

dotnetpro

dotnetpro.de

Das Magazin für Profi-Entwickler

4-5/25

Single Page Applications

Schnellere Ladezeiten, nahtlose UX, reduzierte Serverlast und effiziente
Entwicklung durch wiederverwendbare Komponenten. Ab Seite 6



30. Juni – 3. Juli 2025 | Mannheim
PROGRAMM + EXPO INSIDE
www.developer-week.de

DEV ACADEMY

Hands-on Workshops und Weiterbildung für Software-Entwickler und -Architekten



PATRICK SCHNELL

**START IN
ANGULAR FÜR
.NET-ENTWICKLER**

Der Kurs vermittelt Grundlagen von Angular, Googles Open-Source-Framework für Frontend-Webentwicklung. Es bietet Tools zur Erstellung robuster, skalierbarer und wartbarer Webanwendungen. Nach Abschluss können Sie selbständig dynamische, interaktive Webanwendungen entwickeln.



JAN POHLMAYER

**KI IN DER PRAXIS:
ANBINDUNG AN JAKARTAEE
MIT LANGCHAIN4J**

Der Workshop gibt einen Überblick über neuronale Netze und LLMs mit Fokus auf die Anbindung von AI-Schnittstellen via LangChain4j zur Inferenz. Die LLMs laufen lokal, das Training von Modellen wird nicht behandelt.



ROLAND KÖNIG

**BOOTCAMP:
CROSS-PLATTFORM
MIT AVALONIA UI**

Avalonia UI ist eine OpenSource-Klassenbibliothek für UI-Applikationen mit .NET. Als Cross-Plattform-Alternative zu WPF unterstützt es heute Windows, macOS und Linux sowie mobile Plattformen (Android, iOS) und Browser (via WebAssembly).



JENS SEEKAMP

**JAVA FÜR EIN-
UND UMSTEIGER**

In der Schulung lernen Sie Klassen, Methoden und Vererbung in Java sowie die grundlegenden Sprach-Features und Java 8. Durch praktische Übungen entwickeln Sie Anwendungen mit Java-Klassen-Bibliotheken und erlernen die fundamentalen Konzepte.



Mehr Informationen unter www.developer-media.de/trainings
Preise auf Anfrage!

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



Spaß an der Freud

Weshalb wird die Arbeit als Entwickler immer öfter als Belastung wahrgenommen – und was können Sie dagegen tun?

Sie kennen das Phänomen: Wenn wir etwas mit Freude tun, geht uns das gleich viel leichter von der Hand. Wir trotzen allen Mühen, schieben Hindernisse mühelos beiseite, und manchmal freuen wir uns sogar darauf, wenn wir gefordert werden und so richtig anpacken müssen – es macht regelrecht Spaß, sich in eine Aufgabe reinzuknien. Doch wie gelingt uns das?

Daran musste ich denken, als ich die Schlussfassung dieser dotnetpro las. Natürlich spürt man in allen Beiträgen eine gewisse Begeisterung der Autorin oder des Autors. Das Interview meines Kollegen Tilman Börner mit Christian Weyer ist aber ein besonderer Fall. In Zeiten, in denen KI gerne als Herausforderung oder sogar als Bedrohung wahrgenommen wird, ist Christians Begeisterung für die Chancen, die GenAI und Co. für die Softwareentwicklung bedeuten, sehr wohlthuend und ein Stück weit geradezu ansteckend.

Was für ein Kontrast, als ich mich dem Beitrag von Tom Wendel zuwandte! Da ist gleich zu Beginn die Rede von früheren Zeiten, in denen Coden noch Bock machte und das Programmieren pure Euphorie war, und dass dieser Flow inzwischen verloren und Softwareentwicklung heute nur noch bloße Pflicht sei. Ich rieb mir die Augen: Echt jetzt? Ich will nicht vorgreifen, aber selbstverständlich bleibt der Artikel nicht bei dieser Ausgangsthese stehen.

„Oft entscheidet die eigene innere Haltung darüber, wie wir unsere Arbeit wahrnehmen.“

Warum verliere ich gerade über diese beiden Beiträge so viele Worte? Nun, beide haben mich nachdenklich gestimmt, denn in beiden fand ich die Beschreibung von Verhaltensmustern, die mir beim Umgang mit Herausforderungen nicht fremd sind, und zugleich auch eine mögliche Lösung, wie diese sich überwinden lassen: Letztlich geht es um die eigene innere Haltung und um die Frage, wie man mit äußeren Einflüssen umgeht, um in der Arbeit Zufriedenheit und Motivation zu finden.

Möglicherweise geht es Ihnen so wie mir. In diesem Fall trägt diese Ausgabe der dotnetpro vielleicht dazu bei, die Freude an der Softwareentwicklung wiederzufinden.

Ich wünsche Ihnen eine schöne Zeit mit der neuen dotnetpro!

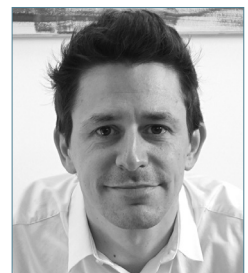
Herzlichst

Fernando Schneider
Chefredakteur dotnetpro



Christian Weyer

verrät im dotnetpro-Interview, dass er sich so jung wie nie zuvor fühlt – und woran das liegt (S. 72)



Tom Wendel

fragt sich, warum das Coden früher richtig Bock machte, heute aber nur noch Pflicht zu sein scheint (S. 130)



INHALT

Focus

Hold my beer!

DIY-Framework für SPAs

Web- und App-Entwicklung vereint

SPAs mit Ionic und Capacitor

Keine Wellness für Sicherheitslücken

Schwachstellen in SPAs aufdecken mit OWASP ZAP

Weißer Schimmel

Single-SPA – Microfrontends

.NET Stack

NETirol: Performance-Falle

EF Core mit Microsoft SQL Server

Keine Innovation ohne Integration

Azure Integration Services

Was Azure Quantum möglich macht

Quantenpower aus der Cloud

Data Access: Hybrider Zwischenspeicher

Das steckt im neuen NuGet-Paket

Microsoft.Extensions.Caching.Hybrid

Donut im Eigenbau

Chart-Komponenten in Visual Basic

Artificial Intelligence

„Sieh die KI als Juniorentwickler“

Interview mit Christian Weyer

Magische Hilfe für das UI

KI als persönlicher Assistent bei der UX-Gestaltung

Cross Concerns

Dauids Deep Dive: Die Schuld

An der Wurzel ansetzen

SOLiDe Architektur

Entwurf und Entwicklung wartbarer Softwaresysteme

6

13

20

26



Bild: Shutterstock / passion artist

30

Single Page Applications: Welche Möglichkeiten und Vorteile bieten SPAs, wo liegen Fallstricke bei ihrem Einsatz?

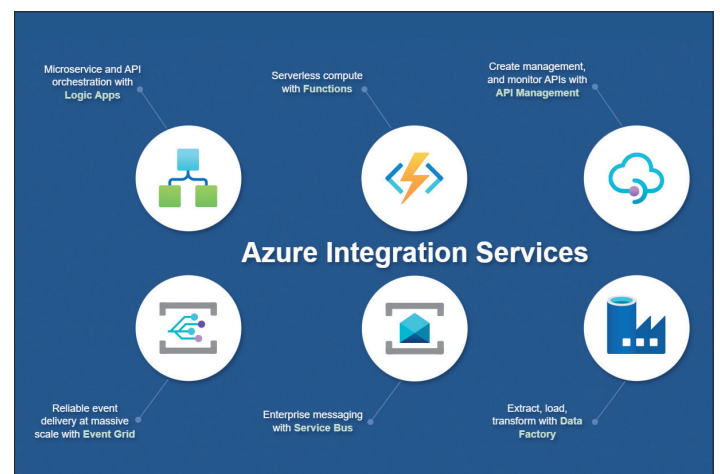
6

32

40

48

54



72

Nomen est omen: Wie Sie mit Azure Integration Services Daten, Prozessen und Systeme effizient verknüpfen.

