "ID": {
      "type": "integer"
    },
    "Name": {
      "type": [
        "string",
        "null"
      ]
    }
  }
}
```

Versieht man die Klasse *PersonWithAutoID* mit der folgenden Annotation

```
[JsonDerivedType(typeof(
    PersonWithAutoID),
    typeDiscriminator:
    "Person")]
```

erhält man in den Metadaten den Typ-Diskriminator als *\$type*:

```
{
  "type": [
    "object",
    "null"
  ],
  "required": [
    "$type"
  ],
  "anyOf": [
    {
      "properties": {
        "$type": {
          "const": "Person"
        },
        "ID": {

```

```
AllowOutOfOrderMetadataProperties: False
AllowTrailingCommas: False
Converters:
Default: System.Text.Json.JsonSerializerOptions
DefaultBufferSize: 16384
DefaultIgnoreCondition: Never
DictionaryKeyPolicy:
Encoder:
IgnoreNullValues: False
IgnoreReadOnlyFields: False
IgnoreReadOnlyProperties: False
IncludeFields: False
IndentCharacter:
IndentSize: 2
IsReadOnly: True
MaxDepth: 0
NewLine:

NumberHandling: AllowReadingFromString
PreferredObjectCreationHandling: Replace
PropertyNameCaseInsensitive: True
PropertyNamingPolicy: System.Text.Json.JsonCamelCaseNamingPolicy
ReadCommentHandling: Disallow
ReferenceHandler:
TypeInfoResolver: System.Text.Json.Serialization.Metadata.DefaultJsonTypeInfoResolver
TypeInfoResolverChain: System.Text.Json.Serialization.Metadata.DefaultJsonTypeInfoResolver
UnknownTypeHandling: JsonElement
UnmappedMemberHandling: Skip
Web: System.Text.Json.JsonSerializerOptions
WriteIndented: False
```

Die konkreten Einstellungen in JsonSerializerOptions.Web (Bild 3)

JsonSerializerOptions.Web

```
{"languages":["C#","Visual Basic .NET","JavaScript","TypeScript"],"personalWebsite":
"www.dotnet-doktor.de","id":42,"fullName":"Holger Schwichtenberg","salutation":"Dr."
,"address":null}
```

Ausgabe des Beispiels in Listing 2 (Bild 4)

● Listing 3: PersonWithAutoID

```
public class PersonWithAutoID
{
    int counter = 0;

    private int id;

    public partial int ID
    {
        get
        {
            if (id == 0) id = ++counter;
            return id;
        }
    }

    set
    {
        if (ID > 0) throw new ApplicationException(
            "ID ist bereits gesetzt");
        id = value;
    }

    public partial void Print()
    {
        Console.WriteLine($"{this.ID}: {this.Name}");
    }
}
```

● Listing 4: JsonSchemaExporter mit Optionen

```

var options = new JsonSerializerOptions
{
    TypeInfoResolver = new DefaultJsonTypeInfoResolver(),
    IndentSize = 5,
    WriteIndented = true,
    PropertyNamingPolicy = JsonNamingPolicy.SnakeCaseLower
};

var schemaOptions = new JsonSchemaExporterOptions
{
    TransformSchemaNode = (context, node) =>
    {
        Console.WriteLine(context.TypeInfo);

        if (context.TypeInfo.Type == typeof(int))
        {
            node["min"] = "1";
            node["max"] = "1000000000";
        }
        return node;
    };

    JsonNode schema2 =
        JsonSchemaExporter.GetJsonSchemaAsNode(
            options, typeof(PersonWithAutoID), schemaOptions);
    Console.WriteLine(schema2);
}

```

```

    "type": "integer"
  },
  "Name": {
    "type": [
      "string",
      "null"
    ]
  }
}

```

Anpassungen für die Schemagenerierung sind möglich. So sorgt der Programmcode in Listing 4 dafür, dass bei allen Integer-Zahlen ein Wertebereich via "min" und "max" im ►

DEV ACADEMY

HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN

3 TAGE
28.-30.10.2024

Clean Code und Software Design

TRAINING!

Was wird behandelt

- Auswirkungen von schlechtem Code
- Grundprinzipien von Clean Code
- Entwurfsprinzipien und -muster
- Grundregeln für strukturierten Quellcode
- Analysetechniken
- Clean Code im Team



David Tielke
Trainer

Listing 5: Einsatz von RespectNullableAnnotations

```
#nullable enable

namespace NET9_Console.FCL90;

internal class FCL9_JSON
{
    record Entwickler(int ID, string Name);

    public void JSON_NullableAnnotations()
    {
        JsonSerializerOptions options = new() {
            RespectNullableAnnotations = true };

        CUI.H1("Serialisieren von einem Objekt mit
            Not-Nullable-Property");
        try
        {
            var json = JsonSerializer.Serialize(
                new Entwickler(42, null), options);
            Console.WriteLine(json);
        }
        catch (Exception ex)
        {
            CUI.Error(ex);
        }
    }
}
```

JSON-Schema angegeben wird. Die Ausgabe für die Property "ID" im JSON-Schema ist dann:

```
"ID": {
  "type": "integer",
  "min": "1",
  "max": "100000000"
```

Nullable-Kontext kann beachtet werden

Beim Serialisieren und Deserialisieren kann System.Text.Json 9.0 nun auch beachten, ob eine Property den Wert null verbietet. Wenn null verboten ist und die neue Serialisierungsoption *RespectNullableAnnotations* auf *true* steht, erzeugt die Serialisierung beziehungsweise Deserialisierung einen Laufzeitfehler, siehe [Listing 5](#).

Die Versuche, das Entwickler-Objekt zu serialisieren beziehungsweise zu deserialisieren, scheitern, weil durch den Nullable-C#-Kontext, der zu Beginn der Datei mit *#nullable enable* aktiviert wird, die Eigenschaft *Name* in der *Record*-Klasse *Entwickler* keinen Null-Wert erlaubt. In beiden Fällen kommt es zum Laufzeitfehler: „The constructor parameter 'Name' on type 'NET9_Console.FCL90.FCL9_JSON+Entwickler' doesn't allow null values.“

Sofern statt *#nullable enable* die Annotation *[DisallowNull]* eingesetzt wird, kommt es lediglich zu einem Laufzeitfehler bei der Deserialisierung, während die Serialisierung funktioniert:

```
record Entwickler(int ID, [DisallowNull] string Name);
```

Schön ist: Man kann den JSON-Serialisierer mit einer Einstellung in der Projektdatei (*.csproj*) global dazu bewegen, diese neue Prüfung durchzuführen:

```
<ItemGroup>
  <RuntimeHostConfigurationOption Include=
    "System.Text.Json.JsonSerializerOptions
      .RespectNullableAnnotations" Value="true" />
</ItemGroup>
```

- [1] NuGet-Paket *System.Text.Json*, www.dotnetpro.de/SL2412DataAccess1
- [2] Microsoft Learn, *JSON serialization and deserialization (marshalling and unmarshalling) in .NET – overview*, www.dotnetpro.de/SL2412DataAccess2
- [3] Microsoft Learn, *What's new in .NET 9*, www.dotnetpro.de/SL2412DataAccess3

Dr. Holger Schwichtenberg

alias der „Dotnet-Doktor“ ist Leiter des Expertennetzwerks www.IT-Visions.de, das mit 53 renommierten Experten mittlere und große Unternehmen durch Beratungen und Schulungen sowie bei der Softwareentwicklung unterstützt. Durch seine Fachbücher und Vorträge auf Fachkonferenzen gehört er zu den bekanntesten Experten für .NET und Webtechniken in Deutschland. www.dotnet-doktor.de

dnpCode A2412DataAccess

DACPAC UND BACPAC AUS ENTWICKLERSICHT

Handlich verpackt

Ein Transfer-Dateiformat ermöglicht die fehlerfreie Synchronisierung eines Datenbankschemas mit einer Datei, einer Codebasis, einer anderen Datenbank – und umgekehrt.

Als .NET-Entwickler hat man relativ häufig auch mit dem Microsoft SQL Server zu tun. Bei der täglichen Arbeit kam eine Frage auf, und die Suche nach einer Antwort gab zugleich den Anstoß für diesen Artikel: Wie kann ich meine Daten elegant von einem SQL Server zu einem anderen oder in die Cloud übertragen? Kann man das automatisieren, oder gibt es da eigentlich auch ein API dafür?

Alle Datenbankadministratoren werden jetzt vermutlich lachend vom Stuhl fallen, tatsächlich aber war das zumindest für mich – als .NET-Entwickler – durchaus eine Frage.

In diesem Artikel sehen wir uns daher etwas genauer die Datenformate DACPAC und BACPAC an, und wir werden klären, was Microsoft unter einer Data-Tier Application versteht und welche Möglichkeiten diese uns bietet.

BAK-Format zum Exportieren und Importieren?

Der erste Gedanke zu der ursprünglichen Frage geht vermutlich in Richtung einer Backup-Datei (Dateiendung *.bak*). Natürlich lässt sich über das SQL Server Management Studio eine BAK-Datei der Datenbank erstellen, allerdings sind damit gewisse Einschränkungen verbunden. So lassen sich BAK-Dateien aus einer neueren SQL-Server-Version nicht auf einem älteren SQL Server einspielen. Bei einer Migration in die Cloud kann der Import ebenfalls erschwert oder gänzlich unmöglich sein. Azure SQL etwa unterstützt keinen BAK-Import.

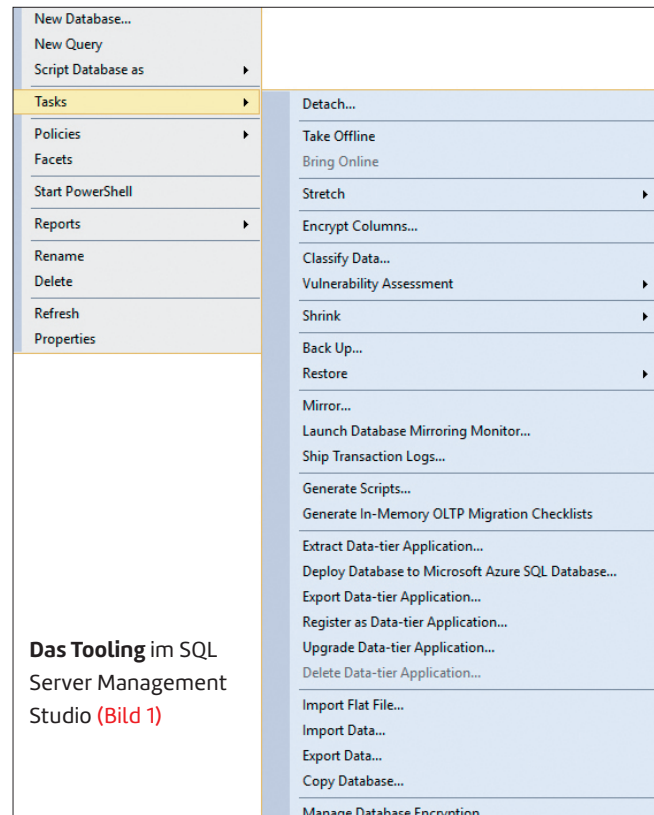
Grundsätzlich sind BAK-Dateien Datenbankabbilder und funktionieren als solche prächtig für alltägliche Backups. Als Transportformat sind sie allerdings nur mäßig gut geeignet.

Data-Tier Application

Blickt man genauer in die Kontextmenüs des SQL Server Management Studios (SSMS), gibt es dort ein paar Vertreter, die auf ein „Kopieren“ hindeuten.

Da ist zunächst die bereits angesprochene Variante über *Backup & Restore*. Daneben gibt es (zumindest im SSMS) noch *Export Data* und *Copy Database*, aber diese Optionen haben – zumindest bei meinen Versuchen – entweder nur mäßige oder gar seltsame Resultate zutage gefördert. Es kann aber sein, dass dies bei einfacheren Datenmodellen durchaus gut funktioniert.

Eine etwas kryptisch anmutende Kategorie enthält allerdings eine Funktion, die genau für das hier angesprochene Problem eine Lösung bietet: *Extract Data-tier Application...* beziehungsweise *Export Data-tier Application...* zum Exportieren sowie *Register as Data-tier Application...* beziehungsweise *Upgrade Data-tier Application...* zum Importieren (Bild 1).



Das Tooling im SQL Server Management Studio (Bild 1)

DACPAC

Die Option *Extract Data-tier Application...* erzeugt eine DACPAC-Datei, die in der Regel nur die Schema-Informationen der Datenbank enthält.

BACPAC

Im Unterschied zu DACPAC beinhaltet das Format BACPAC neben den Schema- und Daten-Infos auch Login-Informationen und fungiert eher als eine Art klassisches Backup, ähnlich einer BAK-Datei. Eine solche Datei wird ebenfalls über den Menüpunkt *Export Data-tier Application...* erzeugt.

Möchte man eine komplette Datenbank von A nach B kopieren, ist dies über BACPAC durchaus möglich, doch dies hat nicht den Anspruch, ein komplettes Backup zu ersetzen.

Im Unterschied zu einer BAK-Datei wird die BACPAC-Datei sequenziell Tabelle für Tabelle geschrieben, und Änderungen während des Erstellungsprozesses werden einfach mit exportiert, anders als bei einer BAK-Datei, die als Backup genau einen Snapshot zu genau einem Zeitpunkt einfangen kann. ►

Einsatzzweck

Wenn also eine DACPAC-Datei nur das Schema enthält und eine BACPAC-Datei zwar eine Art Backup darstellt, aber durchaus Probleme mit sich bringt, wofür sind die beiden denn dann gut?

Der Ursprung kommt aus der SQL-Entwicklerwelt, um eine Art Paket zu haben, das sich einfach verteilen lässt und das Änderungen sauber in einem Entwicklungsprozess abbilden kann. Vor diesem Hintergrund ist auch der Name entstanden, denn DACPAC steht für „Data Application Component – Package“. Hinzu kommt, dass zum Beispiel der SQL-Azure-Dienst keine BAK-Dateien nativ importieren kann, jedoch mit DACPAC und BACPAC umgehen kann.

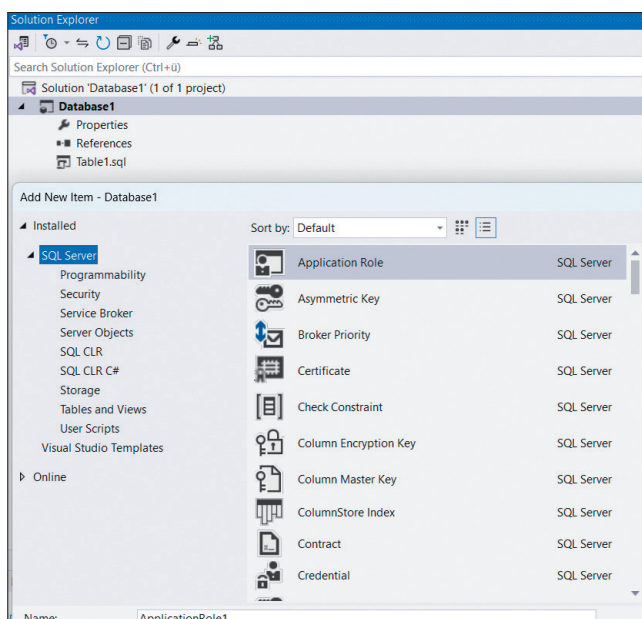
Tooling in Visual Studio

Wie bereits gezeigt, kann ein DACPAC ebenso wie ein BACPAC über das SQL Server Management Studio erstellt werden. Möchte man die SQL-Datenbank, die neben Schema und Daten auch Trigger oder Stored Procedures beinhalten kann, über einen eher entwicklungstypischen Prozess aufsetzen, gibt es in Visual Studio eigens ein SQL Database Project, das als Build-Artefakt ein DACPAC erzeugt (Bild 2).

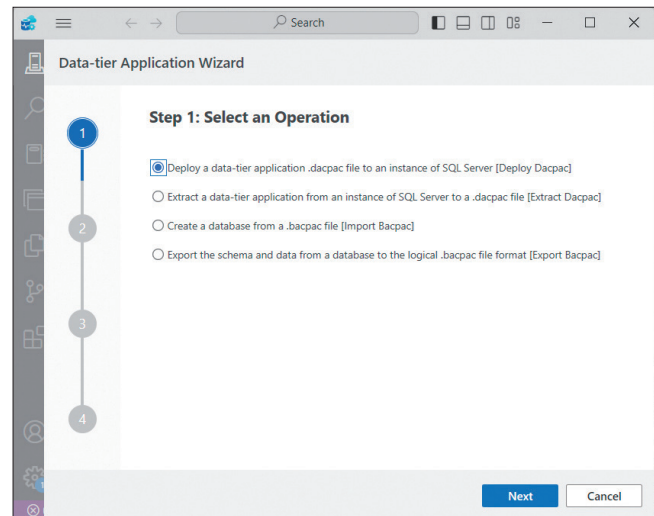
Über diesen Weg ist es möglich, solche Pakete zu bauen und so zum Beispiel Schemaänderungen von der Entwicklungsumgebung hin zu den produktiven Systemen zu bringen. Das Wording, um Änderungen zu publizieren, ist auf den ersten Blick etwas merkwürdig, aber man muss das Ziel eines DACPAC-Imports eher als Applikation begreifen, und dann ergibt „Deploy“ auch tatsächlich Sinn.

Tooling im Azure Data Studio

Neben dem SQL Server Management Studio (SSMS) kann man über das recht junge Azure Data Studio ebenfalls DACPAC- und BACPAC-Dateien erstellen oder importieren. Der Funktionsumfang entspricht aber – zumindest aktuell –



Das Tooling in Visual Studio (Bild 2)



Das Tooling im Azure Data Studio (Bild 3)

so ziemlich dem, was man auch im SSMS findet. Der Hauptunterschied liegt darin, dass die Beschreibungen hier präziser sind und man nicht sofort von einem riesigen Kontextmenü von Optionen erschlagen wird (Bild 3).

SqlPackage.exe zur Automatisierung

Wer lieber ein Tool nutzen möchte, um diese Prozesse zu automatisieren, kann SqlPackage.exe einsetzen. Dieses Werkzeug kann man entweder direkt bei der SQL-Installation mit installieren lassen oder separat installieren. Am einfachsten geht der Weg als .NET Tool:

```
dotnet tool install -g microsoft.sqlpackage
```

Danach steht einem über *sqlpackage* ein sehr mächtiges Werkzeug zur Erzeugung und zum Importieren von DACPAC-beziehungsweise BACPAC-Dateien zur Verfügung (Bild 4).

.NET API

Die bisher gezeigten Möglichkeiten sind insbesondere für Datenbankadministratoren ein Segen. Allerdings geht es in dem Artikel ja um Entwickler, und für diese gibt es das NuGet-Paket DacFx, um via Code DACPAC-Dateien zu erstellen oder zu deployen. Hat man DacFx einmal hinzugefügt, ist das Erzeugen einer Datei äußerst einfach:

```
string connectionString = "...";
DacServices dacServices = new(connectionString);
DacExtractOptions options = new DacExtractOptions
{
    ExtractAllTableData = true,
    IgnorePermissions = true,
    IgnoreUserLoginMappings = true,
};

Version version = new Version("1.0");
dacServices.Extract(@"C:\Temp\",
    "DatabaseName",
    "AppName",
    version, extractOptions: options);
```

```

PS C:\Users\muehsig> sqlpackage
SqlPackage: Command-line tool for creating and deploying SQL Server databases and DACPAC packages. (16
2.3.566.1)
Copyright (c) 2024 Microsoft Corporation. All rights reserved.

Help for dynamic property usage.
/@@<file>:<string>
    Read response file for more options.

/help:[True|False]
    (short form /?)

/version:[True|False]
    Display the tool version in use.

Help for command actions.
/AccessToken:<string>
    Specifies the token-based authentication access token to use when connect to the target database.
    (short form /at)

/Action:{Extract|DeployReport|DriftReport|Publish|Script|Export|Import}
    Specifies the action to be performed. (short form /a)

/AzureCloudConfig:<string>
    Specifies the custom endpoints for connecting to Azure Active Directory in the format:
    "AzureActiveDirectoryAuthority={value};DatabaseServicePrincipalName={value}". (short form /acc)

/AzureKeyVaultAuthMethod:{Interactive|ClientIdSecret}
    Specifies what authentication method is used for accessing Azure Key Vault if a publish operation
    includes
    modifications to an encrypted table/column. (short form /akv)

```

SqlPackage.exe im Einsatz (Bild 4)

Wichtig zu erwähnen ist hierbei, dass die Datei direkt auf dem Client erzeugt wird. Bei einem klassischen SQL-Backup muss man wesentlich genauer überlegen, wohin das resultierende Backup geschrieben wird und ob der aufrufende Client auf diesen Pfad Zugriff hat. Über DacFx entfällt das komplett.

Das Importieren einer Datei geschieht über folgende Codezeilen:

```

string connectionString = "";
DacServices dacServices = new(connectionString);
DacPackage dacpac = DacPackage.Load(
    @"C:\Temp\test.dacpac");

DacDeployOptions options = new()
{
    IgnorePermissions = true,
    IgnoreUserSettingsObjects = true,
    IgnoreLoginSids = true,
    IgnoreRoleMembership = true,
    AllowIncompatiblePlatform = true,
    ExcludeObjectTypes = new[]
    {
        ObjectType.Users,
        ObjectType.Logins,
        ObjectType.RoleMembership
    }
};

```

```

dacServices.Deploy(
    dacpac,
    "DatabaseName",
    true,
    options);

```

Optionen

Es scheint zwar, als wären DACPAC und BACPAC völlig verschiedene Dateiformate, aber tatsächlich ist der Grundauf-

bau bei beiden identisch. Es ist bei beiden ein gezippter Ordner, der aus einer Handvoll XML-Dateien und den eigentlichen Daten besteht. Letztere sind in einem binären Format hinterlegt.

Grundsätzlich kann man auch in einem DACPAC über die Eigenschaft *ExtractAllTableData* Daten exportieren, und diese werden dann auch wieder importiert. Allerdings kann dann nicht jedes Tooling mit den Daten umgehen. Ein DACPAC mit Daten wird zum Beispiel vom Visual-Studio-Tooling als invalide betrachtet.

Welche Eigenschaften zur Verfügung stehen, sieht man am besten in der Dokumentation

von SqlPackage.exe und dem jeweiligen Befehl, zum Beispiel bietet *Extract* noch diverse zusätzliche Eigenschaften. Alle diese Optionen können auch via DacFx genutzt werden.

Kombination mit dem Entity Framework und Co.

Wer in seinen Projekten das Entity Framework oder einen ähnlichen OR-Mapper einsetzt, der die Datenbank-Entitäten auch via Code erstellen kann, wird vermutlich DACPAC im Sinne eines Applikationspakets nicht unbedingt benötigen. Grundsätzlich kann man natürlich SQL Database Projects einsetzen, um Migrationen so abzubilden, aber hier ist die Antwort vermutlich wie so oft: Es kommt darauf an. Besteht ein Kunde auf einen „sauberen“ Migrationsprozess der Datenbank und möchte er ein DACPAC, um Schemaänderungen so auszurollen, ist das ein valider Anwendungsfall. Meist kommt man aber über einen Code-First-Ansatz bereits sehr weit.

Fazit

Eine Data-Tier Application ist nichts, wovor man Angst haben müsste. Möchte man eine Datenbank designen und in einem geordneten Entwicklungsprozess abbilden, eignet sich das SQL-Database-Projekt – zumindest überall dort, wo Entity Framework und Co. nicht im Einsatz sind.

DACPAC- und BACPAC-Dateien lassen sich über diverse Werkzeuge erstellen, und per API ist dies auch als Entwickler einfach in bestehende Anwendungen integrierbar. ■



Robert Mühsig

ist Softwareentwickler und arbeitet vornehmlich mit dem Microsoft-Technologie-Stack. Durch sein Community-Engagement – von User Groups bis hin zu Blogs – ist er seit mehr als 15 Jahren Microsoft MVP.

robert.muehsig@codeinside.eu

dnpCode

A2412BacPac



LERNSTOFF

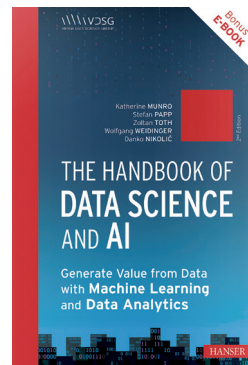
dnpCode A2412Lernstoff

MACHINE LEARNING UND DATA ANALYTICS

KI-Handbuch

Dieses umfassende Werk richtet sich an Fachleute, Studierende und Interessierte im Bereich Data Science und künstliche Intelligenz. Hauptziel des Handbuchs ist es, einen ganzheitlichen Überblick über die Felder Datenanalyse und Machine Learning zu liefern. Die Autoren streben danach, sowohl theoretische Grundlagen als auch praktische Anwendungen zu vermitteln und dabei die neuesten Entwicklungen und Trends in diesen sich schnell entwickelnden Bereichen zu berücksichtigen. Besonders hervorzuheben ist die Behandlung ethischer Fragen und rechtlicher Rahmenbedingungen sowie die Präsentation von Anwendungsfällen aus unterschiedlichen Branchen.

Die Neuauflage zeichnet sich durch ihre Aktualität aus, indem sie die neuesten Entwicklungen und Trends berücksichtigt. Trotz der Komplexität der Themen ist das Buch verständlich geschrieben und eignet sich sowohl für Einsteiger als auch für erfahrene Praktiker. Es verbindet theoretische Grundlagen mit praktischen Anwendungen und legt Wert



K. Munro, S. Papp, Z. Toth, W. Weidinger, D. Nikolic und weitere

The Handbook of Data Science and AI
Hanser, 2nd Edition, 2024, 876 Seiten,
Englisch, ISBN 978-1-56990-235-6,
79,99 Euro

Zielgruppe: Softwareentwickler,
Data Scientists, KI-Entwickler

Voraussetzungen: keine

Leseprobe:
www.dotnetpro.de/SL2412Lernstoff1

auf Interdisziplinarität, indem es Konzepte aus verschiedenen Fachbereichen verknüpft und viele Beispiele aus der realen Welt liefert.

Damit ist das Handbuch eine wertvolle Ressource für Leser, die ihr Wissen im Bereich Data Science und KI vertiefen oder erweitern möchten, und ein unverzichtbares Werk für alle, die sich professionell mit Data Science und KI beschäftigen. Für Dezember 2024 plant der Verlag eine erweiterte deutschsprachige Neuauflage des Handbuchs. (fd) ■

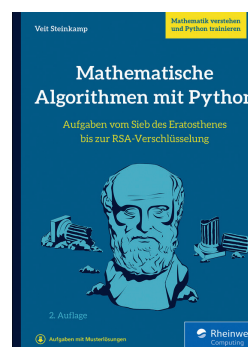
MATHEMATISCHES TRAINING

Py-Algorithmen

Dank professioneller Bibliotheken eignet sich Python optimal für mathematische Aufgaben. Dieses Buch bietet ein systematisches Trainingsprogramm für die Codierung an, das quasi nebenbei eine fundierte Einführung in die Programmiersprache einschließt. Andererseits führen praktische Erfahrungen mit leistungsfähigen Algorithmen zu einem vertrauteren Umgang mit der Mathematik.

Zunächst werden Datenstrukturen wie Tupel, Sets, Listen und Dictionaries behandelt, anschließend die maßgeblichen Programmstrukturen Sequenz, Verzweigung und Wiederholung, jeweils an einfachen Beispielen. Im nächsten Schritt stellt der Autor die für mathematische Lösungen entworfenen Erweiterungsmodule NumPy, Matplotlib, SymPy und SciPy kurz vor, die in den weiteren Kapiteln zum Einsatz kommen.

Die nachfolgenden Abschnitte führen auf dieser Basis die Codierung bekannter Algorithmen in leicht nachvollziehba-



Veit Steinkamp

Mathematische Algorithmen mit Python
mitp, 2024, 567 Seiten, Deutsch,
ISBN 978-3-367-10005-7, 34,90 Euro,
E-Book 24,90 Euro

Zielgruppe: Python-Training

Voraussetzungen: keine

Leseprobe:
www.dotnetpro.de/SL2412Lernstoff2

rer Weise vor. Den Anfang machen Algorithmen mit Zahlen, vom Sieb des Eratosthenes bis zur eulerschen Zahl. Zusätzliche Themen sind die Lösung von Gleichungssystemen, die Visualisierung von Folgen sowie die Nullstellenberechnung. Es folgen die klassischen Themen der Differenzialrechnung, die Visualisierung von Reihen, die numerische Integration, der Umgang mit Differenzialgleichungen, Regressionsrechnungen und die Berechnung statistischer Kennzahlen. Den Abschluss bildet ein Kapitel über Fraktale. (vh) ■



DATENSCHUTZ IN PROJEKTEN

Privatheit wahren

Dieser Titel zielt darauf ab, sich den durch KI und Big Data wachsenden Herausforderungen mit Leidenschaft entgegenzustemmen, damit die Nutzung von Daten etwa für die Bewältigung von Krisen oder Epidemien nicht behindert wird. Angesprochen sind dabei vor allem Data-Science-Fachleute. Nach einer kurzen Einführung zur Bedeutung von Data Governance werden als brauchbare Verfahren zur Anonymisierung von Daten die Lösungen unter dem Label „Differential Privacy“ beschrieben, die einerseits die Privatsphäre schützen, aber andererseits den Informationsgehalt der Daten erhalten sollen. Anschließend wird demonstriert, wie solche Methoden in Datenpipelines integriert werden können.

Für typische Formen von Angriffen auf die Privatsphäre werden Erfolg versprechende Vorkehrungen vorgestellt. Ein eigenes Kapitel geht auf die Möglichkeiten ein, Datenschutz in umfangreichen Machine-Learning-Projekten zu gewährleisten. Als mögliche Alternative zur zentralisierten Massen-datenverarbeitung werden unter dem Begriff „Federated



Katharine Jarmul

Data Privacy in der Praxis. Datenschutz und Sicherheit in Daten- und KI-Projekten

O'Reilly, 2024, 414 Seiten, Deutsch, ISBN 978-3-96009-233-9, 49,90 Euro, E-Book 39,90 Euro

Zielgruppe: Data-Science-Fachleute

Voraussetzungen: keine

Leseprobe:

www.dotnetpro.de/SL2412Lernstoff3

Learning“ Verfahren beschrieben, bei denen die Daten lokal verteilt gehalten werden. Unter „Encrypted Computation“ werden neue Ansätze vorgestellt, Berechnungen mit verschlüsselten Daten anzustellen, ohne sie vorher entschlüsseln zu müssen.

Ein eigenes Kapitel ist den rechtlichen Aspekten gewidmet, die insbesondere durch die umfangreiche Datenschutzverordnung ausdifferenziert worden sind. Abschließend geht es um die praktische Umsetzung des Datenschutzes. (vh) ■

Kurz vorgestellt

Judith Andresen

Agile Organisationsentwicklung

Die Autorin beschreibt, wie man agile Organisationen entwickelt, dazu den Veränderungsdruck positiv nutzt und den häufigsten Stolperfallen der agilen Organisationsentwicklung aus dem Weg geht. Mit zahlreichen Fallbeispielen sowie Kontrollfragen.

Hanser, 2024, 432 Seiten, Deutsch, ISBN 978-3-446-46336-3, 39,99 Euro

Maurice Knopp

IT-Freelancer

Der Autor, IT-Consultant und Softwareentwickler Maurice Knopp teilt in diesem Buch seine langjährigen Erfahrungen. Er liefert Tools und Tipps für die Durchführung der ersten Projekte sowie Hinweise für die nachhaltige Planung einer Selbstständigkeit. Mit Praxistipps zu Finanzen und Akquise.

Rheinwerk, 2024, 261 Seiten, Deutsch, ISBN 978-3-367-10359-1, 34,90 Euro

Wolfgang Keller

IT-Unternehmensarchitektur

Das Handbuch beschreibt, wie das Softwareportfolio eines Unternehmens optimiert werden kann – vom Erarbeiten der IT-Strategie bis zur Tagesarbeit der IT- und Architektur-Governance. Die Neuauflage berücksichtigt unter anderem das Open-Source-Tool EDGY und Muster für digitale Strategien.

dpunkt.verlag, 4. Auflage, 2024, 566 Seiten, Deutsch, ISBN 978-3-86490-992-4, 54,90 Euro

Erik Schönwälder

SQL Schnelleinstieg

Das Buch verspricht einen einfachen Einstieg in das Datenbankmanagement mit der Abfragesprache SQL. Alle Grundlagen werden Schritt für Schritt anhand kurzer Beispiele und leicht verständlich erläutert – inklusive der Besonderheiten von MySQL und PostgreSQL.

mitp, 2024, 300 Seiten, Deutsch, ISBN 978-3-7475-0868-8, 19,99 Euro

EIN PASSWORTMANAGER AUS ENTWICKLERSICHT

Tresore per Schnittstelle

Passwortmanager erfreuen sich steigender Beliebtheit; viele bieten umfangreiche APIs an. Am Beispiel von Bitwarden zeigt der Artikel, was möglich ist.

Ob beim Online-Banking, in den sozialen Medien oder bei E-Mail-Konten, überall gilt es persönliche Informationen vor unbefugtem Zugriff zu schützen. Weit verbreitet als Zugriffsschutz sind Passwörter in Kombination mit E-Mail-Adressen oder Benutzernamen. Aufgrund der steigenden Zahl an Konten und, im besten Fall, immer komplexeren Passwörtern wird es immer schwieriger, den Überblick zu behalten. Hier bieten sich Passwortmanager wie 1Password, LastPass und Bitwarden an, um diese wachsende Menge an Zugangsdaten zu verwalten.

Der Zugriff auf Passwortmanager ist schon lange nicht mehr nur über den Browser, Desktop- oder mobile Anwendungen möglich, sondern auch per Schnittstellen. Das erlaubt es, die Tresore dieser Passwortmanager in Anwendungen zu nutzen, um darüber Passwörter zu generieren und zu verwalten. Zusätzlich bieten viele Tools Management-APIs an, damit auch die Verwaltung der Passwortmanager über Schnittstellen automatisierbar ist.

Der Artikel wirft in diesem Kontext einen Blick auf Bitwarden und zeigt im Folgenden, wie Sie damit per CLI, SDK oder RESTful-API Tresore verwalten und beispielsweise auf gesicherte Einträge wie Passwörter zugreifen sowie neue Einträge erstellen.

Was ist Bitwarden?

Bitwarden ist eine Softwarelösung, die dazu dient, Passwörter und andere vertrauliche Informationen zu speichern, zu verwalten und zu schützen. Bitwarden hebt sich durch seine Open-Source-Natur und umfassenden Sicherheitsfunktionen von vielen anderen Passwortmanagern ab und will damit eine vertrauenswürdige Lösung für die Verwaltung digitaler Identitäten bieten.

Die Hauptfunktion besteht darin, alle Passwörter des Benutzers in einem verschlüsselten Tresor zu speichern, auf den nur der Benutzer zugreifen kann. Bitwarden verwendet moderne Verschlüsselungstechnologien, um sicherzustellen, dass die gespeicherten Daten vor unbefugtem Zugriff geschützt sind. Im Kasten **Zero-Knowledge Encryption** finden Sie hierzu weitere Informationen.

Die Funktionsweise von Bitwarden ist einfach und benutzerfreundlich. Benutzer können ihre Passwörter und andere sensible Daten in Bitwarden eingeben und kategorisieren. Diese Daten sind dann verschlüsselt gesichert, was bedeutet, dass selbst im Fall eines Datenlecks die Informationen nahezu unzugänglich bleiben. Benutzer können von verschiedenen Geräten und Plattformen aus auf ihre gespeicherten

Daten zugreifen, was Bitwarden zu einer flexiblen und praktischen Lösung macht. Neben der Passwortverwaltung bietet Bitwarden außerdem die Möglichkeit, sichere Notizen zu speichern. Diese Funktion ist nützlich, um sensible Informationen wie PINs, Lizenzschlüssel oder persönliche Notizen sicher zu verwahren. Genau wie bei Passwörtern sind diese Notizen mit starker Verschlüsselung geschützt und nur für den Benutzer zugänglich.

Ein weiterer wichtiger Aspekt von Bitwarden ist die Multi-Plattform-Unterstützung. Bitwarden ist auf einer Vielzahl von Geräten und Betriebssystemen verfügbar, darunter Windows, macOS, Linux, Android und iOS. Es gibt auch Browser-Erweiterungen für alle gängigen Webbrowser, die es Benutzern ermöglichen, ihre Passwörter direkt in ihrem Browser zu verwalten. Diese breite Unterstützung stellt sicher, dass Benutzer ihre Passwörter und alle anderen gespeicherten Informationen immer und überall zur Verfügung haben.

Die Nutzung eines Passwortmanagers wie Bitwarden bietet zahlreiche Vorteile. Einer der offensichtlichsten ist die erhöhte Sicherheit. Statt sich viele verschiedene Passwörter merken zu müssen, können Benutzer starke, einzigartige Passwörter für jede Anwendung erstellen und sicher in Bitwarden speichern. Dies reduziert das Risiko von Sicherheitsverletzungen erheblich, da schwache und wiederverwendete Passwörter eine der häufigsten Ursachen für Datenlecks sind.

Ein weiterer Vorteil ist die Bequemlichkeit. Passwortmanager speichern nicht nur Passwörter, sondern auch andere sensible Informationen wie Kreditkartendaten, sichere Notizen

● Zero-Knowledge Encryption

Bitwarden verwendet eine Zero-Knowledge-Verschlüsselung, was bedeutet, dass das Unternehmen die Daten seiner Nutzer nicht einsehen kann. Dies wird durch eine Ende-zu-Ende-Verschlüsselung der Tresordaten mit AES-CBC 256 Bit und durch die Verwendung von PBKDF2 SHA-256 / Argon2id zur Ableitung des Verschlüsselungsschlüssels erreicht [7]. Die Codebasen der PC-Clients, der mobilen Anwendungen und des Servers sind Open Source [8]. Jährlich führen Dritte Sicherheitsprüfungen durch. Zudem ist ein Programm zur Offenlegung von Schwachstellen eingerichtet [9]. Bitwarden ist konform mit HIPAA, GDPR, CCPA, SOC 2, SOC 3 sowie dem EU-US und Swiss-US Privacy Shield Framework [10].

und persönliche Dokumente. Dies ermöglicht es Benutzern, schnell und einfach auf alle wichtigen Informationen zuzugreifen, ohne sich diese merken oder sie unsicher aufbewahren zu müssen.

Der Open-Source-Ansatz

Die Transparenz, die durch Open-Source-Software gewährleistet wird, ist ein wesentlicher Vorteil. Da der Quellcode von jedem überprüft werden kann, ist es möglich, Sicherheitslücken und Schwachstellen schnell zu identifizieren und zu beheben. Dies erhöht das Vertrauen der Benutzer, da sie sicher sein können, dass keine versteckten Hintertüren oder unsicheren Praktiken im Code vorhanden sind. Darüber hinaus fördert

Das Bitwarden-API bietet umfassenden Zugriff auf die Funktionen von Bitwarden. Mit dem API lassen sich zahlreiche Aufgaben automatisieren und Bitwarden nahtlos in andere Systeme integrieren. Das API ist gut dokumentiert und beinhaltet zwei Hauptbereiche: das Organization Management API und das Vault Management API.

Das **Organization Management API** ermöglicht es, organisatorische Aspekte von Bitwarden zu verwalten. Dieses API ist besonders nützlich für Unternehmen, die eine große Anzahl von Benutzern und Daten verwalten müssen. Zu den Hauptfunktionen gehören:

- Benutzerverwaltung: Hinzufügen, Entfernen und Verwalten von Benutzern innerhalb einer Organisation.