32

78

Experten in dieser Ausgabe



Bernhard Lauer zeigt, wie Sie Dashboard-Charts à la Excel mithilfe von VB.NET und WPF erstellen.

54

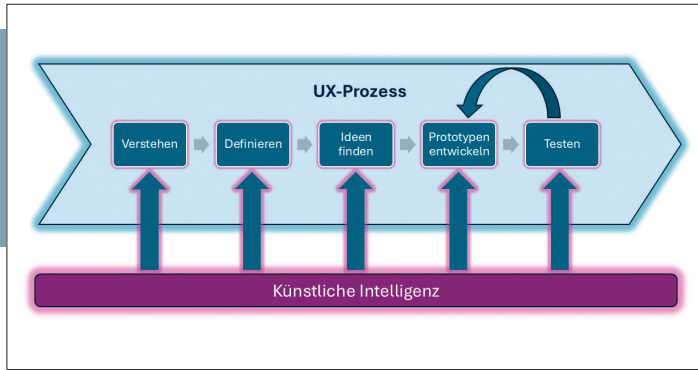


Klaus Aschenbrenner nimmt Möglichkeiten zur Performancesteigerung von PostgreSQL unter die Lupe.

112

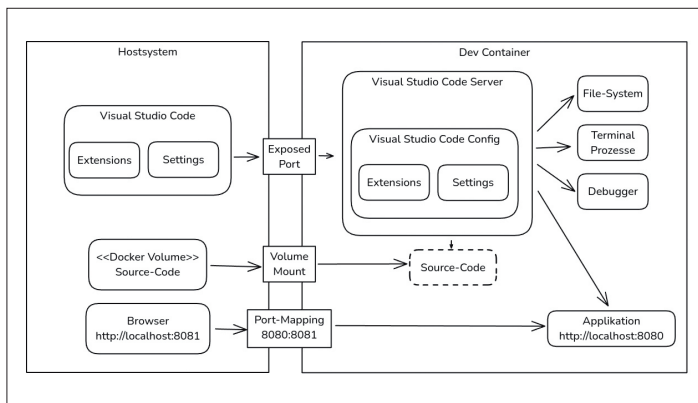
86

88



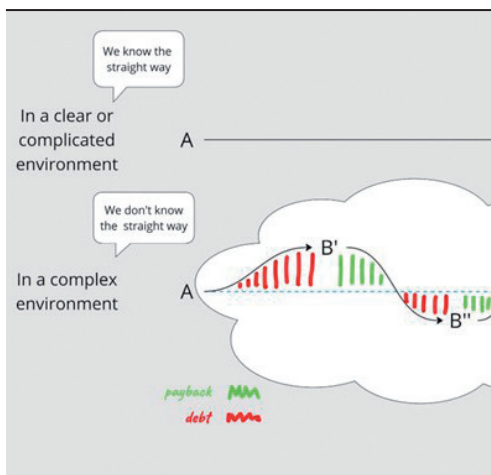
Magischer Helfer: Wie KI den UX-Design-Prozess revolutionieren kann und bei Designaufgaben unterstützt.

78



Per Dev Container lassen sich ganze Projektumgebungen in der Versionskontrolle ablegen und wiederherstellen.

107



Technische Schulden haben in Entstehung und Abbau vor allem mit einem Mangel an Führung zu tun. Womit noch nicht gesagt ist, wer verantwortlich ist und handeln sollte.

124

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.

Der Wert von Zielen

Management und Führung

97

What else?

Die Lego-ToyPad-Challenge

Spielerisch programmieren lernen

102

Entwickeln im Container

Dev Container mit Visual Studio Code

107

Performance Tuning

PostgreSQL

112

People & Soft Skills

Time-out: Alle für alle oder alle für mich?

Vertrauen als Grundlage für gemeinsamen Erfolg

122

Eine Frage der Führung

Technische Schulden in der Softwareentwicklung

124

Als Coden noch Bock machte

Wie Motivation entsteht

130

Knowledge

Die ganze Welt der Softwareentwicklung

Die Developer Week 2025 und ihre Track Chairs

140

Editorial

3

Arbeitsmarkt

136

Lernstoff

142

Impressum/Inserentenverzeichnis

144

Anbieterverzeichnis

145

Vorschau

146



DIY-FRAMEWORK FÜR SPAS

Hold my beer!

Das Wie und Warum selbst gebauter Frontend-Frameworks.

Die Liste der Dinge, die man als normalsterbliche:r Softwareentwickler:in nach Ansicht graubärtiger Senior-Nerds unter gar keinen Umständen jemals selbst programmieren sollte, ist lang. Datums- und Zeitverarbeitung? Sehr gefährlich. Kryptografie? Um Himmels willen! Ein eigenes Frontend-Framework für das neue SPA-Projekt? Klares Zeichen von Naivität und/oder Selbstüberschätzung und außerdem ein Affront gegen die natürliche Ordnung der Dinge.

Doch tief im Herzen wissen wir natürlich, dass solches Gatekeeper-Gemurmel, wie alles in der Softwareentwicklung, eher den Charakter von Richtlinien als von Regeln hat. So haben natürlich alle Entwickler:innen der aktuellen Industriestandards im Hinblick auf Zeit, Kryptografie und SPA-Frameworks irgendwann einmal ihre jeweils ersten Schritte in den jeweiligen Disziplinen unternommen – Lernen und Testen muss immer gestattet sein. Und selbst außerhalb von Experimenten muss auch ein:e Diplom-Ingenieur:in von und zu Senior-Standardlösung einsehen, dass besondere Umstände häufig an den jeweiligen Sonderfall angepasste Software er-

fordern. Die Frage ist dann nur: Was zählt als „besonderer Umstand“? Wann wäre es im Fall von Webfrontends an der Zeit, den etablierten Frontend-Frameworks den Rücken zuzuwenden und sich ein eigenes SPA-Framework zu schreiben?

Frontend-Frameworks im Jahr 2025

Das Webfrontend-Ökosystem lässt sich mit etwas Wohlwollen als „unübersichtlich“ charakterisieren. Neben zahlreichen Klassikern wie React, Vue und Angular haben sich mittlerweile die sogenannten Full-Stack-Frameworks nach Machart von Next.js etabliert, die die APIs und Paradigmen von etwa React (im Fall von Next.js) oder Vue (im Fall von Nuxt) mit ins Backend holen (**Bild 1**) [1]. „Etabliert“ heißt hierbei keinesfalls „fertig“ – bei allen relevanten Kandidaten läuft die Dependency-Umwälzpumpe am Drehzahlbegrenzer, und ein Prozess des permanenten Neuerfindens ist Standard. Was aus der Außenperspektive wie das schiere Chaos wirkt, ergibt in der historischen Betrachtung ein gewisses Maß an Sinn.

Bild: Shutterstock / passion artist

Die Geschichte von Frontend-Frameworks beginnt mit der Feststellung, dass statische HTML-Seiten ungeeignet waren, um die hyperinteraktiven Web-Apps der Generation Web 2.0 (zum Beispiel Facebook und Twitter) abzubilden. Interaktive User Interfaces im Browser selbst waren dank JavaScript zwar prinzipiell machbar, aber aufgrund von Einschränkungen und Kompatibilitätsproblemen seitens der damaligen Browser-APIs tatsächlich niemandem zuzumuten. Unter diesen Prämissen (native Browserfeatures sind nicht zu gebrauchen und UIs sind Aufgabe des Clients) entstanden die klassischen Frontend-Frameworks und auf ihrer Basis klassische Single-Page-Apps: komplexe clientseitige JavaScript-Programme, die jeden Aspekt des Browsers abstrahieren und den Server zum reinen API-Provider degradierten. Die offensichtlichen Probleme mit diesem Ansatz sind mannigfaltig: Komplexen Content auf anämischen Antik-Androids zu rendern ist nicht vergnügungssteuerepflichtig, und die seinerzeit relevanten Suchmaschinen waren von leeren HTML-Gerüsten, die erst in einem hypothetischen Browser mit Inhalt versehen werden sollten, auch nicht begeistert. Doch zum Glück ist JavaScript, wenn man die entsprechende Geisteshaltung pflegt, wie Gewalt: Wo ein bisschen JS ein bisschen hilft, hilft ein Mehr an JS mit Sicherheit noch viel weiter.

Full-Stack-Frameworks bringen die Programmierparadigmen und Problemlösungen von Single-Page-App-Frameworks auf den Server. Mit ausreichend hohem Abstraktionsniveau ist es möglich, SPA-Frameworks sowohl in den JavaScript-Engines von Browsern als auch JS-Runtimes auf Servern lauffähig zu machen. Auf diese Weise lässt sich der Traum von „isomorphic JavaScript“ verwirklichen: eine einzige UI-Beschreibung, die auf dem Server in einen statischen HTML-String übersetzt wird, auf dem Client aber ein modernes, hyperinteraktives UI rendert. Dies beschleunigt, so die Idee, das initiale Laden einer SPA durch Pre-Rendering und macht nebenher auch noch die Suchmaschinen-Crawler froh. Der Hydration genannte In-Browser-Mutationsprozess, der aus statischem HTML eine interaktive Web-App macht, ist nicht komplett trivial, da HTML eine sehr verlustbehaftete Serialisierung der JavaScript-Repräsentation einer Web-App darstellt – doch wo ein Wille ist, ist bekanntlich ein Weg.

Mehrere Variationen dieses Ansatzes werden durch steten Input aus den Geldbörsen von Big-Tech-Sugardaddys und/oder Venture Capital genährt. Diversität ist daher wirklich gegeben: Obwohl ziemlich allen Frontend-Frameworks im Jahr 2025 die bereits beschriebene Ideologie zugrunde liegt, ist die Developer Experience sehr unterschiedlich. React biegt jedes einzelne Web-API auf funktionale Programmierung um, lässt Entwickler:innen aber ansonsten viel Auswahl bei der Komposition des sonstigen Tech-Stacks. Bei Angular hat hingegen jedes Modul seinen ihm eigens zugewiesenen Platz, und einen offiziellen Compiler gibt es obendrein. Das Svelte-Framework implementiert sich konsequenterweise direkt komplett als Compiler, während Vue.js aussieht, als wür-

de es auf normale Webstandards aufbauen – wenn auch mit Ergänzungen, die Marty McFly aus dem Jahr 2062 mitgebracht zu haben scheint.

Der Markt für JavaScript-Frameworks ist also groß (denn alle machen irgendwas mit Webtech), die Marktteilnehmer (alle, die irgendwas mit Webtech machen) sind divers, und das Gottvertrauen der Geldgeber ist unerschütterlich. Wie kann da das Produkt „JavaScript-Framework“ etwas anderes sein als eine mit Höchstgeschwindigkeit rotierende Neuerfindungs-Maschine? Und wie kann ein JavaScript-Framework unter diesen Umständen etwas anderes sein als ein Produkt? Denn eines ist unübersehbar: Obwohl alle relevanten Frontend-Frameworks dank freizügigster Open-Source-Lizenzen zum Nulltarif zu haben und ohne Einschränkungen einsetzbar sind, operieren sie aus der Kundenperspektive der Entwickler:innen wie Hochglanzprodukte. Sie leisten sich durchgestylte Webseiten, richten ausschweifende Konferenzen aus und sorgen mit kurzen Produktlebenszyklen dafür, dass die Frameworks Feature-technisch auf Augenhöhe bleiben. Das alles ist per se unproblematisch: Wenn Webseiten von Open-Source-Projekten kein akutes Augenbluten auslösen, Konferenzen ein gewisses Maß an spätrömischer Dekadenz mitbringen und die Software stets bugfrei und konkurrenzfähig gehalten wird, ist das zunächst positiv. Von Big Tech oder Big VC finanziert zu werden ist ebenfalls allgemein akzeptiertes Geschäftsgebarren.

Doch wenn Webframeworks als Produkte in einem Markt operieren und wir Entwickler:innen die Konsumenten und Zielgruppen bilden, dürfen wir die feilgebotenen Waren objektiv und gnadenlos bewerten. Wir haben es schließlich nicht mehr mit den Hobbyprojekten kauziger Free-Software-Philosophen zu tun, sondern mit den Angeboten großer, gut kapitalisierter Organisationen. Wir dürfen und sollten uns fragen, ob das,

was uns der Markt vorsetzt, für unsere jeweiligen Einsatzzwecke auch tatsächlich geeignet ist.

„Das Webfrontend-Ökosystem lässt sich als ‚unübersichtlich‘ charakterisieren.“

Welchen Job machen Frameworks überhaupt?

Wikipedia beschreibt Softwareframeworks nicht unzutreffend als ein „Programmiergerüst“, das „den Rahmen zur Verfügung [stellt], innerhalb dessen der Programmierer eine Anwendung erstellt“. Das Hauptmerkmal eines solchen Gerüsts (oder idealerweise eines jeden Gerüsts, sei es in Software, auf Baustellen oder auf Spielplätzen) sollte Stabilität sein – ein Merkmal, für das das Frontend-JavaScript-Ökosystem nicht gerade bekannt ist. Mit modernen SPA-Frameworks umgesetzte Projekte zeichnen sich oft durch extravagant dimensionierte Dependency-Trees aus, die täglich gepflegt werden wollen. Breaking Changes sind nicht selten und betreffen mal die betroffenen Frameworks selbst, mal Third-Party-Komponenten, mal Sub-Sub-Sub-Dependencies. Im günstigsten Fall verbraucht das dauerhafte Updaten nur Zeit, Budget und Lebensfreude aufseiten der Ausführenden, im schlechtesten Fall verliert das Projekt den Kampf gegen den Wildwuchs und ►