```

Windows PowerShell
Fabia@BELLEROPHON E:\git\bitwarden-demo [10:17:40]
> .\bw.exe login
? Email address: fabian@
? Master password: [hidden]
You are logged in!

To unlock your vault, set your session key to the 'BW_SESSION' environment variable. ex:
$ export BW_SESSION="Si
> $env:BW_SESSION="Siv

You can also pass the session key to any command with the '--session' option. ex:
$ bw list items --session Si
Fabia@BELLEROPHON E:\git\bitwarden-demo [10:18:02]
>
  
```

Login im Bitwarden-CLI mittels E-Mail-Adresse und Passwort (Bild 1)

die Transparenz eine aktive Community von Entwicklern und Sicherheitsexperten, die kontinuierlich daran arbeiten, die Sicherheit und Funktionalität der Software zu verbessern.

Der Quelltext der Bitwarden-Projekte steht auf GitHub [1] zur Verfügung; die gleichnamige Organisation auf GitHub verwaltet die Repositories zum Bitwarden-Server, dem Web-Interface, den mobilen Versionen, der Browser-Version und dem CLI. Bitwarden lässt sich zudem selbst hosten, sodass jeder den Bitwarden-Server auf eigenen Servern installieren und betreiben kann.

Dies ist besonders vorteilhaft für Unternehmen oder Privatpersonen, die volle Kontrolle über ihre Daten haben möchten und daher keine Daten in der Cloud eines Drittanbieters speichern wollen. Die Selbst-Hosting-Option bietet somit maximale Kontrolle und Sicherheit für die Benutzerdaten. Bitwarden stellt ausführliche Anleitungen und Dokumentationen zur Verfügung, wie man den Server auf verschiedenen Plattformen einrichtet.

Integrationsmöglichkeiten

Bitwarden bietet eine Vielzahl von Integrationsmöglichkeiten, die es Benutzern und Entwicklern erlauben, die Funktionalität der Passwortverwaltung in verschiedene Anwendungen und Workflows zu integrieren. Aufgrund dieser Flexibilität und Anpassbarkeit ist Bitwarden eine Option sowohl für individuelle Nutzer als auch für Unternehmen, die nach maßgeschneiderten Lösungen suchen.

- Gruppenmanagement: Erstellen und Verwalten von Gruppen, um den Zugriff auf bestimmte Tresorelemente zu steuern.
- Rollen und Berechtigungen: Festlegen von Rollen und Berechtigungen für Benutzer, um sicherzustellen, dass jeder nur Zugang zu den Daten hat, die er tatsächlich benötigt.
- Audit-Logs: Überwachung und Protokollierung von Aktivitäten innerhalb der Organisation, um Sicherheitsvorfälle schnell zu identifizieren und zu beheben.

Das **Vault Management API** bietet direkten Zugriff auf die in Bitwarden gespeicherten Daten. Entwickler können automatisierte Workflows erstellen und Daten nahtlos zwischen verschiedenen Systemen austauschen. Zu den wesentlichen Funktionen zählen:

- Erstellung und Verwaltung von Tresorelementen: Hinzufügen, Aktualisieren und Löschen von Passwörtern, sicheren Notizen und anderen gespeicherten Informationen.
- Suchen und Filtern: Effiziente Suche nach bestimmten Tresorelementen, basierend auf verschiedenen Kriterien.
- Datenverschlüsselung: Sicherstellung, dass alle über das API übertragenen Daten verschlüsselt sind, um die Sicherheit zu maximieren.

Das **Bitwarden Command-Line Interface** ist ein weiteres nützliches Werkzeug. Es ermöglicht, Bitwarden direkt über die Kommandozeile zu steuern. Das ist besonders nützlich für ►

Entwickler und Systemadministratoren, die häufig Skripte und Automatisierungen verwenden. Das CLI ist plattformübergreifend und funktioniert auf Windows, macOS und Linux. Es bietet ebenfalls zahlreiche Funktionen:

- **Anmeldung und Authentifizierung:** Ermöglicht die Anmeldung bei Bitwarden und den Zugriff auf gespeicherte Daten.
- **Verwaltung von Tresorelementen:** Erstellen, Abrufen, Aktualisieren und Löschen von Tresorelementen direkt über die Kommandozeile.
- **Sicherheitsüberprüfungen:** Durchführung von Sicherheitsüberprüfungen und Audits, um sicherzustellen, dass die gespeicherten Daten sicher sind.
- **Integration in Skripte:** Einfache Integration in Bash-Skripte und andere Automatisierungswerkzeuge, um wiederholbare Aufgaben zu automatisieren.

Neben dem API und dem CLI bietet Bitwarden auch Software Development Kits (SDKs) und Bibliotheken, die dabei helfen, Bitwarden in eigene Anwendungen zu integrieren. Ein prominentes Beispiel ist das Bitwarden-C#-SDK. Hierbei handelt es sich um eine Bibliothek, mit deren Hilfe sich Bitwarden-Funktionen in .NET-Anwendungen integrieren lassen.

Das SDK bietet eine einfache und intuitive Schnittstelle zum Bitwarden-API und umfasst viele Funktionen. Dazu gehört die Benutzerverwaltung, um Benutzer innerhalb von .NET-Anwendungen zu erstellen und zu verwalten. Auch der

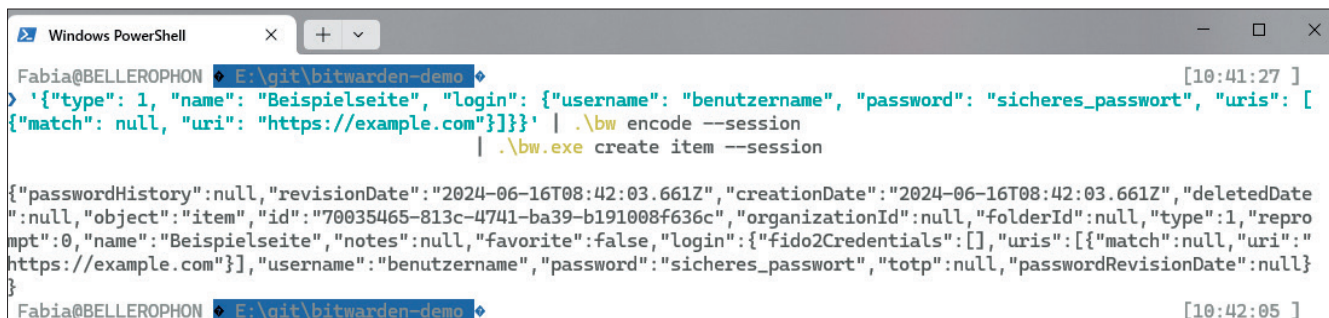
der DSGVO, während US-Unternehmen die US-Cloud wählen könnten. Auch Performance und Latenz sind betroffen: Europäische Nutzer profitieren von geringerer Latenz bei der EU-Cloud, US-Nutzer bei der US-Cloud.

Die Wahl zwischen EU- und US-Cloud hängt von den Anforderungen an Datenschutz, rechtliche Vorgaben und Performance ab, wobei Bitwarden diese Optionen für verschiedene Bedürfnisse anbietet.

Nutzung des CLI

Die Nutzung des Bitwarden Command-Line Interface bietet wie bereits erwähnt eine leistungsstarke und flexible Möglichkeit, die Funktionen von Bitwarden direkt über die Kommandozeile zu steuern. Das CLI ermöglicht eine Vielzahl von Aktionen, die von der Verwaltung von Tresorelementen bis zur Durchführung von Sicherheitsüberprüfungen reicht.

Die Anwendungsfälle des CLI sind vielfältig. Es lassen sich Routineaufgaben automatisieren, die Integration in DevOps-Workflows, beispielsweise in CI/CD-Pipelines, ist denkbar, und auch der schnelle Zugriff auf Zugangsdaten ist möglich, sodass in Bitwarden gespeicherte Passwörter auch in Skripten oder einfach nur direkt auf der Kommandozeile zur Verfügung stehen. Zudem sind automatisierbare Sicherheitsüberprüfungen und Audits möglich, beispielsweise, um regelmäßig sicherzustellen, dass alle gespeicherten Passwörter stark und sicher sind.



```

Windows PowerShell
Fabia@BELLEROPHON E:\git\bitwarden-demo [10:41:27]
> '{"type": 1, "name": "Beispielseite", "login": {"username": "benutzername", "password": "sicheres_passwort", "uris": [{"match": null, "uri": "https://example.com"}]}}' | .\bw encode --session
| .\bw.exe create item --session

{"passwordHistory":null,"revisionDate":"2024-06-16T08:42:03.661Z","creationDate":"2024-06-16T08:42:03.661Z","deletedDate":null,"object":"item","id":"70035465-813c-4741-ba39-b191008f636c","organizationId":null,"folderId":null,"type":1,"repro
mpt":0,"name":"Beispielseite","notes":null,"favorite":false,"login":{"fido2Credentials":[],"uris":[{"match":null,"uri":"
https://example.com"}],"username":"benutzername","password":"sicheres_passwort","totp":null,"passwordRevisionDate":null}
}
Fabia@BELLEROPHON E:\git\bitwarden-demo [10:42:05]
  
```

Erzeugen eines Eintrags über das Bitwarden CLI (Bild 2)

Tresorzugriff ist Teil des Funktionsumfangs. Damit ist der Zugriff auf gespeicherte Passwörter und sichere Notizen direkt aus der .NET-Anwendung heraus möglich.

Cloud: EU versus US

Die Unterscheidung zwischen EU- und US-Cloud bei Bitwarden betrifft den Standort der Rechenzentren für die Datenspeicherung und -verarbeitung: Die EU-Cloud speichert Daten innerhalb der EU, während die US-Cloud dies in den USA tut. Ein wichtiger Unterschied sind die zugrunde liegenden Datenschutzbestimmungen: Die EU-Cloud unterliegt der strengen DSGVO, die den Schutz personenbezogener Daten sicherstellt, die US-Cloud hingegen den weniger strengen US-Datenschutzgesetzen.

Rechtliche und regulatorische Aspekte sind ebenfalls relevant: EU-Unternehmen bevorzugen oft die EU-Cloud wegen

Das CLI ist vor der ersten Nutzung zu installieren. Es gibt native Pakete für Windows, macOS und Linux sowie Pakete für NPM, Chocolatey [2] und Snapcraft. Nach der Installation ist sicherzustellen, dass die richtige Bitwarden-Cloud zum Einsatz kommt. Der Login ist über drei Wege möglich. Wir erhalten Zugang über E-Mail-Adresse und Master-Passwort, unter Verwendung eines API-Schlüssels und über Single Sign-on (SSO). Das Kommando für den Login ist simpel:

```
bw login
```

Ohne weitere Angaben erscheint die Abfrage nach der E-Mail-Adresse und dem Passwort. Bild 1 zeigt, wie es aussieht, wenn der Login erfolgreich war. Die Session-Daten sind jetzt zu hinterlegen oder bei einem Befehl über die `--session`-Option zu übergeben.

Wenn das Ziel ist, mit den Daten im Tresor zu arbeiten, muss dieser noch per Kommando entsperrt werden. Das ist beispielsweise über

```
bw unlock
```

möglich. Hier wird erneut nach dem Master-Passwort gefragt, um den Tresor zu entsperren. Das Resultat ist ein Sitzungsschlüssel, der ebenfalls gespeichert oder bei zukünftigen Befehlen mit angegeben werden muss. Über die Bash lassen sich die Daten beispielsweise direkt mit einem Kommando speichern:

```
export BW_SESSION=$(bw unlock --raw)
```

Der Parameter `--raw` stellt sicher, dass nur der Sitzungstoken zurückgegeben wird, der dann vom Skript als Umgebungsvariable gesetzt wird. Über den Sitzungsschlüssel lassen sich jetzt alle Daten im Tresor abfragen, beispielsweise über das folgende Kommando mit direkter Angabe der Session:

```
bw.exe list items --session Wo...oAA==
```

Das Resultat ist ein mitunter sehr umfangreiches JSON-Objekt, je nachdem, wie viele Elemente im Tresor vorhanden sind. Zurückgegeben werden Informationen zum URL, Notizen, Benutzername, Passwort, Alter des Passworts, Revisionsdaten zu Passwörtern und viele weitere mehr.

Nach dem Login und Öffnen des Tresors lassen sich zudem viele andere Aktionen ausführen. Beispielsweise ist das Hinzufügen von Elementen möglich, wie der nachfolgende Befehl zeigt:

```
{ "type": 1, "name": "Beispielseite", "login":
  { "username": "benutzername", "password":
    "sicheres_passwort", "uris": [ { "match": null, "uri":
      "https://example.com" } ] } } | bw encode --session
QRm...kQ== | bw.exe create item --session QRm...kQ==
```

Der Befehl nutzt die JSON-Dateien für das neue Element im Tresor, die zunächst codiert werden müssen, um sie anschließend im `create item`-Kommando zu nutzen. In beiden Fällen ist die Session zu übergeben, wenn diese nicht wie beschrieben zuvor abgelegt wurde (siehe **Bild 2**). Das zeigt auch, dass es sinnvoll ist, die Session zu speichern, da ansonsten diese Befehle sehr unübersichtlich werden können.

Auch das Abrufen von Elementen ist über ein einfaches Kommando möglich:

```
bw.exe get password <item-id> --session QRm...kQ==
```

Für die Item-ID ist entweder die genaue ID im Tresor wichtig oder eine Zeichenkette. Ist keine genaue ID angegeben, wird der Tresor nach einem passenden Element durchsucht. Ein Aufruf von

```
bw get password Github
```

gibt somit das Passwort des gespeicherten Eintrags von GitHub zurück. In vielen Fällen funktioniert dieser Weg, da die Elemente recht eindeutig benannt sind. Darüber hinaus existieren zahlreiche Befehle zum Bearbeiten, Löschen, Wiederherstellen, Senden und Empfangen von Elementen. Auf diese Weise lassen sich zahlreiche Aktionen direkt über das CLI steuern und insbesondere automatisieren.

Anbindung des API

Das Bitwarden-API ermöglicht es, zahlreiche Funktionen von Bitwarden zu nutzen und in eigene Anwendungen zu integrieren. Dieses Mal nicht über das CLI, sondern über direkte ►

● Listing 1: Einen Benutzer über das API erstellen

```
public class BitwardenApi
{
    private static readonly HttpClient client =
        new HttpClient();
    private const string baseUrl =
        "https://api.bitwarden.com/v1/";

    public async Task CreateUser(string apiKey, string
        organizationId, string email, string name)
    {
        client.DefaultRequestHeaders.Add(
            "Authorization", $"Bearer {apiKey}");

        var newUser = new
        {
            Email = email,
            Name = name,
            Type = 2 // Member type
        };

        var content = new StringContent(
            JsonConvert.SerializeObject(newUser),
            Encoding.UTF8, "application/json");

        var response = await client.PostAsync(
            $"{baseUrl}organizations/{organizationId}/
            members", content);
        if (response.IsSuccessStatusCode)
        {
            Console.WriteLine("User created successfully.");
        }
        else
        {
            Console.WriteLine(
                $"Error: {response.StatusCode}");
        }
    }
}
```


RESTful-Schnittstellen. Der Zugriff umfasst sowohl das Management von Organisationen als auch die Verwaltung von Tresoren.

Das Organization Management API von Bitwarden erlaubt es, organisatorische Aufgaben zu automatisieren und effizient zu verwalten, wie bereits weiter oben im Artikel beschrieben. Dies umfasst die Verwaltung von Benutzern, Gruppen und Berechtigungen innerhalb einer Organisation. [Listing 1](#) zeigt dazu ein Beispiel.

In diesem Beispiel wird ein neuer Benutzer in einer Organisation angelegt. Die `CreateUser`-Methode nimmt den API-Schlüssel, die Organisations-ID, die E-Mail und den Namen des neuen Benutzers als Parameter und sendet eine `POST`-Anfrage an das Bitwarden-API. Die Authentifizierung geschieht über den API-Schlüssel, der über die Bitwarden-Website erzeugt wird.

Das Vault Management API bietet direkten Zugriff auf die in Bitwarden gespeicherten Daten. Dies umfasst das Erstellen, Abrufen, Aktualisieren und Löschen von Tresorelementen. [Listing 2](#) zeigt dazu eine Klasse `BitwardenApi`, um verschiedene Operationen durchzuführen.

In diesem Beispiel dient die Methode `CreateVaultItem` dazu, ein neues Login-Element zu erstellen, während die Methode `GetVaultItem` ein spezifisches Tresorelement anhand seiner ID abrufen. Beide Methoden verwenden den API-Schlüssel zur Authentifizierung und senden entsprechende HTTP-Anfragen an das Bitwarden-API.

Das Bitwarden-API bietet somit umfassende Möglichkeiten zur Integration von Passwortmanagement-Funktionen in eigene Anwendungen. Die Beispiele in C# zeigen, wie man das API für die Verwaltung von Organisationen und Tresoren nutzen kann.

● Listing 2: Beispiele für das Vault Management API

```
public class BitwardenApi
{
    private static readonly HttpClient client =
        new HttpClient();
    private const string baseUrl =
        "https://api.bitwarden.com/v1/";

    public async Task CreateVaultItem(
        string apiKey, string name, string username,
        string password, string url)
    {
        client.DefaultRequestHeaders.Add(
            "Authorization", $"Bearer {apiKey}");

        var newItem = new
        {
            type = 1, // Login type
            name = name,
            login = new
            {
                username = username,
                password = password,
                uris = new[]
                {
                    new { match = (int?)null, uri = url }
                }
            }
        };

        var content = new StringContent(
            JsonConvert.SerializeObject(newItem), Encoding.
            UTF8, "application/json");

        var response = await client.PostAsync(
            $"{baseUrl}items", content);

        if (response.IsSuccessStatusCode)
        {
            Console.WriteLine(
                "Vault item created successfully.");
        }
        else
        {
            Console.WriteLine(
                $"Error: {response.StatusCode}");
        }
    }

    public async Task GetVaultItem(
        string apiKey, string itemId)
    {
        client.DefaultRequestHeaders.Add(
            "Authorization", $"Bearer {apiKey}");

        var response = await client.GetAsync(
            $"{baseUrl}items/{itemId}");
        if (response.IsSuccessStatusCode)
        {
            var responseBody =
                await response.Content.ReadAsStringAsync();
            Console.WriteLine(
                $"Vault item: {responseBody}");
        }
        else
        {
            Console.WriteLine(
                $"Error: {response.StatusCode}");
        }
    }
}
```

● Listing 3: Beispiele für den Einsatz des Bitwarden-SDK

```
using Bitwarden.Sdk;

const string accessToken = "<access-token>";
using var bitwardenClient = new BitwardenClient();
bitwardenClient.AccessTokenLogin(accessToken);

var organizationId = Guid.Parse("<organization-id>");
var projectResponse = bitwardenClient.Projects().
    Create(organizationId, "TestProject");

var response = bitwardenClient.Projects.
    List(organizationId);

Guid projectId = projectResponse.Id;

projectResponse = bitwardenClient.Projects.
    Get(projectId);
projectResponse = bitwardenClient.Projects.
    Update(projectId, organizationId,
        "TestProjectUpdated");

const string key = "key";
const string value = "value";
const string note = "note";
var secretResponse = bitwardenClient.Secrets.
    Create(key, value, note, organizationId,
        new[] { projectId });
var secretId = secretResponse.Id;
```

Integration des SDK

Die Integration des Bitwarden-C#-SDK [3] in eine Anwendung ermöglicht es, die Funktionen des Bitwarden Secrets Manager zu nutzen. Wie bei den anderen Methoden ist es auch mit dem SDK möglich, sicher auf sensible Informationen zuzugreifen, diese zu verwalten und in eigene Anwendungen zu integrieren. Der Secrets Manager von Bitwarden bietet eine sichere und effiziente Methode, um Geheimnisse wie API-Schlüssel, Passwörter und andere vertrauliche Daten zu speichern und abzurufen.

Listing 3 soll einige Beispiele für die Nutzung des Bitwarden-SDK unter C# demonstrieren. Es zeigt zunächst die Authentifizierung mittels Access-Token, bevor in den folgenden Zeilen eine neue Organisation erstellt wird. Anschließend kommt die ID der Organisation zum Einsatz, um ein neues Projekt zu erstellen und zusätzlich alle Projekte abzurufen und aufzulisten.

Auf Basis eines Projekts lassen sich die Informationen in diesem Projekt abrufen und aktualisieren. Auch das Anlegen eines neuen, sicheren Elements ist über das SDK möglich. ►

● Listing 4: Konfiguration einer GitHub Action für Bitwarden

```
name: Use Bitwarden Secret

on:
  push:
    branches:
      - main

jobs:
  retrieve-secret:
    runs-on: ubuntu-latest

    steps:
      - name: Checkout code
        uses: actions/checkout@v2

      - name: Install Bitwarden CLI
        run: |
          sudo apt-get update
          sudo apt-get install -y jq
          curl -Lso /usr/local/bin/bw
            https://github.com/bitwarden/cli/releases/
            download/v1.17.0/bw-linux-1.17.0.zip
            chmod +x /usr/local/bin/bw

      - name: Login to Bitwarden
        env:
          BW_API_KEY: ${ secrets.BITWARDEN_API_KEY }
        run: |
          bw login --apikey
          export BW_SESSION=$(bw unlock --raw)

      - name: Get Secret
        env:
          BW_SESSION:
            ${ steps.login.outputs.BW_SESSION }
        run: |
          SECRET=$(
            bw get item my-secret-item-id --raw
          )
          echo "SECRET=$SECRET" >> $GITHUB_ENV

      - name: Use Secret
        run: echo "The secret value is $SECRET"
```

Auf diese Weise lassen sich zahlreiche Features direkt im Code nutzen, ohne Schnittstellen wie RESTful-APIs anzusteuern. Im Hintergrund erzeugt das SDK diese Anfragen, führt sie aus und verarbeitet Daten beziehungsweise Fehler. Welcher Weg einfacher ist, hängt letztlich vom jeweiligen Projekt und den Vorlieben der Entwickler ab.

Zum Zeitpunkt der Entstehung dieses Artikels war die genaue Roadmap des SDK noch nicht sicher. Laut dem Team hinter Bitwarden ist das Passwortmanager-SDK nicht für den öffentlichen Gebrauch bestimmt, und Bitwarden unterstützt das SDK in diesem Stadium aktuell nicht offiziell. Es ist ausschließlich dazu gedacht, die Geschäftslogik zu zentralisieren und eine einzige Quelle der Wahrheit für die internen Anwendungen bereitzustellen. Wenn sich das SDK zu einem stabileren und vollständigeren Zustand entwickelt hat, soll laut Bitwarden die Möglichkeit neu bewertet werden, stabile Bindings für die Öffentlichkeit freizugeben. Das bedeutet, dass das Passwortmanager-API zum aktuellen Zeitpunkt nicht stabil ist und sich ohne Vorwarnung ändern kann.

Beispiele für Integrationen

Die Integration von Bitwarden in verschiedene Entwicklungstools und -prozesse kann die Sicherheit und Effizienz erheblich steigern. Ein besonders nützliches Einsatzgebiet ist die Integration von Bitwarden in CI/CD-Pipelines, wo es für die sichere Verwaltung von Geheimnissen (Secrets) und Zugangsdaten verwendet werden kann. Ein konkretes Beispiel hierfür ist die Verwendung von Bitwarden in GitHub Actions.

GitHub Actions ist ein Tool zur Automatisierung von Software-Workflows direkt in GitHub. Es ermöglicht, verschiedene Aktionen auszuführen, so zum Beispiel das Testen, Bauen und Bereitstellen von Code. Durch die Integration von Bitwarden in GitHub Actions lässt sich sicherstellen, dass sensible Informationen wie API-Schlüssel und Zugangsdaten sicher in den Workflows verwendet werden.

Listing 4 zeigt ein Beispiel für die Nutzung von Bitwarden in einer GitHub Action. Die Action sorgt dafür, dass der Code aus dem Repository ausgecheckt und anschließend das Bit-

warden-CLI installiert wird. Danach erfolgt die Anmeldung bei Bitwarden per API-Schlüssel, um ein Secret (Geheimnis) aus Bitwarden abrufen und in einer Umgebungsvariable speichern zu können. Dieses Geheimnis, beispielsweise Passwörter für einen Teil der Infrastruktur, lässt sich dann in der Action in weiteren Schritten nutzen.

Passwordless.dev

Bitwarden Passwordless.dev zielt darauf ab, die Abhängigkeit von herkömmlichen Passwörtern zu beseitigen und stattdessen sicherere und bequemere Methoden zur Authentifizierung anzubieten. Passwordless.dev bietet dazu eine Plattform für passwortlose Authentifizierung, die eine Reihe von modernen Technologien und Methoden nutzt, um die Sicherheit zu erhöhen und die Benutzererfahrung zu verbessern. Die Plattform ermöglicht biometrische Authentifizierungen über Fingerabdrücke oder Gesichtserkennung und unterstützt das Web Authentication API (WebAuthn), das von W3C und FIDO Alliance entwickelt wurde. WebAuthn ermöglicht es Benutzern, sich sicher mit öffentlichen und privaten Schlüsselpaaren zu authentifizieren, die auf dem Gerät des Benutzers generiert werden.

Ein weiterer sicherer Authentifizierungsmechanismus ist die Verwendung von Einmalpasscodes (OTP), die an die E-Mail-Adresse oder das Mobiltelefon des Benutzers gesendet werden. Dies bietet eine zusätzliche Sicherheitsschicht, da der Code nur einmalig verwendet werden kann und zeitlich begrenzt ist. Auch Magic Links sind möglich, damit sich Nutzer mit einem Klick authentifizieren können, ohne ein Passwort eingeben zu müssen.

Zu Passwordless.dev existiert auch ein API sowie zahlreiche Bibliotheken, zum Beispiel für .NET und JavaScript. Dadurch lassen sich die Funktionen direkt in eigene Anwendungen integrieren. **Listing 5** zeigt beispielhaft die Integration zum Erstellen eines Access-Tokens in ASP.NET, in **Listing 6** sehen Sie den Prozess zum Einloggen eines Nutzers durch das Verifizieren eines Tokens. Dieser Code basiert auf dem Passwordless-NuGet-Paket:

● Listing 5: Create-Token-Code in Kombination mit Passwordless.dev

```
[HttpGet("/create-token")]
public async Task

```

● Listing 6: Code zur Verifikation eines Tokens in Kombination mit Passwordless.dev