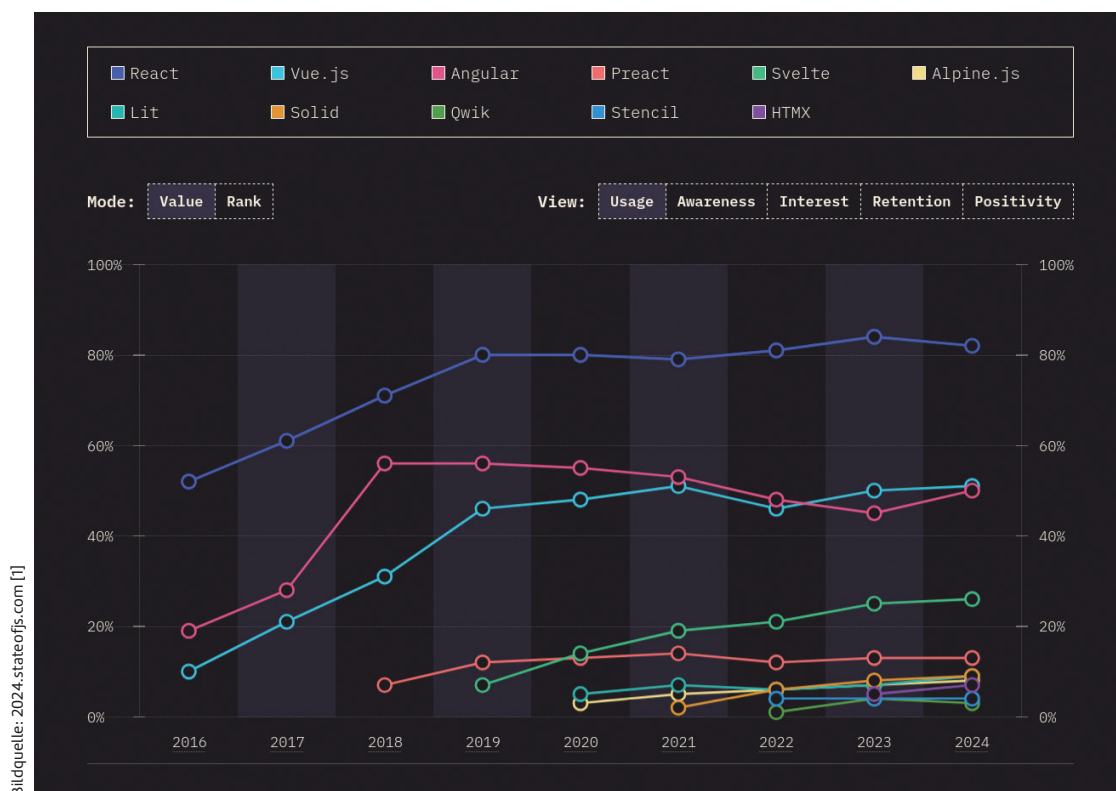
ist zu radikalen Klagemaßnahmen gezwungen. Drakonische NDAs hindern den Autor dieser Zeilen daran, die blutigen Details über die zahlreichen (ungeplanten!) Inhouse-Framework-Forks auszulaudern, mit denen Entwickler:innen in namhaften Firmen kämpfen müssen, doch er kann zumindest verraten, dass es keine Seltenheit ist, einen eigenen Angular-Entwicklungszweig auf Basis einer Version aus dem Frühmittelalter zu pflegen. Denn was sonst will man machen, wenn man zum Beispiel für über zehn Jahre an unveränderliche Hardware gebunden ist, neue Framework-Versionen aber annehmen, auf immer neueren und schnelleren Geräten zu laufen? Was sonst will man tun, wenn Abhängigkeiten von Abhängigkeiten kombiniert mit den schieren Ausmaßen des Projekts quartalsweise aufschlagende Breaking Changes nicht bewältigbar machen?

„Quartalsweise aufschlagende Breaking Changes“ sind dabei keine Übertreibung, denn die Instabilität von Frontend-JavaScript hat viele Ausprägungen und Gründe. Manchmal ist das Gewackel Programm, wie die tatsächlich alle drei Monate stattfindenden Major-Releases von Angular. Andere Kandidaten, wie React, unterstützen alte APIs auch noch nach dem Release der Nachfolger ihrer Nachfolger lange weiter, doch das Ökosystem tendiert dazu, auf die neuen Züge umgehend aufzuspringen. Third-Party-Komponenten, Dokumentation, Tutorials und Personal (Entwickler:innen wie Influencer:innen) sind stets an der Bleeding Edge orientiert, was zu faktisch nicht begründbarem (aber auch nicht ignorierbarem) Refactoring-Druck auf ansonsten stabile Codebases führt. Ein Stabilitätsanker sieht anders aus.

Das permanente Neuerfinden ist auf den Produktcharakter der Frontend-Frameworks zurückzuführen. Neue Versionen

und Innovationen füttern den News-Cycle, und auch die marginalsten Performance-Verbesserungen in Edge Cases reichen, um die Konkurrenz beim Benchmark zu überholen. Ein Vergleich mit der Autoindustrie drängt sich geradezu auf. Inkrementelle Verbesserungen an der Substanz der Produkte werden mit großem Marketing-Aufwand verkauft, und die Implementierung der inkrementellen Verbesserungen treibt die Komplexität der Produkte – was bei Kleinwagen die kilometerlangen Kabelbäume sind, sind bei modernen Frontend-Frameworks die Codezeilen, Compiler und Komplexitäten. Hinzu kommt der Wartungsaufwand eines Frontend-Framework-Produkts für seine Endnutzer (also uns Entwickler:innen) mit seinen ständigen Dependency-Updates – von dem wir noch nicht mal in allen Fällen etwas haben.

Denn wo wir bei einem Auto bei fast allen Fahrten die meisten der vier verfügbaren Räder verwenden, haben Frontend-Frameworks Ballast an Bord, der für viele Projekte irrelevant ist, aber die Komplexität treibt. Ein offensichtliches Beispiel ist, dass General-Purpose-Frameworks definitionsgemäß mehr Features an Bord haben, als ein einzelnes Projekt je verwenden könnte. Das ist nicht nur im Hinblick auf Download-Performance lästig, sondern auch der Stabilität abträglich. Breaking Changes in Updates bedeuten immer Test- und Deployment-Aufwand, selbst wenn unser spezifisches Projekt die betroffenen APIs nicht oder kaum nutzt. Außerdem sind, ihrem Übergewicht zum Trotz, alle gängigen SPA-Setups auf die in ihrem Rahmen maximal mögliche Performance gebürstet. Daran ist in Isolation auch nichts auszusetzen, doch wer schon mal versucht hat, aus einem Programm die perfekte Performance herauszuquetschen, kennt den Komplexitäts-Kippunkt: Irgendwann ist mit Vereinfachungen des Codes



Die (nicht auch nur ansatzweise repräsentative) Umfrage „State of JavaScript 2024“ zeigt, dass es unter den Frontend-Frameworks keinen unangefochtenen Marktführer gibt (Bild 1)

nichts mehr zu erreichen, und elaborierte Caching-Strategien und Mikrooptimierungen beginnen das einst elegante Programm zu verkomplizieren.

Ebenfalls als Ballast bewertbar sind lebenserhaltende Maßnahmen für antike APIs, die eigentlich niemand mehr verwenden möchte. Reacts Klassenkomponenten haben beispielsweise ihren Hauptwohnsitz in der Mottenkiste, sind aber weiterhin Bestandteil des Kernprodukts. Das ist nicht nur buchstäblicher Ballast in Form von nutzlosen Kilobytes, sondern verkompliziert, wie jedes Subsystem in einem Programm, durch Wechselwirkungen jedes andere Subsystem – von der Angriffsfläche für Sicherheitslücken und Dependency-Verfall ganz zu schweigen. Kein Code ist so effizient, schnell, sicher und wartungsfreundlich wie kein Code – doch „kein Code“ entspricht nicht dem Ethos des modernen SPA-Stacks.

Die Aufgabe, die Frontend-Frameworks im Jahr 2025 wahrnehmen, ist eine vielfältige. Sie kombinieren ausgesuchte Toolchains mit Hochleistungs-Abstraktionen über Browser-APIs und verbinden beides mit einer dazu passenden Entwicklungs- und Designphilosophie. Marketing und Community-Pflege sorgen dabei für ein passendes Ambiente. Sie sind die Bentleys der Webentwicklung: komfortabel, leistungsstark und technisch ausgefeilt. Aber sind sie (mit ihren immensen Unterhaltskosten) das passende Vehikel, um als normaler Mensch im Jahr 2025 ein ganz normales Discount-Dashboard zu bauen? Sind sie nötig? Sinnvoll? Kosteneffektiv?

In welchem Jahrhundert leben wir eigentlich wirklich?

Was Frontend-Frameworks faktisch leisten, ist das eine. Das andere ist ihr Raison d'Être, der auf zwei vermeintliche Problemstellungen zurückzuführen ist:

1. Web-Apps erfordern ein Maß an clientseitiger Interaktivität, das nur durch die Priorisierung von Frontend-JavaScript über alle anderen Überlegungen umsetzbar ist.
2. Browser haben hierfür nicht die passenden Bordmittel, weshalb massiver Abstraktions- und Engineering-Aufwand zu betreiben ist.

In den frühen 2010er-Jahren, als die aktuelle Generation der SPA-Frameworks (ihrerseits die Grundlage der zurzeit so populären Full-Stack-Frameworks) aus der Ursuppe gekrochen kam, war das eine zutreffende Zustandsbeschreibung. Dinosaurier wie der Internet Explorer waren noch weit vom Aussterben entfernt, und über 60 Prozent aller Websites diesseits der Andromeda-Galaxie sahen sich zur Verwendung von jQuery gezwungen, um zahlloser Browser-Inkonsistenzen Herr zu werden und fehlende UI-Features via Plug-in einzurüsten. Nur sind die frühen 2010er mittlerweile vorbei!

Webbrowser wurden in allen Dimensionen massiv aufgerüstet. DOM-APIs haben ein Komfortniveau erreicht, das ein

Programmieren ohne Abstraktionen für Normalsterbliche möglich macht. Die Einführung von Web Components bringt ein natives Komponentenmodell in den Browser, ECMA-Script-Module machen viele Aspekte klassischer Toolchains überflüssig, HTML hat mit Formular-Features und Neuerungen wie `<dialog>` massiv aufgerüstet, und moderne CSS-Features wirken aus der Perspektive von 2013 wie Science-Fiction. Die Performance hat sich ebenfalls stark verbessert, und durch schnelleres Ausrollen von neuen Releases erreichen neue Browser-Versionen in kürzester Zeit die Massen der Nutzer:innen. Der gesamte Punkt 2 der obigen Kausalkette

ist nicht mehr als eine von Opas Geschichten von der fernen Front des Browserkriegs.

Und Punkt 1 sieht bei nüchterner Betrachtung nicht viel besser aus. Es ist keinesfalls zwingend nötig, Web-Apps primär mit JavaScript zu entwickeln. Das ist nur der aktuelle Zeitgeist. Viele Use Cases (Dateneingabe, Content-Websites) sind hervorragend mit serverseitigen Mitteln abzubilden, sofern alles schnell genug

lädt – was ohne den Overhead eines fetten Frontend-Frameworks problemlos machbar ist. Wenn mal etwas mehr clientseitige Interaktivität erforderlich ist, gibt es zahllose Optionen, die nicht zu 100 Prozent aus JavaScript-Frameworks bestehen. HMX [2] etwa motzt HTML so auf, dass mit ein paar Attributen API-Requests durchgeführt und Views mit Updates versorgt werden können:

```
<!-- Wie zu Opas Zeiten einfach die Library laden -->
<script src="https://unpkg.com/htmx.org@2.0.4"></script>
```

```
<!-- Button-Klick triggert einen Post-Request. Die
Antwort ersetzt, wenn eingetroffen, den Button
(das "outerHTML") -->
<button hx-post="/clicked" hx-swap="outerHTML">
  Klick!
</button>
```

Hotwired [3] funktioniert hingegen wie ein SPA-Layer über eine klassische serverseitige Web-App, indem es HTML-Updates ohne Seiten-Reload in die gerade aktive Seite hineinpatcht. View Transitions [4] sind ein in fast allen Browsern verfügbarer Webstandard, der einen ähnlichen Effekt erreicht und ganz ohne Third-Party-Software auskommt. Und dank der bereits benannten Browser-Updates können moderat kompetente Entwickler:innen auch einfach ein wenig Vanilla-JS schreiben und ein paar Web Components zusammenstecken.

Lösen die genannten Alternativen zu SPA-Frameworks jedes Problem in der Frontendentwicklung? Natürlich nicht, keine Software löst alle Probleme in ihrer Domäne unter allen Umständen. Aber sie sind Alternativen. Sie lassen Endprodukte entstehen, die leichtgewichtiger, weniger komplex und weniger fragil sind als das, was der State of the Art verordnet. ►

„Stabilität ist ein
Merkmal, für das
das Frontend-
JavaScript-Ökosystem
nicht gerade
bekannt ist.“

Moderne Webentwicklung des Jahres 2025 besteht leider überwiegend darin, unter der Annahme der Alternativlosigkeit eine hochgezüchtete JavaScript-Maximallösung durch das Realitäts-Nadelöhr zu zwängen. Das ist kein Engineering, sondern Ideologie. Engineering findet Kompromisse und lokale Maxima für die jeweiligen Anwendungsfälle. Spezialisierte Bereiche wie Raumfahrt oder Gelddruckmaschinen wie der Facebook-Feed profitieren davon, dass keine Kosten und Mühen gescheut werden, sollten aber Ausnahmen bleiben. Kosten und Mühen zu scheuen und zu reduzieren ist gutes Engineering. Vereinfachung ist gut. Weniger Wartungsaufwand ist gut. Billiger ist besser. Solange der Gegenstand unserer Engineering-Bemühungen ein brauchbares Endprodukt wird, sollten wir versuchen, das Meiste mit dem Wenigsten zu erreichen.

Hold my beer

Woraus besteht ein Softwareframework? Eigentlich nur aus Software und Entscheidungen. Es legt Trade-offs fest, definiert APIs und bestimmt, was in welcher Reihenfolge zu passieren hat. Und da wir als Softwareentwickler:innen den ganzen Tag nichts anderes machen, als APIs zu definieren und Trade-offs festzulegen, sind wir allesamt prädestiniert für die Fertigung eines eigenen Frontend-Frameworks. Wird es das brillianteste Frontend-Framework aller Zeiten werden? Vielleicht, vielleicht nicht. Es wird aber unser eigenes sein, mit den Zutaten und Entscheidungen, die uns zusagen. Fangen wir doch gleich mit ein paar Entscheidungen an!