```
[HttpGet("/verify-signin")]
public async Task<IActionResult>
VerifySignInToken(string token)
{
    try
    {
        var verifiedUser = await
            _passwordlessClient.VerifyTokenAsync(token);
        return Ok(verifiedUser);
    }
    catch (PasswordlessApiException e)
    {
        return new JsonResult(e.Details)
        {
            StatusCode = (int?)e.StatusCode
        };
    }
}
```

dotnet add package Passwordless

Diese ASP.NET-Core-Implementierung verwendet .NET 7 und etwas JavaScript für eine Passwordless.dev-Implementierung.

Alternative Passwortmanager

Die zunehmende Notwendigkeit, sensible Informationen sicher zu verwalten, hat zur Entwicklung vieler Passwortmanager geführt. Drei seien hier kurz vorgestellt.

LastPass [4] ist einer der bekanntesten Passwortmanager, der eine Vielzahl von Funktionen zur sicheren Verwaltung von Passwörtern und anderen sensiblen Daten bietet. LastPass hat ebenfalls begonnen, passwortlose Authentifizierungsmethoden zu unterstützen, wie zum Beispiel Biometrie und Multifaktor-Authentifizierung (MFA). Über ein API ist auch die Integration mit Anwendungen möglich.

Dashlane [5] ist ein weiterer beliebter Passwortmanager, der ebenfalls Funktionen für passwortlose Authentifizierung bietet. Dashlane verwendet Biometrie und MFA, um die Sicherheit zu erhöhen und den Anmeldeprozess zu vereinfachen. Dashlane bietet verschiedene Integrationsmöglichkeiten, darunter auch SDKs für die Nutzung in verschiedenen Programmiersprachen. Obwohl es kein spezifisches .NET-SDK gibt, lässt sich das RESTful API von Dashlane nutzen, um Anwendungen zu erstellen.

1Password [6] ist bekannt für seine Benutzerfreundlichkeit und robusten Sicherheitsfunktionen. Es bietet auch passwortlose Anmeldungen durch die Verwendung von WebAuthn und anderen modernen Authentifizierungsmethoden. 1Password bietet ein umfassendes API und CLI-Tools, die in .NET-Anwendungen integrierbar sind. Dies ermöglicht es, auf sichere Weise mit den gespeicherten Daten zu interagieren und passwortlose Authentifizierung zu implementieren.

Fazit

Die Integration von Bitwarden in .NET-Anwendungen bietet eine effiziente Möglichkeit zur Verwaltung von Anmeldeinformationen. Als Open-Source-Passwortmanager gewährleistet Bitwarden durch seine starke Verschlüsselung und plattformübergreifende Verfügbarkeit hohe Sicherheitsstandards. In .NET-Anwendungen lässt sich Bitwarden über sein

API oder Bibliotheken einbinden, was eine nahtlose Integration in bestehende Projekte ermöglicht.

Ob diese Integration zum Projekt passt, ist von Anwendung zu Anwendung zu entscheiden. Vielleicht ist ein anderer Passwortmanager die bessere Wahl oder ein gänzlich anderes System. Zugänge ohne Passwörter bieten auch andere Authentifizierungs-Anbieter an, ohne dass ein vollständiger Passwortmanager zum Einsatz kommen muss. Durch die größer werdende Verbreitung und die stärkere Akzeptanz von Passwortmanagern erscheint es aber wahrscheinlich, dass auch die Integration in Anwendungen zunehmen wird. Hier haben Open-Source-Projekte wie Bitwarden aufgrund des öffentlichen Codes einen entscheidenden Vorteil. ■

[1] Die Organisation zu Bitwarden auf GitHub,
<https://github.com/bitwarden>

[2] Bitwarden-CLI auf Chocolatey,
www.dotnetpro.de/SL2412Bitwarden1

[3] Bitwarden-SDK auf GitHub,
www.dotnetpro.de/SL2412Bitwarden2

[4] LastPass, <https://www.lastpass.com/de>

[5] Dashlane, <https://www.dashlane.com/de>

[6] 1Password, <https://1password.com/de>

[7] Verschlüsselung in Bitwarden,
www.dotnetpro.de/SL2412Bitwarden4

[8] Zero-Knowledge Encryption in Bitwarden,
www.dotnetpro.de/SL2412Bitwarden4

[9] Programm zur Offenlegung von Schwachstellen,
www.dotnetpro.de/SL2412Bitwarden5

[10] Informationen zum Audit-Programm von Bitwarden,
www.dotnetpro.de/SL2412Bitwarden6



Dr. Fabian Deitelhoff

arbeitet als IT-Leiter, Head of Software Development und Head of Product bei der Education Partners GmbH. Darüber hinaus ist er als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig.

@FDeitelhoff

dnpCode

A2412Bitwarden

C#-GRAPHEN-ALGORITHMEN

Knoten- und Kantenkunde

Ein Programmbeispiel zeigt, wie Sie Graphen-Algorithmen für die Tiefen- und die Breitensuche in C# entwickeln.

Graphen-Algorithmen spielen in der heutigen Softwareentwicklung eine entscheidende Rolle. So liefern sie Antworten auf komplexe Fragestellungen in der Routenplanung, in Kommunikations-, Vertriebs- und Versorgungsnetzwerken sowie in Such- und Optimierungsverfahren.

Ein Graph ist hierbei ein abstraktes mathematisches Konstrukt, das als Werkzeug zur Modellierung von strukturellen Zusammenhängen verwendet wird.

Der Graph besteht aus Knoten (englisch: Vertices), die durch Kanten (englisch: Edges) miteinander verbunden sein können. Konkret ist ein Knoten (Vertex) in der grafischen Darstellung ein Punkt, und die Kante (Edge) ist eine Linie, die zwei Knoten verbindet. **Bild 1** zeigt beispielhaft die Bahnverbindung mit Städten in Deutschland als Graphenmodell.

Der Umgang mit einem solchen Modell ist ein wichtiges Hilfsmittel in der Informatik. Einige der Anwendungsfälle von Graphen sind:

- **Routenplanung:** Ein klassisches Problem der Graphentheorie. Hier repräsentieren die Knoten wichtige Orte, die Kanten stellen verbindende Straßen dar.
- **Kürzeste Wege:** Hier werden Graphen für die Darstellung von Infrastruktur- und Kommunikationsnetzen genutzt, um den Weg zwischen zwei gegebenen Knoten zu ermitteln. Hierbei kann es sich um den kürzesten Weg oder aber auch um den schnellsten Weg handeln. Auch Wege mit dem geringsten Energieverbrauch, den geringsten Kosten und anderen Optimierungen können bestimmt werden.
- **Matchings:** Das Matching-Problem fragt nach einer Zuordnung zwischen Elementen von verschiedenen Mengen. Ein Matching besteht dann in der Bildung von Paaren, die alle miteinander übereinstimmend sind.
- **Flüsse:** Hierbei handelt es sich um die Problemstellung, einen Fluss (Güter-, Kommunikation-, Transport- oder Versorgungsfluss) zu bestimmen. Meist ist das Ziel, einen maximalen Fluss zwischen zwei gegebenen Knoten zu ermitteln, wobei die Kapazität jeder einzelnen Kante beschränkt ist.

Das ist nur ein kleiner Überblick über die Möglichkeiten, mit Graphen-Algorithmen zu arbeiten. Wenden wir uns im nächsten Schritt der Grunddatenstruktur von Graphen zu.

Graphen-Datenstruktur

Auch im Programmcode stellt ein Graph eine nichtlineare Datenstruktur dar, die aus Knoten und Kanten besteht. Kanten sind hierbei immer Linien oder Bögen, die zwei beliebige Knoten in einem Graphen miteinander verbinden. Knoten



Bahnverbindungen in Deutschland (Bild 1)

werden auch als Scheitelpunkte bezeichnet. Somit besteht ein Graph aus einer endlichen Anzahl an Knoten und einer Reihe von Kanten, die sie verbinden.

Bild 2 zeigt einen ungerichteten Graphen ohne Mehrfachkanten. Dargestellt ist dort die Menge der Knoten (Vertices) $V=\{A,B,C,D,E\}$ und die Menge der Kanten (Edges) $E=\{AB,AC,AD,BD,BE,CD,DE\}$.

Es gibt verschiedene Arten von Graphen, deren jeweilige Eigenschaften und Besonderheiten auf der Position dieser Knoten und Kanten basieren. Die wichtigsten sind:

- **Digraphen:** Man unterscheidet ungerichtete Graphen (**Bild 2**) von gerichteten Graphen (**Bild 3**), bei denen Kanten nur in der angegebenen Richtung durchlaufen werden dürfen. Diese Graphen werden als Digraph (Directed Graph) bezeichnet.
- **Multigraphen:** Ein Multigraph ist ein ungerichteter Graph, der zahlreiche Kanten zulässt. Zwei oder mehr Kanten, die dieselben zwei Knoten verbinden, sind sogenannte Mehrfachkanten. Normalerweise gibt es zwischen zwei Knoten maximal eine Kante.

- **Gewichteter Graph:** In einem gewichteten Graphen bekommt jede Kante noch eine Zusatzinformation, das Gewicht. Dieses Gewicht kann zum Beispiel die Länge der Kante darstellen. Es ist aber auch möglich, über das Gewicht anzugeben, wie viele Kanten zwischen zwei Knoten existieren.
- **Teilgraph:** Ein Teilgraph ist eine Teilmenge der Knotenmenge und der Kantenmenge innerhalb eines Graphen.
- **Vollständiger Graph:** In einem vollständigen Graphen ist jeder Knoten mit jedem anderen Knoten durch genau eine Kante verbunden.
- **Getrennter Graph:** Wenn mindestens zwei der Knoten des Graphen nicht durch eine Kante verbunden sind, bezeichnet man den Graphen als unzusammenhängend.
- **Zyklusfreier Graph:** Ein Zyklus beschreibt den geschlossenen Kantenzug in einem Graphen. Kommt im Zyklus jeder Knoten maximal einmal vor, dann bezeichnet man den Zyklus als Kreis. Graphen ohne Zyklen heißen zyklusfrei oder azyklisch. Ein zyklusfreier, zusammenhängender Graph ist immer ein Baum.

Es gibt noch eine Vielzahl von weiteren Graphen; aus den beschriebenen und aufgezeigten Graphen lässt sich aber eine Datenstruktur erkennen, die es ermöglicht, eine Graphenterminologie für Algorithmen und Datenstrukturen zu entwickeln.

Graphenterminologie

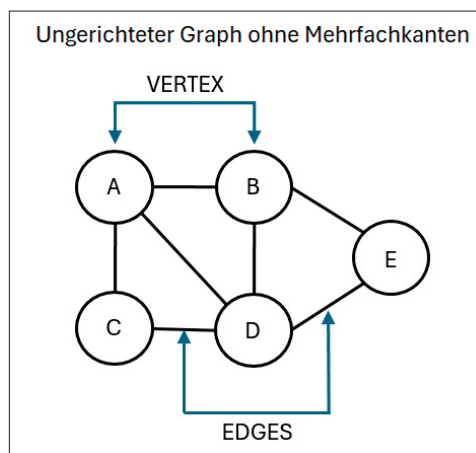
Für die Entwicklung eines Graph-Algorithmus müssen als Erstes die Begrifflichkeiten der Graphdatenstruktur bestimmt werden:

- Kanten sind die grundlegenden Elemente, aus denen Graphen aufgebaut sind. Hierbei hat jede Kante zwei Enden, mit denen sie verbunden ist.
- Wenn zwei Eckpunkte Endpunkte derselben Kante sind, sind sie benachbart.
- Die ausgehenden Kanten eines Knotens (Scheitelpunkts) sind gerichtete Kanten, bei denen der Knoten der Ursprung ist.
- Die eingehenden Kanten eines Knotens sind gerichtete Kanten mit dem Knoten als Ziel.
- Die Anzahl der Kanten, die zu einem Knoten in einem Diagramm gehören, entspricht seinem Grad.
- Jede Kante hat zwei Enden, die sogenannten Scheitelpunkte, mit denen sie verbunden ist.
- Ein Zyklus ist definiert als eine Reise, die gleichzeitig beginnt und endet.
- Ein Pfad mit eindeutigen Scheitelpunkten wird als einfacher Pfad bezeichnet.

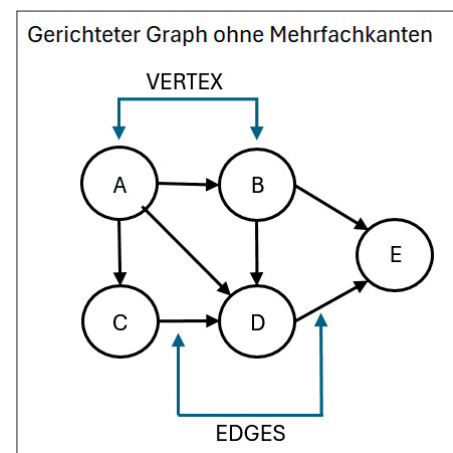
- Ein Graphdiagramm, das keine Zyklen aufweist, wird als Wald bezeichnet.
- Ein Baum ist ein verknüpfter Graph, der keine Zyklen hat. Wenn alle Zyklen in einem Directed Graph entfernt werden, wird er zu einem Baum, und wenn eine Kante in einem Baum entfernt wird, wird er zu einem Wald.

Aus dieser Terminologie ergibt sich für einen Algorithmus folgende formale Beschreibung:

Die Sammlung von Eckpunkten und die Nachbarn jedes Eckpunktes werden benötigt, um einen Graphen darzustellen (Eckpunkte, die direkt durch eine Kante mit ihm verbunden



Ungerichteter Graph ohne Mehrfachkanten
(Bild 2)

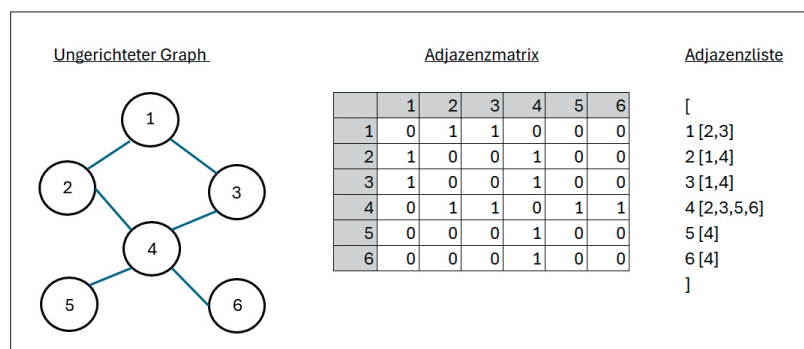


Gerichteter Graph ohne Mehrfachkanten
(Bild 3)

den sind). Ist der Graph gewichtet, wird die Gewichtung jeder Kante zugewiesen. Je nach Dichte der Kante des Graphen, der Art der auszuführenden Operationen und der Einfachheit der Verzweigung gibt es mehrere Möglichkeiten, den Graphen optimal zu beschreiben.

Darstellung von Graphen

Zwei wichtige Darstellungsformen von Graphen sind die Adjazenzmatrix und die Adjazenliste. Bei der Darstellung als Adjazenzmatrix wird eine Matrixstruktur verwendet, um einen Graphen repräsentieren. Die Zeilen- und Spaltenindi-



Ungerichteter Graph als Adjazenzmatrix und -liste (Bild 4)

Operation	Adj. Matrix	Adj. Liste
add Vertex	$O(1)$	$O(V^2)$
add Edge	$O(1)$	$O(1)$
Remove Vertex	$O(V+E)$	$O(V^2)$
Remove Edge	$O(E)$	$O(1)$
Search	$O(V+E)$	$O(1)$
Space	$O(V+E)$	$O(V^2)$

Vor- und Nachteile bei der Berechnung (Bild 5)

katoren stehen für die Knoten und die Werte in jeder Zelle für die Kanten. Der Wert einer Zelle wird als 1 markiert, wenn es eine Kante zwischen zwei Knoten gibt, andernfalls wird dieser mit 0 markiert.

In der Adjazenzlisten-Darstellung wird eine Array-Struktur verwendet, um einen Graphen darzustellen. Bild 4 zeigt die Darstellung eines ungerichteten Graphen sowohl als Matrix wie auch als Liste.

Beide Darstellungen haben ihre eigenen Vor- und Nachteile. Wenn Sie insgesamt V Knoten und E Kanten haben, sollte der $Null(0)$ -Vergleich aus Bild 5 eine Vorstellung davon vermitteln, welche Darstellung zu verwenden ist.

Die Adjazenzliste verbraucht weniger Arbeitsspeicher und ist schneller darin, alle Kanten zu finden und zu iterieren; allerdings ist sie langsamer, um eine Kante zwischen zwei Eckpunkten zu finden.

Die Adjazenzmatrix verbraucht mehr Speicher und ist langsamer dabei, alle Kanten zu finden und darüber zu iterieren; allerdings gelingt es mit ihr schneller, eine Kante zwischen zwei Scheitelpunkten zu finden.

So weit die theoretische Einführung. Sie können nun mit der Implementierung eines Graphen mithilfe einer Adjazenzliste beginnen. Mehr theoretisches Wissen über Graphen finden Sie unter [1].

Die Aufgabe

Mit Ihrem jetzigen Wissen über die Graphdatenstruktur können Sie ohne weitere Umwege in das Beispiel einsteigen. Die Aufgabenstellung ist im Bild 6 abgebildet: Es geht um das Durchlaufen der Datenstruktur eines Graphen, bei dem es noch keine eindeutige Identifizierung der untergeordneten Ebenen gibt.

Um die Datenstruktur eines solchen Graphen zu erschließen, werden zunächst alle Scheitelpunkte im Graphen identifiziert, indem sie besucht werden; dieser Vorgang wird als Graph Traversal bezeichnet. Das Graph Traversal findet im Suchprozess auch die verwendeten Kanten und verwendet dabei Verfahren, die keine Schleifendurchläufe erzeugen, um den Suchprozess möglichst kurz zu halten. Dazu ist es erforderlich, bereits besuchte Scheitelpunkte als solche zu kennzeichnen, um sie nicht häufiger als nötig anzusteuern.

Das Graph Traversal nutzt im Wesentlichen zwei Algorithmen zum Durchlaufen von Graphen, die sich in der Reihenfolge unterscheiden, in der die Knoten besucht werden: Depth First Search (DFS) und Breadth First Search (BFS).

Depth First Search (DFS)

Der Depth-First-Search-Algorithmus wird auch als Tiefensuche bezeichnet. Die Tiefensuche stellt einen rekursiven Algorithmus zum Erfassen aller Knoten eines Graphen oder einer Baumdatenstruktur zur Verfügung. Bei der Durchquerung des Graphen werden alle Knoten besucht.

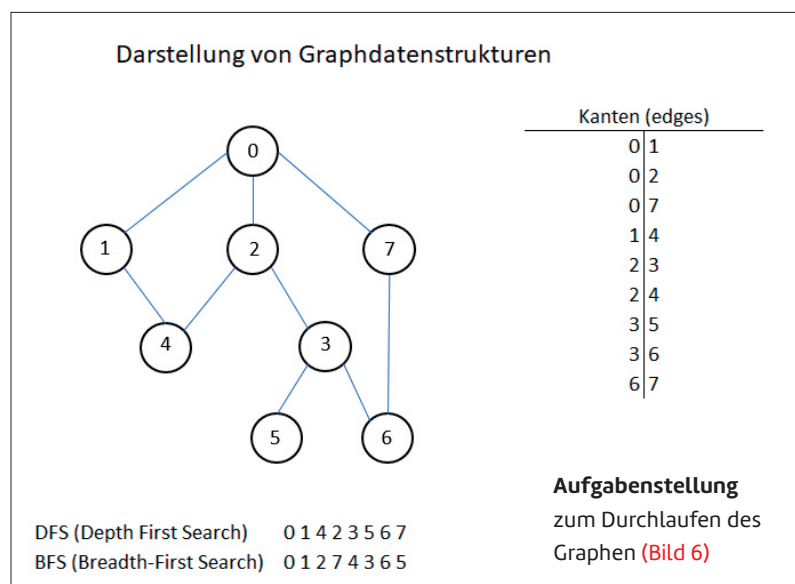
Die Standard-Implementierung ordnet jeden Scheitelpunkt des Graphen einer von zwei Kategorien zu: „besucht“ und „nicht besucht“. Die angestrebte Lösung des Algorithmus besteht darin, jeden Knoten als besucht zu markieren und gleichzeitig Zyklen zu vermeiden.

Der Depth-First-Search-Algorithmus durchläuft folgende Schritte:

- **Schritt 1:** Legen Sie einen beliebigen Knoten des Graphen auf einen Stapel.
- **Schritt 2:** Nehmen Sie das oberste Element des Stapels und fügen Sie es der Liste der besuchten Elemente hinzu.
- **Schritt 3:** Erstellen Sie eine Liste der benachbarten Knoten dieses Scheitelpunkts. Fügen Sie diejenigen, die nicht in der besuchten Liste enthalten sind, oben im Stapel hinzu.
- **Schritt 4:** Wiederholen Sie die Schritte 2 und 3, bis der Stapel leer ist.
- **Schritt 5:** Wenn der Stapel leer ist, erstellen Sie den endgültigen Spannbaum (Spanning Tree) [2], indem Sie die nicht verwendeten Kanten des Graphen löschen.

Listing 1 zeigt den Pseudocode für die Implementierung des Depth-First-Search-Algorithmus. Die *init()*-Funktion wird auf jeden Knoten ausgeführt. Das soll sicherstellen, dass jeder Knoten im Graphen abgedeckt wird.

Der Depth-First-Search-Algorithmus findet in folgenden Anwendungen Verwendung:



- Um Wege im Graphen zu finden.
- Zum Testen, ob es sich um einen Bipartiter-Graphen [3] handelt.
- Zum Finden stark verbundener Komponenten eines Graphen.
- Zur Erkennung von Zyklen in einem Diagramm.

Breadth First Search (BFS)

Auch bei diesem Algorithmus sorgt die Traversierung dafür, dass alle Knoten eines Graphen besucht werden. Die angewandte Breitensuche ist ein rekursiver Algorithmus zum Durchsuchen aller Knoten eines Graphen oder einer Baumdatenstruktur.

Die Standardimplementierung des Breadth-First-Search-Algorithmus ordnet jeden Scheitelpunkt des Graphen den

Listing 1: Pseudocode für Depth First Search

```
DFS(G, u)
    u.besucht = wahr
    für jedes v G.Adj[u]
        wenn v.besucht == false
            DFS(G,v)

init() {
    Für jedes u G
        u.besucht = false
    Für jedes u G
        DFS(G, u)
}
```

Kategorien „besucht“ oder „nicht besucht“ zu. Die Umsetzung des Breadth-First-Search-Traversals erfolgt in folgenden Schritten:

- **Schritt 1:** Legen Sie einen beliebigen Knoten des Graphen an das Ende einer Warteschlange.
- **Schritt 2:** Nehmen Sie das vordere Element der Warteschlange und fügen Sie es der besuchten Liste hinzu.
- **Schritt 3:** Erstellen Sie eine Liste der benachbarten Knoten dieses Scheitelpunkts. Fügen Sie diejenigen, die nicht in der besuchten Liste enthalten sind, am Ende der Warteschlange hinzu.
- **Schritt 4:** Wiederholen Sie die Schritte 2 und 3, bis die Warteschlange leer ist.
- **Schritt 5:** Da die Warteschlange leer ist, ist die Breitensuche des Graphen abgeschlossen.

Der Graph kann zwei verschiedene, getrennte Teile haben. Um sicherzustellen, dass jeder Knoten abgedeckt wird, kann der Breadth-First-Search-Algorithmus auch auf jeden Knoten ausgeführt werden. Listing 2 zeigt die Implementierung eines Pseudocodes für den Algorithmus.

Der Breadth-First-Search-Algorithmus kann für folgende Bereiche verwendet werden:

- Erstellen eines Index anhand des Suchindexes,
- GPS-Navigation,
- Pfadfindungsalgorithmen,
- im Ford-Fulkerson-Algorithmus [4] zum Finden des maximalen Flusses in einem Netzwerk,
- Zyklenerkennung in einem ungerichteten Graphen sowie
- im minimalen Spannbaum [2].

Jetzt können Sie anhand der Vorgehensweise und des Pseudocodes die Implementierung für den Depth-First-Search- und Breadth-First-Search-Algorithmus in C# vornehmen. Der Code wird vereinfacht ausgeführt, um sich auf den Algorithmus konzentrieren zu können.

Implementierung in C#

Zur Erstellung dieses Programmbeispiels wird als Entwicklungsumgebung Visual Studio 2022 (in einer Version ab Community Edition) mit dem .NET Framework 8.0 verwendet. Alternativ können Sie auch .NET Framework 6.0 benutzen.

Legen Sie für das Beispiel ein neues Console-App-Projekt in Visual Studio an. Vergeben Sie als *Project name* den Begriff *GraphenAlgorithmus* und wählen Sie .NET 8.0 als Framework aus. Nach dem Klick auf die Schaltfläche *Create* erstellt Visual Studio automatisch ein lauffähiges Konsolenprogramm mit der Ausgabe *Hello World!*

Im nächsten Schritt können Sie für das Beispiel die beiden Codezeilen in der Klasse *Program.cs* entfernen.

Im Beispiel soll der Graph aus Bild 6 durchlaufen werden, um die beiden Algorithmen Depth First Search und Breadth First Search auszuführen. Die Angaben zum Kantenverlauf und die Ergebnisse der Spannbäume sind entsprechend aufgeführt.

Der Programmcode in Listing 3 ist die Hauptmethode, mit der Sie die Algorithmen-Traversierung eines Graphen auf der Konsole ausgeben können. Bei der ersten Tiefensuche (DFS) beginnen Sie mit einem Knoten und durchlaufen einen Zweig, wobei alle Knoten abgedeckt werden, bevor Sie zum nächsten Zweig kommen. Es kann verschiedene Varianten geben, die auf *Pre Order*, *In Order* oder *Post Order* basieren.

Der Code aus Listing 3 zeigt zunächst die Depth-First-Search-Traversierung des Graphen. Die Variablen $vx=8$ und $e=9$ geben die Knoten- und Kanten-Anzahl für den Graphen aus Bild 6 wieder. Die Variablen *vertex*, *edges* und *visited* speichern die Kanten des Graphen, alle Knoten und besuchte Knoten. ►

Listing 2: Pseudocode für Breadth First Search

```
eine Warteschlange erstellen Q
v als besucht markieren und v in Q einfügen
während Q nicht leer ist
    entferne den Kopf u von Q
    markiere und stelle alle (nicht besuchten)
    Nachbarn von u in die Warteschlange
```


Listing 3: Implementierung der Algorithmen

```

using System.Threading;

int vx = 8;
int e = 9;

int[] vertex = new int[vx];
int[,] edges = new int[vx, vx];
bool[] visited = new bool[vx];

Console.WriteLine("Enter Kantenverbindungen");
for (int i = 0; i < e; i++)
{
    int s, d;
    string[] input = Console.ReadLine().Split(' ');
    int.TryParse(input[0], out s);
    int.TryParse(input[1], out d);
    edges[s, d] = 1;
    edges[d, s] = 1;
}

Console.WriteLine(
    "Ausgabe Depth First Search Algorithmus");
for (int i = 0; i < vx; i++)
{
    if (!visited[i])
        DepthFirstSearch(edges, vx, visited, i);
}

for (int i = 0; i < vx; i++)
{
    visited[i] = false;
}

Console.WriteLine();
Console.WriteLine(
    "Ausgabe Breadth First Search Algorithmus");

for (int i = 0; i < vx; i++)
{
    if (!visited[i])
        BreadthFirstSearch(edges, vx, visited, i);
}

Console.ReadLine();

static void DepthFirstSearch(int[,] edges, int v,
    bool[] visited, int si)
{
    visited[si] = true;
    Console.Write(si + " ");
    for (int i = 0; i < v; i++)
    {
        if (i == si)
            continue;
        if (!visited[i] && edges[si, i] == 1)
        {
            DepthFirstSearch(edges, v, visited, i);
        }
    }
}

static void BreadthFirstSearch(int[,] edges, int v,
    bool[] visited, int si)
{
    Queue<int> queue = new Queue<int>();
    queue.Enqueue(si);
    visited[si] = true;
    while (queue.Count != 0)
    {
        int currentVertex = queue.Dequeue();
        Console.Write(currentVertex + " ");
        for (int i = 0; i < v; i++)
        {
            if (i == currentVertex)
                continue;
            if (!visited[i] && edges[currentVertex, i] == 1)
            {
                queue.Enqueue(i);
                visited[i] = true;
            }
        }
    }
}

```

Die erste *for*-Schleife dient zur Eingabe der Kantenverbindung. Danach beginnen Sie mit einem Knoten, der mit der Variablen *i* (*si*) an die Methode *DepthFirstSearch(...)* übergeben wird. Die *for*-Schleife iteriert über jeden Scheitelpunkt und markiert jeden besuchten Knoten. Außerdem wird für jeden Knoten rekursiv die *DepthFirstSearch*-Methode aufgerufen.

Bei der anschließenden Breitensuche im Breadth-First-Search-Algorithmus werden zunächst alle Knoten auf derselben Ebene oder in derselben Tiefe durchsucht, bevor die nächste Ebene erreicht wird. Im Beispiel wird die Warteschlange aus der .NET-Klasse *Queue* zum Speichern aller

Kanten desjenigen Knotens genutzt, der gerade iteriert wird. Die Methode *Enqueue* fügt am Ende der Warteschlange (*Queue*) ein Element hinzu. *Dequeue* entfernt das älteste Element vom Anfang der Queue.

Bild 7 zeigt das lauffähige Programmbeispiel mit der entsprechenden Ausgabe der Tiefen- und Breitensuche.

Fazit

Dieser kleine Workshop sollte aufzeigen, welche Anwendungen von Graphen im Alltag eine Rolle spielen. Ob es sich dabei um Netzwerke, Landkarten oder um das soziale Netzwerk

```

D:\Entwicklung\Graphalgorithmen\Graphalgorithmen\bin\D
Enter Kantenverbindungen
0 1
0 2
0 7
1 4
2 3
2 4
3 5
3 6
6 7
Ausgabe Depth First Search Algorithmus
0 1 4 2 3 5 6 7
Ausgabe Breadth First Search Algorithmus
0 1 2 7 4 3 6 5

```

Die beiden Algorithmen im Test (Bild 7)

handelt – in diesem stellt die Person den Knoten dar und die Kanten die Verbindung zu Freunden –, überall trifft man auf Graphen.

Des Weiteren sollte aufgezeigt werden, wie man aus einer formal aufgezeigten Terminologie selbstständig einen Algorithmus für die Tiefen- und Breitensuche entwickeln kann.

Werfen Sie doch einfach mal einen Blick auf weitere Graphen-Algorithmen wie Dijkstra [5] und Jarník [6] und erweitern Sie den Programmcode beziehungsweise die Algorithmen

nach Belieben weiter, um zum Beispiel mit Gewichten in Graphen zu arbeiten. Für Sie als Entwickler bleibt das Thema Graphen auf jeden Fall spannend. ■

[1] Graphentheorie, www.dotnetpro.de/SL2412Graphen1

[2] Spannbaum bei Wikipedia, www.dotnetpro.de/SL2412Graphen2

[3] Bipartiter Graph bei Wikipedia, www.dotnetpro.de/SL2412Graphen3

[4] Algorithmus von Ford und Fulkerson bei Wikipedia, www.dotnetpro.de/SL2412Graphen4

[5] Dijkstra-Algorithmus bei Wikipedia, www.dotnetpro.de/SL2412Graphen5

[6] Jarník bei Wikipedia, www.dotnetpro.de/SL2412Graphen6



Daniel Basler

arbeitet als Lead Developer und Softwarearchitekt. Seine Schwerpunkte liegen auf Cross-Plattform-Apps, Android, JavaScript und Microsoft-Technologien.

dnpCode

A2412Graphen

DEV ACADEMY

HANDS-ON-WORKSHOPS UND
WEITERBILDUNG FÜR SOFTWARE-
ENTWICKLER UND -ARCHITEKTEN

2 TAGE

AI-driven Software Development mit GPT und Co.

TRAINING!

Was wird behandelt

- GPT-Modelle und ihre Unterschiede
- AI-Tools für Software-Entwickler
- Grundlagen des Prompt Engineering
- Verwenden der OpenAI API
- AI Search und Vektordatenbanken
- Eigene LLMs bereitstellen mit Azure OpenAI
- Entwicklung eigener Assistenten
- Anbindung Backendsysteme



Jörg Neumann,
MVP OpenAI
Trainer

LEGO EDUCATION SPIKE PRIME IM BLICK

Soll es Block oder Text sein?

Spike Prime ist eine moderne Robotikplattform für kreatives Lernen, die mit visueller und textbasierter Programmierung eine Fülle von Anwendungsmöglichkeiten bietet.

Unser Bildungssystem steht in der heutigen Zeit vor großen Herausforderungen und Veränderungen. Die zunehmende Digitalisierung und neue pädagogische Konzepte prägen die Bildungslandschaft und erfordern innovative Ansätze. Darunter gewinnt die digitale Bildung immer mehr an Bedeutung und verändert die Art und Weise, wie Wissen vermittelt und erworben wird. Sie umfasst den Einsatz digitaler Medien und Technologien im Unterricht sowie die Vermittlung von Kompetenzen für die digitale Welt. Schüler lernen den verantwortungsvollen Umgang mit digitalen Werkzeugen und entwickeln wichtige Fähigkeiten für die Zukunft.

In diesem Kontext ist das Stellvertretermodell ein wichtiger Aspekt. Dieses Modell im Bereich Bildung und Lernen bezieht sich auf eine Form des modernen Lernens, bei der eine bestimmte Person, Gruppe oder Technologie als „Stellvertreter“ fungiert, um Lerninhalte zu vermitteln oder Lernprozesse zu unterstützen. Es gibt verschiedene Ausprägungen dieses Modells, aber generell geht es um die Idee, dass nicht der eigentliche Lehrer oder Dozent, sondern ein Stellvertreter Lernprozesse leitet oder erleichtert.

In der modernen Bildungstechnologie wird das Stellvertretermodell oft auf digitale Werkzeuge übertragen. Beispielsweise können künstliche Intelligenzen, Lernplattformen oder Hardwaresysteme wie Roboter als Stellvertreter agieren, indem sie Lehrinhalte präsentieren, Tests verwalten oder Feedback geben. Auch hier wird die Rolle des Lehrers durch ein System oder eine Technologie teilweise ersetzt oder ergänzt.

Lego Education Spike Prime

Lego Education Spike Prime [1] ist ein Lernwerkzeug aus dem Hause Lego Education. Es kombiniert die bekannten Lego-Bausteine mit fortschrittlicher Technologie und pädagogischen Konzepten, um zum Beispiel Schülern die Möglichkeit zu bieten, Konzepte aus den Bereichen Technik, Umwelt und Softwareentwicklung selbst zu erleben. Damit es einfacher zu lesen ist, bezeichnen wir das System im weiteren Verlauf des Artikels kurz als Spike Prime.

Lego Education hat sich seit Jahrzehnten der Förderung von praktischem, spielerischem Lernen verschrieben. Spike Prime stellt dabei einen wichtigen Schritt dar, da es die neueste technische Entwicklung mit bewährten pädagogischen Methoden verbindet. Das System ermöglicht es Schülern, komplexe MINT-Konzepte (Mathematik, Informatik, Naturwissenschaften und Technik) auf eine greifbare und unterhaltsame Weise zu verstehen.

Spike Prime richtet sich primär an Schulen und Bildungseinrichtungen und eignet sich insbesondere für Schüler im



Komponenten im Spike-Prime-Set von Lego (Bild 1)

Alter von 10 bis 14 Jahren. Es ist jedoch auch in Maker-Spaces und MINT-Clubs beliebt, wo es kreative und technische Fähigkeiten fördert. Die Vielseitigkeit des Systems macht es zu einem wertvollen Werkzeug für verschiedene Lernumgebungen und Altersgruppen.

Das Herzstück von Spike Prime ist der programmierbare Hub, der als Gehirn der Kreationen fungiert. Er verfügt über einen leistungsstarken Prozessor, Bluetooth-Konnektivität und ein 5x5-LED-Display. Dazu kommen präzise Motoren, verschiedene Sensoren (wie Farb-, Kraft- und Distanzsensoren) sowie eine Vielzahl von Lego-Technic-Bausteinen. Diese Komponenten ermöglichen den Bau komplexer, interaktiver Modelle und Roboter.

Spike Prime zielt darauf ab, Schüler mit essenziellen Fähigkeiten für das 21. Jahrhundert auszustatten. Durch praktische Projekte und Herausforderungen entwickeln die Lernenden ein tiefes Verständnis für MINT-Konzepte. Sie üben kritisches Denken und Problemlösung, indem sie komplexe Aufgaben in kleinere, bewältigbare Schritte zerlegen. Die Arbeit in Teams fördert zudem wichtige soziale Kompetenzen wie Kommunikation und Zusammenarbeit.

Die Hardware

Zum Lieferumfang von Spike Prime gehört eine Palette an Hardware-Komponenten, die es ermöglichen, komplexe robotische Systeme zu bauen und zu programmieren. **Bild 1** zeigt eine Übersicht des Lieferumfangs. Die Kombination aus dem Hub, präzisen Motoren, vielseitigen Sensoren und dem modularen Baukastensystem basierend auf Lego Technic macht Spike Prime zu einem motivierenden Werkzeug für den MINT-Unterricht.

Der Spike Prime Hub ist das Herzstück des Systems und fungiert als programmierbarer Controller für alle angeschlossenen Komponenten. Er ermöglicht die Steuerung von Motoren und Sensoren sowie die Verarbeitung von Daten.

Der Hub verfügt über sechs Ein- beziehungsweise Ausgänge, die symmetrisch auf beiden Seiten angeordnet sind. Diese universellen Ports lassen sich sowohl für Motoren als auch für Sensoren nutzen, was eine gute Flexibilität bei der Konstruktion von Modellen bietet. Im Vergleich zum Vorgängermodell EV3, das separate Anschlüsse für Motoren und Sensoren hatte, ist dies eine Verbesserung in Bezug auf Benutzerfreundlichkeit und Vielseitigkeit. Nachteilig an Spike Prime gegenüber dem EV3 ist, dass Letzterer vier Anschlüsse für Sensoren und vier für Motoren besitzt, also insgesamt acht Anschlüsse. Bezogen auf die Anzahl der Anschlüsse bedeutet Spike Prime also einen kleinen Rückschritt.

Der Spike Prime Hub ist mit einem M4-Prozessor mit 100 MHz und 320 KB RAM ausgestattet [2]. Obwohl dies auf den

ersten Blick weniger leistungsfähig erscheint als der ARM9-Prozessor des EV3 mit 300 MHz, bietet der Spike Prime Hub andere Vorteile:

- **eine 5x5-LED-Matrix** für visuelle Ausgaben und Animationen,
- **Bluetooth-Konnektivität** (sowohl BTC 4.2 als auch BLE 4.2),
- **ein integrierter Lautsprecher** für Audioausgaben,
- **ein 6-Achsen-Gyroskop** für präzise Bewegungs- und Lageerfassung,
- **einen wiederaufladbaren Lithium-Ionen-Akku** für lange Betriebszeiten,
- **einen Micro-USB-Anschluss** für einfache Verbindung mit Computern und Tablets.

Der Hub bietet 32 MByte Speicher für Projekte, was deutlich mehr ist als die effektiv nutzbaren 5 MByte des EV3. Dies ermöglicht die Speicherung umfangreicherer Programme und Daten direkt auf dem Gerät.

Spike Prime bietet im Lieferumfang drei Motoren: zwei mittlere und einen großen Motor. Die mittleren Motoren haben ein Drehmoment von 3,5 Ncm und erreichen 135 Umdrehungen pro Minute. Der große Motor bietet ein Drehmoment von 8 Ncm und schafft 175 Umdrehungen pro Minute. Alle Motoren verfügen über eingebaute Encoder, die eine präzise Positionsbestimmung ermöglichen. Dies ist besonders nützlich für genaue Bewegungssteuerung und Robotik-Anwendungen.

Spike Prime bietet zudem eine Reihe von Sensoren für verschiedene Anwendungsbereiche:

- **Kraftmesser:** Misst Kräfte und Druck, ideal für physikalische Experimente.
- **Ultraschallsensor:** Misst Abstände bis zu zwei Metern und verfügt über vier individuell ansteuerbare LED-Segmente.
- **Farbsensor:** Erkennt acht verschiedene Farben und kann auch als Lichtsensor verwendet werden.
- **Gyroskop:** Im Hub integriert, ermöglicht präzise Lage- und Bewegungserfassung.

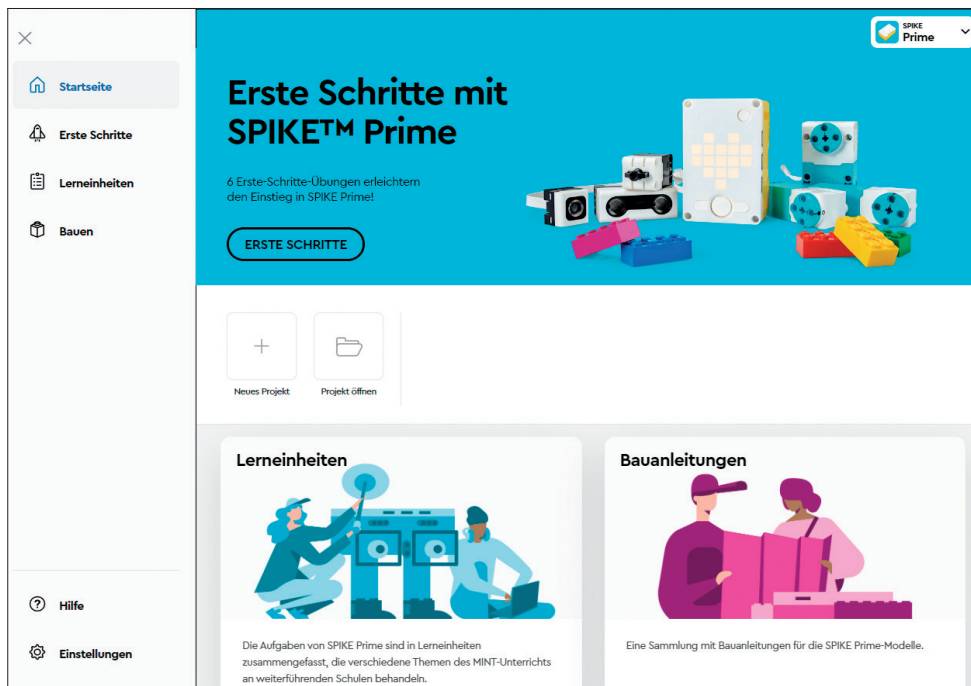
Diese Sensoren ermöglichen es Schülern, mit den Spike-Prime-Produkten ihre Umgebung zu erfassen, Daten zu sammeln und auf Veränderungen zu reagieren. Sie bilden die Grundlage für jegliche Art von interaktiven Robotern und komplexen MINT-Projekten.

Das Spike-Prime-Set enthält darüber hinaus über 500 Lego-Technic-Elemente in verschiedenen Farben. Diese Vielfalt an Bauteilen ermöglicht es Schülern, ihrer Kreativität freien Lauf zu lassen und eine breite Palette von Konstruktionen zu erstellen. Die Elemente sind essenziell, um mit Spike Prime Robotikprojekte umzusetzen. Die Modularität des Systems ist eine seiner größten Stärken. Schüler können schnell zwischen verschiedenen Konfigurationen wechseln, Teile austauschen und ihre Designs iterativ verbessern. Dies fördert das Problemlösungsdenken und ermutigt zum Experimentieren.

Spike Prime bietet neue Räder. Sie haben einen Durchmesser von 55,5 mm und sind so konzipiert, dass sie eine geringe Reibung bei gleichzeitig gutem Grip bieten. Die Reifen ►

• Lego Education Spike Prime versus Mindstorms EV3

Lego Education Spike Prime und Lego Mindstorms EV3 sind beide leistungsfähige Robotik-Plattformen für den Bildungsbereich, unterscheiden sich jedoch in einigen wichtigen Aspekten. Spike Prime verfügt über modernere Hardware mit einem leistungstärkeren Hub, universellen Ports und integrierten Sensoren. Seine Programmierungsumgebung basiert auf Scratch und Python, was den Einstieg und späteren Übergang zur textbasierten Programmierung erleichtert. Spike Prime zielt auf jüngere Schüler ab und bietet eine flachere Lernkurve, während der EV3 sich eher für ältere Schüler und komplexere Projekte eignet. Obwohl EV3 mittlerweile eingestellt wurde, bleibt es dank seiner großen Community relevant. Spike Prime bietet als neueres Produkt bessere Zukunftsfähigkeit und eignet sich besonders gut für den Einstieg in die Robotik, während der EV3 für fortgeschrittene Anwender weiterhin attraktiv bleibt.



Startbildschirm der Spike-App unter Windows 10 (Bild 2)

sind fest mit den Felgen verbunden, was die Stabilität erhöht und Fehler beim Aufbau minimiert.

Allerdings kann es auch in einigen Anwendungsfällen zu Problemen kommen, weil die Räder schmal sind – zum Beispiel in Wettbewerbssituationen oder bei ungewohnten Bodenbeschaffenheiten.

• Lego Education Spike Prime versus Spike Essential

Lego Education Spike Essential [3] ist ein Einstiegsset für jüngere Schüler im MINT-Bereich. Es enthält einen kompakten gelben Technic Hub mit zwei Anschlüssen, zwei kleine Motoren und eine 3x3-LED-Matrix. Das Set umfasst über 500 Lego-Technic-Elemente und ist für Kinder ab etwa sechs Jahren konzipiert.

Das Lego Education Spike Essential Set verfügt über einen kompakten, aufladbaren Hub mit Bluetooth-Verbindung, der eine einfache visuelle Programmierung über eine Scratch-basierte App ermöglicht. Es legt den Fokus auf grundlegende MINT-Konzepte und fördert spielerisches Lernen. Zudem ist es kompatibel mit anderen Powered-Up-Komponenten und eignet sich für ein bis zwei Schulstunden, abhängig von der Akkulaufzeit. Als Teil des Lego-Learning-Systems unterstützt es ein ganzheitliches MINT-Lernen und bietet eine ideale Einführung in Robotik und Programmierung für jüngere Schüler.

Spike Essential bietet eine sanfte Einführung in Robotik und Programmierung, mit altersgerechten Herausforderungen und einer benutzerfreundlichen Oberfläche. Es dient als Vorstufe zum komplexeren Spike-Prime-Set und ermöglicht einen nahtlosen Übergang zu fortgeschrittenen MINT-Konzepten.

Programmierung mit der Lego-Software

Die Programmierung erfolgt bei Spike Prime hauptsächlich über die Lego-Spike-App, eine auf Scratch basierende Programmierungsumgebung. Diese Plattform ermöglicht es, schnell und effektiv mit der Programmierung zu beginnen und gleichzeitig komplexe Projekte zu realisieren.

Das Herzstück der Programmierungsumgebung in Spike Prime ist die visuelle, blockbasierte Programmiersprache, die auf Scratch aufbaut, einer weit verbreiteten Plattform für den Einstieg in die Programmierung. Diese Umgebung ist speziell auf die Bedürfnisse von Schülern und Lehrern im MINT-Bereich zu-

geschnitten. Die Spike-App gibt es als Download [4] für verschiedene Plattformen, wie zum Beispiel macOS, Windows, iOS und Android. Bild 2 zeigt einen Screenshot der Windows-App unter Windows 10.

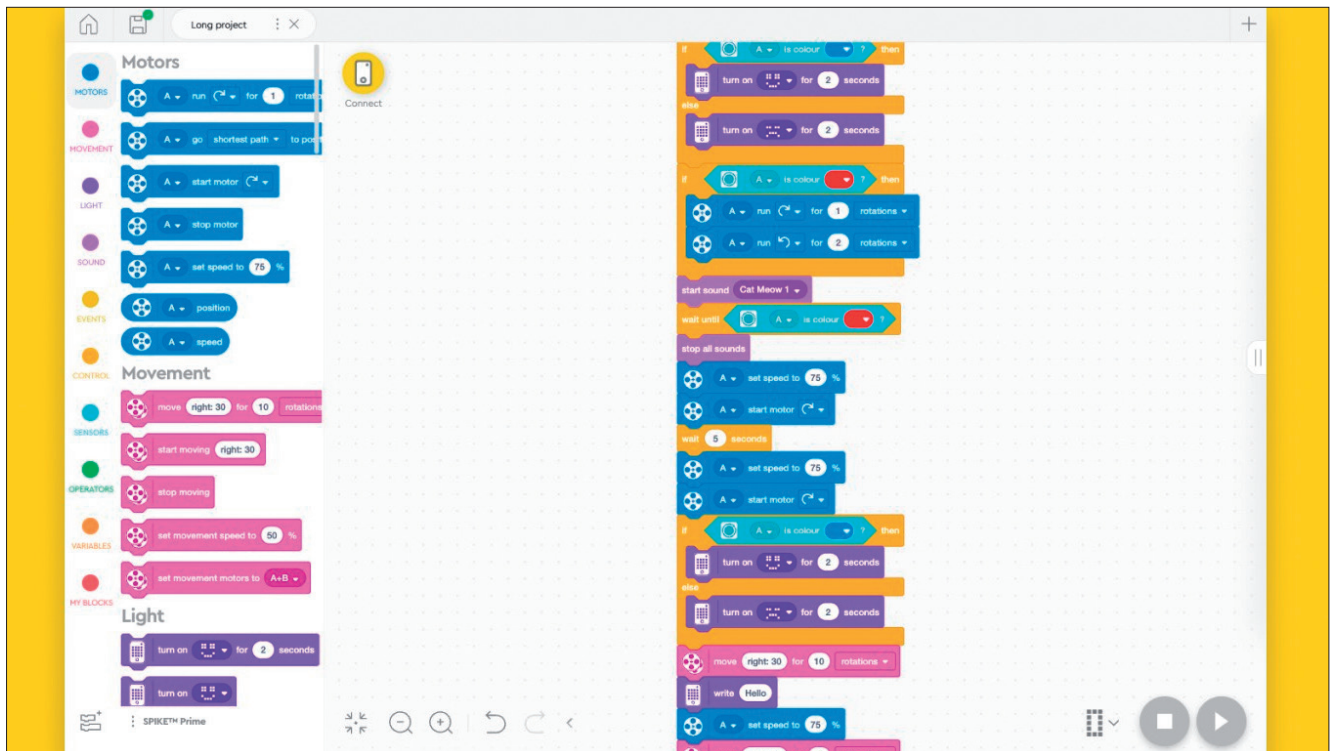
Zum Zeitpunkt der Erstellung dieses Artikels war die Version 3.4.3 der Lego-Spike-App aktuell. Alternativ bietet Lego Education die App als Webversion [5] im Browser. Bild 3 zeigt ein Beispiel für die Web-App in Chrome.

Die Benutzeroberfläche der Spike-App ist übersichtlich und intuitiv gestaltet. Sie besteht aus mehreren Hauptbereichen. Im Programmierbereich lassen sich die Codeblöcke platzieren und miteinander verbinden. Die Blockpalette enthält alle verfügbaren Programmierblöcke, kategorisiert nach Funktionen. Im Hub-Verbindungsbereich ist der Status der Verbindung zum Spike Prime Hub zu sehen, und im Bereich des Projektmanagements ist es möglich, Projekte zu speichern, zu laden und generell zu verwalten.

Die Programmierung an sich erfolgt durch Drag-and-drop von Codeblöcken. Diese Methode ist besonders für Einsteiger geeignet, da sie keine Vorkenntnisse der Syntax traditioneller Programmiersprachen erfordert. Die Blöcke sind farbcodiert und in logische Kategorien unterteilt, was hilfreich für Navigation und Verständnis ist. Einige der wichtigsten Blockkategorien sind die folgenden:

- **Bewegung:** Zur Steuerung der Motoren.
- **Sensoren:** Für die Verarbeitung von Sensorinformationen.
- **Operatoren:** Für mathematische und logische Operationen.
- **Kontrollstrukturen:** Für Schleifen und Verzweigungen.
- **Variablen:** Zum Speichern und Verwalten von Daten.

Die Blöcke lassen sich einfach ineinanderschieben, wobei die Form der Blöcke sicherstellt, dass nur syntaktisch korrekte Verbindungen möglich sind. Dies reduziert Frustration durch



Ein Projekt in der Spike-App als Web-Version in Chrome (Bild 3)

Syntaxfehler und ermöglicht es, sich auf die Logik und Struktur von Programmen zu konzentrieren.

Erste Schritte mit der visuellen Programmierung

Die Spike-App bietet eine Reihe von vordefinierten Beispielprojekten und Tutorials, die den Einstieg in die Programmierung erleichtern. Diese Projekte sind sorgfältig konzipiert und leiten die Nutzer durch verschiedene Programmierkonzepte und Hardwarefunktionen von Spike Prime. Die Tutorials sind typischerweise so aufgebaut, dass alle wichtigen Schritte nach und nach abgearbeitet werden: zum Beispiel die Verbindung zum Hub, eine einfache Motorsteuerung, Sensorauswertung, bedingte Anweisungen, Schleifen und ähnliche Konzepte. Zudem bauen die Projekte aufeinander auf und führen schrittweise in komplexere Konzepte ein – für Einsteiger daher eine gute Wahl.

Die Steuerung der Motoren ist ein zentraler Aspekt vieler Spike-Prime-Projekte. Die App bietet verschiedene Blöcke zur präzisen Kontrolle der Motoren:

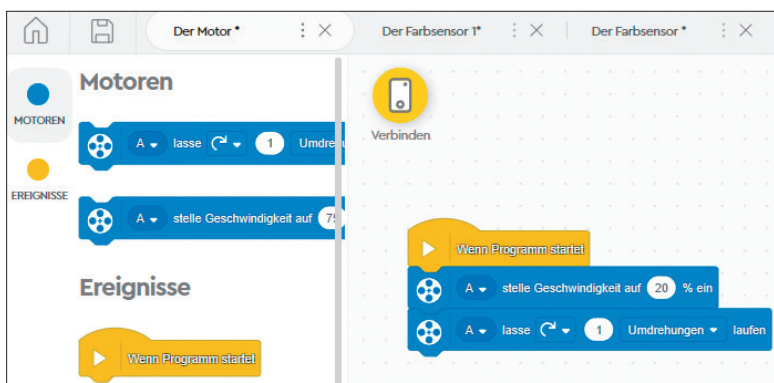
- **Drehung um einen bestimmte Grad:** Ermöglicht die genaue Positionierung der Motoren.
- **Drehung für eine bestimmte Zeit:** Nützlich für zeitbasierte Bewegungen.
- **Geschwindigkeitssteuerung:** Erlaubt die Anpassung der Motorgeschwindigkeit.
- **Synchrone Motorsteuerung:** Für koordinierte Bewegungen mehrerer Motoren.

Ein Beispiel für einen einfachen Motorsteuerungsblock zeigt das zugehörige Bild 4 der Scratch-Blöcke.

Die Sensorauswertung ist ein weiterer wichtiger Aspekt der Spike-Prime-Programmierung. Die App bietet spezielle Blöcke für jeden verfügbaren Sensortyp, wie den Farbsensor, den Ultraschallsensor, den Kraftsensor und das Gyroskop. Ein typischer Sensorauswertungsblock ist in Bild 5 zu sehen.

Selbst die einfache Kombination von Motor- und Sensorsteuerung ermöglicht die Erstellung komplexer und interaktiver Roboter. Beispielsweise lässt sich ein Roboter programmieren, der unter Verwendung des Farbsensors einer Linie folgt und durch Auswertung des Ultraschallsensors Hindernissen ausweicht.

Darüber hinaus bietet die Spike-App auch fortgeschrittene Programmiermöglichkeiten. Dazu gehört die Datenprotokollierung zum Sammeln und Analysieren von Sensordaten über die ►



Scratch-Programm zur Motoransteuerung (Bild 4)

Zeit. Die parallele Ausführung ermöglicht es, dass mehrere Programmteile gleichzeitig ablaufen können, und die ereignisgesteuerte Programmierung sorgt dafür, dass Programme auf spezifische Ereignisse reagieren können, um anschließend Aktionen auszuführen.

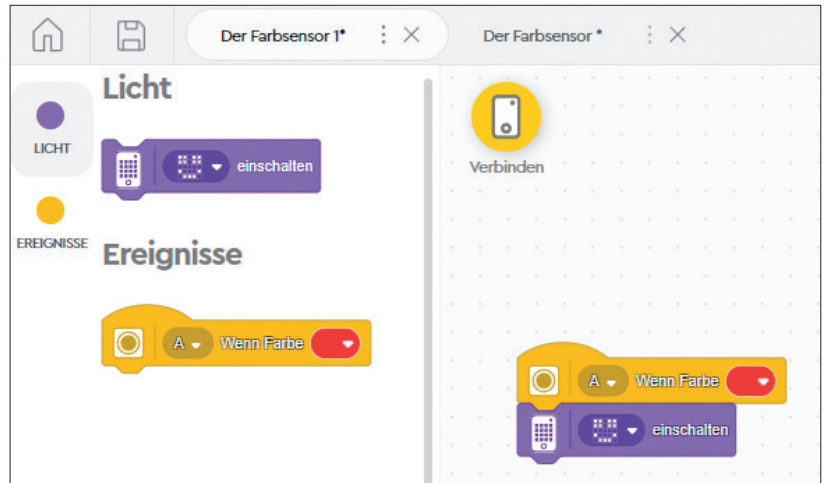
Python als Alternative?

Zusätzlich zur blockbasierten Programmierung bietet die Spike-App die Möglichkeit, mit Python zu programmieren. Dies ermöglicht einen sanften Übergang von der visuellen zur textbasierten Programmierung und bereitet auf fortgeschrittenere Programmierkonzepte vor. Python bietet eine leistungsfähige und flexible Plattform für komplexere Projekte und tieferes Verständnis von Programmierkonzepten.

Die Python-Unterstützung für Spike Prime ermöglicht es, textbasierte Programme zu erstellen und auszuführen. Dies bietet mehrere Vorteile gegenüber der blockbasierten Programmierung in der Lego-Software, hat aber auch gewisse Grenzen, die bei der Nutzung und Einführung im Bildungskontext zu berücksichtigen sind.

Die Python-Programmierung für Spike Prime bietet im Vergleich zur Lego-Software erweiterte Möglichkeiten wie die Implementierung komplexerer Algorithmen und Datenstrukturen, eine präzisere Kontrolle über Hardware-Ressourcen sowie die Nutzung externer Python-Bibliotheken. Dies bereitet Lernende effektiv auf professionelle Programmierung vor.

Allerdings bringt dieser Ansatz auch Herausforderungen mit sich: Anfänger sehen sich einer steileren Lernkurve ge-



Scratch-Programm zur Ansteuerung des Lichtsensors (Bild 5)

genüber, die visuelle Unterstützung ist reduziert, und es können Einschränkungen bei der Hardwareunterstützung auftreten. Trotz dieser Grenzen bietet die Python-Programmierung insgesamt ein leistungsfähigeres und flexibleres Entwicklungsumfeld für fortgeschrittene Spike-Prime-Projekte.

Es ist wichtig zu beachten, dass es verschiedene Python-Implementierungen für Spike Prime gibt, etwa Spike 3, Spike 2 und Pybricks [6]. Jede hat ihre eigenen Vor- und Nachteile in Bezug auf Funktionalität und Kompatibilität.

Installation und erstes Python-Beispiel

Um mit Python auf Spike Prime zu arbeiten, ist zunächst die entsprechende Firmware auf dem Hub zu installieren. Dies kann je nach gewählter Python-Implementierung variieren. Für die offiziell unterstützte Lego-Python-Umgebung sind die Schritte recht einfach. Über die installierte Spike-Software lässt sich der Spike Prime Hub verbinden, um die aktuelle Firmware des Hubs zu installieren. Damit steht die von Lego bereitgestellte Python-Umgebung zur Verfügung. Für alternative Implementierungen wie Pybricks gibt es spezifische Anweisungen [7] auf der Website.

Das Python-API für Spike Prime bietet eine Reihe von Bibliotheken zur Steuerung der verschiedenen Hardwarekomponenten. Listing 1 zeigt ein erstes Beispiel, in dem verschiedene Komponenten importiert werden und dann zum Einsatz kommen, um beispielsweise den Hub zu initialisieren, die LED-Matrix anzusteuern und Sensoren auszuwerten. Diese Grundlagen ermöglichen es, die verschiedenen Komponenten von Spike Prime zu steuern und auszulesen. Das Beispiel verdeutlicht zudem, warum diese Art der Programmierung sich so sehr von der Scratch-basierten Umgebung unterscheidet. Das ist logisch, da es sich um zwei grundverschiedene Ansätze handelt, ist aber bei der Arbeit mit Lernenden unbedingt zu berücksichtigen.

Roboterbewegung und Sensordatenverarbeitung

Das zweite Python-Beispiel zeigt, wie sich die Bewegung eines Roboters steuern lässt und Sensordaten verarbeitet werden können. Der Python-Code steuert einen Spike-Prime-Ro-

Listing 1: Das erste Python-Programm für Spike Prime