Erste Design-Entscheidung: JavaScript depriorisieren. Das klingt nach einer für ein Frontend-Framework widersinnigen

„Das Web ist eine stabile Plattform für Content und User Interfaces aller Arten.“

Entscheidung, aber es ist unser Framework – wir können machen, was wir wollen! Wenn wir unser Tool so viele Aufgaben wie möglich an eingebaute APIs und Standards wie HTML und CSS delegieren lassen, müssen wir weniger reimplementieren und benötigen weniger Tooling. Wir nutzen die Browser des Jahres 2025 und werden uns ihres vollen Funktionsumfangs bedienen! Das Komponentenmodell? Web Components. Styling-System? Nacktes, standardkonformes CSS. State Management? Wenn wir ohnehin Web Components mit JavaScript-Klassen machen, benötigen wir keine zusätzlichen State-Container oder Signals – und falls sich das eines Tages ändert, können wir das revidieren, denn unsere zweite Designentscheidung ist die für den radikalen Pragmatismus.

Es gibt keinen Grund, päpstlicher zu sein als der Papst. Wenn der Browser ein bestimmtes Feature nicht bietet, installieren wir uns eine passende Library – im Idealfall eine mit wenig Overhead und vielen Alternativen, damit bei unangenehmen Breaking Changes oder einem

Ausfall der Maintainer Alternativen parat stehen. uhtml [5] ist eine Rendering-Engine auf Basis von Template Literals. Sie ist kompakt, kann viel und es existieren buchstäblich Hunderte fast identische Alternativen. Ornament [6] macht das Handling von Web Components etwas angenehmer und besteht aus lediglich einer Handvoll Funktionen, die explizit für den „Bau dein eigenes Framework“-Anwendungsfall ausgelegt sind.

Auf Basis dieser zwei Entscheidungen ist es uns bereits möglich, eine einfache Komponenten-Basisklasse zusammenzustecken, wie in [Listing 1](#) gezeigt.

● Listing 1: Eine einfache Komponenten-Basisklasse

```
// src/framework.js

import { reactive, connected, debounce } from
  "@sirpepe/ornament";
import { html, render } from "uhtml";

export class BaseElement extends HTMLElement {
  // Enthält das UI des Elements. Details unwichtig :)
  #root = this.attachShadow({ mode: "closed" });

  // Macht uhtml in Subklassen als this.html verfügbar
  html(...args) {
    return html(...args);
  }

  // Subklassen können eine Methode namens render()
  // definieren, die das UI des Elements definiert.
  // Sie wird von dieser Basisklasse aufgerufen,

  // wenn sich etwas an den Daten der Klasse (z.B.
  // an Attributen) ändert, das aber höchstens
  // einmal pro Render-Zyklus. DOM-Diffing patcht
  // UI-Updates nahtlos und schnell ein.
  @reactive()
  @connected()
  @debounce()
  #render() {
    if (this.render) {
      render(this.#root, this.render());
    }
  }

  // Stellt den Funktionsumfang von Ornament dem Rest
  // des Frameworks bereit
  export * from "@sirpepe/ornament";
}
```


Eine Frontend-Komponente kann von dieser Basisklasse erben, und sobald wir die Komponenten-Klasse mit einem HTML-Tag assoziieren, übernimmt die HTML-Engine des Browsers den Rest. Das könnte für ein einfaches „Hallo Welt“ wie in [Listing 2](#) gezeigt aussehen.

Diese Klasse werfen wir in einen Browser, konfrontieren diesen dann mit dem HTML-Tag `<say-hello>` (optional mit dem `name`-Attribut), und fertig ist eine Web Component ... jedenfalls fast. Denn die Decorators-Syntax mit dem `@`-Zeichen ist für Browser Mitte 2025 noch Terra incognita, auch wenn sie kommender Bestandteil des ECMAScript-Standards ist. Wir kommen also doch nicht ganz um etwas Tooling herum, doch das muss kein Problem sein – wenn wir es clever anstellen!

Die dritte Entscheidung für dieses Framework ist nämlich, Developer Experience zugunsten von allem anderen dezent zu depriorisieren, wenn auch nicht komplett aufzugeben. Die Marketing-Strategie der gängigen SPA-Frameworks hat seit jeher die Entwicklerschaft im Fadenkreuz und preist den nur mit ihnen erreichbaren Komfortfaktor an. Tatsächlich hat sich aber die Webplattform mittlerweile so weit entwickelt, dass der maximale nativ gebotene Komfortfaktor als „gar nicht mal so schlecht“ einzuordnen ist. Das einzig echte Problem ist, dass bisher nicht alle Browser die für „gar nicht mal so schlecht“ benötigten Features an Bord haben. Mit den passenden Tools lässt sich aber genau diese Lücke schließen – und zwar so, dass wir stets auf dem Pfad der Webstandards verweilen. Wir sind mal an der Spitze der nativen Features und manchmal sogar weiter vorne als die Browser. Wir achten aber strikt darauf, den Pfad der Webstandards nie vollends zu verlassen, sodass jedes Investment in Tools eine Investition in Zukunft und kein fahrlässiges Gezocke auf das Überleben irgendwelcher Frameworks oder die Launen irgendwelcher Maintainer ist. Da die Decorators-Syntax mit dem `@`-Zeichen ein definitiv kommender Webstandard ist, der bereits in allen möglichen Tools außer Browsern implementiert ist, ist ihr Einsatz so unriskant wie nur möglich – selbst wenn wir als Enabler ein Tool wie esbuild [7] brauchen. Dieses sorgt für den Support von Decorators, unterstützt Hot Reloading sowie einen Dev-Server und entschärft ein paar kleinere Papercuts ... aber nicht mehr! Wer Node.js installiert hat, kann sich all das mit einem einzigen CLI-Kommando ins Haus holen:

```
$ npx esbuild src/index.js --bundle --target=es2022
  --format=esm --outdir=dist --watch --servedir=.
```

Und das war es schon! Mit zwei mikroskopischen Libraries, einem generischen JavaScript-Compiler, der Entscheidung für Webstandards und einem lächerlichen Maß an Glue Code haben wir ein sehr brauchbares Feature-Set implementiert:

- Unsere „Toolchain“ (mit exakt einem Tool darin) erlaubt die Verwendung der neuesten Standard-Features und versorgt uns mit DX-Grundbedürfnissen wie Hot Reloading.
- Mit Web Components haben wir ein Komponentenmodell, das (weil in HTML-Elementen resultierend) allen Entwickler:innen bekannt ist und mit jeder Software, vom PHP-CMS bis zum React-Frontend, kompatibel ist.

● Listing 2: Hallo-Welt-Beispiel

```
// src/say-hello.js

import { BaseElement, define, attr, string } from
  "../framework.js";

// Die Komponenten-Basisklasse erbt von unserer
// Framework-Basisklasse und kann daher ihre
// Renderlogik bequem in einer render()-Methode
// definieren. Der Decorator @define() meldet die
// Klasse beim Browser als ein neues HTML-Element
// an.
@define("say-hello")
export class SayHello extends BaseElement {
  // Definiert ein HTML-Attribut namens "name",
  // das vom Typ "string" ist und "Anonymous" als
  // Standardwert verwendet.
  @attr(string()) accessor name = "Anonymous";

  // Definiert das von <say-hello> implementierte
  // UI. Änderungen an relevanten Daten wie etwa
  // dem "name"-Attribut lösen automatisch ein
  // UI-Update aus.
  render() {
    return this.html`<p>Hello ${this.name}</p>`;
  }
}
```

- Das Komponentenmodell verwendet Klassen, was die Frage, wo wir den Zustand der Komponenten hinterlegen, praktischerweise vollautomatisch beantwortet.
- DOM-Diffing und ein paar Decorators machen das Programmiermodell relativ deklarativ, relativ performant und relativ komfortabel.

Diese Liste ist eher kurz, aber sehr viel mehr als „React out of the box“ braucht sie auch nicht zu bieten. Ähnlich wie bei React müsste auch unser Framework für komplexere Aufgaben allerlei Ergänzungen erhalten, doch das ist kein Problem – wir nehmen Webstandards, wo möglich, und an Standards orientierte Libraries, wo nötig. Mit diesem Standards-first-Ansatz haben wir aber bereits mehr geleistet, als nur eine so gerade eben adäquate Featureliste zusammenzustellen.

Quo vadis, DIY-Framework?

Featurelisten sind das eine. Nachdem wir aber die ersten 99 Prozent dieses Artikels damit verbracht haben, uns über die Instabilität und Aufgeblasenheit der gängigen SPA-Frameworks zu ereifern, lohnt sich ein Blick darauf, wie sich unser Werk unter diesen Gesichtspunkten schlägt. Was macht unser Mini-Framework alles nicht?

- Wir haben null Vendor-Lock-in. Das Build-Tool ist komplett austauschbar, da wir es allein als Mittel der Standard- ►

Durchsetzung verwenden – keine exotischen Plug-ins oder Spracherweiterungen weit und breit! Wenn Browser eines Tages Decorators unterstützen, können wir das Tooling ersatzlos entfernen, und sollte der Maintainer dem Irrsinn der Welt anheimfallen, gibt es mehrere Eins-zu-eins-Ersatz-Optionen ... denn wir brauchen nicht mehr als Webstandards und das absolute Minimum an Komfort-Features. Stringbasierte Templating-Systeme mit DOM-Diffing wie `uhtml` gibt es wie Sand am Meer, Eins-zu-eins-Ersatz und Ornament wurde von einem untalentierten Spinner (dem Autor dieses Artikels) innerhalb einer Woche unter dem Einfluss bewusstseinsentrübender Elemente (SARS-CoV-2) zusammengeschraubt. Bei Bedarf kann es durch richtige Software wie Lit [8] ersetzt oder von durchschnittlich begabten Entwickler:innen reproduziert oder geforkt und gewartet werden.

- Es gibt nichts zu lernen, was Kenner des Browsers nicht ohnehin schon auf dem Kasten haben. Web Components sind nichts weiter als HTML-Elemente, also etwas, das alle Webentwickler:innen wie ihre Westentaschen kennen. Und alle, die schon mal mit Absicht einen Code-Editor geöffnet haben, sind in der Lage, eine Klasse mit den basalsten aller OOP-Techniken herunterzuschreiben. Vom verkalkten Netscape-Boomer bis zum jungen Hüpfer: Wo ein Wille ist, kann die Deklaration `class A extends B` entstehen. Selbst LLMs, die lobotomisiertesten aller Junior-Devs, sind in der Lage, unfallfreie JavaScript-Klassen zu generieren!
- Mangels Vendor- oder Technik-Lock-in sind unsere Komponenten außerdem kompatibel zu allem, denn alles – auch das neueste SPA-Framework – muss am Ende des Tages Webstandards wie HTML sprechen. Sollte unser Projekt also eines Tages von der reinen Lehre abweichen und ein „richtiges“ Frontend-Framework übergeholfen bekommen, werden unsere Komponenten nahtlos in das neue System integrierbar sein.
- Da wir fast alle signifikanten Aufgaben an den Browser selbst abwälzen, besteht unser Framework aus praktisch nichts. Es gibt ein wenig Glue Code, doch der hält nur Features zusammen, die uns die Plattform bietet. Die wenigen Dependencies sind austauschbar und erhalten wenige Updates, da bei der Entwicklung von Template-Parsers und API-Umbau-Decorators wenig schiefgehen kann und wenig Innovationsspielraum bleibt. Kein Code ist so gut wie kein Code – was nicht da ist, kann nicht veralten, nicht langsam sein, nicht Ressourcen für die Wartung verschlingen.

Unser selbst gestricktes SPA-Toolkit macht zwar nicht viel, doch das ist in Ordnung – was nicht ist, kann immer noch werden. Und im Zuge des Werdens kann das Framework sich weiterhin treu bleiben, denn anders als Features sind die Qualitäten des Frameworks etwas, das allein auf unseren Entscheidungen beruht. Indem wir uns auf Standards besinnen und Tooling und Dependencies nur installieren, wenn sie eine brauchbare Exit Strategy erlauben, fördern wir zutage, was heutzutage allzu oft unter Megabytes an JavaScript begraben liegt: dass die Webplattform eine stabile Grundlage für Content und UIs aller Arten ist. Sie zeigt Texte an, rendert

komplexe UIs für Industrieprozesse und ist die Grundlage des Infotainment-Systems von Luxusautos. Sie leistet all das, obwohl Webentwickler:innen auf einem bestehenden, soliden Fundament eine fragile JavaScript-Lasagne nach der anderen auftürmen. Sie leistet das alles aber auch, wenn wir die Lasagne im Ofen lassen.

Und diese Grundidee kann problemlos am Leben erhalten werden, selbst wenn das Framework wächst und sich entwickelt. Jedes neue Feature und jede neue Anforderung kann unter der Maßgabe „Standards first“ betrachtet und auf eine Weise umgesetzt werden, die in weniger Code, weniger Dependency-Updates und weniger Risiko mündet.

Der Normalzustand im Frontend ist heutzutage das glatte Gegenteil: Jedes Framework ist der neue große Entwurf, der alles neu und alles richtig macht, gleich wie viel JavaScript dafür aufgewendet werden muss und wie viel Instabilität damit einhergeht. Doch mit einer etwas anderen Herangehensweise kann die Entwicklung eines Webfrontends auch ganz anders laufen. Wer einen Use Case hat, der nicht eine hyperinteraktive Frontend-App ist, benötigt vielleicht gar kein SPA-Framework, sondern schnell ladendes HTML plus ein bisschen JavaScript-Würze. Wenn besondere Anforderungen an Performance oder Stabilität gestellt werden, kann man sich, wie gesehen, mit wenig Aufwand einen Framework-Maßanzug schneiden.

Nur wer gedenkt, genau das Gleiche wie der Rest der Welt zu machen, und an den gleichen Problemen wie der Rest der Welt scheitern möchte, ist mit den gleichen Lösungen, wie sie der Rest der Welt verwendet, gut bedient. Aber wie wäre es mal mit einem Projekt, das schnell lädt, einmal pro Woche statt einmal pro Stunde ein Dependency-Update benötigt und das auch nach Jahren noch aus seinem Quellcode lauffähig gemacht werden kann? Der Framework-Markt hat hierfür kein Angebot. Also machen wir uns einfach, wie wir gesehen haben, unser eigenes. ■

- [1] State of JavaScript 2024, <https://2024.stateofjs.com/en-US>
- [2] HTMX, <https://htmx.org>
- [3] Hotwired, <https://hotwired.dev>
- [4] View Transitions, www.dotnetpro.de/SL2504-05SPADIY1
- [5] `uhtml` auf GitHub, <https://github.com/WebReflection/uhtml>
- [6] Ornament auf GitHub, <https://github.com/SirPepe/ornament>
- [7] `esbuild` auf GitHub, <https://esbuild.github.io>
- [8] Lit, <https://lit.dev>



Peter Kröner

war selbstständiger Webdesigner und –entwickler, bevor er 2010 Trainer und Berater für Webtechnologien wurde. Seither bereist er das Land im Namen von JavaScript, HTML5, React und TypeScript, über die er auch regelmäßig bloggt und podcastet.

dnpCode

A2504–05SPADIY

SPAS MIT IONIC UND CAPACITOR

Web- und App-Entwicklung vereint

Warum SPAs und Hybrid-Apps mit Ionic und Capacitor den entscheidenden Wettbewerbsvorteil bieten und welche Herausforderungen dabei zu meistern sind.