```
from Spike import PrimeHub, LightMatrix, Button,
    StatusLight, ForceSensor, MotionSensor, Speaker,
    ColorSensor, App, DistanceSensor, Motor, MotorPair
from Spike.control import wait_for_seconds,
    wait_until, Timer

# Initialisierung des Hubs
hub = PrimeHub()

# Steuerung der LED-Matrix
hub.light_matrix.show_image('HAPPY')

# Motorsteuerung
motor = Motor('A')
motor.run_for_degrees(360)

# Sensorauswertung
color_sensor = ColorSensor('E')
print(color_sensor.get_color())
```

boter, der mithilfe eines Farbsensors einer Linie folgt, bis er ein Hindernis erkennt, und dann stoppt und umkehrt.

Listing 2 zeigt dazu den Code. Zunächst werden der Roboter-Hub, ein Farbsensor, ein Abstandssensor und zwei Motoren initialisiert. Der Farbsensor misst kontinuierlich die Lichtreflexion, um die Position des Roboters relativ zur Linie zu bestimmen. Ein proportionaler Regelalgorithmus (P-Regler) berechnet anhand des Unterschieds zwischen dem gemessenen Reflexionswert und einem Zielwert die notwendige Korrektur für die Steuerung. Der Roboter passt seine Bewegung entsprechend an, um der Linie zu folgen.

Wenn der Abstandssensor ein Hindernis innerhalb von 10 cm Distanz erkennt, hält der Roboter an, fährt kurz rückwärts, um sich Platz für eine Drehung zu verschaffen, dreht sich um 180 Grad und fährt dann der Linie folgend wieder zurück. Währenddessen zeigt der Roboter über eine LED-Matrix eine Startanzeige („GO“) an, bevor die Linienfolge beginnt.

Dieses Beispiel zeigt mehrere zentrale Konzepte, darunter die Verarbeitung von Sensordaten, bei der ein Farbsensor zur Erkennung von Oberflächenreflexionen verwendet wird, um einer Linie zu folgen. Die Motorensteuerung erfolgt basierend auf diesen Sensordaten. Ein Distanzsensor erkennt Hindernisse und ermöglicht es, eine Kollision zu vermeiden. Zudem wird eine einfache proportionale Regelung (PID) eingesetzt, um eine gleichmäßige Linienfolge zu gewährleisten. Das Programm könnte durch die Implementierung einer voll-

ständigen PID-Regelung oder weiterer Funktionen, wie die Erkennung von Kreuzungen, weiter verbessert werden.

Python im Bildungskontext

Die Programmierung von Spike Prime mit Python bietet eine Vielzahl von Möglichkeiten für fortgeschrittene Projekte und tieferes Lernen. Allerdings ist es wichtig zu beachten, dass die Wahl der Python-Implementierung (Spike 3, Spike 2, Pybricks) die verfügbaren Funktionen und die Kompatibilität beeinflussen kann. Pybricks wird oft als die stabilste und leistungsfähigste Option angesehen, obwohl es einige Einschränkungen bei der UART-Unterstützung gibt.

Für Pädagogen und Schüler, die tiefer in die Python-Programmierung mit Spike Prime eintauchen möchten, empfiehlt es sich, die offizielle Lego-Education-Dokumentation sowie die verfügbaren Community-Ressourcen zu konsultieren. Diese bieten oft detaillierte Anleitungen, Beispielprojekte und Best Practices für die Entwicklung mit Python auf der Spike-Prime-Plattform.

Zusätzliche technische Möglichkeiten

Die Programmierung von Spike Prime lässt sich über die Standardfunktionen hinaus erweitern, um komplexere und anspruchsvollere Projekte zu realisieren. Open-Source-Frameworks wie OpenMV [8] und MicroPython [9] ermöglichen fortgeschrittene Funktionen innerhalb der Spike- ►

● Listing 2: Python-Programm für einen PID zur Sensordatenverarbeitung

```
from Spike import PrimeHub, ColorSensor,
    DistanceSensor, MotorPair
from Spike.control import wait_for_seconds

# Initialisierung der Komponenten
hub = PrimeHub()
color_sensor = ColorSensor('E')
distance_sensor = DistanceSensor('F')
drive_motors = MotorPair('A', 'B')

# Konstanten für die Linienfolge
TARGET_REFLECTION = 50
KP = 0.5 # Proportionaler Faktor für PID-Regelung

def follow_line():
    while True:
        # Lesen des Reflexionswertes
        reflection =
            color_sensor.get_reflected_light()

        # Berechnung des Fehlers
        error = reflection - TARGET_REFLECTION

        # Berechnung der Steuerung (einfache
        # P-Regelung)

        steering = KP * error

        # Fahren mit berechneter Steuerung
        drive_motors.start(steering=steering)

        # Überprüfung auf Hindernisse
        if distance_sensor.get_distance_cm() < 10:
            avoid_obstacle()

    def avoid_obstacle():
        # Stoppen und zurückfahren
        drive_motors.stop()
        drive_motors.move(-10, 'cm', 0, 20)

        # Drehen
        drive_motors.move(180, 'degrees', 100, 20)

        # Vorwärts fahren
        drive_motors.start()

# Hauptprogramm
hub.light_matrix.show_image('GO')
wait_for_seconds(1)
follow_line()
```


Prime-Programmierung wie Bildverarbeitung und maschinelles Lernen direkt auf dem Spike Prime Hub.

Die Kombination von Spike Prime mit Geräten wie Raspberry Pi oder Arduino erweitert die Fähigkeiten des Systems erheblich. Ein Raspberry Pi kann beispielsweise als leistungsfähiger Prozessor für komplexe Berechnungen dienen, während ein Arduino zusätzliche Sensoren und Aktoren einbinden kann. Beispielsweise ist der Raspberry Pi Build HAT eine solche Erweiterungsplatine für Raspberry-Pi-Computer, die in Zusammenarbeit mit Lego Education entwickelt wurde. Sie ermöglicht die einfache Steuerung von Lego-Technic-Motoren und die Auswertung von Sensoren aus dem Spike-Portfolio mit einem Raspberry Pi. Die Erweiterungsplatine bietet vier Anschlüsse für Lego-Technic-Motoren und Sensoren und ist mit allen 40-Pin-GPIO-Raspberry-Pi-Modellen kompatibel. Zur einfachen Steuerung der Lego-Komponenten steht eine spezielle Python-Bibliothek zur Verfügung. Ein integrierter RP2040-Mikrocontroller übernimmt die Verwaltung der angeschlossenen Lego-Geräte. Für eine stabile Stromversorgung des Raspberry Pi und der Lego-Komponenten lässt sich ein optionales Netzteil nutzen.

Die Verbindung von Spike Prime mit Cloud-Diensten wie Google Cloud oder AWS ermöglicht die Nutzung fortschrittlicher Datenanalyse- und KI-Funktionen. Sensordaten können in Echtzeit an diese Dienste gesendet und dort verarbeitet werden, um tiefere Einblicke zu gewinnen oder komplexe Steuerungsalgorithmen zu implementieren. Die Bluetooth- und WiFi-Fähigkeiten des Spike Prime Hub eröffnen vielfältige Kommunikationsmöglichkeiten. Über Bluetooth lassen sich mehrere Spike Prime Hubs miteinander vernetzen, um koordinierte Aktionen durchzuführen. WiFi ermöglicht die Anbindung an das Internet und damit den Zugriff auf Webservices oder die Fernsteuerung über mobile Geräte. Diese erweiterten Programmierungsmöglichkeiten transformieren Spike Prime von einem einfachen Lernwerkzeug zu einer vielseitigen Plattform für fortgeschrittene MINT-Projekte.

Vergleich der beiden Sprachen

Lego Spike Prime bietet sowohl visuelle Programmierung mit Scratch als auch textbasierte Programmierung mit Python. Scratch ist intuitiv und einsteigerfreundlich, hat jedoch Einschränkungen bei komplexeren Projekten. Python hingegen ist mächtiger und flexibler, erfordert aber eine längere Einarbeitungszeit. Die Wahl der Programmiersprache hängt vom Erfahrungsniveau der Lernenden, der Projektkomplexität, der verfügbaren Zeit und den Lernzielen ab. Für Einsteiger und einfachere Projekte eignet sich Scratch, während Python für fortgeschrittene Lernende und anspruchsvollere Aufgaben vorteilhaft ist. Ein effektiver Ansatz kombiniert beide Methoden: Man beginnt mit Scratch für grundlegende Konzepte, wechselt schrittweise zu Python für fortgeschrittene Funktionen und nutzt Scratch für schnelle Prototypen, während Python für die Feinabstimmung eingesetzt wird. Das ermöglicht eine sanfte Lernkurve und maximale Flexibilität in der Projektentwicklung. Diese Kombination ermöglicht es Lernenden, die Vorteile beider Welten zu nutzen. Sie können mit der intuitiven visuellen Programmierung beginnen und

allmählich zu textbasierter Programmierung übergehen, was eine sanfte Lernkurve und tieferes Verständnis fördert.

Die Spike-App unterstützt diesen Ansatz, indem sie sowohl visuelle als auch textbasierte Programmierung in einer Umgebung anbietet. Dies erleichtert den Übergang und ermöglicht es Lehrenden, den Unterricht flexibel an die Bedürfnisse der Lernenden anzupassen.

Fazit

Lego Education Spike Prime bietet eine vielseitige Robotik-Plattform, die speziell für den MINT-Unterricht entwickelt wurde. Sie verbindet kreative, praktische Lernansätze mit technologischem Fortschritt und richtet sich an Schüler im Alter von 10 bis 14 Jahren.

Spike Prime ermöglicht durch visuelle und textbasierte Programmierung, insbesondere mit Scratch und Python, eine einfache Einführung in die Robotik und in MINT-Konzepte. Gefördert werden dabei kritisches Denken, Problemlösung und Teamarbeit.

Im Vergleich zu älteren Systemen wie dem Mindstorms EV3 bietet Spike Prime moderne Hardware, bleibt jedoch zugänglich für Einsteiger und ist zukunftssicher für den Einsatz in Bildungsumgebungen. Die geringere Anzahl an Ports und die Qualität der Teile, zum Beispiel der Reifen, haben allerdings auch Kritik hervorgerufen. Nicht alle finden, insbesondere im Bildungskontext beispielsweise an Schulen, dass Spike Prime ein würdiger Nachfolger des EV3 ist. Wie sich das System weiterentwickelt, beobachtet die Community mit großer Spannung. Ein Nachfolgemodell ist bisher nicht angekündigt. ■

- [1] Website zu Lego Education Spike Prime, www.dotnetpro.de/SI2412LegoSpike1
- [2] Spezifikationen von Education Spike Prime, www.dotnetpro.de/SI2412LegoSpike2
- [3] Website zu Lego Education Spike Essential, www.dotnetpro.de/SI2412LegoSpike3
- [4] Download der Spike-App, www.dotnetpro.de/SI2412LegoSpike4
- [5] Webversion der Spike-App, <https://Spike.legoeducation.com/>
- [6] Website zu Pybricks, <https://pybricks.com>
- [7] Tutorial zur Installation von Pybricks, www.dotnetpro.de/SI2412LegoSpike5
- [8] Website zum OpenMV-Projekt, <https://openmv.io>
- [9] Website zu MicroPython, <https://micropython.org>



Dr. Fabian Deitelhoff

arbeitet als IT-Leiter, Head of Software Development und Head of Product bei der Education Partners GmbH. Darüber hinaus ist er als Autor, Dozent und Softwareentwickler im .NET- und Webumfeld tätig.

@FDeitelhoff

dnpCode

A2412LegoSpike

RUST-KURS, TEIL 8

Asynchron

Cooperative Scheduling in
Rust-Anwendungen realisieren.



Rust ist eine neuartige, moderne Programmiersprache, die sich auf die Sicherheit, der Geschwindigkeit und auf die effiziente, fehlerfreie parallele Programmierung konzentriert. In dieser Artikelserie geht es um den Einstieg in Rust und nebenbei gelegentlich um einen Vergleich mit C/C++ sowie C#. Die zurückliegende Folge [1] hat sich näher mit Multi-Threading, Channels und parallelen Iteratoren beschäftigt. Die heutige Folge erklärt, wie Sie die asynchrone Programmierung in Rust realisieren können, mit der Sie in einer anderen Art und Weise mehrere Code-Pfade parallel ausführen können. Zunächst werden ein paar wichtige Begriffe vorgestellt.

Parallelism, Concurrency, Preemptive- und Cooperative Scheduling

Die beiden Begriffe Parallelism und Concurrency werden sehr oft synonym verwendet, obwohl es sich um unterschiedliche Konzepte handelt. Parallelism bedeutet, dass mehrere Tasks ihre Arbeit parallel zueinander verrichten. Die vorangegangene Folge hat hierzu das Konzept des Multi-Threa-

ding vorgestellt, mit dem das Parallelisieren von Threads auf einem Multi-Core-System einfach realisiert werden kann.

Beim Preemptive Scheduling führt das Betriebssystem in regelmäßigen Abständen (unter Windows etwa alle vier Millisekunden) sogenannte Kontext-Switches durch. Dabei wird der Status des aktuellen Threads – also alle aktuellen Registerinhalte – in einer Datenstruktur gesichert und anschließend der Status eines anderen Threads in die Register geladen, damit dieser weiter ausgeführt werden kann. Da hierfür sehr viele Assembly-Instruktionen benötigt werden, ist ein Kontext-Switch sehr zeitintensiv und kann unter Umständen zu Skalierungsproblemen führen.

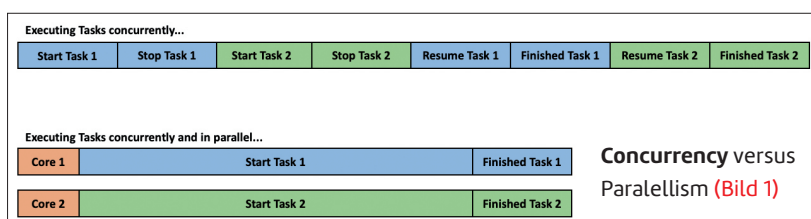
Stellen Sie sich beispielsweise bei einem Datenbank-Server vor, dass für jede eingehende Abfrage ein neuer Thread gestartet wird. Bei Tausenden von gleichzeitig ablaufenden Abfragen müsste das Betriebssystem ein Kontext-Switching für Tausende Threads durchführen. Es ist leicht ersichtlich, dass davon die Performance kaum profitieren kann.

Jeder Thread besteht zusätzlich aus verschiedenen Stack-Frames, welche die verwendeten lokalen Variablen speichern und den aktuellen Callstack umfassen. Stack-Frames behält das Betriebssystem im Hauptspeicher und generiert somit zusätzlichen Overhead. Beispiel Windows: Hier hat ein Stack-Frame für einen Thread eine initiale Größe von einem Megabyte. Trotz dieser Nachteile ist das Preemptive Scheduling die Multitasking-Variante, die moderne Betriebssysteme aktuell einsetzen.

Neben dem Preemptive Scheduling gibt es aber noch das Cooperative Scheduling, bei dem das Scheduling primär von den Anwendungen selbst realisiert wird. Das heißt, dass eine Anwendung einen Code-Pfad für eine bestimmte Zeitdauer ausführt und nach Ablauf der Zeitdauer die CPU von sich aus freigibt, damit ein anderer Code-Pfad ausgeführt werden kann. Das Freigeben der CPU wird als CPU-Yielding bezeichnet.

Da in diesem Verfahren keine Kontext-Switches seitens des Betriebssystems durchgeführt werden müssen, ist das Cooperative Scheduling um einiges schneller als das Preemptive Scheduling. Das Problem ist jedoch, dass Anwendungen selbst für das Freigeben der CPU verantwortlich sind. Dadurch kann ein einfacher Programmierfehler (das fehlende regelmäßige CPU-Yielding) das komplette Betriebssystem zum Absturz bringen. Diese Variante des Scheduling hat Microsoft unter Windows 3.1 eingesetzt – also vor einer sehr langen Zeit.

Nebenbei sei noch ein weiteres Beispiel erwähnt: Der SQL Server implementiert intern ebenfalls ein Cooperative



ding vorgestellt, mit dem das Parallelisieren von Threads auf einem Multi-Core-System einfach realisiert werden kann.

Concurrency bedeutet auch, dass mehrere Tasks ausgeführt werden, die aber zeitlich zueinander versetzt ablaufen können. **Bild 1** veranschaulicht diesen wichtigen Unterschied.

Um eine Concurrency von Tasks erreichen zu können, müssen Sie in der Lage sein, Tasks zu stoppen und ihren aktuellen Status zu speichern, damit Sie den Task zu einem späteren Zeitpunkt fortführen können.

Das Betriebssystem stellt spezielle Multitasking-Funktionalitäten zur Verfügung, mit deren Hilfe Parallelism und Concurrency von Code-Pfaden realisiert werden können. Programmiersprachen wie Rust setzen auf diese Funktionalitäten auf. Multitasking kann hierbei auf zwei unterschiedliche Arten durchgeführt werden: per Cooperative Scheduling oder Preemptive Scheduling.

Scheduling zum Ausführen von Datenbank-Abfragen. Der SQL Server umgeht dabei das Preemptive Scheduling des Betriebssystems und führt seine Worker-Threads selbstständig aus. Daraus folgt auch, dass die Code-Pfade innerhalb des SQL Servers eigenständig dafür Sorge tragen müssen, dass die CPU in regelmäßigen Intervallen freigegeben wird. Jedes Mal, wenn ein CPU-Yielding innerhalb der Implementierung des SQL Servers durchgeführt wird, wird für die betreffende Abfrage der Wartetyp *SOS_SCHEDULER_YIELD* zurückgeliefert, die Abfrage wird in den Status *RUNNABLE* verschoben und wartet dann auf freie CPU-Zeit. In solchen und ähnlichen Fällen ist das Cooperative Scheduling durchaus sinnvoll.

Als Daumenregel kann gesagt werden, dass Preemptive Scheduling immer dann die richtige Wahl ist, wenn CPU-intensive Arbeiten durchzuführen sind. Dazu können Sie Threads nutzen, deren Verwendung Sie bereits in der vorangegangenen Folge dieser Serie kennengelernt haben.

Besteht Ihr Code jedoch aus Funktionsaufrufen, die zu Blockaden führen können (weil die Code-Ausführung gestoppt werden muss, bis der Funktionsaufruf seitens des Betriebssystems vollständig durchgeführt wurde), ist der Einsatz von Cooperative Scheduling sinnvoller. Dazu zählen unter anderem:

- Netzwerkzugriffe,
- Datenbankzugriffe,
- Internetzugriffe,
- Interaktion mit dem Storage-Subsystem,
- Warten bis ein Timer abgelaufen ist.

Rust bietet deshalb neben klassischen Threads auch asynchrone Funktionen (*Async Functions*) an, die mithilfe einer zusätzlichen asynchronen Runtime über Cooperative Sche-

● **Tabelle 1: Asynchrone Runtimes für Rust**

Runtime	Beschreibung
async-std [7]	Diese Runtime bietet asynchrone Implementierungen der Komponenten der Rust Standard Library an.
Tokio [8]	Eine der bekanntesten und leistungsfähigsten Runtimes. Zu empfehlen für ernsthafte Rust-Projekte.
smol [9]	Eine kleine, sehr einfache Runtime.
fuchsia-async [10]	Eine Runtime, die für das Fuchsia OS von Google eingesetzt werden kann.

duling ausgeführt werden können. In bestimmten Bereichen erreichen Sie mit dem asynchronen Programmiermodell eine bessere Performance als mit klassischen Threads.

Asynchrone Rust Runtimes

Das asynchrone Programmiermodell von Rust ist nichts Neues oder Außergewöhnliches. Vergleichbar mit den Funktionalitäten in Rust sind beispielsweise die von JavaScript angebotenen Promises [2] oder das Task Asynchronous Programming Model (TAP) von .NET [3].

Eine Besonderheit der asynchronen Programmierung in Rust ist die Tatsache, dass Rust nur die notwendige Infrastruktur zur Verfügung stellt. Hierbei handelt es sich um die Trait *Future* [4] und die beiden Schlüsselwörter *async* und *await*, die Bestandteil des Rust-Compilers *rustc* sind.

Eine asynchrone Runtime gehört jedoch nicht zu den Bestandteilen eines klassischen Rust-Programms. Sie müssen sie deshalb explizit über eine passende Crate in Ihrem Programm referenzieren (über die Datei *cargo.toml*). Durch diesen Ansatz wird sichergestellt, dass Rust-Binaries sehr klein bleiben, da eine asynchrone Runtime nur dann mitkompiliert wird, wenn die Anwendung sie auch benötigt.

Tabelle 1 gibt einen Überblick über die bekanntesten asynchronen Runtimes für Rust, die immer wieder in unterschiedlichen Projekten eingesetzt werden.

Alle Beispiele in diesem Artikel basieren auf der Runtime *async-std*, da diese für die ersten Schritte leicht verständlich und programmiertechnisch an die synchrone Standard-Library von Rust angelehnt ist. Für ernsthafte Rust-Projekte ist jedoch die Tokio-Runtime zu empfehlen, da diese aktiv von der Rust-Community weiterentwickelt wird und für sie laufend neue Releases bereitgestellt werden.

Abhängig von der gewählten Runtime und deren Konfiguration kann diese asynchrone Code mit nur einem Worker-Thread (Single-Threaded) oder mehreren Worker-Threads (Multi-Threaded) ausführen. Im Hintergrund nutzt die Runtime dafür einen Thread-Pool, dessen Größe man ebenfalls anpassen kann.

Asynchrone Programmierung mit Rust

Als Beispiel für die asynchrone Programmierung in Rust dient hier die Ausführung von Abfragen auf einem SQL Server.

● Listing 1: Synchroner Zugriff auf den SQL Server

```
// Exemplarisch. Der Code lässt sich nicht
// kompilieren!

fn main()
{
    // 1. Connect to SQL Server.
    let sql_client =
        SQLClient::connect(connectionString);

    // 2. Execute a simple SELECT query.
    let select_query = Query::new(sql_statement);
    let mut data_stream =
        select_query.query(&client);

    // 3. Retrieve and process each row...
    while (let Some(row) = data_stream.try_next())
    {
        println!("{:?}", row);
    }
}
```

Zum besseren Verständnis wird in [Listing 1](#) zunächst exemplarischer Code gezeigt, der sich zwar nicht kompilieren lässt, aber demonstriert, wie der Zugriff auf den SQL Server mit normalen synchronen Funktionen funktionieren würde und wo die Performance-Probleme versteckt sind. Wie Sie im Listing erkennen können, interagiert dieser Code an drei Stellen mit dem SQL Server:

- Im ersten Schritt wird eine neue Datenbankverbindung zum SQL Server aufgebaut.
- Im nächsten Schritt wird eine *SELECT*-Abfrage zum SQL Server gesendet.
- Danach werden die zurückgelieferten Datensätze in der Anwendung verarbeitet.

Ohne asynchrone Programmierung laufen diese Funktionsaufrufe nacheinander ab, das heißt, dass der aktuelle Thread so lange blockiert wird, bis der Funktionsaufruf ein Ergebnis zurückliefert. Seitens des Betriebssystems wird der blockierte Thread in einen sogenannten *SUSPENDED*-Status versetzt, und es wird ein Kontext-Switch durch das Preemptive Scheduling durchgeführt, damit ein anderer Thread in der Zwischenzeit ausgeführt werden kann.

Wie Sie bereits von den Erläuterungen zu Kontext-Switches weiter oben im Text wissen, sind Kontext Switches auf Betriebssystem-Ebene sehr teuer und skalieren auch nicht besonders gut.

Damit anstelle des synchronen ein asynchroner Funktionsaufruf durchgeführt werden kann (der die CPU nicht blockiert), ist eine entsprechende Implementierung erforderlich. Einen normalen synchronen Funktionsaufruf durch eine Art Magie in einen asynchronen Funktionsaufruf zu transformieren funktioniert leider nicht.

Wie oben geschildert, benötigt man zunächst eine asynchrone Runtime für Rust, die asynchrone Funktionsaufrufe unterstützt. Ich habe mich im Rahmen dieses Beispiels für die Runtime *async-std* entschieden. Zusätzlich verwende ich die Crate *Tiberius* [5], um über asynchrone Funktionsaufrufe auf den SQL Server zugreifen zu können.

Eine Besonderheit ist hierbei, dass Sie die Abhängigkeit zu dieser Crate unter macOS anders konfigurieren müssen als unter Windows, da unter macOS ebenfalls eine OpenSSL-Bibliothek referenziert werden muss, damit sich der Rust-Code kompilieren lässt. [Listing 2](#) zeigt, wie Sie die plattformabhängigen Parameter in die Datei *cargo.toml* eintragen. Nachdem

Listing 2: Die Datei *cargo.toml*

```
[package]
name = "sqlserver_async"
version = "0.1.0"
edition = "2021"

[dependencies]
async-std = {
  version = "1.12.0",
  features = ["attributes"]
}
anyhow = "1.0.0"
futures-util = "0.3"

# Windows
[target.'cfg(
  target_os = "windows"').dependencies]
tiberius = {
  version="0.12.2",
  features = ["chrono", "time"]
}

# Mac OS
[target.'cfg(
  target_os = "macos"').dependencies]
tiberius = {
  version="0.12.2",
  default-features = false,
  features = ["chrono", "time",
    "vendored-openssl"]
}
```

Sie die Crate *Tiberius* in der Datei *cargo.toml* referenziert haben, können Sie nun auf den SQL Server zugreifen, siehe [Listing 3](#). Wie Sie erkennen können, sieht dieser Code der fiktiven synchronen Variante aus [Listing 1](#) ziemlich ähnlich. Aber es gibt dennoch einige Unterschiede: Eine der wichtigsten Änderungen ist die Verwendung der Schlüsselwörter *async* und *await*.

Die Funktion *execute_query()* wurde mit dem Schlüsselwort *async* definiert, wodurch Sie dem Rust-Compiler mitteilen, dass diese Funktion asynchron über Cooperative Scheduling ausgeführt werden soll. Jedes Mal, wenn Sie eine asynchrone Funktion aufrufen, die zu einer Blockade im Betriebssystem führt, wechselt die asynchrone Runtime über Cooperative Scheduling zu einem anderen Code-Pfad (der zuvor durch einen Funktionsaufruf blockiert war) und führt diesen weiter aus. Dabei handelt es sich um die zuvor angesprochene Technik des CPU-Yieldings. Wichtig zu wissen ►

```
Klauss-MBP:sqlserver_async klausaschenbrenner$ cargo run
warning: unused implementer of `futures_util::Future` that must be used
  -> src/main.rs:143:5
143 |     execute_query("SELECT @@VERSION".to_string());
    |     ~~~~~
= note: futures do nothing unless you `await` or poll them
= note: `[warn(unused_must_use)]` on by default
warning: `sqlserver_async` (bin "sqlserver_async") generated 1 warning
Finished `dev` profile [unoptimized + debuginfo] target(s) in 0.16s
Running `target/debug/sqlserver_async`
Done!
Klauss-MBP:sqlserver_async klausaschenbrenner$
```

Futures tun zunächst einmal nichts ([Bild 2](#))

ist, dass hier das Betriebssystem nicht involviert ist, da das Cooperative Scheduling vollständig im User-Mode erfolgt. Die Frage, die sich nun stellt, ist, an welchen Punkten das CPU-Yielding durchgeführt werden soll. Dazu definiert die Programmiersprache Rust das Schlüsselwort *await*.

Dieses Schlüsselwort können Sie beim Aufruf einer asynchronen Funktion angeben, wodurch seitens der verwendeten asynchronen Runtime das CPU-Yielding durchgeführt wird, wenn der Funktionsaufruf blockiert.

In **Listing 3** werden die folgenden CPU-Yielding-Points über das Schlüsselwort *await* definiert:

- *TcpStream::connect(...).await?*
- *Client::connect(...).await?*
- *select_query.query(...).await?*
- *data_stream.try_next().await?*

Hinweis: Das Fragezeichen (?) am Ende des Funktionsaufrufs ist für die Fehlerbehandlung in Rust notwendig, die in einer späteren Folge dieser Serie noch näher betrachtet werden wird.

Futures

Jetzt geht es darum, wie man eine asynchrone Funktion in Rust aufrufen und verwenden kann. Der hier zuerst abgedruckte, logischste Ansatz – nämlich der direkte Aufruf der asynchronen Funktion – führt leider noch nicht zum gewünschten Ergebnis:

```
fn main()
{
    execute_query(
        "SELECT @@VERSION".to_string(), 0);
    println!("Done!");
}
```

Rufen Sie die asynchrone Funktion über einen normalen Funktionsaufruf, passiert einfach gar nichts, wie Sie in **Bild 2** sehen können. Lediglich das Makro *println!* der Funktion *main()* wird ausgeführt, der Code innerhalb der Funktion *execute_query()* allerdings nicht. Im Bild hervorgehoben ist, dass der Rust-Compiler eine Warnung generiert hat, die besagt, dass eine sogenannte *Future* nichts durchführt, solange das Schlüsselwort *await* fehlt oder die Funktion *poll()* ausgeführt wird. Was genau ist nun aber eine Future, und was bedeutet diese Warnung?

Eine Future in Rust ist nichts anderes als eine Datenstruktur, welche die Trait *Future* [3] implementiert, die zur Standard Library von Rust gehört. Die Definition dieser Trait ist einfach gehalten, da nur ein Output-Typ und die Funktion *poll()* definiert wird. Hier die Definition dieser Trait:

```
pub trait Future
{
    type Output;
    // Required method
```

Listing 3: Asynchroner Zugriff

```
async fn execute_query(sql_statement: String,
    duration: u64) -> anyhow::Result<>
{
    // Create a new connection string.
    let config = Config::from_ado_string(
        CONNECTION_STRING?);

    // Create a new TcpStream to the
    // SQL Server endpoint.
    println!("Connecting to SQL Server...");

    let socket = TcpStream::connect(
        config.get_addr()).await?;
    socket.set_nodelay(true)?;

    // Connect to SQL Server.
    let mut client = Client::connect(
        config, socket).await?;
    println!("Successfully connected to
        SQL Server.");

    // Execute a simple query against SQL Server.
    let select_query = Query::new(
        &sql_statement);

    let mut data_stream = select_query.query(
        &mut client).await?;

    // Loop over the received data.
    while let Some(row) =
        data_stream.try_next().await?
    {
        match row
        {
            QueryItem::Metadata(meta) =>
            {
                println!("{:?}", meta);
                println!("");
            },
            QueryItem::Row(row) =>
            {
                println!("{:?}", row);
                println!("");
            }
        }
    }

    Ok(())
}
```

```
fn poll(
    self: Pin<&mut Self>,
    cx: &mut Context<'_>) ->
    Poll<Self::Output>;
}
```

Wie Sie erkennen können, hat die Funktion *poll()* der Trait *Future* eine veränderliche Referenz auf sich selbst (Parameter *self*) und eine veränderliche Referenz auf einen sogenannten *Context* (Parameter *cx*).

Interessant ist hierbei der Parameter *self*, da dieser in eine Datenstruktur vom Typ *Pin<T>* eingebettet ist. In Rust müssen Futures immer gepinnt werden, das heißt, dass diese im Hauptspeicher nicht verschoben werden dürfen. Diese Vorgehensweise ist notwendig, da Futures sogenannte Self-Referential Structures [6] sind, also Verweise auf sich selbst beinhalten.

Würde der Speicherort einer Future im Hauptspeicher verändert, würden diese Verweise ungültig. Durch Verwenden des Datentyps *Pin<T>* ist sichergestellt, dass der Speicherort einer Future nicht geändert werden kann, und somit bleiben auch die Verweise innerhalb der Future intakt.

Rufen Sie nun eine asynchrone Funktion auf, generiert der Rust-Compiler eine Datenstruktur, die diese Trait implementiert, und liefert sie als Funktionsergebnis zurück. Innerhalb der Funktion *poll()* ist die Funktion enthalten, die Sie zuvor in der asynchronen Funktion implementiert haben.

Der Rust-Compiler führt eine Code-Transformation der asynchronen Funktion in eine Future durch. Darum wird beim Funktionsaufruf auch die eigentliche Funktionsimplementierung nicht ausgeführt, da nur die Future-Datenstruktur generiert wird.

Damit nun die asynchrone Funktion ausgeführt werden kann, müssen Sie die zurückgelieferte Future-Datenstruktur an einen sogenannten *Executor* übergeben, der in regelmä-

Listing 4: Eine Future ausführen

```
fn execute_single_query()
{
    // The current thread will be blocked until
    // the future is resolved.
    let response1 = task::block_on(
        execute_query(
            "SELECT TOP 10
            BusinessEntityID, FirstName, LastName,
            ModifiedDate FROM
            Person.Person".to_string(), 0));
    println!("{}", response1);

    // The current thread will be blocked until
    // the future is resolved
    let response2 = task::block_on(
        execute_query(
            "SELECT @@VERSION".to_string(), 0));
    println!("{}", response2);
}
```

ßigen Intervallen die Funktion *poll()* aufruft und so die Future Schritt für Schritt abarbeitet.

Die *Executor*-Komponente ist Bestandteil der asynchronen Runtime, die in der Rust-Anwendung referenziert wurde – im Beispiel ist sie also Bestandteil der Runtime *async-std*.

Listing 4 zeigt nun, wie Sie die Future über die Funktion *task::block_on()* der asynchronen Runtime ausführen. Das Ergebnis dieser Ausführung zeigt Bild 3.

Wie Sie sehen, wird nun das Ergebnis der SQL-Abfrage auf der Konsole ausgegeben – die asynchrone Funktion, ►

```
● Klaus-MBP:sqlserver_async klausaschenbrenner$ cargo run
  Compiling sqlserver_async v0.1.0 (/Users/klausaschenbrenner/Desktop/sqlserver_async)
  Finished `dev` profile [unoptimized + debuginfo] target(s) in 2.95s
  Running `target/debug/sqlserver_async`
Connecting to SQL Server...
Successfully connected to SQL Server.
ResultMetadata { columns: [Column { name: "BusinessEntityID", column_type: Int4 }, Column { name: "FirstName", column_type: NVarchar }, Column { name: "LastName", column_type: NVarchar }, Column { name: "ModifiedDate", column_type: Datetime }], result_index: 0 }

Row { columns: [Column { name: "BusinessEntityID", column_type: Int4 }, Column { name: "FirstName", column_type: NVarchar }, Column { name: "LastName", column_type: NVarchar }, Column { name: "ModifiedDate", column_type: Datetime }], data: TokenRow { data: [I32(Some(1)), String(Some("Ken")), String(Some("Sánchez")), DateTime(Some(DateTime { date: 2022-04-14, time: 10:00:00}))] } }

Row { columns: [Column { name: "BusinessEntityID", column_type: Int4 }, Column { name: "FirstName", column_type: NVarchar }, Column { name: "LastName", column_type: NVarchar }, Column { name: "ModifiedDate", column_type: Datetime }], data: TokenRow { data: [I32(Some(2)), String(Some("Terri")), String(Some("Duffy")), DateTime(Some(DateTime { date: 2022-04-14, time: 10:00:00}))] } }

Ok(())
Connecting to SQL Server...
Successfully connected to SQL Server.
ResultMetadata { columns: [Column { name: "", column_type: NVarchar }], result_index: 0 }

Row { columns: [Column { name: "", column_type: NVarchar }], data: TokenRow { data: [String(Some("Microsoft SQL Server 2022 (RTM-CU14) 022 Microsoft Corporation\n\tDeveloper Edition (64-bit) on Linux (Ubuntu 22.04.4 LTS) <X64>"))] }, result_index: 0 }

Ok(())
Done!
● Klaus-MBP:sqlserver_async klausaschenbrenner$
```

Die asynchrone Funktion wurde erfolgreich ausgeführt (Bild 3)

Ausschnitt
des Codes,
den der Rust-
Compiler für
asynchrone
Funktionen
generiert
(Bild 4)

```
let socket =
  match #[lang = "branch"] (match #[lang = "into_future"] (TcpStream::connect(config.get_addr()))
  {
    mut __awaitee =>
      loop {
        match unsafe {
          #[lang = "poll"] (#[lang = "new_unchecked"] (&mut __awaitee),
            #[lang = "get_context"] (_task_context))
          {
            #[lang = "Ready"] { 0: result } => break result,
            #[lang = "Pending"] {} => { }
          }
        },
        _task_context = (yield ());
      },
    }
  ) {
    #[lang = "Break"] { 0: residual } =>
      #[allow(unreachable_code)]
      return #[lang = "from_residual"] (residual),
    #[lang = "Continue"] { 0: val } =>
      #[allow(unreachable_code)]
      val,
  };
```

beziehungsweise die Future, in die die asynchrone Funktion transformiert wurde, ist korrekt ausgeführt worden.

Ein Problem, das sich in Listing 4 versteckt hat, ist die Tatsache, dass die beiden SQL-Statements seriell, also der Reihe nach ausgeführt werden. Die Funktion `task::block_on()` blockiert nämlich den aktuellen Thread so lange, bis die übergebene Future vollständig abgearbeitet wurde.

Daraus folgt, dass das zweite SQL-Statement erst dann ausgeführt werden kann, wenn das erste Statement beendet wurde. Daher wird später noch besprochen, wie Sie mehrere Futures parallel abarbeiten können.

Futures als State Machines

Im vorangegangenen Abschnitt haben Sie gelernt, dass der Rust-Compiler aus einer asynchronen Funktion eine Future erzeugt, welche die Trait `Future` aus der Standard-Library implementiert. Und zum Ausführen einer Future benötigen Sie einen Executor, der Bestandteil der gewählten asynchronen Runtime ist.

Was genau ist nun aber eine Future, und wie kann ein Executor eine solche Future ausführen? Die zurückgelieferte Datenstruktur, welche die Trait `Future` implementiert, ist nichts anderes als eine State Machine, die aus mehreren unterschiedlichen Status besteht.

Wie Sie zuvor in Listing 3 gesehen haben, wurde beim Aufruf einer asynchronen Funktion das Schlüsselwort `await` verwendet. Bei jeder Verwendung dieses Schlüsselworts handelt es sich einerseits um einen CPU-Yielding-Point und andererseits um einen Status innerhalb der State Machine, der in der zurückgelieferten Future durch den Rust-Compiler generiert wird. Die folgenden Code-Zeilen zeigen exemplarisch, welche Stati für die asynchrone Funktion aus Listing 3 durch den Rust Compiler generiert werden.

```
enum State
{
  Unpolled,
  State0, // TcpStream::connect()
  State1, // Client::connect()
```

```
State2, // select_query.query()
State3  // data_stream.try_next()
Resolved
}
```

Wie Sie erkennen können, startet eine Future immer im Status *Unpolled*. Bei jeder Verwendung des Schlüsselworts `await` wird ein weiterer Status zur State Machine hinzugefügt, und wenn die asynchrone Funktion vollständig abgearbeitet wurde endet jede State Machine im Status *Resolved*.

Um nun von einem Status in den nächsten Status wechseln zu können, muss die Executor-Komponente auf der Future-Datenstruktur die Funktion `poll()` aufrufen (die Bestandteil der Trait `Future` ist). Diese führt dann eine entsprechende State Transition durch und wechselt vom aktuellen in den nächsten Status. Dies geschieht immer dann, wenn in der Code-Ausführung ein Punkt erreicht wird, an dem das Schlüsselwort `await` verwendet wurde.

Da diese Code-Punkte jedoch einen asynchronen Funktionsaufruf beinhalten, wird mit höchster Wahrscheinlichkeit der Funktionsaufruf noch nicht vollständig abgearbeitet sein, wodurch der generierte Code ein CPU-Yielding durchführt, wodurch auf dem gleichen Thread ein anderer Code-Pfad zur Ausführung kommt.

Sobald der ursprüngliche asynchrone Funktionsaufruf seine Arbeit erledigt hat, wird dies der asynchronen Runtime über eine sogenannte Waker-Komponente mitgeteilt, wodurch bei einer der nächsten CPU-Yielding-Operationen der ursprüngliche Task weiter ausgeführt wird – und zwar genau an der Stelle, wo er zuvor angehalten wurde.

Das ist die Grundidee des Cooperative Scheduling, da hier seitens des Betriebssystems kein Kontext-Switching erforderlich ist.

Sämtliche Scheduling-Operationen finden direkt im User-Mode statt, ohne dass hier Funktionalitäten aufgerufen werden müssen, die im Kernel-Mode des Betriebssystems ablaufen. Dadurch ist das Konzept des Cooperative Scheduling für blockierende Funktionsaufrufe um einiges besser geeignet als das Konzept des Preemptive Scheduling. Bild 4 zeigt einen Aus-

schnitt aus dem Code, den der Rust-Compiler für asynchrone Funktionen generiert. Diesen sogenannten HIR Code (High Level Intermediate Representation Code) können Sie sich über folgenden Kommandozeilenbefehl ausgeben lassen:

```
cargo rustc -- -Z unpretty=hir
```

Wichtig ist dabei, dass Sie eine Nightly-Build-Version von Rust einsetzen, da ein normaler Release-Build diesen Befehl nicht unterstützt. Installieren lässt sich eine solche Version mithilfe des folgenden Kommandozeilenbefehls:

```
rustup default nightly
```

Möchten Sie zur normalen Release-Build zurückwechseln, verwenden Sie den folgenden Kommandozeilenbefehl:

```
rustup default stable
```

Parallel ausgeführte Futures

In [Listing 4](#) haben Sie gesehen, wie Sie asynchrone Funktionen über die Funktion `task::block_on()` aufrufen. Übergeben Sie dieser Funktion jedoch nur eine Future zur Ausführung, wird der aktuelle Thread so lange blockiert, bis diese Future ihre Code-Ausführung beendet hat. Damit erreichen Sie aber keine richtige Parallelisierung, da bei einem CPU-Yielding-Point keine anderen Code-Pfade zur Ausführung zur Verfügung stehen.

Daher bietet Ihnen die asynchrone Runtime `async-std` die Funktion `task::spawn()` an, mit der Sie eine asynchrone Funktion asynchron im Hintergrund ausführen können, ohne dass der aktuelle Thread blockiert wird. Durch diese Vorgehensweise erreichen Sie, dass Futures parallel ausgeführt werden. [Listing 5](#) zeigt ein einfaches Beispiel.

Im Code werden innerhalb der Funktion `execute_multiple_queries_concurrently()` die verschiedenen Abfragen zusammengefasst, die anschließend über die Funktion `many_select_queries()` asynchron aufgerufen werden. Hierbei wird der Thread über die Funktion `task::block_on()` so lange blockiert, bis alle Abfragen beendet wurden. Daraus folgt, dass es sich bei der Funktion `many_select_queries()` wiederum um eine asynchrone Funktion handelt, die als Ergebnis eine Future zurückliefert.

Innerhalb der Implementierung dieser Funktion wird nun für jede übergebene Abfrage die Funktion `task::spawn()` aufgerufen, die als Ergebnis eine Variable vom Typ `JoinHandle` zurückliefert. Auf dieser Variablen wird anschließend in einer Schleife wiederum die Funktion `await` aufgerufen, welche die internen State Machines der Futures von einem Status in den nächsten Status bewegt und somit die asynchronen Funktionen Schritt für Schritt abarbeitet. [Bild 5](#) zeigt die Ergebnisse der verschiedenen Abfragen. Wenn Sie genau hinschauen, erkennen Sie, dass die Datensätze der einzelnen Abfragen auf der Konsole in einer anderen Reihenfolge ausgegeben werden, da sie nun parallelisiert über das Cooperative Scheduling ausgeführt wurden. ▶

Listing 5: Mehrere Futures ausführen

```
fn execute_multiple_queries_concurrently()
{
    let mut queries = vec![];
    queries.push(
        "SELECT TOP 10 BusinessEntityID,
        FirstName, LastName, ModifiedDate FROM
        Person.Person".to_string());

    queries.push("SELECT @@VERSION".to_string());

    queries.push("SELECT COUNT(*) FROM
        Person.Person".to_string());

    queries.push("SELECT TOP 10 AddressID,
        AddressLine1, PostalCode FROM
        Person.Address".to_string());

    // The current thread will be blocked until
    // the top future is resolved.
    let response = task::block_on(
        many_select_queries(queries));
    println!("{:?}", response);
}

async fn many_select_queries(queries: Vec<String>)
{
    let mut duration: u64 = queries.len() as u64 + 1;
    let mut handles = vec![];

    for query in queries
    {
        // Runs the new task asynchronously
        // in the background.
        // Other code can run concurrently...

        handles.push(task::spawn(
            execute_query(query.to_string(), duration)));
    };

    // Wait until all futures are resolved.
    for handle in handles
    {
        let _ = handle.await;
    }
}
```



```
ResultMetadata { columns: [Column { name: "", column_type: Nvarchar },], result_index: 0 }
ResultMetadata { columns: [Column { name: "BusinessEntityID", column_type: Int4 }, Column { name: "FirstName", column_type: Nvarchar }, Column { name: "LastName", column_type: Nvarchar }, Column { name: "ModifiedDate",
ResultMetadata { columns: [Column { name: "AddressID", column_type: Int4 }, Column { name: "AddressLine1", column_type: Nvarchar }, Column { name: "PostalCode", column_type: Nvarchar },], result_index: 0 }
Row { columns: [Column { name: "BusinessEntityID", column_type: Int4 }, Column { name: "FirstName", column_type: Nvarchar }, Column { name: "LastName", column_type: Nvarchar }, Column { name: "ModifiedDate", column_type:
ing(Some("Sánchez")), DateTime(Some(DateTime { days: 39818, seconds_fragments: 0 }))] }, result_index: 0 }
Row { columns: [Column { name: "AddressID", column_type: Int4 }, Column { name: "AddressLine1", column_type: Nvarchar }, Column { name: "PostalCode", column_type: Nvarchar },], data: TokenRow { data: [I32(Some(1)), Stri
Row { columns: [Column { name: "BusinessEntityID", column_type: Int4 }, Column { name: "FirstName", column_type: Nvarchar }, Column { name: "LastName", column_type: Nvarchar }, Column { name: "ModifiedDate", column_type:
tring(Some("Duffy")), DateTime(Some(DateTime { days: 39469, seconds_fragments: 0 }))] }, result_index: 0 }
Row { columns: [Column { name: "AddressID", column_type: Int4 }, Column { name: "AddressLine1", column_type: Nvarchar }, Column { name: "PostalCode", column_type: Nvarchar },], data: TokenRow { data: [I32(Some(2)), Stri
Row { columns: [Column { name: "BusinessEntityID", column_type: Int4 }, Column { name: "FirstName", column_type: Nvarchar }, Column { name: "LastName", column_type: Nvarchar }, Column { name: "ModifiedDate", column_type:
String(Some("Tamburello")), DateTime(Some(DateTime { days: 39388, seconds_fragments: 0 }))] }, result_index: 0 }
ResultMetadata { columns: [Column { name: "", column_type: Int4 },], result_index: 0 }
Row { columns: [Column { name: "", column_type: Nvarchar },], data: TokenRow { data: [String(Some("Microsoft SQL Server 2022 (RTM-CU11) (KB5032679) - 16.0.4105.2 (X64) \n\tNov 14 2023 18:33:19 \n\tCopyright (C) 2022 Mic
<X64>"))], result_index: 0 }
Row { columns: [Column { name: "AddressID", column_type: Int4 }, Column { name: "AddressLine1", column_type: Nvarchar }, Column { name: "PostalCode", column_type: Nvarchar },], data: TokenRow { data: [I32(Some(3)), Stri
```

Ergebnisse der parallelen SQL-Abfragen (Bild 5)

Synchroner Code in Futures

Eine weitere Besonderheit von asynchronen Funktionen in Rust ist, wie Sie in diesen Funktionen synchrone Funktionsaufrufe durchführen sollten. Wenn Sie eine synchrone Funktion ganz normal aufrufen, blockiert der aktuelle Thread so lange, bis der synchrone Funktionsaufruf vollständig durchgeführt wurde.

Daraus folgt aber auch, dass kein Cooperative Scheduling seitens des Executors durchgeführt werden kann, da bis zur Fertigstellung des synchronen Funktionsaufrufs kein CPU-Yielding-Point innerhalb der asynchronen Funktion erreicht werden kann. Dadurch wird auch die Executor-Komponente der asynchronen Runtime blockiert. Daraus ergeben sich dann entsprechende Performanceprobleme.

Listing 6 illustriert dieses Problem innerhalb der Funktion `execute_query()` durch den expliziten Aufruf der synchronen Funktion `std::thread::sleep()`, die den aktuellen Thread für die angegebene Zeit pausiert.

Rufen Sie diesen Code mit `std::thread::sleep()` auf, werden Sie feststellen, dass die Ausführung um einiges länger dauert, da nach jedem zurückgelieferten Datensatz der aktuelle Thread für einige Sekunden angehalten wird – wie lange, ist

abhängig vom übergebenen Parameter `duration`. Während dieser Zwangspause können jedoch auch kein Cooperative Scheduling durchgeführt und keine anderen Codepfade ausgeführt werden.

Um dieses Problem zu vermeiden, können Sie in der Runtime `async-std` die Funktion `task::spawn_blocking()` verwenden. Diese Funktion erwartet eine Closure, die anschließend in einem eigenen Thread ausgeführt wird. Dadurch wird der Haupt-Thread nicht blockiert, und seitens des Executors kann das Cooperative Scheduling durchgeführt werden. In Listing 7 sehen Sie die dafür erforderlichen Änderungen.

Der Thread, der über die Funktion `task::spawn_blocking()` gestartet wird, wird aus einem Thread-Pool genommen, der für blockierende Funktionsaufrufe reserviert ist. Wenn Sie – wie in diesem Beispiel – beim Aufruf der Funktion `task::spawn_blocking()` das Schlüsselwort `await` weglassen, führt der Haupt-Thread einfach seine Arbeit fort, ohne auf die neu gestarteten Threads zu warten.

Daraus resultiert, dass der Haupt-Thread seine Arbeit um einiges schneller beendet hat als die nebenläufig gestarteten Threads, wodurch Ihr Programm beendet wird und die nebenläufigen Threads abgebrochen werden. Dieses Problem

● Listing 6: Synchrone Funktionsaufrufe 1/2

```
// Loop over the received data.
while let Some(row) =
    data_stream.try_next().await?
{
    match row
    {
        QueryItem::Metadata(meta) =>
        {
            println!("{:?}", meta);
            println!("{}", meta);
        },
        QueryItem::Row(row) =>
        {
            println!("{:?}", row);
            println!("{}", row);

            // This call blocks the executor of the
            // runtime, because there is no .await!

            println!("Long running CPU intensive task...");

            std::thread::sleep(
                core::time::Duration::from_secs(duration));
        }
    }
}
```

Listing 7: Synchroner Funktionsaufrufe 2/2

```
// Loop over the received data.
while let Some(row) = data_stream.try_next().await?
{
    match row
    {
        QueryItem::Metadata(meta) =>
        {
            println!("{:?}", meta);
            println!("");
        },
        QueryItem::Row(row) =>
        {
            println!("{:?}", row);
            println!("");
        }
    }

    // The CPU intensive task is running on a
    // separate thread, and will not block the
    // executor.

    task::spawn_blocking(move ||
    {
        println!("Long running CPU
        intensive task...");

        std::thread::sleep(
            core::time::Duration::from_secs(duration));
    });
}
```

können Sie vermeiden, indem die nebenläufigen Threads mithilfe eines Channels mit dem Haupt-Thread kommunizieren und sich untereinander koordinieren.

Zusätzlich bietet die gewählte asynchrone Runtime die Möglichkeit an, freiwillig an beliebig gewählten Code-Punkten ein CPU-Yielding durchzuführen. Dies geschieht durch den Aufruf der Funktion `task::yield_now()`. Dieser Ansatz ist zum Beispiel dann sehr von Vorteil, wenn Sie in einer asynchronen Funktion einen längeren CPU-intensiven Algorithmus implementieren möchten.

Generell sollten Sie aufgrund der größer werdenden Komplexität vermeiden, in einer asynchronen Funktion CPU-intensive Operationen durchzuführen beziehungsweise synchrone Funktionen auszuführen. Je mehr asynchrone Funktionen mit entsprechenden CPU-Yielding-Points zur Verfügung stehen, desto besser kann der Executor im Hintergrund das Cooperative Scheduling durchführen.

Eine weitere Funktionalität, die Rust in Kombination mit asynchronen Funktionen anbietet, sind sogenannte Async Blocks. Ein Async Block ist nichts anderes als ein Code-Abschnitt, der als Ergebnis eine Future zurückliefert, die wiederum über eine asynchrone Runtime ausgeführt werden kann. Hier ein einfaches Beispiel:

```
fn async_blocks()
{
    let query1 = async
    {
        let _ = execute_query(
            "SELECT TOP 10 BusinessEntityID,
            FirstName, LastName, ModifiedDate FROM
            Person.Person".to_string(), 0).await;
    };

    task::block_on(query1);
}
```

Ein Async Block bietet ebenfalls die Möglichkeit, CPU-Yielding-Points über das Schlüsselwort `await` zu definieren, an denen ein Cooperative Scheduling durch die asynchrone Runtime durchgeführt wird.

Fazit

In diesem Artikel haben Sie das asynchrone Programmiermodell von Rust kennengelernt, mit dem Sie in einer einfachen Art und Weise ein Cooperative Scheduling in Ihren Anwendungen realisieren können. Die Standard-Library von Rust definiert dafür die Trait `Future`, die über eine asynchrone Runtime Schritt für Schritt abgearbeitet werden kann. ■

- [1] Klaus Aschenbrenner, *Multithreading*, *dotnetpro* 10/2024, Seite 131 ff., www.dotnetpro.de/A2410Rust
- [2] *Promises in JavaScript*, www.dotnetpro.de/SL2411Rust1
- [3] *TAP im .NET Framework*, www.dotnetpro.de/SL2412Rust2
- [4] *Trait Future*, www.dotnetpro.de/SL2412Rust3
- [5] *async-std*, <https://docs.rs/async-std>
- [6] *Self Referential*, www.dotnetpro.de/SL2412Rust4
- [7] *Tokio*, <https://docs.rs/tokio>
- [8] *smol*, <https://docs.rs/smol>
- [9] *fuchsia-async*, www.dotnetpro.de/SL2412Rust5
- [10] *Tiberius*, www.dotnetpro.de/SL2412Rust6



Klaus Aschenbrenner

berät Unternehmen in Europa beim Einsatz des Microsoft SQL Server und beschäftigt sich mit Windows-Programmierung und .NET. Zweimal zeichnete Microsoft ihn für sein Engagement als Microsoft MVP aus. Er ist zu erreichen über www.SQLpassion.at.

dnpCode

A2412Rust

MONOGAME FÜR EINSTEIGER, TEIL 6

Wie neu geboren, und das im Loop

Der Player wird heute wiedergeboren, er rennt als sein eigener Zwilling zu geloopter Hintergrundmusik Rillen in den Teppich, und Sie riskieren endlich einen Blick ins Lager.

Da Sie nun schon den sechsten Teil dieser Serie – die bisherigen Folgen finden Sie unter [1] bis [5] – vor sich liegen haben, wäre die Frage durchaus gerechtfertigt, was noch alles geplant ist. Das Ziel, auf das Sie hinarbeiten, wäre ja weiterhin (idealerweise) ein in sich abgeschlossenes Spiel. Lassen Sie uns daher kurz brainstormen, welche Puzzleteile dafür definitiv noch fehlen.

Offensichtlich natürlich ein sinnvolles Screen-Management, damit Sie einen Startbildschirm haben und mindestens zwischen zwei Räumen hin- und herwechseln können. Zudem (und ich wundere mich, dass mich dazu noch niemand angeschrieben hat) fehlt Sound. Und Musik. Außerdem selbstverständlich NPCs ... und dazu gehören auch die Staubsauger-Roboter. Text fehlt auch auf dem Bildschirm: *Titel, Ab laufende Zeit, Score*, so etwas in der Art. Vielleicht sogar ein wenig Spice. Damit meine ich nicht etwaige Rohstoffe fiktionaler Planeten, sondern Dinge wie die Idle-Animation, die die Gaming-Experience noch mal weiter abrunden. Beispiel gefällig? Wie wäre es mit kleinen Staubwolken, die von Ihrem Spieler beim Laufen aufgewirbelt werden (so ergeben die Staubsauger-Roboter nämlich auch endlich Sinn)? Und für den Abschluss der Serie habe ich schon einen ganzen Sack voller Ideen dafür, wie es danach weitergehen kann.

Refactoring

Wie Refactoring funktioniert, wissen Sie, und dass es in dieser Serie viele Gelegenheiten dazu gibt, haben Sie auch schon erkannt. Die letzten ihrer Art haben wir gemeinsam durchge-

führt, doch ab nun finden Sie dazu nur noch Recaps. Das bedeutet, dass ich ab jetzt bei erforderlichen Refactorings lediglich erläutern werde, warum Sie sie benötigen und worauf Sie dabei grundsätzlich achten sollten. Darüber hinaus finden Sie meine Ergebnisse natürlich weiterhin im Repository, und dann und wann gibt es passend dazu auch Videos, in denen Sie die einzelnen Schritte nachvollziehen können. Auf diese Weise können Sie das Tempo weiter steigern, und zugleich überfordern wir nicht unnötig das Seitenkontingent der dotnetpro.

Bei diesem Artikel habe ich mich zunächst etwas schwergetan. Warum? Der Code ist an einem Punkt angelangt, ab dem wir mittelfristig beginnen werden, diverse Features nach und nach hinzuzufügen. Damit das optimal gelingt, ist es erforderlich, das bis jetzt aufgebaute Wissen anzuwenden, um einiges umzustrukturieren. Da sich einige Teile dabei gegenseitig beeinflussen (und ich hoffe inständig, dass @ralfw nicht mitliest), lassen sie sich beim Refactoring nicht so leicht isoliert betrachten. Zu gegebener Zeit könnte es sich daher lohnen, mal über das Thema #CleanCode zu sprechen.

Sie werden verstehen, dass geschriebener Text daher nicht das geeignete Medium ist, diesen Prozess in aller Ausführlichkeit zu beschreiben. Daher habe ich das in aller Heimlichkeit erledigt und werde das Ergebnis in den folgenden Abschnitten nur kurz zusammenfassen. So können wir im Anschluss direkt fortfahren!

Doch zuerst ein Einschub. Ich wurde darauf angesprochen, wie genau die Idee hinter dem Repository wäre. Da das an dieser Stelle sehr gut passt, folgt eine Erläuterung für alle.

Doch noch nicht Refactoring

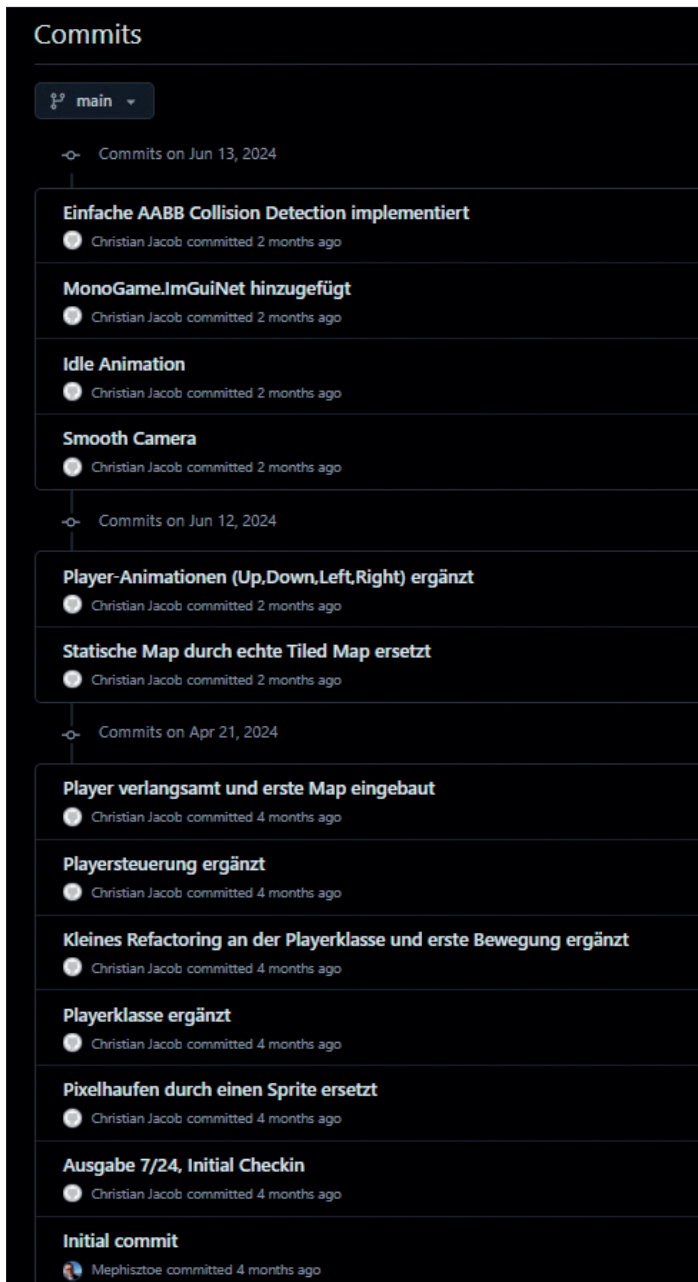
Unter [6] befindet sich – das sollte bekannt sein – das zentrale Repository mit dem Code, passend zu dieser Artikelserie. Wenn Sie einmal in die Commit-Historie sehen, ergibt sich eine Übersicht, ähnlich wie in **Bild 1** zu sehen.

Darüber können Sie zum Beispiel sehr schön die Unterschiede zwischen den einzelnen Ständen sichtbar machen. Interessant sind aber vor allem die Tags (**Bild 2**). Denn ab dem zweiten Artikel ist der jeweils finale Commit zum dazu passenden Quellcode entsprechend getaggt. Sie haben nun zwei Möglichkeiten, damit umzugehen. Entweder Sie laden hier direkt das jeweilige ZIP-File herunter und haben damit zum Beispiel den aktuellsten Stand aus einem bestimmten Artikel, oder

• ;tldr

Bei so vielen potenziellen Themen ist es manchmal nicht leicht, sich für einen in sich geschlossenen Artikel zu entscheiden, was zuerst drankommen sollte. Für diesen Artikel aber werden Sie erneut Ihren Code ein wenig auf alles Kommende vorbereiten und bei der Gelegenheit einfach schon mal einen Hauch von NPC einbringen (einfach, weil es so einfach ist). #refactoring

Darüber hinaus implementieren Sie ein Screen-Management und ergänzen sogar ein bisschen was für die Ohren.



Commit-Historie des Repositorys (Bild 1)

aber Sie klonen direkt das gesamte Repository und erstellen für sich einen lokalen Branch auf der Basis eines spezifischen Tags; vielleicht auch einfach einen Fork des Repositorys, sollten Sie einen eigenen GitHub-Account haben. Die Entscheidung liegt ganz bei Ihnen.

In der Annahme, dass Sie entweder mit Ihrem eigenen Quellcode en jour sind mit der Serie oder aber sich den Stand des letzten Teils heruntergeladen haben, geht es nun weiter mit den bereits angekündigten Umbenennungen.

Umbenennungen

Ich denke, es ist mittlerweile an der Zeit, nicht mehr von *Game1* zu spre-

chen. Ich habe die Klasse daher in *GameMain* umbenannt. Außerdem habe ich mich gegen den *LockerRoom* entschieden und spreche ab jetzt von einem *Storage* und orientiere mich für das Suffix fortan an der Technik, die wir einsetzen werden: Screens. Daher gab es eine Umbenennung von *MainRoom* und *LockerRoom* in *MainScreen* und *StorageScreen*. Sämtliche Vorkommnisse (inklusive der entsprechenden Maps) folgen dieser Umbenennung.

Collision Detection

Zum Ende des fünften Artikels hin sprachen wir darüber, dass wir mindestens einen weiteren Raum benötigen werden. Hinzu kommt, dass es mit Sicherheit auch noch einen *StartScreen* geben sollte. Bisher passierte aber praktisch alles in der Klasse *GameMain*. Da unterschiedliche Screens aber sehr individuell funktionieren können und von überall die Collision Detection im Zugriff sein muss, habe ich sie von der Klasse entkoppelt. Dazu gibt es nun eine Klasse *CollisionDetector*. Zudem finden Sie darin eine Änderung vor: Der Detector bekommt nun den *TargetLayer* übergeben und prüft nicht mehr generisch nur den *FloorLayer*.

DebugRects

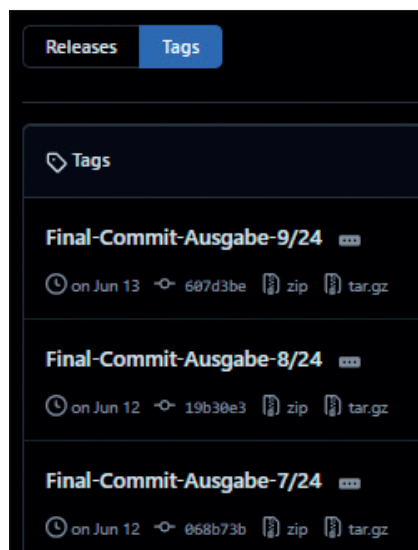
Sie hatten bereits eine Auflistung von *DebugRects* in *GameMain* implementiert und diese genutzt, um aus verschiedenen Stellen des Codes Items hinzuzufügen. Auf eine eigene Methode zum Zeichnen habe ich inzwischen verzichtet und stattdessen das Zeichnen direkt in die Methode *Draw* eingebaut:

```
spriteBatch.Begin(
    transformMatrix: Camera.GetViewMatrix(),
    samplerState: SamplerState.PointClamp);
{
    if (enableDebugRect)
        DebugRects.ForEach(debugRect => spriteBatch.
            DrawRectangle(debugRect.Item1,
                debugRect.Item2, thickness: .2f));
}
spriteBatch.End();
```

Um auch tatsächlich von überall aus dem Code Bounding Boxes hinzufügen zu können, ist die Liste nun *public static* – beispielsweise wird das im *CollisionDetector* genutzt.

Der Player

Bevor wir zum Hauptthema dieses Artikels kommen, den Screens, hier noch ein paar Erläuterungen zum Player. Bisher gab es nur einen davon, und den be- ►



Repository-Tags (Bild 2)

● CollisionCheck

Während die Methode *CollisionCheck* übrigens aktuell noch die Information zurückgibt, ob es zu einer Kollision gekommen ist oder nicht, brauchen Sie künftig idealerweise noch zusätzlich eine Referenz zu dem Tile, mit dem die Kollision stattgefunden hat. Warum, dazu kommen wir noch. Falls Sie dazu schon eine Ahnung haben: Go for it. Ansonsten kümmern wir uns später im entsprechenden Kontext darum.

weg? Richtig, Sie. Im Spiel wird es später aber mehrere Charaktere geben, und die meisten davon werden vom Computer gesteuert. Auch hier wollen Sie möglichst wenig Code doppeln. Daher habe ich für spätere NPCs und den Player den entsprechend relevanten Code in eine gemeinsame Basisklasse ausgegliedert und sie *GameCharacter* genannt. Das Ergebnis sehen Sie in [Listing 1](#).

Die *Player*-Klasse erbt nun direkt von der Klasse *GameCharacter*. Abschließend gab es noch ein paar kleine Anpassungen in der Klasse *GameMain*, um diesen Änderungen gerecht zu werden. Das betrifft die Initialisierung des Players und die Aufrufe zum Laden des Contents, *Update* und *Draw*. Durch das Verschieben einiger Methoden konnte ich diese

auch aus der Klasse *GameMain* entfernen (beispielsweise *AddAnimationCycle*).

Einschub: Game Components

Moment mal. Time-out. Kurze Pause. Da wäre nun also eine Basisklasse und Vererbung, und um das Ganze zu steuern – das haben Sie sofort erkannt (und sich die Umsetzung genau angeschaut) –, ist es nötig, manuell an den richtigen Stellen die Methoden *LoadContent*, *Update* und *Draw* aufzurufen. „Geht das nicht einfacher? Automatisch?“, fragen Sie, und ich antworte: „Ja. Grundsätzlich schon.“ Und falls Sie im Internet bereits nach Möglichkeiten dazu gesucht haben sollten, ist Ihnen aufgefallen, dass es in MonoGame sogenannte Komponenten gibt (*GameComponent*). Und davon gibt es verschiedene Varianten – so etwa die sogenannte *DrawableGameComponent*. Die Verlockung ist groß, damit zu arbeiten, und die Idee klingt simpel: Sie erstellen eine entsprechende Komponente, registrieren sie zu Beginn des Spiels, und überall dort, wo in der Klasse *GameMain* zum Beispiel *Draw* oder *Update* aufgerufen wird, werden automatisch auch die entsprechenden Methoden aller registrierten Komponenten aufgerufen. Warum sich also die Mühe machen, an dieser Stelle das Rad neu zu erfinden?

Nun, die Antwort ist einfach: Das Rad – um bei dieser Metapher zu bleiben – hat einen porösen Gummischlauch, mehrere Nägel im abgenutzten Profil und ist übrigens auch nicht

● Listing 1: GameCharacter

```
public abstract class GameCharacter(
    Game game, Vector2 startPosition, int movementSpeed)
{
    protected readonly Game1 Game = (Game1)game;
    protected AnimatedSprite sprite;
    protected Vector2 position = startPosition;
    protected readonly int speed = movementSpeed;
    protected Direction direction = Direction.Right;
    protected bool isMoving = false;

    public Vector2 Position => position;

    public Direction Direction => direction;

    // Öffentliche Methoden
    public abstract void LoadContent();

    public virtual void Draw(
        GameTime gameTime, SpriteBatch spriteBatch)
    {
        spriteBatch.Draw(sprite, position);
    }

    public abstract void Update(GameTime gameTime);

    public void SetX(float newX) => position.X = newX;

    public void SetY(float newY) => position.Y = newY;

    protected void AddAnimationCycle(SpriteSheet
        spriteSheet, string name, int[] frames,
        bool isLooping = true, float frameDuration = .15f)
    {
        var cycle = new SpriteSheetAnimationCycle
        {
            IsLooping = isLooping,
            FrameDuration = frameDuration
        };

        foreach (var frame in frames)
        {
            cycle.Frames.Add(new SpriteSheetAnimationFrame(
                frame, frameDuration));
        }

        spriteSheet.Cycles.Add(name, cycle);
    }
}
```

● Listing 2: ImGui Handler

```
private void RegisterImGuiHandlers()
{
    ImGui.Begin("Collision Detection");
    ImGui.Checkbox("Enabled",
        ref enableCollisionDetection);
    ImGui.End();

    ImGui.Begin("Tilemap");
    ImGui.Checkbox("Show Floor Layer",
        ref enableFloorLayer);
    ImGui.Checkbox("Show Decoration Layer",
        ref enableDecorationLayer);
    ImGui.Checkbox("Show Foreground Layer",
        ref enableForegroundLayer);
    ImGui.End();
}
```

für die Witterungsbedingungen geeignet, in denen Sie unterwegs sind.

Beispielsweise werden die entsprechenden Methoden durch die jeweiligen *Base.Draw*- und *-Update*-Aufrufe ausgelöst, und diese müssen Sie weiterhin manuell in Ihrem Code unterbringen. Zudem ist es oft essenziell, in welcher Reihenfolge die Methoden aufgerufen werden, und dazu müssten Sie zusätzlich alles konfigurieren.

Sie merken: Es ist deutlich einfacher, eigene Klassen zu schreiben und die entsprechenden Methoden einfach so aufzurufen, wie Sie sie benötigen. In der Tat ist es sogar so, dass im Internet eine rege Diskussion zu dem Thema entbrannt ist, in die sich sogar die ursprünglichen Entwickler der Basisklassen einmischen. Deren Argumentation lautet: Die Existenz dieser Klassen geht auf die ersten Versionen von XNA zurück, und der ursprüngliche Versuch war, die Idee von Windows Forms nachzuahmen. Tatsächlich aber stellt sich heraus, dass der Weg über eigene Klassen weit flexibler ist und sich damit sogar die ursprünglichen Entwickler ganz klar gegen die Nutzung von Game Components aussprechen. Eine interessante Diskussion dazu findet sich übrigens unter [7]. Meine Empfehlung lautet: Erstellen Sie einen Branch und experimentieren Sie damit – oder warten Sie noch einen Moment, denn wir werden die Game Components irgendwie doch gleich einsetzen ...

ImGui

Bisher wurden alle Debug-Fenster direkt in *GameMain* erzeugt. Nun aber müsste das in verschiedenen Klassen passieren, zum Beispiel in den Screens. Ich persönlich mag das aber an einem Ort. Daher belasse ich die Steuerung in *GameMain*. Damit das funktioniert, finden Sie nun in *GameMain* einen öffentlichen Event:

```
public event Action ImGuiRenderRequested;
```

Nach dem Beenden des *SpriteBatches* in *Draw* werden nun alle registrierten *ImGui*-Render-Requests gerendert:

```
GuiRenderer.BeginLayout(gameTime);
{
    // Lokales ImGui Overlay
    DrawImGuiOverlay(frameRate);

    // Registrierte ImGui Overlays
    ImGuiRenderRequested?.Invoke();
}
GuiRenderer.EndLayout();
```

In *MainScreen* wird passend dazu nun der entsprechende Handler registriert (und in *UnloadContent* wieder entfernt):

```
Game.ImGuiRenderRequested += RegisterImGuiHandlers;
```

Die Implementierung dazu finden Sie in [Listing 2](#).

MonoGame.Extended Milestone v4.0

Erinnern Sie sich an den Hinweis im letzten Artikel? MGE hat den Milestone v4.0 erreicht und ist damit wieder gleichauf mit der Entwicklung von MonoGame. Außerdem kam es zu so vielen Verbesserungen, dass ich sie hier gar nicht alle aufzählen kann. Ich versuche es daher auch nicht, sondern verweise auf die Release-Notes unter [8].

Ich hatte überlegt, die Aktualisierung auf MGE4 zu Beginn dieses Artikels durchzuführen, habe dann aber entschieden, sie an dieser Stelle lediglich zu erläutern. Denn was dazu erforderlich ist, ist viel besser auf der Website von MGE dokumentiert, und Sie halten mit dem entsprechenden Commit aus dem Repository sowieso das fertige Ergebnis in der Hand.

Neben den offensichtlichen Anpassungen – namentlich der Aktualisierung der Referenzen und der Tatsache, dass nun alle MGE-Bibliotheken in (fast) nur eine Bibliothek zusam- ►

● Listing 3: AddAnimationCycle für MGE4

```
protected void AddAnimationCycle(SpriteSheet
    spriteSheet, string name, int[] frames, bool
    isLooping = true, float frameDuration = .15f)
{
    spriteSheet.DefineAnimation(name, builder =>
    {
        builder.IsLooping(isLooping);

        for (int i = 0; i < frames.Length; i++)
        {
            builder.AddFrame(frames[i],
                TimeSpan.FromSeconds(frameDuration));
        }
    });
}
```

mengefasst wurden und es ein paar Änderungen in den Namenskonventionen gab – ist eine der für uns sichtbarsten Änderungen in *AnimatedSprite* zu finden.

Die Art, wie die Animationen geladen und im Sprite registriert werden, unterscheidet sich von der bisherigen Vorgehensweise. Eine angepasste Methode *AddAnimationCycle* finden Sie in [Listing 3](#).

Auch im *TextureAtlas* und in der Art, wie ein *SpriteSheet* initialisiert wird, gab es kleine Änderungen. Daher gab es Anpassungen in der Methode *LoadContent* in der *Player*-Klasse:

```
var spriteTexture = Game.Content.Load<Texture2D>("Player/SpriteSheet");
```

```
var spriteAtlas = Texture2DAtlas.Create("spriteAtlas", spriteTexture, 16, 16);
var spriteSheet = new SpriteSheet("spriteSheet", spriteAtlas);
```

Wichtig – und leider so nicht dokumentiert – ist vor allem aber auch, dass es am Sprite nun keine *Play*-Methode mehr gibt, mit der Sie die Animation abspielen können, so wie Sie es bisher in der Methode *Update* in der *Player*-Klasse getan haben. Dafür wird nun die Methode *SetAnimation* genutzt, der Sie den Namen der jeweiligen Animation übergeben. Dabei gibt es eine Besonderheit: Während das wiederholte Aufrufen der *Play*-Methode auf der gleichen Animation keine Auswirkung

Listing 4: Player-Update

```
public override void Update(GameTime gameTime)
{
    float dt =
        (float)gameTime.ElapsedGameTime.TotalSeconds;
    var keyboardState = Keyboard.GetState();

    isMoving = false;

    if (keyboardState.IsKeyDown(Keys.Up))
    {
        direction = Direction.Up;
        isMoving = true;
    }

    if (keyboardState.IsKeyDown(Keys.Down))
    {
        direction = Direction.Down;
        isMoving = true;
    }

    if (keyboardState.IsKeyDown(Keys.Left))
    {
        direction = Direction.Left;
        isMoving = true;
    }

    if (keyboardState.IsKeyDown(Keys.Right))
    {
        direction = Direction.Right;
        isMoving = true;
    }

    if (isMoving)
    {
        var movement = direction switch
        {
            Direction.Up => new Vector2(0, -speed * dt),
            Direction.Down => new Vector2(0, speed * dt),
            Direction.Left => new Vector2(-speed * dt, 0),
            Direction.Right => new Vector2(speed * dt, 0),
            _ => Vector2.Zero
        };

        position += movement;

        var animation = direction switch
        {
            Direction.Up => "walkUp",
            Direction.Down => "walkDown",
            Direction.Left => "walkLeft",
            Direction.Right => "walkRight",
            _ => sprite.CurrentAnimation // falls
            direction nicht passt, bleibt die aktuelle
            Animation
        };

        if (!sprite.CurrentAnimation.Equals(animation))
        {
            sprite.SetAnimation(animation);
        }
        else
        {
            if (!sprite.CurrentAnimation.Equals("idleDown"))
            {
                sprite.SetAnimation("idleDown");
            }
        }

        position = Vector2.Round(position * 5) / 5.0f;

        Game1.DebugRects.Add(new Tuple<RectangleF, Color>(
            new RectangleF(position.X + 2, position.Y + 12,
                12, 4), Color.Red));

        sprite.Update(gameTime);
    }
}
```

● Listing 5: GameScreen-Rumpf

```
using Microsoft.Xna.Framework;
using MonoGame.Extended.Screens;
using System;

namespace DevConfGame;

public class MainRoom(Game game) : GameScreen(game)
{
    private new Game1 Game => (Game1)base.Game;

    public override void Draw(GameTime gameTime)
    {
        throw new NotImplementedException();
    }

    public override void Update(GameTime gameTime)
    {
        throw new NotImplementedException();
    }
}
```

gen hatte, führt das wiederholte Aufrufen von *SetAnimation* dazu, dass die Animationssequenz jedes Mal zurückgesetzt wird. Sie müssen also prüfen, ob sich die Laufrichtung geändert hat, bevor Sie eine neue Animation setzen.

Eine Möglichkeit wäre hier die Anpassung des *Switch*-States in der *Update*-Methode der *Player*-Klasse, um vor Setzen einer Animation zunächst mittels *sprite.CurrentAnimation* zu prüfen, ob sich die Animation geändert hat. Ich schlage an dieser Stelle aber eine Alternative vor, einfach nur, um neue Sprachkonstrukte auszuprobieren: die *Switch*-Expression.

In Listing 4 finden Sie die neue Methode *Update*. Abschließend kam es zu einer Optimierung hinsichtlich der Einbindung der MGE-Content-Pipeline. In der Projektdatei habe ich folgende Zeilen hinzugefügt:

```
<PropertyGroup>
  <MonoGameExtendedPipelineReferencePath>$(
    MSBuildThisFileDirectory)pipeline-references</
    MonoGameExtendedPipelineReferencePath>
</PropertyGroup>
```

Damit wird die DLL der MGE-Content-Pipeline automatisch in das Projektverzeichnis kopiert, und Sie können sie nun in der Datei *Content.mgcb* direkt von dort referenzieren.

Die entsprechende Zeile sollte so aussehen:

```
/reference:..\pipeline-references\
  MonoGame.Extended.Content.Pipeline.dll
```

Damit ist das Projekt nun erfolgreich auf MonoGame.Extended 4.0.0 migriert.

Erkennen Sie im Laufzeitverhalten irgendwelche Unterschiede zu vorher? Ja? Macht nichts. Nein? Auch gut. Kommen wir nun zum Kernthema dieses Artikels.

MonoGame.Extended.Screens

Screen-Management also. MGE bietet hierzu den sogenannten *ScreenManager* an. Enthalten ist die Klasse *GameScreen* als Basis für Ihre separaten Screens. Damit haben diese eigene *Initialize*-, *LoadContent*-, *Update*- und *Draw*-Methoden. Auch sehr nett: Für den Wechsel zwischen zwei Screens können Sie sich eine der vorhandenen Transitionen als Übergangseffekt aussuchen ... ganz so wie in PowerPoint. Allerdings gibt es in MGE derzeit nur zwei vorgefertigte Übergangseffekte.

Hier die Idee:

- Namespaces hinzufügen
- GameScreens implementieren
- ScreenManager instanzieren
- ScreenManager als Game Component registrieren
- Screens entsprechend den Anforderungen laden

Aktuell verfügen Sie in Ihrem Spiel über zwei Maps: *StorageScreen* und *MainScreen*. Entsprechend habe ich für Sie bereits zwei neue Klassen ergänzt, die genauso heißen und von *GameScreen* erben. Der Rumpf der Klassen sieht aus wie in Listing 5.

Mit der privaten Variable *Game* überschreiben Sie die *Game*-Variable aus der Basisklasse *GameScreen* und wandeln sie in Ihren spezifischen Typ *GameMain* um, was geht, da beide von der Klasse *Game* erben.

Bis zuletzt passierte alles hinsichtlich der Steuerung des ursprünglichen *MainRoom* in der Klasse *Game1*. Die *TiledMap* wurde geladen und gerendert, die *Collision Detection* gesteuert und mehr. Das wurde nun in die jeweiligen GameScreens überführt. Beziehungsweise für den *MainScreen*. Zum *StorageScreen* kommen wir noch. Auf diese Weise kann sich die Klasse *GameMain* nun nicht nur auf die Steuerung des Spiels selbst konzentrieren, sondern auch sehr leicht dem Spiel neue GameScreens hinzufügen.

In *GameMain* finden Sie dazu nun einige Variablen, über die die Screens registriert und verwaltet werden:

```
readonly Dictionary<ScreenName, GameScreen>
  screens = [];
readonly ScreenManager screenManager;
ScreenName currentScreen;
```

Im Constructor wird der *ScreenManager* instanziiert und als Game Component registriert:

```
screenManager = new ScreenManager();
Components.Add(screenManager);
```

Damit gilt es nun darauf zu achten, dass das Laden, Aktualisieren und Zeichnen der Screens sinnvoll in den entsprechenden Methoden in *GameMain* ausgelöst wird.

In *Initialize* fügen Sie nun die Screens hinzu: ►

Listing 6: LoadScreen

```
public void LoadScreen(ScreenName screen,
    EventHandler onStateChanged = null)
{
    var transition = new
    FadeTransition(GraphicsDevice, Color.CornflowerBlue,
        0.5f);
    if (onStateChanged != null)
    {
        transition.StateChanged += onStateChanged;
    }

    screenManager.LoadScreen(screens[screen],
        transition);
    currentScreen = screen;
}
```

```
screens.Add(ScreenName.MainScreen, new MainScreen(
    this, spriteBatch));
screens.Add(ScreenName.Storage-
Screen, new StorageScreen(this, spriteBatch));
```

Zudem benötigen Sie die Möglichkeit, einen Screen zu laden. Dazu finden Sie im Code die Methode *LoadScreen* (Listing 6). Sie wird einfach in *GameMain.LoadContent* aufgerufen.

Listing 7: CollisionCheck

```
public TiledMapTilesetTile CollisionCheck(
    TiledMapTileLayer layer, Vector2 position,
    Direction direction, string tileName = "")
{
    var tilePositions = GetRelevantTiles(
        position, direction);

    TiledMapTilesetTile collisionTile;

    foreach (var tilePos in tilePositions)
    {
        collisionTile = CollisionDetected(
            layer, tilePos, position, tileName);

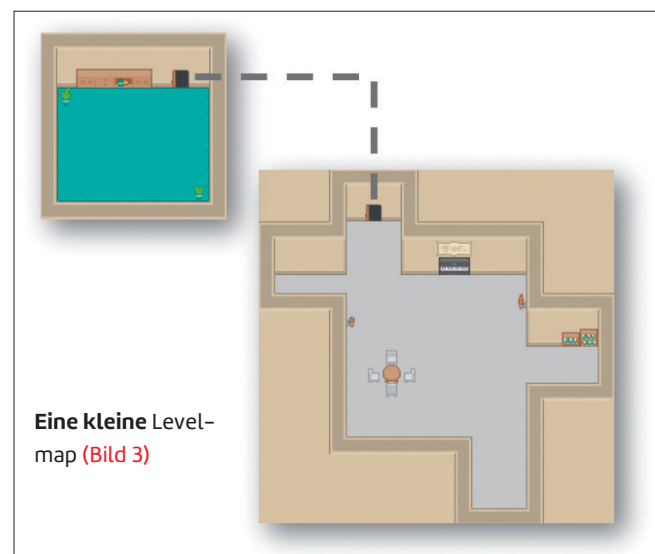
        if (collisionTile != null)
        {
            return collisionTile;
        }
    }

    return null;
}
```

The Doors

Um nun zwischen GameScreens hin- und herwechseln zu können, muss das Spiel erkennen, dass der Player zum Beispiel „durch eine Tür geht“. Im Grunde genommen genügt es dazu, die Kollision mit der Tür zu erkennen und darauf zu reagieren. Sie erinnern sich daran, dass ich zu Beginn des Artikels im Zusammenhang mit der Collision Detection kurz erwähnt habe, dass an dieser Stelle eine Änderung erforderlich sein wird. In der Annahme, dass Sie sich entweder den Refactoring Commit zu diesem Artikel aus dem Repository geholt haben oder aber Ihren eigenen Code selbstständig so weit aktualisiert haben, besteht der nächste Schritt nun darin, die Collision Detection zu erweitern.

Bisher gibt die Methode *CollisionCheck* nur ein *true* oder *false* zurück, je nachdem ob eine Kollision erkannt wurde – oder eben nicht. Sie benötigen nun das konkrete Tile, mit dem der Player kollidiert ist, um anhand von dessen Eigenschaften prüfen zu können, ob eine Aktion erforderlich ist (zum Beispiel ein Wechsel des GameScreens).



Beginnen Sie damit, das *TiledMapTilesetTile* bereits in *CollisionDetected* zurückzugeben:

```
private TiledMapTilesetTile CollisionDetected(
    TiledMapTileLayer layer, Point tilePosition,
    Vector2 playerPosition, string tileName = "")
```

Wie Sie erkennen, können Sie nun auch einen *tileName* mitgeben. Sofern Sie das tun, fragen Sie ihn beim Prüfen auf Kollision auch ab:

```
if (tilesetTile?.Objects?.Count > 0 && (
    string.IsNullOrEmpty(tileName) ||
    tilesetTile.Objects[0].Name == tileName))
```

Und bei einer Kollision geben Sie das Tile dann entsprechend zurück:

● Listing 8: Screen-Wechsel

```
public override void Update(GameTime gameTime)
{
    // Backup Position
    var playerPos = Game.Player.Position;

    Game.Player.Update(gameTime);
    tiledMapRenderer.Update(gameTime);

    var collisionTile =
        collisionDetector.CollisionCheck(floorLayer,
            Game.Player.Position, Game.Player.Direction);
    var collisionStorageDoor =
        collisionDetector.CollisionCheck(
            decorationLayer, Game.Player.Position,
            Game.Player.Direction, "StorageDoor");

    if (enableCollisionDetection && (collisionTile
        != null || collisionStorageDoor != null))
    {
        // Revert Position
        Game.Player.SetX(playerPos.X);
        Game.Player.SetY(playerPos.Y);
    }

    if (collisionStorageDoor != null)
    {
        Game.LoadScene(ScreenName.StorageScreen,
            (sender, e) =>
            {
                Game.Player.SetX(80);
                Game.Player.SetY(90);
            });
    }
}
```

```
if (collision)
{
    Game1.DebugRects.Add(new Tuple<RectangleF, Color>(
        globalRect, Color.Red));
    return tilesetTile;
}
else
{
    Game1.DebugRects.Add(new Tuple<RectangleF, Color>(
        globalRect, Color.Green));
    return null;
}
```

Die Rückgabe von *collision* (*true/false*) entfällt nun, und auch wenn keine Kollision vorliegt, wird null zurückgegeben.

Die aufrufende Methode *CollisionCheck* erhält nun als Rückgabe von *CollisionDetected* entweder das Tile, mit dem

die Kollision stattgefunden hat, oder aber eben nichts. Auch hier ändern Sie die Signatur und übergeben bei Bedarf einen *tileName*. Den vollständigen Code finden Sie in [Listing 7](#).

Sobald Sie nun die Methode *Update* in der Klasse *MainScreen* entsprechend anpassen, sodass neben allgemeinen Kollisionen auch solche mit der Tür zum Lager geprüft werden (Achtung, diese befindet sich in einem anderen Layer!), können Sie auch den Wechsel in den *StorageScreen* auslösen. Die Implementierung dazu finden Sie in [Listing 8](#).

Das wird im Moment zwar insofern funktionieren, als dass der *MainScreen* bei Kontakt mit der Tür ausgeblendet wird. Dann erfolgt aktuell allerdings eine *NotImplementedException*, denn der *StorageScreen* wurde noch nicht implementiert. Sowas.

StorageScreen

Probieren Sie die Klasse *StorageScreen* selbst auszuimplementieren. Tipp: Der Code ist praktisch identisch zu dem aus *MainScreen*, da es noch kaum Unterscheidungen im Verhalten gibt. Denken Sie lediglich daran, beim Zusammenstoß mit der Tür zurück zum *MainScreen* zu wechseln und dabei sinnvolle Startkoordinaten für den Player anzugeben. Und voilà – schon haben Sie eine kleine Levelmap ([Bild 3](#)).

NPCs

Für dieses Spiel werden die NPCs nicht gerade mit Intelligenz ausgestattet sein. Im Grunde genommen genügt es, wenn sich die NPCs entlang festgelegter Pfade bewegen. Da solche Pfade auch nur aus einzelnen Wegpunkten bestehen, ist es nicht schwer, das zu implementieren. Für den Einstieg erstellen Sie der Einfachheit halber eine Klasse *NonPlayer* mit einer entsprechenden Signatur (siehe [Listing 9](#)). ►

● Listing 9: NonPlayer

```
public class NonPlayer(
    Game game, List<Vector2> waypoints) :
    GameCharacter(game, new Vector2(100, 100), 85)
{
    private int currentWaypointIndex = 0;

    // Radius um einen Wegpunkt, in dem
    // er als erreicht gilt

    private float waypointRadius = 5f;

    public override void LoadContent()
    {
    }

    public override void Update(GameTime gameTime)
    {
    }
}
```

Beim Instanzieren des NPC können Sie dann einfach eine Liste von Wegpunkten übergeben. Interessant ist, dass der NPC nun selbstständig die Punkte wird ablaufen müssen. Das wird in der Methode *Update* passieren, und dort ist es auch erforderlich, dass Sie einen Weg finden, dafür zu sorgen, dass der NPC „merkt“, dass er einen Punkt auch erreicht hat. Damit das funktioniert, und weil die Strecke, die der NPC pro Frame zurücklegt, nicht immer identisch sein muss, ist es sinnvoll, einen Radius festzulegen, innerhalb dessen der Zielpunkt als erreicht gilt.

Übernehmen Sie als Nächstes die Methode *LoadContent* aus der *Player*-Klasse. Aktuell gibt es nur ein einziges SpriteSheet, sprich: Wir erschaffen zunächst einen Zwilling.

Wenden Sie sich nun der Methode *Update* zu. Der Teil, der die Animation steuert, bleibt identisch zum Player:

```
public override void Update(GameTime gameTime)
{
    float dt = (float)gameTime.ElapsedGameTime.TotalSeconds;

    var animation = direction switch
    {
        Direction.Up => "walkUp",
        Direction.Down => "walkDown",
        Direction.Left => "walkLeft",
        Direction.Right => "walkRight",
        _ => sprite.CurrentAnimation // falls
        // direction nicht passt, bleibt die aktuelle
        // Animation
    };

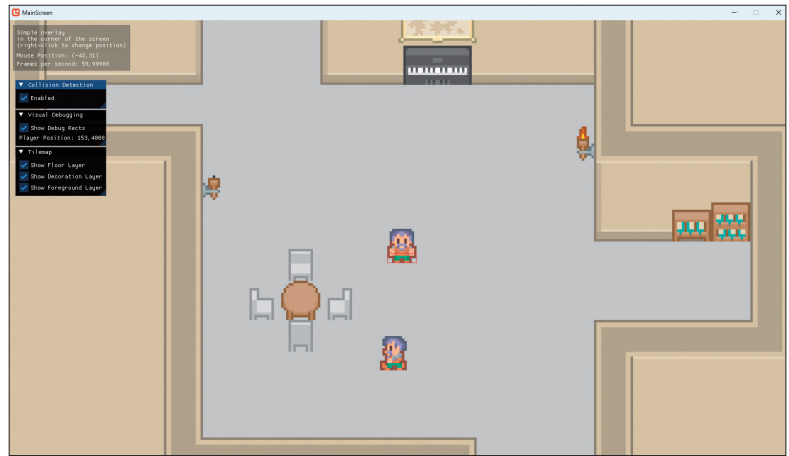
    if (!sprite.CurrentAnimation.Equals(animation))
    {
        sprite.SetAnimation(animation);
    }

    if (directionToWaypoint.LengthSquared() == 0)
    {
        if (!sprite.CurrentAnimation.Equals("idleDown"))
            sprite.SetAnimation("idleDown2");
    }

    position = Vector2.Round(position * 5) / 5.0f;
    sprite.Update(gameTime);
}
```

Nach Ermittlung der Deltatime müssen Sie nun zunächst prüfen, ob der nächste Wegpunkt erreicht wurde. Dazu berechnen Sie einfach die Distanz zwischen der aktuellen Position und dem Wegpunkt und finden heraus, ob sie kleiner als der vorgegebene Radius ist. Ist das der Fall, setzen Sie den nächsten Wegpunkt fest.

Die Klasse *Vector2* hält für die Berechnung praktischerweise eine entsprechende Methode für Sie bereit:



Lauf, Forrest, lauf (Bild 4)

```
// Prüfe, ob der aktuelle Wegpunkt erreicht wurde
if (Vector2.Distance(position,
    waypoints[currentWaypointIndex]) < waypointRadius)
{
    // Gehe zum nächsten Wegpunkt
    currentWaypointIndex =
        (currentWaypointIndex + 1) % waypoints.Count;
}
```

Als Nächstes ermitteln Sie die Richtung zum nächsten Wegpunkt und implementieren dann eine Strategie, um den NPC dorthin zu bewegen. Da wir aktuell die Spieler nur horizontal und vertikal bewegen, wäre es eine Möglichkeit, zunächst in die Richtung zu laufen, deren Abstand zum Wegpunkt am größten ist.

Fügen Sie also folgenden Code vor der Animation ein:

```
// Bestimme die Bewegungsrichtung basierend auf der
// dominanten Komponente
if (Math.Abs(directionToWaypoint.X) >
    Math.Abs(directionToWaypoint.Y))
{
    // Bewege horizontal
    if (directionToWaypoint.X < 0)
    {
        direction = Direction.Left;
        position.X -= speed * dt;
    }
    else if (directionToWaypoint.X > 0)
    {
        direction = Direction.Right;
        position.X += speed * dt;
    }
}
else
{
    // Bewege vertikal
    if (directionToWaypoint.Y < 0)
    {
        direction = Direction.Up;
        position.Y -= speed * dt;
    }
}
```

```

else if (directionToWaypoint.Y > 0)
{
    direction = Direction.Down;
    position.Y += speed * dt;
}
}

```

Das war es auch schon. Nun müssen Sie den NPC noch instanzieren. Ergänzen Sie daher in *GameMain* eine Property *NPC* analog zu *Player*. Im Constructor erzeugen Sie sie und übergeben dabei eine Liste von Wegpunkten:

```

NPC = new NonPlayer(this, waypoints:
[
    new(80, 100),
    new(200, 100),
    new(200, 200),
    new(80, 200)
]);

```

In der Methode *LoadContent* lassen Sie den NPC dessen Assets laden – ebenfalls analog zum Player. Und abschließend fehlt noch die Kontrolle über den NPC. Um konform mit den aktuellen Design-Entscheidungen zu bleiben, lösen Sie das Update in *MainScreen.Update* aus, direkt im Anschluss an den Player, und ergänzen ebenfalls einen Aufruf auf dessen *Draw*-Methode im *SpriteBatch* der Methode *MainScreen.Draw*.

Probieren Sie nun aus, ob alles funktioniert. Falls nicht, beheben Sie etwaige Probleme, oder vergleichen Sie Ihren Code mit dem aus dem Repository. Falls alles geklappt hat, sollte der NPC nun im *MainScreen* immer schön im Rechteck laufen (Bild 4).

Challenge 1

Hier eine kleine Challenge: Damit das Spiel zum Schluss nicht aussieht wie diese Sequenz in *Matrix*, in der Neo gegen unendlich viele Iterationen von Agent Smith kämpft ... und in unserem Fall schlimmer noch: mit sich selbst spielt (Sie wissen, wie das gemeint ist!), probieren Sie doch einmal, für einen NPC ein weiteres *PlayerSprite* zu entwerfen. Vielleicht möchten Sie ja sogar schon mehrere NPCs ergänzen.

Challenge 2

Für das finale Spiel wird es wenig sinnvoll sein, dass die NPCs in permanenter Bewegung sind. Sie brauchen eine Möglichkeit, Rastpunkte festzulegen. Vorstellbar wären mehrere Stehtische, an denen die NPCs verweilen, um ihren Kaffee zu genießen, bevor sie zurück in die Konferenzsäle laufen.

Challenge 3

Angenommen, es wäre in Zukunft auch sinnvoll, dass sich ein Game Character auch diagonal bewegen kann. Was für Änderungen wären dazu erforderlich? Welche Animationen würden Sie verwenden? Gibt es etwas Wichtiges zu beachten? Probieren Sie es einfach aus, und beginnen Sie dabei beim Player selbst, bevor Sie sich der Frage stellen, wie das

mit dem NPC umgesetzt werden kann. An dieser Stelle übrigens auch der Hinweis, dass wir das vielleicht gar nicht benötigen für das Spiel, aber Sie brauchen ja auch etwas, um die Zeit bis zum nächsten Artikel zu überbrücken, oder?

Nachdem Sie nun ein großes Refactoring hinter sich gebracht sowie ein Screen-Management implementiert haben und NPCs Ihr Spiel zu bevölkern beginnen, kommen wir abschließend zu der versprochenen Ergänzung für die Ohren. Im Moment ist das Spiel noch recht leise, und es wäre für den Anfang nett, dass uns der *MainScreen* mit einer Hintergrundmusik begrüßt. Außerdem – und um der Trennung der Game-Screens etwas mehr Sinn abzugewinnen – könnte im *StorageScreen* eine andere Musik laufen.

Musik

An dieser Stelle muss ich mal etwas loswerden. Ich liebe das Internet. Ich war dabei, als es in Deutschland langsam ankam, und habe alle Phasen miterlebt, bis es zu dem wurde, was wir heute erleben. Ich habe mittels Modem über eine direkte Telefonverbindung Dateien von Mailboxen heruntergeladen, war über AOL mit dem IRC verbunden, hatte Post (und die Stimme dazu noch immer im Kopf), habe Musik über Napster heruntergeladen, mich fremdgeschämt, als Boris sagte: „Bin ich schon drin? Das war ja einfach!“, und heute unterhalte ich mich mit KI-Bots über Code-Optimierungen.

„Wozu dieser Ausflug in die Historie des Internets?“, fragen Sie sich, und hier kommt die Antwort: Vergessen Sie das Mantra, das Internet sei so groß, dass man darin nichts wiederfinden würde. Denn während es früher nur wenige Nodes gab und dafür überschaubare „Gelbe Seiten“, gibt es heute

● Contact Challenge

Du bist schon so lange online unterwegs, dass sich das „Siezen“ im Internet für dich noch immer irgendwie falsch anfühlt? Auf *ChatCity* war die erste Frage, die du damals gestellt hast, immer „asl?“? Die Quota von 2kb Up für 4kb Down in Mailboxen hat dich auch genervt? Du bist mit Ralph Browns Interrupt-Liste groß geworden, und manchmal, wenn eine Nachricht von irgendwoher hochpoppt, hörst du noch immer dieses „Oh oh!“ von ICQ? Dann kommt hier ganz speziell deine Challenge: Schreib mir eine Netmail an 2:240/1120.34, und du bekommst von mir einen Link für einen **kostenlosen** Download des All-in-One Asset Packs von Kenney.

Du hast richtig gelesen! Ich habe mit Kenney gesprochen, und er hat ein paar Voucher-Codes springen lassen. Für den Fall, dass du mit der Adresse nichts anfangen kannst: Keine Sorge. Es wird noch mehr Chancen geben.

zwar viele Nodes, allerdings auch wirklich gute Suchmaschinen. Und wir haben schon einmal darüber gesprochen, dass es für einen Spieleentwickler essenziell ist, gute Assets zu finden. Also trainieren Sie Ihr Such-Mojo. ►

Zurück zum Thema Musik. Durch einen Unity-Kurs auf Udem.org wurde ich aufmerksam auf einen Loop, der mir so sehr gefiel, dass ich mir vornahm, ihn unbedingt auch zu nutzen, sofern ich mal einen Kurs für Spieleentwicklung schreiben würde. Da bin ich nun und wollte diesen Loop entsprechend in diese Artikelserie einbinden. Doch wie ist das mit den Lizenzen?

Sie erinnern sich daran, dass ich auch dieses Thema in ausreichendem Maße adressiert habe. Unglücklicherweise fand sich ausgerechnet zur Musik in dem Udem.org-Kurs keinerlei Hinweis auf die Quelle der verwendeten Stücke, und im Internet zufällig nach .wav-Dateien zu suchen, die über keinerlei Metadaten verfügen, klingt nach einer aussichtslosen Sache, richtig? Falsch. Es hat – zugegebenermaßen – etwas gedauert, aber ich habe die Quelle gefunden. Auf der Seite Opengameart.org gibt es viele Loops für Spieleentwickler, und ich klickte mich durch, hoffend, wenigstens auf etwas Ähnliches zu stoßen.

Und in der Tat führte mich eine der Previews zu einem Stück, das sowohl vom Stil wie auch den verwendeten Samples her dem Stück, das ich suchte, extrem ähnlich war. Ich wurde schlagartig hellwach, wechselte zur Seite des Autors, fand dort vier Stücke, klickte dort das erste an, und bam! Das Gefühl, das richtige Sandkorn an einem der größten Strände unserer Asset-Meere gefunden zu haben, war unbeschreiblich! „Zu viel Pathos!“, denken Sie? Akzeptiert. Immerhin aber profitieren Sie nun von einer exquisiten Auswahl an Loops für diese Serie und beginnen hier und jetzt mit der Implementierung der Hintergrundmusik!

Laden Sie zunächst die vier Loops von [9] herunter. Auf der Website finden Sie Archive sowohl mit WAV- wie auch OGG-Dateien. Sie entscheiden sich für die OGG-Dateien, denn als Alternative zu MP3 ist das OGG-Vorbis-Format patentfrei; so riskieren Sie bei kommerzieller Nutzung keine Lizenzprobleme. Werfen Sie einen Blick in die *INFO.txt* darin: Die Musik steht unter CC0-Lizenz, und Sie wissen ja, was das bedeutet.

Öffnen Sie nun das Content-Builder-UI mit einem Doppelklick auf die Datei *Content.mgcb*, und ergänzen Sie direkt unter *Content* ein neues Verzeichnis mit dem Namen *Music*. Kopieren Sie die entpackten vier OGG-Dateien über den Windows-Explorer in das neue Verzeichnis und benennen Sie um:

- *Stage 1* --> *StorageScreenBG*
- *Stage 2* --> *MainScreenBG*
- *Boss Fight* --> *HurryUpBG*
- *Stage Select* --> *StartScreenBG*

HurryUpBG und *StartScreenBG* benötigen wir zwar noch nicht. Aber wenn wir die Loops schon einmal haben ...

Fügen Sie die vier existierenden Dateien dem Musik-Verzeichnis im Content Builder UI hinzu und kompilieren Sie die Assets testhalber. Das Ergebnis sollte in etwa aussehen wie in **Bild 5**. Schließen Sie das Tool wieder.

Die Einbettung in die vorhandenen Screens ist denkbar einfach: Sie beginnen mit dem *MainScreen* und ergänzen in der dazugehörigen Klasse eine Variable *backgroundMusic* vom Typ *Song*:

```
Song backgroundMusic;
```

In *LoadContent* laden Sie die Music und starten direkt das Playback:

```
backgroundMusic = Content.Load<Song>("Music/MainScreenBG");
MediaPlayer.Play(backgroundMusic);
MediaPlayer.IsRepeating = true;
```

Um sicherzugehen, dass die Musik nicht aufgrund irgendwelcher Ereignisse unterbrochen wird, können Sie präventiv in der *Update*-Methode auch noch ein paar Zeilen ergänzen, die die Wiedergabe im Zweifelsfall erneut starten oder fortsetzen:

```
if (MediaPlayer.State == MediaState.Stopped)
    MediaPlayer.Play(backgroundMusic);
else if (MediaPlayer.State == MediaState.Paused)
    MediaPlayer.Resume();
```

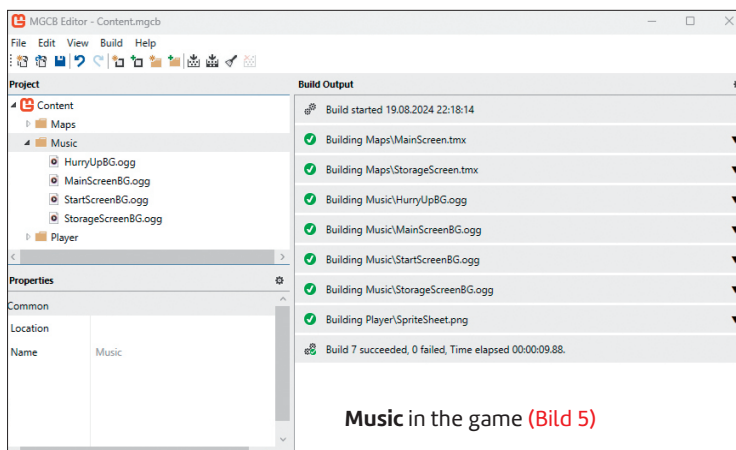
Starten Sie das Spiel – und bitte sagen Sie auch, dass der Loop süchtig macht!

Wenn Sie nun mit dem Player in den *StorageScreen* wechseln, werden Sie feststellen, dass die Musik dort einfach weiterspielt. Das liegt schlicht daran, dass der *MediaPlayer* eine statische Klasse ist. Da aktuell aber beim Wechsel zurück zum *MainScreen* dieser praktisch neu geladen wird (darum kümmern wir uns später noch), wird die Musik entsprechend einfach zurückgesetzt. Wollen Sie, dass die Musik permanent im Hintergrund den gleichen Loop spielt, würden Sie die Implementierung in der Klasse *GameMain* umsetzen.

Im Moment möchten Sie aber in den beiden Screens unterschiedliche Loops laufen lassen. Ergänzen Sie daher selbstständig die Hintergrundmusik für den *StorageScreen*.

Tipp

Gibt es aber vielleicht einen Kompromiss, um die geloopte Hintergrundmusik dennoch sauber abzubil-



den? Ja, gibt es. Der Grund für den Gap ist nämlich – vereinfacht gesagt – die Tatsache, dass MonoGame Songs nicht in den Speicher lädt, sondern aus der Datei streamt. Soundeffekte aber werden zuvor in den Speicher geladen, und da das

● Qualität des MediaPlayers

Ihnen wird mittlerweile aufgefallen sein, dass der in MonoGame integrierte MediaPlayer den Loop scheinbar nicht sauber abbilden kann. In dem Moment, in dem die Musik wieder an den Anfang springt, gibt es eine wahrnehmbare Pause. Das ist unter anderem auch der Grund, warum viele Spieleentwickler davon abraten, den MonoGame-MediaPlayer zu verwenden. Natürlich gibt es diverse alternative Lösungen. Die bekannteste dürfte vermutlich FMOD sein. Allerdings benötigt man dazu ein Konto auf der Website des Herstellers, und zusätzlich gibt es Lizenzbedingungen, die unter anderem auch Gebühren verlangen. Sollten Sie sich dennoch mit FMOD vertraut machen wollen, finden Sie unter [10] einen C#-Wrapper, der sich in MonoGame nutzen lässt.

Abspielen selbst (unter der Haube) im Grunde genommen identisch ist, können Sie also Folgendes tun:

Ändern Sie den Content Processor in MGCB von *Song* auf *SoundEffect* für die vier Loops. Und dann ändern Sie den Code, um statt eines Songs einen Soundeffekt zu laden:

Im Headerbereich der Klasse *MainScreen*:

```
private SoundEffect song = null;
private SoundEffectInstance songInstance = null;
```

In *LoadContent*:

```
song = Content.Load<SoundEffect>("Music/MainScreenBG");
songInstance = song.CreateInstance();
songInstance.IsLooped = true;
songInstance.Play();
```

In *UnloadContent*:

```
songInstance.Stop();
```

Wiederholen Sie das auch für den *StorageScreen*, und vergessen Sie dabei nicht, dass Sie den Teil, der Songs in der *Update*-Methode fortsetzt, nicht mehr benötigen. Sobald Sie das Spiel nun starten, sollte der Loop perfekt abgespielt werden.

Es gibt eine rege Diskussion im Internet zu der Frage, ob das eine gute Lösung ist oder nicht. Für unseren Zweck ist es für den Moment jedenfalls ausreichend. Probleme könnten auftreten, wenn das Spiel für mobile Plattformen entwickelt würde, da sich die Entscheidung, Musik als Soundeffekt zu laden, möglicherweise auf die Größe der kompilierten Content-Dateien auswirkt.

Fazit

Mehrere Screens, NPCs, Hintergrundmusik – langsam, aber stetig entwickelt sich das Spiel zu etwas, das wirklich Spaß macht. Was mir im Moment noch fehlt, ist Feedback von Ihnen. In diesem Artikel haben Sie den ersten Aufruf gefunden, mit mir einfach mal Kontakt aufzunehmen. Okay, „einfach“ sei einmal dahingestellt. Denn schätzungsweise gibt es in Deutschland vielleicht noch 100 Menschen, die mit der oben genannten Adresse etwas anfangen könnten, und die gehören sicher nicht zur Zielgruppe dieser Artikelserie. Wichtiger aber war mir, dass Sie nun wissen, dass ich etwas zu verschenken habe. Vielleicht lassen Sie sich ja auch von dieser obskuren Adresse nicht aufhalten ...

Ansonsten freuen Sie sich auf den kommenden Artikel. Wir werden uns mit Soundeffekten (also wirklichen Soundeffekten, nicht noch mehr geloopter Musik) auseinandersetzen, uns mehr Gedanken um die NPCs machen, erstmals Textausgaben einbauen (zum Beispiel ablaufende Zeit), und falls noch eine oder zwei Seiten übrig sind, beschäftigen wir uns vielleicht noch mit einem Partikeleffekt #Staub.

Bis dahin denken Sie daran: Integrieren Sie ein paar NPCs auf Basis verschiedener SpriteSheets, machen sich schon einmal Gedanken um deren Verhalten, oder basteln Sie an den Maps herum (#Stehtische).

Bis zum nächsten Monat!

- [1] *Christian Jacob, Ja wo läuft er denn hin?*, www.dotnetpro.de/A2408MonoGame
- [2] *Christian Jacob, Es begann mit XNA ...*, www.dotnetpro.de/A2406MonoGame
- [3] *Christian Jacob, Da bewegt sich doch was?*, www.dotnetpro.de/A2407MonoGame
- [4] *Christian Jacob, Perfekte Pixel für Enthusiasten*, www.dotnetpro.de/A2409MonoGame
- [5] *Christian Jacob, Zwischen den Ebenen*, www.dotnetpro.de/A2410MonoGame
- [6] *GitHub-Repository zur Artikelserie*, www.dotnetpro.de/SL2412MonoGame1
- [7] *Game-Component-Kommentar auf StackExchange*, www.dotnetpro.de/SL2412MonoGame2
- [8] *MonoGame.Extended 4.0.0 Release Notes*, www.dotnetpro.de/SL2412MonoGame3
- [9] *Chiptunes*, www.dotnetpro.de/SL2412MonoGame4
- [10] *FMOD-Wrapper*, www.dotnetpro.de/SL2412MonoGame5



Christian Jacob

ist Entwickler und Co-Founder der Blue Sky Innovations GmbH. Er unterstützt Unternehmen bei der Einführung moderner Strategien sowie bei Planung und Umsetzung individueller Softwarelösungen auf Basis von Microsoft .NET.

cjacob@blueskyinnovations.de

dnpCode

A2412MonoGame

.NET MAUI AUF SAMSUNG-SMART-TVS

Die große Freiheit

Samsung setzt im Bereich der Smart-TV-Applikationsentwicklung schon länger auf .NET-Technologien. Vergleichsweise neu ist dagegen die Möglichkeit der MAUI-Entwicklung.

Ziel dieses Artikels ist eine kurze Vorstellung der Möglichkeiten und Grenzen der MAUI-Entwicklung im Bereich der Smart-TVs. Außerdem soll auf einige Besonderheiten und Änderungen im Bereich der Entwicklungswerkzeuge eingegangen werden.

Gerade in Bezug auf Usability und Co. sei darauf hingewiesen, dass die in der dotnetpro-Ausgabe 1/2024 getroffenen Feststellungen [1] nach wie vor uneingeschränkt gültig sind. Insbesondere gilt, dass ein Smart-TV weder eine Workstation noch ein Handcomputer ist – nicht jeder Inhalt, der auf den letztgenannten beiden Plattformen problemlos funktioniert, lässt sich auch auf Fernsehgeräten gewinnbringend oder zu einer allgemein hohen Benutzerzufriedenheit deployen.

Zwangsupgrades

Ein Grund, warum Fernsehgeräte-Hersteller den Trend zum „Smart-TV“ forcieren, ist nach Ansicht des Autors die Sicherstellung von permanenten Aktualisierungen der Fernsehgeräte. Der Unterschied zwischen HD und 4K mag merkbar sein, der Sprung von 4K auf 8K jedoch ist kaum noch sichtbar und dürfte darum nur wenige Benutzer zum Umstieg animieren. Um neue Kaufanreize zu schaffen, setzt man in Korea deshalb konsequent auf Aktualisierungen des Betriebssystems: Bei der Erstellung dieses Artikels ist Tizen 7.0 im Handel, das SDK wurde mittlerweile allerdings bereits auf Version 8.0 aktualisiert. Allgemein gilt dabei die unter [2] bereitstehende Kompatibilitätsliste, welche die Zuordnung von TV-Geräteserien zu den Betriebssystemversionen ermöglicht.

Wenige Tage vor Abschluss der Arbeiten an diesem Artikel kündigte Samsung an, in Zukunft die mit Tizen 8 ausgelieferten Geräte mit Updates auf neue Versionen zu versorgen – für

existierende Hardware mit Tizen 7 gilt dieses Angebot dem Vernehmen nach nicht oder nur sehr eingeschränkt.

Allerdings kommen insbesondere in Europa oft leicht unterschiedliche Produktbezeichnungen zum Einsatz. Der einfachste Weg zum schnellen Feststellen der Plattform-Version eines vorliegenden Smart-TV ist ein Aufruf des URL `http://j2i.net/apps/userAgent/`, der – wie **Bild 1** zeigt – eindeutig über den Versionsstand auf dem jeweiligen Gerät informiert.

Im Hintergrund gibt es dabei allerdings ebenfalls Veränderungen: Smart-TVs mit älteren Versionen inklusive Tizen 6.5 werden von MAUI nicht unterstützt. Cross-Plattform-Benutzerinterfaces für diese ältere Gerätegeneration entstehen ausschließlich unter Xamarin Forms, wobei anzumerken ist, dass solche Xamarin-Forms-Applikationen auch auf Fernsehgeräten mit Tizen 7.0 problemlos funktionieren.

Außerdem bietet Samsung mit NUI noch einen komplett eigenständigen GUI-Stack an, der allerdings nicht hier, sondern erst in einem Folgeartikel zur Sprache kommen soll.

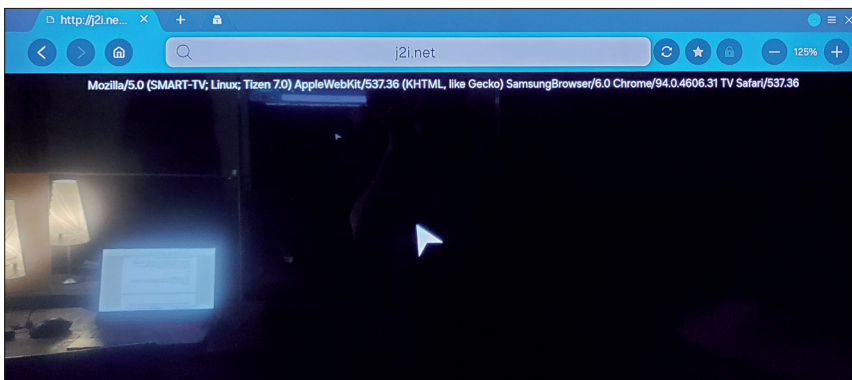
Aktualisierung des Stand-alone-SDK

Trotz der Änderungen im Bereich des Smartwatch-Betriebssystems hat sich Samsung zumindest zum Zeitpunkt der Abfassung dieses Artikels noch nicht von der Illusion gelöst, dass Tizen irgendwann auch im Handcomputer- und Smartwatch-Markt Marktdominanz erreichen könnte.

Konsequenz für Entwickler ist, dass das SDK nach wie vor „janusköpfig“ aufgebaut ist – die in Visual Studio lebende Instanz interagiert, analog zum Fall von Android, mit einer alleinstehenden, vollständigen Variante der sonstigen Entwicklungswerkzeuge. Wer bisher keine größere Zertifikats-Infrastruktur eingerichtet hat, ist deshalb gut beraten, das Tizen-

Plug-in zu löschen und zudem den Tizen-Deinstallationsassistenten für das SDK zu verwenden. Im Visual Studio Marketplace ist danach über *Extras | TiZen | TiZen Package Manager* die Erweiterung *Visual Studio Tools for Tizen* herunterzuladen – der darin enthaltene Paketmanager erlaubt dann das abermalige Deployen der diversen SDK-Komponenten.

Alternativ dazu ist es möglich, den Paketmanager mit der Aktualisierung der vorliegenden Komponenten zu beauftragen. Wichtig ist dabei, dass die in Visual Studio enthaltene Komponente ebenfalls aktualisiert werden muss.



Dieser Smart-TV läuft mit Tizen 7.0 (**Bild 1**)

● Aktivierung des Samsung-Entwicklermodus

Die auf Samsungs Smart-TVs zum Freischalten des Entwicklermodus notwendigen Fernbedienungsmanöver sind mitunter etwas verwirrend. Auf YouTube findet sich unter [7] ein Video des Autors, das die Vorgehensweise illustriert.

Erzeugen einer neuen Projektvorlage

Sobald alle auf der Workstation des Entwicklers befindlichen Komponenten auf den aktuellen Stand gebracht sind, ist es abermals an der Zeit, den Projektgenerator-Assistenten von Visual Studio anzuwerfen. Dass im Rahmen des Starts der IDE erneut eine UAC-Aufforderung samt Verweis auf die PowerShell erscheint, ist ärgerlich, war allerdings beim Schreiben dieses Artikels nicht zu eliminieren. Das Verneinen dieser Aufforderung führt übrigens dazu, dass das Tizen-SDK später nicht ordnungsgemäß funktioniert.

Die MAUI-Entwicklung für den Smart-TV setzt eine weitere Besonderheit im Bereich der Projekterstellung voraus: Die korrekte Vorlage ist nämlich die in **Bild 2** gezeigte Variante *.NET MAUI-App*. Samsung hat sich vom Sonderweg der eigenen Tizen-Projektvorlagen mittlerweile verabschiedet.

Unschön ist, dass bei Abfassung dieses Artikels Tizen selbst in der neuesten Version 8, die noch nicht auf realer Hardware verfügbar ist, das .NET Framework ausschließlich in Version 6.0 unterstützt – die in der iOS- und Android-Welt weit verbreiteten .NET-Versionen 7 und 8 dürften demgemäß frühestens im Jahr 2025 verfügbar werden. Insbesondere auf sonst produktiv genutzten Workstations scheint jedoch häufig die Version 6.0 gar nicht mehr in der Zielauswahl-Combobox auf.

Zur Umgehung dieses Problems öffnen Sie im ersten Schritt den Visual Studio Installer und laden danach die Paket-Einstellungen der für die Arbeit mit dem Smart-TV vorgesehenen Instanz von Visual Studio. Prüfen Sie im Installer dann nach, ob die Komponente *.NET 6.0-Runtime (Langfristige Unterstützung)* als auf der Workstation vorhanden markiert ist.

Aufgrund eines bekannten Problems in Visual Studio reicht dies normalerweise nicht aus, um die Version 2022 zum Anbieten von .NET 6.0 als Ziel zu animieren. Im nächsten Schritt folgt deshalb der Besuch der Microsoft-Download-Webseite für .NET 6.0 unter [3], wo Sie die aktuellste Version von .NET 6.0 herunterladen. Beim Schreiben dieses Artikels lautet der korrekte Link *.NET 6.0 SDK (v6.0.421) – Windows x64 Installer*; er führt zum Herunterladen der Datei *dotnet-sdk-6.0.421-win-x64.exe*. Diese wird danach wie gewohnt ausgeführt.

Nach dem obligatorischen System-Neustart kehren wir in Visual Studio 2022 zurück und erzeugen auf Basis der MAUI-Vorlage ein neues Projekt. Im Auswahl-Assistenten entscheiden wir uns vorerst für die Plattform-Version 7.0, um die Kompilation durchlaufen zu lassen – in den folgenden Schritten wird der Autor den String *NMGMauiTV* als Solution-Namen vergeben.

Nach dem erfolgreichen Durchlaufen des Projektgenerators öffnen wir die Eigenschaften der Projektmappe und scrollen – wie in **Bild 3** gezeigt – in die Rubrik für die Tizen-Ziele.

Im ersten Schritt muss dabei die Checkbox *Aktivieren Sie das Targeting der Tizen-Plattform* aktiviert werden. Im Bereich der .NET-Ziel-Runtime wird die Version 6.0 ausgewählt; als Ziel-Tizen-Framework und minimales Framework ist jeweils die Version 6.5 auszuwählen. Zu beachten ist, dass es sich dabei um eine Inkonsistenz zwischen dem Visual-Studio-Tooling und den von Samsung tatsächlich implementierten Komponenten handelt – MAUI-basierte Applikationen dürfen ausschließlich auf Smart-TVs zur Ausführung gelangen, die auf Tizen 7.0 basieren.

Nach dem Speichern der Eigenschaften blendet Visual Studio im Projektmappen-Explorer normalerweise eine Aufforderung zur Installation zusätzlicher Komponenten ein. Diese sollten Sie anklicken und eventuell fehlende Komponenten bereitstellen lassen.

Nach dem Schließen des Einstellungs- und Installationsfensters folgt ein Klick auf die Ziel-Auswahl, wo Sie in der Option *Tizen-Framework* darauf achten müssen, wie in **Bild 3** gezeigt *Tizen 6.5* auszuwählen.

Nach dem Durchführen dieser Manipulationen empfiehlt sich ein Neustart von Visual Studio – im Idealfall lassen sich die Projektmappe beziehungsweise ihre Eigenschaften danach bereits öffnen. In der Praxis kommt es allerdings stattdessen oft zum in **Bild 4** gezeigten Fehlerbild. Kritisch daran ist, dass in diesem Fall keine Kompilation des resultierenden Projekts möglich ist.

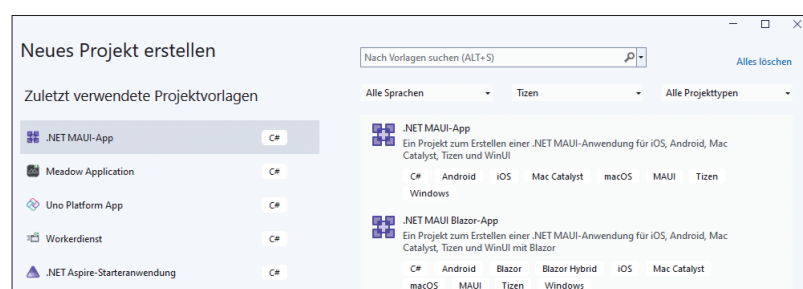
Umgehungsversuch, der erste – Erzeugen der Applikation über die Windows-Kommandozeile

Da sich das Tooling in Visual Studio an dieser Stelle als renitent erwiesen hat, bietet sich der Umstieg in die .NET-Entwicklerkommandozeile an. Unsere erste Aktion ist hier die Erzeugung eines neuen Arbeitsverzeichnisses nach folgendem Schema:

```
C:\>mkdir tizennetspace
```

```
C:\>cd tizennetspace
```

Im nächsten Schritt ist eine Steuerungsdatei erforderlich, welche die bei der Arbeit in diesem Ordner zu verwendende Version des .NET Frameworks beschreibt. Der einfachste Weg zu ihrer Generierung ist die Nutzung von Notepad: ►



Das Tizen-Callout informiert über die Eignung dieser Projektvorlage (**Bild 2**)

```
C:\tizennetspace>notepad global.json
```

Inhaltlich finden sich in diesem File keine Überraschungen. Achten Sie lediglich darauf, die Versionsinformationen an die von Ihnen installierte Version des SDK anzupassen; in unserem Fall bedeutet das:

```
{
  "sdk": {
    "version": "6.0.421"
  }
}
```

Nach getaner Arbeit ist im ersten Schritt das Aktualisieren der im SDK lokal vorgehaltenen Workload-Liste erforderlich. Sinn dieser Operation ist die Möglichkeit zum Freischalten des eigentlichen Install-Befehls, der die notwendigen Komponenten auf den Rechner schafft:

```
C:\tizennetspace>dotnet workload search
C:\tizennetspace>dotnet workload install maui-tizen
Workload(s) maui-tizen wurde(n) erfolgreich installiert.
```

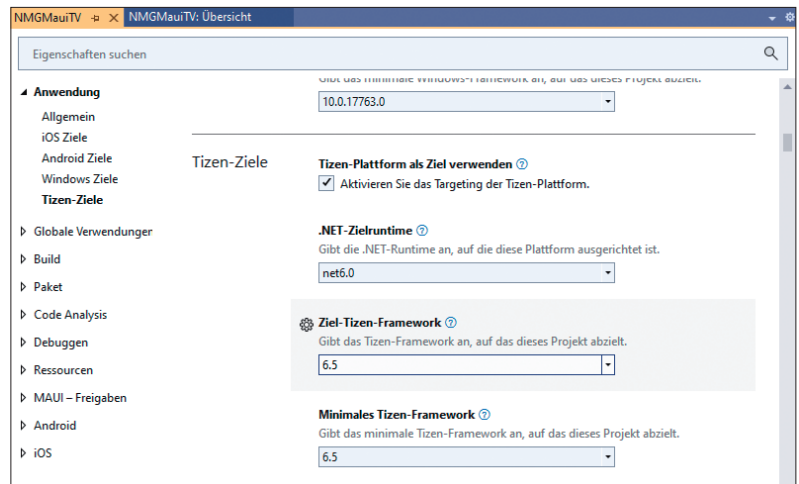
Nach dem Herunterladen der Payload haben Sie die Möglichkeit, über den .NET-Projektgenerator nach folgendem Schema ein neues Projektskelett ins Leben zu rufen:

```
C:\tizennetspace>dotnet new maui -n HelloMaui
```

Die Vorlage ".NET MAUI-App" wurde erfolgreich erstellt.

Im nächsten Schritt bietet es sich an, die Struktur der Projektdatei zu analysieren. Von Haus aus deaktiviert der Generator – wie im nachfolgenden Code-Snippet gezeigt – die relevanten Deklarationen, die das Projekt als für Tizen kompilierbar beziehungsweise nutzbar markieren:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFramework>net6.0-tizen</TargetFramework>
    <!-- Uncomment to also build the tizen app. You will
         need to install tizen by following this:
         https://github.com/Samsung/Tizen.NET -->
```



Die komplette Aktivierung von Tizen setzt im Visual-Studio-GUI einige Kunstgriffe voraus (Bild 3)

```
<!-- <TargetFrameworks>$(TargetFrameworks);
      net6.0-tizen</TargetFrameworks> -->
<OutputType>Exe</OutputType>
```

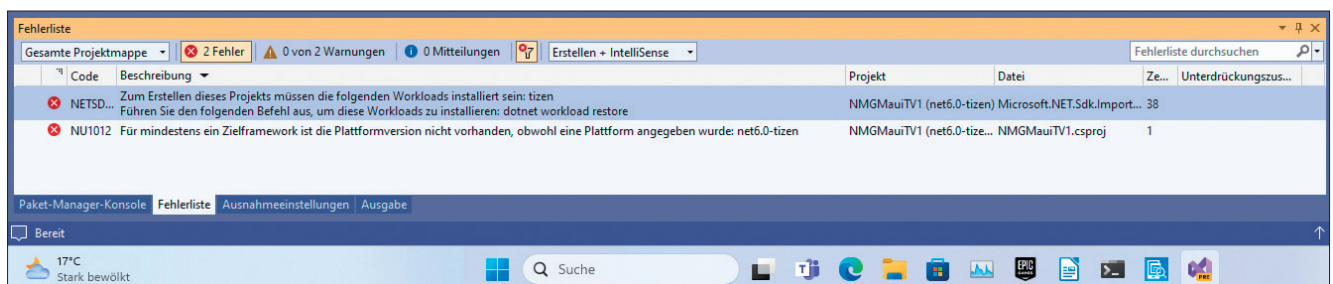
Zweckmäßig ist es nun, die vom Kommandozeilenwerkzeug generierte Solution in Visual Studio 2022 zu laden und zu probieren, ob die Kompilation erfolgreich verläuft.

In den meisten Fällen kommt es stattdessen abermals zum in Bild 4 gezeigten Fehler. Als Behebungsmaßnahmen bieten sich dann zwei Methoden an. Erstens könnte man in der PowerShell die beiden folgenden Kommandos ausprobieren, um abermals zusätzliche Payloads herunterzuladen:

```
PS C:\tizennetspace\HelloMaui> dotnet workload restore
Workloads werden installiert: android ios maccatalyst
PS C:\tizennetspace\HelloMaui> dotnet workload install
maui-tizen
Workloads 'maui-tizen' sind bereits installiert.
```

Lösungsweg Nummer zwei ist das von Samsung unter [4] zum Download bereitgestellte Skript, das sich um die Armierung der diversen notwendigen Komponenten kümmern soll. Laden Sie es herunter und führen Sie es danach in einer mit Administratorrechten ausgestatteten PowerShell aus:

```
PS C:\tizennetspace> .\workload-install.ps1
```



Dieser Bildschirm dient als eine Art Lackmustr (Bild 4)

Während der Programmausführung kommt es dabei mitunter zu Warnmeldungen, wie in **Bild 5** gezeigt. Die Ursache dafür ist, dass das Samsung-Skript einfach nur alle zur Verfügung stehenden unbekannten SDK-Versionen ausprobiert – findet es für die jeweilige .NET-Variante keine Binärdateien auf dem Server, so wird eine Statusmeldung ausgegeben. In den Tests des Autors führte allerdings auch dies nicht zu einer vollständig lauffähigen Projektstruktur.

Vielmehr ließ sich in den Tests des Autors – siehe die Ausgabe des Strings *Done installing Tizen workload 8.0.154* in **Bild 5** – nur in einer bestimmten Situation eine ausführbare Datei generieren. Anhand der unter [5] bereitstehenden Informationen lässt sich nämlich feststellen, dass das heruntergeladene Tizen-Unterstützungspaket mit der Framework-Version SDK 8.0.200 funktioniert.

Wer diese nach den weiter oben besprochenen Anweisungen installiert, kann über *global.json* nach folgendem Schema eine auf diese SDK-Version optimierte Version des MAUI-Projektskeletts rehydrieren:

```
{
  "sdk": {
    "version": "8.0.200"
  }
}
```

Im Test des Autors ließ sich diese problemlos kompilieren – die Auslieferung auf den Smart-TV scheiterte indes, obwohl dieses in den Einstellungen des Tizen-Plug-ins als anzielbares Gerät aufgelistet war.

Umgehungsversuch, der zweite – zwangsweise Kompilation

Die Probleme im Bereich der Visual-Studio-Integration hat der Autor bei Samsung bereits gemeldet. Wer zum aktuellen Zeitpunkt MAUI-Payloads ausführen möchte, sollte stattdes-

sen auf Ubuntu 20.04 setzen. Zu beachten ist, dass die IDE-Unterstützung dort noch rudimentärer ausfällt. Andererseits ist es dort möglich, eine ausführbare *.tpk*-Datei zu erhalten.

Die Einrichtung des .NET-SDK unter Ubuntu wollen wir hier als bekannt annehmen. Unsere erste Aktion ist deshalb die Erzeugung einer Arbeitsumgebung, in der wir danach den Tizen-Workload nach folgendem Schema installieren:

```
tamhan@TAMHAN18:~$ mkdir tizenspace
tamhan@TAMHAN18:~$ cd tizenspace/
tamhan@TAMHAN18:~/tizenspace$ curl -sSL
https://raw.githubusercontent.com/Samsung/Tizen.NET/
main/workload/scripts/workload-install.sh | sudo bash
[sudo] password for tamhan:
...
```

Nach der Abarbeitung ist es erforderlich, einige zusätzliche Workloads zu installieren:

```
tamhan@TAMHAN18:~/tizenspace$ dotnet workload search
...
tamhan@TAMHAN18:~/tizenspace$ sudo dotnet workload
install maui-tizen tizen
[sudo] password for tamhan:
```

Der erste Befehl aktualisiert dabei die auf der Workstation befindliche Workload-Liste, während sich das zweite Kommando um das eigentliche Ausliefern der Komponenten kümmert.

Die Generierung des Projektskeletts erfolgt dann wie erwartet unter Nutzung von *dotnet new*. Die unter Ubuntu auf dem CLI herunterladbare Version des MAUI-Projektskeletts ist dabei etwas „kompatibler“ als ihre für Windows vorgesehene Variante:

```
tamhan@TAMHAN18:~/tizenspace$ dotnet new tizen -n
HelloTizenNet6
```

```
PS C:\tizennetspace> .\workload-install.ps1

Sicherheitswarnung
Führen Sie ausschließlich vertrauenswürdige Skripts aus. Skripts aus dem Internet können zwar nützlich sein, stellen
jedoch auch eine potenzielle Gefahr für Ihren Computer dar. Wenn Sie diesem Skript vertrauen, lassen Sie mit dem Cmdlet
"Unblock-File" die Ausführung des Skripts ohne die Warnmeldung zu. Möchten Sie "C:\tizennetspace\workload-install.ps1"
ausführen?
[N] Nicht ausführen [M] Einmal ausführen [H] Anhalten [?] Hilfe (Standard ist "N"): M

Check Tizen Workload for sdk 6.0.425
Id: Samsung.NET.Sdk.Tizen.Manifest-6.0.400
An exception was caught: Der Remoteserver hat einen Fehler zurückgegeben: (404) Nicht gefunden.
Id: Samsung.NET.Sdk.Tizen.Manifest-6.0.400
An exception was caught: Der Remoteserver hat einen Fehler zurückgegeben: (404) Nicht gefunden.
Id: Samsung.NET.Sdk.Tizen.Manifest-6.0.400
An exception was caught: Der Remoteserver hat einen Fehler zurückgegeben: (404) Nicht gefunden.
Return cached latest version.
Installing Samsung.NET.Sdk.Tizen.Manifest-6.0.400/8.0.154 to C:\Program Files\dotnet\sdk-manifests\6.0.400...
Installing Samsung.Tizen.Sdk/8.0.154...
Installing Samsung.Tizen.Ref/8.0.154...
Installing Samsung.Tizen.Templates/8.0.154...
Installing Samsung.NETCore.App.Runtime.tizen/8.0.154...
Done installing Tizen workload 8.0.154

Done
PS C:\tizennetspace>
```

Das Samsung-Installationsskript zeigt sich kommunikativ (**Bild 5**)

Im nächsten Schritt ist ein Rauchtest erforderlich. Neben einer erstmaligen Kompilation befehlen wir hier auch das Restaurieren der diversen für die Ausführung der Solution notwendigen Pakete:

```
tamhan@TAMHAN18:~/tizenspace/HelloMaui$ dotnet build
-t:Run -f net6.0-tizen
MSBuild version 17.7.4+3ebbd7c49 for .NET
...
tamhan@TAMHAN18:~/tizenspace/HelloMaui$ sudo dotnet
workload restore
...
Successfully installed workload(s) android wasm-tools
maui-tizen tizen.
```

Im nächsten Schritt ist eine Anpassung der Projektsteuerungsdatei erforderlich, denn die Toolchain für Tizen funktioniert beim Schreiben dieses Artikels ausschließlich dann, wenn Tizen als einzige Zielplattform deklariert ist. Die Anpassung lässt sich nach folgendem Schema bewerkstelligen:

```
<Project Sdk="Microsoft.NET.Sdk">
  <PropertyGroup>
    <TargetFrameworks>$(TargetFrameworks);net6.0-tizen</
TargetFrameworks>
    <OutputType>Exe</OutputType>
    <RootNamespace>HelloMaui</RootNamespace>
    <UseMaui>true</UseMaui>
    <SingleProject>true</SingleProject>
    <ImplicitUsings>enable</ImplicitUsings>
```

Zu guter Letzt ist dann eine abermalige Kompilation erforderlich – scheitert diese mit der Fehlermeldung „error: /home/tamhan/tizen-studio/tools/sdb not found. Please set SdbToolPath property“, so ist der erste Teil der Konfiguration erfolgreich verlaufen.

Einrichtung von Tizen Studio zur Bereitstellung der SDK-Komponenten

Die Erzeugung von Tizen-Kompilaten setzt verschiedene Elemente des nativen SDK voraus. Das Herunterladen des MAUI-Tizen-Workloads installiert diese (wohl aus lizenzrechtlichen Gründen) nicht. Zur Umgehung des Problems besuchen wir die Tizen-Studio-Downloadseite unter [6], wo wir die Linux-Variante des Studio-Installers herunterladen und diesen mit folgenden Befehlszeilen ausführen:

```
tamhan@TAMHAN18:~$ cd Downloads/
tamhan@TAMHAN18:~/Downloads$ chmod +x
web-ide_Tizen_Studio_5.6_ubuntu-64.bin
tamhan@TAMHAN18:~/Downloads$
./web-ide_Tizen_Studio_5.6_ubuntu-64.bin
```

Im Rahmen der Ausführung fragt der Installations-Assistent, wo die zu Tizen gehörenden Pakete unterkommen sollen. Die in **Bild 6** gezeigten Standardeinstellungen für die Ablageorte sind dabei korrekt.

Während der Installation erscheinen mehrere UAC-Anfragen, in denen Ubuntu zur Eingabe des Superuser-Passworts auffordert – diesem müssen Sie nachkommen, um dem Installer das Verankern von Tizen Studio im Rechtesystem des Betriebssystems zu ermöglichen.

Nach der erfolgreichen Installation fragt das Werkzeug mitunter, ob es den Paketmanager starten darf. Dies sollten Sie bejahen und die beiden folgenden Komponenten installieren:

- *Main SDK | 8.0 Tizen*
- *Extension SDK | Extras | TV Extensions 8.0*

Nach dem kompletten Deployen der Pakete bringt ein Neustart der Workstation den Path auf den aktuellen Stand.

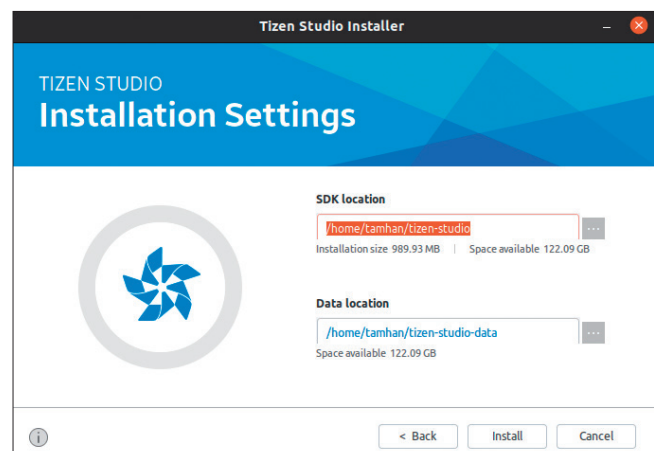
Im nächsten Schritt folgt eine abermalige Rekompilation nach folgendem Schema:

```
tamhan@TAMHAN18:~/tizenspace/HelloMaui$ dotnet build
-t:Run -f net6.0-tizen
MSBuild version 17.7.4+3ebbd7c49 for .NET
Determining projects to restore...
...Samsung.Tizen.Sdk.Packaging.targets(161,5): error :
The api-version specified in tizen-manifest file is 7.
[/home/tamhan/tizenspace/HelloMaui/HelloMaui.
csproj::TargetFramework=net6.0-tizen]
/usr/share/dotnet/packs/Samsung.Tizen.Sdk/8.0.133/
targets/Samsung.Tizen.Sdk.Packaging.targets(161,5):
error : Current target framework net6.0-tizen is not
supported in this api-version.
```

Die Ausgabe informiert dabei darüber, dass die vorliegende Projekt-Konfiguration erfolgreich kompiliert wurde und es erst im Rahmen der Verpackung und der Manifestverifikation zu Problemen kommt.

Zur Behebung dieses Missstands reicht es aus, die Datei *tizen-manifest.xml* zu öffnen und sie nach dem folgenden Schema zu adaptieren – die Datei findet sich im Unterordner *Platforms/Tizen*:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest package="maui-application-id-placeholder"
```



Die Standardeinstellungen reichen völlig aus (**Bild 6**)

```

tamhan@TAMHAN18:~/tizenspace/HelloMaui$ dotnet build -t:Run -f net6.0-tizen
MSBuild version 17.7.4+3ebbd7c49 for .NET
Determining projects to restore...
All projects are up-to-date for restore.
HelloMaui -> /home/tamhan/tizenspace/HelloMaui/bin/Debug/net6.0-tizen/tizen-x86/HelloMaui.dll
/usr/share/dotnet/packs/Samsung.Tizen.Sdk/8.0.133/targets/Samsung.Tizen.Sdk.Packaging.targets(276,9): message : MSB4120: Item 'TizenTpkFile
s' definition within target references itself via (qualified or unqualified) metadatum 'Filename'. This can lead to unintended expansion an
d cross-applying of pre-existing items. More info: https://aka.ms/msbuild/metadata-self-ref [/home/tamhan/tizenspace/HelloMaui/HelloMaui.cs
proj::TargetFramework=net6.0-tizen]
/usr/share/dotnet/packs/Samsung.Tizen.Sdk/8.0.133/targets/Samsung.Tizen.Sdk.Packaging.targets(276,9): message : MSB4120: Item 'TizenTpkFile
s' definition within target references itself via (qualified or unqualified) metadatum 'Extension'. This can lead to unintended expansion a
nd cross-applying of pre-existing items. More info: https://aka.ms/msbuild/metadata-self-ref [/home/tamhan/tizenspace/HelloMaui/HelloMaui.c
sproj::TargetFramework=net6.0-tizen]
HelloMaui is signed with Default Certificates!
HelloMaui -> /home/tamhan/tizenspace/HelloMaui/bin/Debug/net6.0-tizen/tizen-x86/com.companyname.hellomai-1.0.0.tpk
EXEC : error : target not found [/home/tamhan/tizenspace/HelloMaui/HelloMaui.csproj::TargetFramework=net6.0-tizen]
EXEC : error : target not found [/home/tamhan/tizenspace/HelloMaui/HelloMaui.csproj::TargetFramework=net6.0-tizen]
EXEC : error : failed to get package temporary path [/home/tamhan/tizenspace/HelloMaui/HelloMaui.csproj::TargetFramework=net6.0-tizen]
/usr/share/dotnet/packs/Samsung.Tizen.Sdk/8.0.133/targets/Samsung.Tizen.Sdk.Applications.targets(20,5): error MSB3073: The command ""/home/ta
mhan/tizen-studio/tools/sdb" install bin/Debug/net6.0-tizen/tizen-x86/com.companyname.hellomai-1.0.0.tpk" exited with code 1. [/home/ta
mhan/tizenspace/HelloMaui/HelloMaui.csproj::TargetFramework=net6.0-tizen]

Build FAILED.

EXEC : error : target not found [/home/tamhan/tizenspace/HelloMaui/HelloMaui.csproj::TargetFramework=net6.0-tizen]
EXEC : error : target not found [/home/tamhan/tizenspace/HelloMaui/HelloMaui.csproj::TargetFramework=net6.0-tizen]
EXEC : error : failed to get package temporary path [/home/tamhan/tizenspace/HelloMaui/HelloMaui.csproj::TargetFramework=net6.0-tizen]
/usr/share/dotnet/packs/Samsung.Tizen.Sdk/8.0.133/targets/Samsung.Tizen.Sdk.Applications.targets(20,5): error MSB3073: The command ""/home/ta
mhan/tizen-studio/tools/sdb" install bin/Debug/net6.0-tizen/tizen-x86/com.companyname.hellomai-1.0.0.tpk" exited with code 1. [/home/ta
mhan/tizenspace/HelloMaui/HelloMaui.csproj::TargetFramework=net6.0-tizen]
0 Warning(s)
4 Error(s)

Time Elapsed 00:00:02.13
tamhan@TAMHAN18:~/tizenspace/HelloMaui$

```

Probleme beim Deployment der Software führen zu einem vergleichsweise umfangreichen Fehlerbild (Bild 7)

```

version="0.0.0" api-version="8" xmlns=
"http://tizen.org/ns/packages">

```

Danach ist auch schon eine Rekompilation möglich, die durch Generieren der *.tpk*-Datei ein schlüsselfertiges Archiv liefert:

```

tamhan@TAMHAN18:~/tizenspace/HelloMaui$ dotnet build
-t:Run -f net6.0-tizen
MSBuild version 17.7.4+3ebbd7c49 for .NET
Determining projects to restore...
... [/home/tamhan/tizenspace/HelloMaui/HelloMaui.
csproj::TargetFramework=net6.0-tizen]
HelloMaui is signed with Default Certificates!
HelloMaui -> /home/tamhan/tizenspace/HelloMaui/bin/
Debug/net6.0-tizen/tizen-x86/com.companyname.
hellomai-1.0.0.tpk

```

Wer seine Ubuntu-Workstation noch nicht mit dem Smart-TV verbunden hat, sieht dann die sehr umfangreichen in Bild 7 gezeigten Fehler.

An dieser Stelle bietet sich der Start des Device Managers von Tizen Studio an, um eine Verbindung zum im Entwicklermodus befindlichen Fernsehgerät herzustellen. Das Deployment der Applikation erfolgt dann unter Nutzung des sdb-Kommandozeilenwerkzeugs – es verhält sich sehr analog zu dem, was Entwickler von der Android Debug Bridge kennen:

```

$ sdb install helloworld/bin/Debug/tizen80/org.tizen.
example.helloworld-1.0.0.tpk
$ tizen run -p org.tizen.example.helloworld

```

Fazit

Obwohl sich die Tizen-Toolchain zum Zeitpunkt der Abfassung dieses Artikels noch in einem sehr frühen Zustand be-

findet, ist offensichtlich, dass Samsung Unterstützung von MAUI im Smart-TV-Bereich über kurz oder lang nachrüsten wird. Die hier gezeigten Experimente ermöglichen erste Versuche im Bereich der Mobilisierung hauseigener Payloads. Was das Verhältnis von Aufwand zu Nutzen angeht, sei allerdings angemerkt, dass die Verwendung von Xamarin derzeit die größere Reichweite ermöglicht. ■

- [1] Tam Hanna, Auferstehung des siebten Toten, *dotnetpro* 1/2024, Seite 114 ff., www.dotnetpro.de/A2401Tizen
- [2] Samsung-TV-Model-Group-Kompatibilitätsliste, www.dotnetpro.de/SL2412SmartTVMaui1
- [3] Microsoft-Download-Webseite für .NET 6.0, www.dotnetpro.de/SL2412SmartTVMaui2
- [4] Installer für das Tizen Workload Manifest, www.dotnetpro.de/SL2412SmartTVMaui3
- [5] Tizen Workload 8.0.157 – .Net 8.0, www.dotnetpro.de/SL2412SmartTVMaui4
- [6] Tizen-Studio-Download, www.dotnetpro.de/SL2412SmartTVMaui5
- [7] Samsung-TV in den Entwicklermodus schalten, www.dotnetpro.de/SL2412SmartTVMaui6



Tam Hanna

entwickelt Programme für verschiedene Plattformen, betreibt Online-Newsdienste zum Thema und steht für Fragen, Trainings und Vorträge gern zur Verfügung. Sie erreichen ihn unter der E-Mail-Adresse tamhan@tamoggemon.com.

dnpCode

A2412SmartTVMaui

ARBEITSMARKT

TRENDS UND JOBS FÜR ENTWICKLER

Robert Half

Einstellungstrends

Für seine Studie „Prognosen zu Einstiegsgehältern und Einstellungstrends 2025“ fasst der Personaldienstleister Robert Half seine Erkenntnisse so zusammen: „Die Nachfrage nach Talenten und Fachkräften im IT-Bereich bleibt historisch hoch. Angesichts eines volatilen Marktumfeldes und eines entsprechenden Innovationsdrucks müssen Unternehmen die digitale Transformation weiter vorantreiben. Dies hat zur Folge, dass die Gehälter für Fachkräfte aus dem IT-Bereich weiter stark steigen und Unternehmen Schwierigkeiten haben, am Markt rar gesäte Spezialisten für sich zu gewinnen.“

Alle für die Studie befragten Unternehmen erwarten ein positives Jahr 2025. 41 Prozent wollen mehr Mitarbeiter fest anstellen, 27 Prozent mehr Zeitarbeitskräfte ins Unternehmen holen und 34 Prozent mehr projektbasierte Teams engagieren (Bild 1).

Die Bereiche, für die Unternehmen besonders nach neuem Personal suchen, sind gemäß der Studie IT-Service und -Support, Künstliche Intelligenz, Automatisierung, Maschinelles Lernen, IT-Sicherheit und -Vernetzung.

Als besonders stark nachgefragte Fachkenntnisse listet Robert Half Cloud, Künstliche Intelligenz, Automation, Maschinelles Lernen, Software- und Anwendungsentwicklung sowie ERP- und CMS-Systeme.

Zudem gibt Robert Half einen Einblick in die durchschnittlichen Brutto-Jahresgehälter für bestimmte Berufe, die man unter tinyurl.com/rh-it-gehalt gezielt für eine frei wählbare Stadt abfragen kann. In Bild 2 finden Sie die Zahlen für Softwareentwickler (C#, .NET, Python) für die Städte München, Stuttgart, Köln, Hannover und Leipzig. Neben Beruf und Stadt sind keine weiteren Eingrenzungen, etwa Erfahrung oder Unternehmensgröße, erlaubt. Dafür sind die Ergebnisse in drei Perzentil-

Stufen (25, 50, 75) angegeben. Das 25. Perzentil besagt, dass ein Viertel der Datenwerte kleiner oder gleich diesem Wert ist.

Bitkom

Milliarden für Start-ups

Mit der WIN-Initiative wollen Staat und Wirtschaft gemeinsam bis zu 10 Milliarden Euro Kapital für die Start-up-Finanzierung bereitstellen. Zugleich sollen auch weitere Rahmenbedingungen für Start-ups und das Venture-Capital-Ökosystem verbessert werden, etwa mit Blick auf Börsengänge. Bitkom-Präsident Dr. Ralf Wintergerst: „Die WIN-Initiative ist ein wichtiges Signal an Start-ups und ihre Kapitalgeber in Deutschland. Sie wird die Finanzierung vieler deutscher Start-ups erleichtern und verbessern. So wird sie dafür sorgen, dass junge Tech-Unternehmen in Deutschland schneller wachsen; sie stärkt direkt den Technologie- und Innovationsstandort und mit-

telbar die gesamte deutsche Wirtschaft. Um international wettbewerbsfähig zu bleiben, brauchen wir mehr digitale Innovationen. Dabei spielen Start-ups eine herausragende Rolle.“

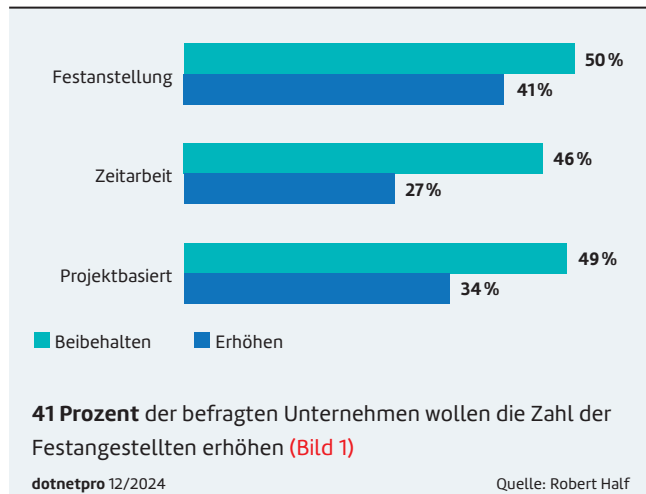
Aktuell sagt laut Bitkom nicht einmal ein Fünftel der Start-ups, dass es hierzulande ausreichend Venture Capital gibt, 8 von 10 haben zudem aufgrund der konjunkturellen Eintrübung zuletzt eine zunehmende Zurückhaltung bei Investoren festgestellt. Die WIN-Initiative könne das dringend benötigte Aufbruchsignal sein, so Wintergerst.

Smart City Index

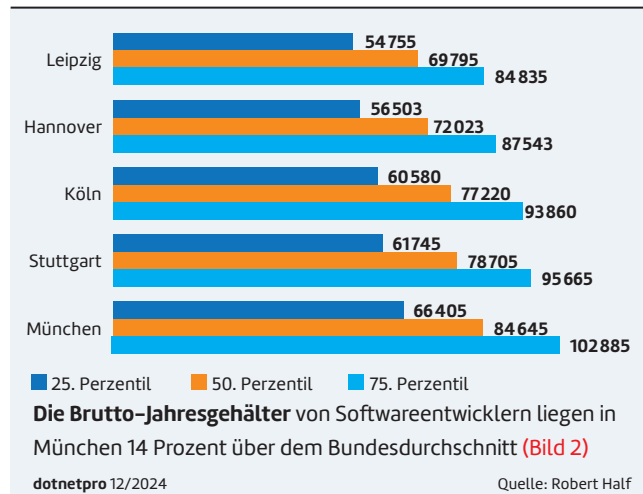
München bleibt vor Hamburg

Alljährlich ermittelt der Branchenverband Bitkom das Ranking der am umfangreichsten digitalisierten deutschen Großstädte. Im zuletzt vorgestellten, sechsten Smart City Index konnte Vorjahressieger München seinen Vorsprung vor dem

● Art der Anstellung 2025



● Gehälter der Softwareentwickler (C#, .NET, Python)



Zahl des Monats

Nachholbedarf: In der Gruppe der Tech-Profis glauben **29 Prozent**, dass Deutschlands Arbeitnehmer nicht einmal die Grundlagen der IT-Sicherheit beherrschen. Rund **17 Prozent** der Befragten haben schon einmal auf einen fremden Link in einer E-Mail geklickt. Auffällig: Gerade diejenigen, die eine große oder sehr große Kompetenz für sich reklamieren, sind anfällig für potenziell riskante Verhaltensweisen.

Quelle: Statista IT-Wissenslücken (33116) und IT-Kompetenzen (33115)

erneut zweitplatzierten Hamburg weiter ausbauen und Köln den dritten Platz verteidigen. Die drei am weitesten digitalisierten Großstädte Deutschlands bleiben auch 2024 an der Spitze, ansonsten aber gibt es viel Bewegung im Digital-Ranking: So schaffen es Bochum (Platz 4, Vorjahr: 11), Freiburg im Breisgau (Platz 6, Vorjahr: 14) und Lübeck (Platz 8, Vorjahr: 19) in diesem Jahr unter die zehn besten Städte, dagegen fallen Aachen (Platz 11, Vorjahr: 5), Osnabrück (Platz 12, Vorjahr: 7) und Karlsruhe (Platz 13, Vorjahr: 10) aus den Top 10. **Bild 3** zeigt die Top 10 der deutschen Smart Cities.

Untersucht wurden alle deutschen Städte ab 100 000 Einwohnern. Den Indexwerten liegen Angaben der untersuchten Städte sowie amtliche Statistiken und öffentlich zugängliche Datenquellen zugrunde. Die 37 Indikatoren berechnen sich aus 162 Parametern pro Stadt und insgesamt 13 284 Datenpunkten. Sie sind normiert,

das heißt auf eine Skala von 0 bis 100 übersetzt. Pro Indikator bedeutet der Indexwert von 100 den höchsten erreichten Wert, nicht aber einen theoretisch erreichbaren Wert. Beispiel: Wiesbaden hat laut amtlicher Zulassungsstatistik einen Anteil von 7,8 Prozent E-Fahrzeugen an allen zugelassenen Fahrzeugen. Das ist der höchste Wert unter allen untersuchten Städten. Damit erreicht Wiesbaden im Indikator „Anteil E-Fahrzeuge“ den Maximalwert 100.

Freelancer Krankenversicherung

Gemäß einer aktuellen Umfrage von Freelancermap sind zwei Drittel der Freiberufler in der gesetzlichen Krankenversicherung (GKV) und ein Drittel privat versichert. Mehr als die Hälfte der PKV-Versicherten (54 Prozent) war davor gesetzlich versichert. Aus heutiger Sicht sagen vier von zehn, sie seien nicht gut genug über die Folgen informiert worden, oder zeigen

sich unsicher über die Qualität der Aufklärung. Gut ein Drittel hat den Schritt in die PKV be-reut oder stellt ihn zumindest infrage. Thomas Maas, CEO von Freelancermap: „Wer ins Freelancing startet, hat seinen Fokus oft auf anderen Themen, wie dem künftigen Einkommen. Diese Perspektive führt Freelancer oft geradewegs zur PKV, denn diese ist – besonders in der Gründungsphase – preislich meist attraktiver.“

Vor fünf Jahren wurde die Mindestbeitragsbemessungsgrundlage der GKV gesenkt, dennoch zahlen heute noch alle Selbstständigen, die weniger als rund 1178 Euro pro Monat verdienen, proportional zum Einkommen immer noch zu hohe Beiträge zur GKV. Für die „Altfälle“, die vor der Reform in die PKV gewechselt sind, gibt es häufig kaum Chancen auf Rückkehr. Viele privat versicherte Freelancer befürchten, im Ruhestand bis zur Hälfte ihrer Alterseinkünfte für die Krankenversicherung aufwenden zu

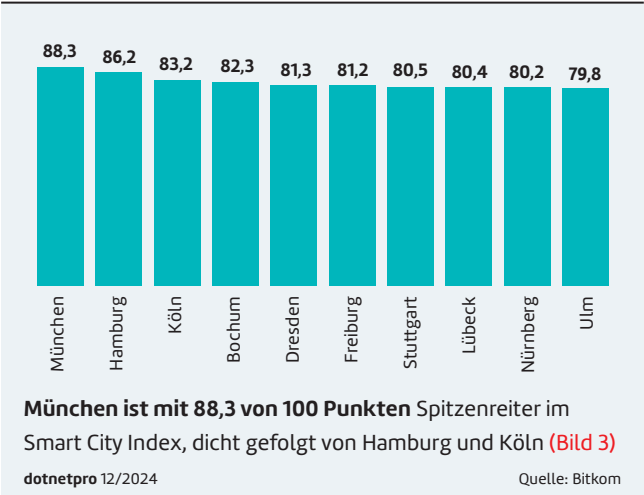
müssen, und dies ist zudem der Hauptgrund für einen Verbleib in der GKV (**Bild 4**).

Thomas Maas: „Aus unserer Sicht wäre es wünschenswert, dass die Initiative und der Unternehmergeist von Freelancern in Deutschland mehr wertgeschätzt und stärker gefördert würden, etwa indem die GKV ihnen in den ersten ein bis drei Jahren darauf zugeschnittene Konditionen anbietet.“

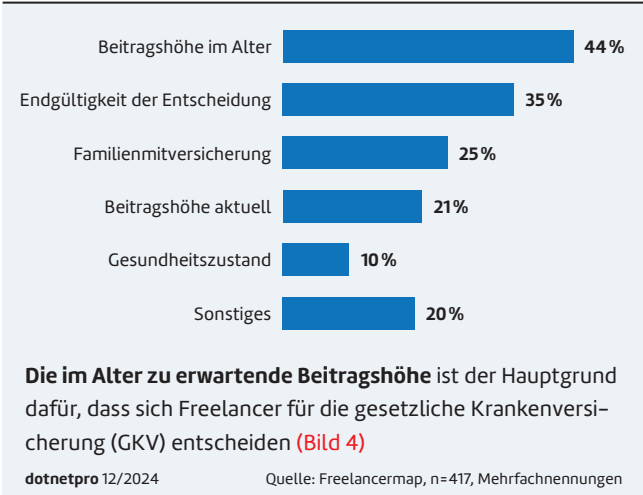
Die PKV punktet mit meist besseren Leistungen, Rück-erstattungen und einkommens-unabhängigen, flexibel anpassbaren Beiträgen: Vor allem für junge und nicht mit Vorerkrankungen belastete Freelancer bleiben dies starke Argumente. Und zwei Drittel der PKV-versicherten Freiberufler haben den Wechsel nie bereut.

„Freelancer, die aktuell vor der Wahl ihrer Krankenversicherung stehen, sollten Wert auf gute Beratung legen, um die Folgen eines Wechsels umfassend einschätzen zu können“, rät deshalb Thomas Maas.

Smart City Index 2024



Freiberufler: Gründe für die GKV





developer media jobs

Der Stellenmarkt für Entwickler

LANGE SUCHEN ODER GLEICH FINDEN. WAS DARF'S SEIN?

**JETZT
ANZEIGE
SCHALTEN!**



jobs.developer-media.de

Reinhold Fritsch, Tel.: +49 731 88005-8285 | Sabine Vockrodt, Tel.: +49 731 88005-8222
oder per E-Mail an: jobs@developer-media.de

JOBS für Entwickler

Auf jobs.developer-media.de finden
Software-Entwickler und -Architekten
ständig aktuelle Job-Angebote

Ihr Job-Angebot fehlt?

Sie suchen Entwickler in Festanstellung oder projektbezogen?
Dann sollten auch Sie unser Online-Jobportal für Ihre Mitarbeitersuche nutzen.
Ihr Stellenmarkt-Team berät Sie gern zu den attraktiven Promotionmöglichkeiten.

Gerne können Sie Ihre Stellenausschreibung zu jederzeit online
selbst einbuchen unter jobs.developer-media.de



Diese und viele weitere aktuelle Job-Angebote finden Sie auf jobs.developer-media.de

Justus-Liebig-Universität Gießen	IT-Administrator/in (m/w/d) für (BOS-) Funksysteme und Gefahrenmeldesysteme	Gießen
Landesbank Hessen-Thüringen	Datenbank Experte (w/m/d)	Offenbach am Main, Erfurt
Hochschule Hamm-Lippstadt	Mitarbeiter*in (m/w/d) technische IT- und Prozessberatung für digitale Transformation	Hamm, Lippstadt
Klinikverbund Südwest gGmbH	Bereichsleitung (m/w/d) für die IT-Infrastruktur	Böblingen
Postnova Analytics GmbH	Junior Softwareentwickler C# (m/w/d)	Landsberg am Lech
Landesbank Hessen-Thüringen	Direktor / Leitung IT-Controlling (m/w/d)	Offenbach am Main
Landesbank Hessen-Thüringen	IT Controller – Schwerpunkt IT-Projektportfoliosteuerung (m/w/d)	Düsseldorf, Offenbach am Main
Bergische Universität Wuppertal	System Engineer IT-Sicherheit, Schwerpunkt Backup und Wiederherstellung (m/w/d)	Wuppertal
Johann Wolfgang Goethe-Universität Frankfurt am Main	IT-Projektmitarbeiter*in Campusmanagement (m/w/d)	Frankfurt am Main
Landesbank Hessen-Thüringen	IT-Risikomanager (m/w/d)	Düsseldorf
Bergische Universität Wuppertal	Wissenschaftliche*r Mitarbeiter*in (Doktorand*in) im Bereich Large Language Models	Wuppertal

Ihre Anzeige ist noch nicht dabei? Rufen Sie uns an:

Reinhold Fritsch, Tel.: +49 731 88005-8285, E-Mail an: jobs@developer-media.de

● Impressum

**Ebner Media Group GmbH & Co. KG****Büro München**

Bayerstraße 16a, 80335 München

Sitz von Redaktion, Anzeigen, Vertrieb

Bayerstraße 16a, 80335 München

Telefon +49 731 88005-8000

info@dotnetpro.de

www.dotnetpro.de

Sitz des Verlags

Ebner Media Group GmbH & Co. KG

Karlstraße 3, 89073 Ulm

Geschäftsführer

Marco Parrillo

Chefredaktion

Fernando Schneider (ViSdP)

Redaktion

Tilman Börner

Freie Mitarbeiter dieser Ausgabe**Redaktion**

Ernst Altmannshofer, Sabine Block,

Bernhard Lauer

Autoren

Klaus Aschenbrenner, Daniel Basler,
 Elena Bochkor, Tim Borowski, Dr. Fabian Deitelhoff,
 Christian Giesswein, Tam Hanna, Christian Jacob,
 Dr. Veikko Krypczyk, Bernhard Lauer, Stefan Mintert,
 Robert Mühsig, Bernhard Pichler, Tobias Richling,
 Dr. Holger Schwichtenberg, Helmut Vonhoegen,
 Tom Wendel, Ralf Westphal, Alexander Zeitler

Art Directorin

Maria-Luise Steinkühler

Gestaltung**EMG Design Unit**

Dagmar Breitenbach, Simone Köhnke,

Petra Reichensperner

Kundenservice / Einzelheftbestellung

EBNER MEDIA GROUP,

Bayerstraße 16a, 80335 München

Telefon +49 731 88005-8205

kundenservice@ebnermedia.de

(Mo.-Do. 9.00 Uhr bis 17.00 Uhr,

Fr. 9.00 Uhr bis 15.00 Uhr)

Erscheinungsweise

dotnetpro erscheint monatlich.

ISSN 1610-1553. Postvertriebskennzeichen B30751

Nachdruck

© Nachdruck nur mit schriftlicher Genehmigung

Developer Events

Charlotte Frieß, Director Live Communication Developer

charlotte.friess@ebnermedia.de

Mediaberatung und Content-Marketing-Lösungen

Thomas Deck

Telefon +49 731 88005-8282

thomas.deck@ebnermedia.de

Beratung Stellenmarkt

Reinhold Fritsch

Telefon +49 731 88005-8285

reinhold.fritsch@ebnermedia.de

Client Success (Disposition)

clientsuccess@ebnermedia.de

Leitung Herstellung / Vertrieb / Kundenservice

Thomas Heydn

Anzeigentarife

Anzeigenpreisliste gültig ab 1.1.2019

digital.dotnetpro.de/mediadaten/

Druck

Silber Druck GmbH & Co. KG

Otto-Hahn-Straße 25, 34253 Lohfelden

Sonderdruck

Von allen Beiträgen dieser Ausgabe können für

Werbezwecke Sonderdrucke hergestellt werden.

Anfragen bitte an fernando.schneider@ebnermedia.de

Bezugspreise

dotnetpro ist das Profi-Magazin für Entwickler
 und erscheint zwölfmal im Jahr.

Das Einzelheft kostet im Inland 19,90 €.

Der Preis für das Premium-Plus-Abonnement

(12 Ausgaben Print + Digital)

beträgt im Inland 182,40 €

(Digital 152,40€ + Print 30,00 €),

Schweiz 304,00 CHF

(Digital 167,70 CHF + Print 136,30 CHF),

sonstiges Ausland 206,40 €

(Digital 152,40 € + Print 54,00 €) pro Jahr.

Abonnement-Bedingungen

Das Abonnement gilt für ein Jahr, anschließend bis
 zur Kündigung zum jeweils geltenden Jahresbezugspreis.

Dein Jahresabo ist ab dem Folgejahr auch monatlich

kündbar. Bei vorzeitiger Kündigung deines bereits

abgerechneten Bezugszeitraums erhältst du den

Betrag der Restlaufzeit zurückerstattet.

Veröffentlichung gemäß Art. 8 Abs. 3 BayPrG:

Alleinige Gesellschafterin der Ebner Media Group

GmbH & Co. KG ist die Ebner Ulm MGV GmbH, Ulm.

● Inserentenverzeichnis

Text Control GmbH

www.textcontrol.com

2

Ebner Media Group

www.ebnermedia.de

23-27, 47,

63, 73, 85, 105, 142-143, 147, 148

Autor:in werden!

JAVA
SCRIPT

C#

ANGULAR



Helfen Sie uns,
Softwareentwickler
weiterzubilden.



Schreiben Sie
Artikel für die

dotnetpro



info@dotnetpro.de

Anbieterverzeichnis

Persönliche Beratung bei Fernando Schneider | +49 731 88005-8831 | fernando.schneider@ebnermedia.de

	COUNT IT Group Softwarepark 49 4232 Hagenberg/AUSTRIA T: +43 (0)7236 – 20077 – 0 F: +43 (0)7236 – 20077 – 6717 m.szoeky@countit.at www.countit.at Markus Szöky	Die COUNT IT Group ist österreichweit einer der führenden Unternehmensdienstleister und mit über 120 MitarbeiterInnen das größte Unternehmen im Softwarepark Hagenberg. Unser agiles Softwarehouse realisiert neben maßgeschneiderten ERP- und DMS-Lösungen vor allem Individualsoftware-Projekte auf Basis modernster .NET-Technologien. Die Bandbreite reicht hier von Web-Anwendungen bis Softwarelösungen für die Industrie 4.0, wobei für uns als Qualitätsdienstleister Testautomatisierung wesentlich ist. Auf uns zählen (inter) nationale Kunden aus Handel, Baubranche und Industrie. Rechnen auch Sie mit uns!
	DATEV eG Paumgartnerstr. 6 – 14 90429 Nürnberg Deutschland Telefon: +49 911 319-0 https://www.datev.de	Die DATEV eG ist der drittgrößte Anbieter für Business-Software in Deutschland und einer der großen europäischen IT-Dienstleister. Das Leistungsspektrum der DATEV reicht von mehr als 200 PC-Programmen über Cloud-Lösungen wie Online-Anwendungen, Datenverarbeitung und -archivierung im Rechenzentrum bis hin zu Outsourcing-Leistungen sowie Sicherheitsdienstleistungen, mit denen die DATEV Unternehmen, Selbstständige, Kommunen, Vereine und Institutionen unterstützt. Beratungsleistungen und Angebote zur Wissensvermittlung in Deutschland und einigen anderen europäischen Ländern runden das Angebot ab.
	Developer Express, Inc 505 N. Brand Blvd. Suite 1450 CA 91203 Glendale USA ClientServices@devexpress.com www.devexpress.com	DevExpress engineers feature-complete UI Controls, enterprise-ready reporting systems, IDE productivity tools and business application frameworks for Visual Studio. Our technologies help you build your best, see complex software with greater clarity, increase your productivity and create stunning touch-enabled applications for Windows, the web and mobile devices without limits or compromise. To learn more and see how you can create business solutions customers expect today and leverage existing skillsets to build next generation applications for tomorrow, visit us online at www.devexpress.com .
	Kutura GmbH Oberaltenallee 20a 22081 Hamburg Telefon: +49 177 216 70 80 Mail: contact@kutura.digital kutura.digital/contact	Unternehmen, in denen sich Menschen entwickeln und wachsen können, gehört die Zukunft. Das gilt besonders in Zeiten, in denen Künstliche Intelligenz, Krisen und Komplexität unsere Aufmerksamkeit erfordern und unsere mentalen Ressourcen beanspruchen. Kutura entwickelt für und mit ihren Kunden eine zukunftsfähige Führungs- und Unternehmenskultur, die drei Zielen dient: Gemeinsam Ergebnisse schaffen, mehr Motivation und weniger Konflikte, zukünftige Stärke sicherstellen. Reimagining Leadership, Inspiring Happiness.
	prodot GmbH Schifferstraße 196 47059 Duisburg T: +49 (0)203 – 396508 – 0 info@prodot.de https://prodot.de	prodot – Software für Marktführer Intelligente Software für internationale Konzerne und mittelständische Unternehmen: prodot stärkt seit über 20 Jahren namhafte Kunden im weltweiten Wettbewerb mit effizienten, stabilen und kostensenkenden Lösungen. Mit Sorgfalt und Enthusiasmus entwickeln wir hochwertige Software, die sich durch intuitives Design auszeichnet. Digitale Prozesse und innovative Technologien wie IoT, AI, Mobile, Cloud sind unser Antrieb – Fortschritt, Qualität und Kontinuität unser Anspruch. Mit rund 80 Mitarbeitern setzen wir die Anforderungen unserer Kunden in hochwertige Softwareprodukte um. ALDI SÜD, Microsoft, Siemens, TK Elevator und weitere vertrauen uns seit vielen Jahren. Gerne zeigen wir Ihnen warum! Sprechen Sie uns an.
	Space Blocks Permissions as a Service Ein Service der wemogy GmbH Graf-Bernadotte-Str. 23 47906 Kempen Deutschland https://www.spaceblocks.cloud hello@spaceblocks.cloud	Die Developer-Plattform für Rollen- und Rechtemanagement Einfach Rollen und Berechtigungen im grafischen Low-Code-Editor anlegen und Zugriffsabfragen mit wenigen Zeilen Quellcode in die eigenen Anwendungen integrieren. Inklusive RBAC, Multi-Tenancy, hierarchischer Berechtigungsvererbung und vielem mehr. Geschrieben in .NET und für .NET-Entwickler konzipiert. Entweder von uns in der Space Blocks Cloud bereitgestellt oder im Selbstbetrieb. Jetzt ausprobieren mit der kostenlosen Entwickler-Version oder eine unverbindliche, kostenfreie Demo buchen!
	Text Control GmbH Überseetor 18 28217 Bremen Deutschland Telefon: +49 421 42706710 https://www.textcontrol.com/	Text Control wurde 1991 gegründet und ist ein preisgekrönter Microsoft Visual Studio Industry Partner und führender Anbieter von Reporting- und Textverarbeitungskomponenten für Microsoft-Entwickler. Die Produkte helfen tausenden von Entwicklern, Text- und Dokumentenverarbeitung in ihre Windows, Web und Mobil-Anwendungen zu integrieren. Text Control ist zu 100% inhabergeführt und damit komplett unabhängig von Investorenentscheidungen. Kunden profitieren von jahrelanger Erfahrung, einer sehr großen Nutzerbasis und den ausgereiften und zuverlässigen Produkten. Kernprodukt ist das Reporting-Framework, welches die Stärken eines Reporting-Tools mit einer, einfach zu handhabenen, Textverarbeitung kombiniert – voll programmierbar und integrierbar in einer .NET Anwendung.
	Carl Zeiss Digital Innovation GmbH Fritz-Foerster-Platz 2 01069 Dresden Telefon: +49 351 49701 – 500 info.digitalinnovation.de@zeiss.com www.zeiss.de/digital-innovation	ZEISS Digital Innovation – Als Teil der ZEISS Gruppe entwickeln wir individuelle und hochwertige digitale Lösungen, die unter anderem auf modernen Web- und Cloud-Technologien basieren. Mit agilen Methoden unterstützen wir unsere Kunden entlang ihrer Digitalisierungsprojekte – und darüber hinaus. Dabei entstehen innovative Plattformen auf einer skalierbaren Gesamtarchitektur. So steigern wir die Innovationskraft unserer Zielgruppe in ihren Märkten. www.zeiss.de/digital-innovation

Die Ausgabe 1/2025 erscheint am 12. Dezember 2024

Foto: EMcGenie



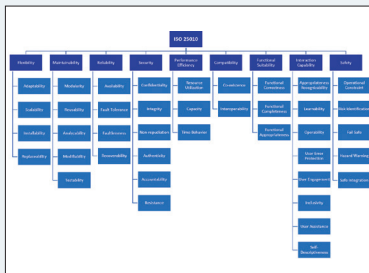
Frontend

Die Welt der Frontend-Entwicklung ist schier grenzenlos. Welch enormes Spektrum an Vielfalt und Möglichkeiten sich bietet, zeigen exemplarisch die Beiträge im Schwerpunkt der **dotnetpro 1/2025**.

Christian Wenz zeigt auf, wie wichtig lokalisierte Anwendungen sind und wie Sie **Web-Apps in ASP.NET Core lokalisieren** können. Elena Bochkor und Veikko Krypczyk nehmen sich das WinUI Community Toolkit vor und beschreiben, wie sich **DataGrids in WinUI 3** realisieren lassen. Tam Hanna schließlich geht der Frage nach, wie sich **Smart-TV-Interfaces mit dem Samsung-Framework NUI** erstellen lassen und welche Rolle .NET MAUI künftig spielen wird.

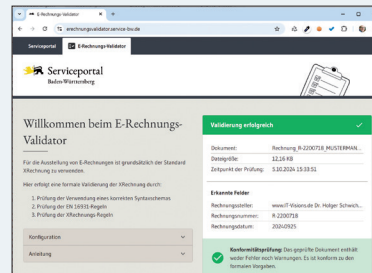
Qualität – was ist das?

Die verschiedenen Sichtweisen auf Softwarequalität anhand der ISO-25000-Familie im Vergleich.



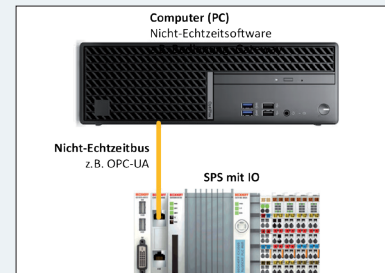
E-Invoicing mit .NET

Dank ZUGFeRD-csharp lassen sich strukturierte elektronische Rechnungen in .NET-Anwendungen erstellen.



Echtzeitsteuerung mit C#

Eine elegante Hybridlösung ermöglicht es, mit C# und OPC UA Echtzeitfunktionalität auf einer SPS zu realisieren.



End of file



A good way to stay flexible is to write less code.

Unsere digitalen Angebote



Wöchentlicher Entwickler-Newsletter
www.dotnetpro.de/newsletter



dotnetpro digital lesen: Android- und iPad-App
www.dotnetpro.de/dnpapps



YouTube: Videos von den Konferenzen
www.youtube.com/@developerweek



Facebook: die News im Stream
<https://de-de.facebook.com/dotnetpro>



X: die News im Stream
https://twitter.com/dotnetpro_mag

KONFERENZ-
PROGRAMM
IM HEFT!

ab Seite 23



.NET DEVELOPER CONFERENCE '24

DIE KONFERENZ FÜR .NET ENTWICKLER
25. - 28.11.2024 | KÖLN

dotnet-developer-conference.de | #DDC24 | Find us on [f](#) [i](#) [X](#) [in](#)

Präsentiert von:



30. JUNI bis 03. JULI 2025
DEVELOPER WEEK '25

MOVING TO MANNHEIM!
#MonnemWirKommen



developer-week.de | #DWX25 | Find us on    

Bild: Shutterstock / Karim Osama