Moderne Softwareentwicklung erfordert flexible Frontends für alle Endgeräte. Nutzer wünschen reibungslose Erlebnisse auf Smartphone, Tablet und Desktop. Hybrid-Apps und SPAs erfüllen diese Erwartungen und sparen deutlich spürbar Zeit sowie Ressourcen.

Technologien wie Capacitor und Ionic bieten mächtige Werkzeuge für Web, iOS und Android. Durch Plug-ins wird nativer Gerätezugriff einfach, ohne separate Codebasen. Teams profitieren von bekannten Frameworks, was Entwicklung und Wartung beschleunigt. Damit lassen sich gemeinsame SPAs umsetzen, die sich wie native Anwendungen anfühlen. Neben der Performance sprechen vereinfachte Updates und einheitliche Designs für diesen Weg. Geschäftsanwendungen überzeugen Kunden, weil sie überall problemlos laufen.

Was die Nutzerwelt erwartet

Der Markt für digitale Geschäftsprozesse erweitert sich stetig. Viele Unternehmen möchten nicht nur den Desktop-Bereich abdecken, sondern auch Mitarbeitern im Außendienst hochwertige Anwendungen bieten. Die Arbeitsweise hat sich verändert: Statt nur im Büro auf Systeme zuzugreifen, erledigen Teams Außeneinsätze oder Kundentermine und wollen jederzeit auf relevante Daten und Funktionen zugreifen. Das Smartphone ist zum unverzichtbaren Werkzeug geworden, sei es beim Erfassen von Auftragsinformationen oder beim Nachschlagen technischer Dokumentationen. Nutzer legen Wert auf Apps, die sich reibungslos bedienen lassen und überall dieselbe Funktionalität bieten. Wer eine Anwendung startet, erwartet kurze Ladezeiten und durchweg logische Navigationswege.

Hybride Konzepte ermöglichen es, diesen Ansprüchen mit einer einzigen Codebasis gerecht zu werden. Ein Unternehmen, das Montagedienstleistungen anbietet, kann beispielsweise einen umfassenden Wartungs- und Reparaturprozess in einer Anwendung abbilden: Im Büro dient der Desktop-Browser zur Terminverwaltung und zur Einsicht in Kundenprofile. Vor Ort nutzt der Techniker dasselbe System, das nun in einer mobilen Version mit offlinefähigen Features bereitsteht. Fotos, Bestandslisten und digitale Checklisten können direkt erfasst werden, ohne dass eine separate App für Android oder iOS programmiert werden muss. Diese Vorgehensweise reduziert Entwicklungs- und Wartungskosten und stei-

gert die Geschwindigkeit, mit der neue Funktionen für alle bereitgestellt werden.

Die Erwartungshaltung an Software bleibt hoch, da komplexe Geschäftsvorfälle abgebildet und gleichzeitig nutzerfreundliche Oberflächen geboten werden sollen. Hybrid-Apps, die von modernen JavaScript-Frameworks angetrieben werden, sind in der Lage, große Datenmengen effizient zu verarbeiten, während sie dank Responsive Design auf unterschiedlichen Displays klar strukturiert wirken. Unternehmen wählen verstärkt diesen Weg, um einheitliche Prozesse ohne Mehraufwand zu gestalten und gleichzeitig sowohl Fachpersonal im Außendienst als auch die Verwaltung im Büro zentral zu unterstützen.

Technologien im Wandel

Browser-Engines und Webstandards haben sich enorm weiterentwickelt. Moderne JavaScript-Laufzeitumgebungen verarbeiten komplexe Logik sehr zügig, was vor einigen Jahren undenkbar war. Frameworks wie Angular, React oder Vue setzen auf Mechanismen wie Virtual DOM oder Web Components und nutzen ausgefeilte Optimierungen beim Rendern. Dadurch laufen Single Page Applications nahezu ohne Seitenwechsel und vermitteln ein reaktionsschnelles Benutzererlebnis. Transpiler-Toolchains wie TypeScript helfen zusätzlich dabei, Code sauber zu strukturieren und Fehler frühzeitig zu vermeiden.

Viele befürchteten früher, dass Hybrid-Apps spürbar langsamer als native Anwendungen seien oder sich nicht nahtlos ins Betriebssystem einfügen könnten. Dieser Vorbehalt stammte aus einer Zeit, als mobile Browser noch starke Einschränkungen hatten und Cordova-Projekte kaum hardwarebeschleunigt liefen. Inzwischen hat sich das grundlegend geändert: WebView-Implementierungen greifen auf dieselben leistungsfähigen Engines zu, die auch im Vollbrowser zum Einsatz kommen. Hardwarebeschleunigte Grafiken, moderne CSS-Layouts und reaktive UIs übertreffen häufig die Erwartungen, sodass Nutzer kaum Unterschiede zu nativen Apps wahrnehmen. Ein weiteres Plus sind UI-Komponenten, die sich dynamisch an das gewohnte Look and Feel von iOS, Android oder dem Web anpassen.

Capacitor liefert zusätzlich den nativen Unterbau, um Gerätedienste wie Kamera, Geolokalisierung oder Push-Be- ►

nachrichtungen fließend anzusprechen. Das System bindet Objective-C oder Java-Code ein, ohne dass separate Projekte gepflegt werden müssen. Build-Tools wie Webpack, Vite oder esbuild optimieren Ressourcen für kurze Ladezeiten und fügen Polyfills nur dort hinzu, wo ältere Browser sie wirklich benötigen. Electron, das dieselbe JavaScript-Codebasis weiterverwendet, dient als Tür zur Desktop-Welt. So lassen sich nahezu alle Endgeräte bedienen, ohne permanent an Performance- oder Integrationsgrenzen zu stoßen. Diese enorme technische Bandbreite unterstreicht, wie viel Potenzial in Hybrid-Apps steckt.

Unternehmen und Entwickler profitieren von modernen Webtechnologien

Wer Hybrid-Apps baut, profitiert von einer schnelleren Time-to-Market. Teams entwickeln nur eine Codebasis und verteilen sie auf mehrere Plattformen. Das verkürzt den Abgleich zwischen iOS, Android und Web und reduziert den Pflegeaufwand nach jedem Release. Updates fließen häufig direkt in die gemeinsame Codebasis, wodurch sich ein neuer Funktionsumfang parallel für alle Nutzergruppen ausliefern lässt. Ein Beispiel wäre etwa die Wartungs-App eines Anlagenbauers: Statt separate Versionen zu pflegen, bekommt die Außendienstmannschaft gleichzeitig mit den Büroangestellten Zugriff auf neue Features, unabhängig vom Endgerät. Laut einer Case Study von Ionic [1] können Unternehmen damit bis zu 30 Prozent Entwicklungszeit sparen, weil sie Prozesse nicht doppelt durchlaufen müssen.

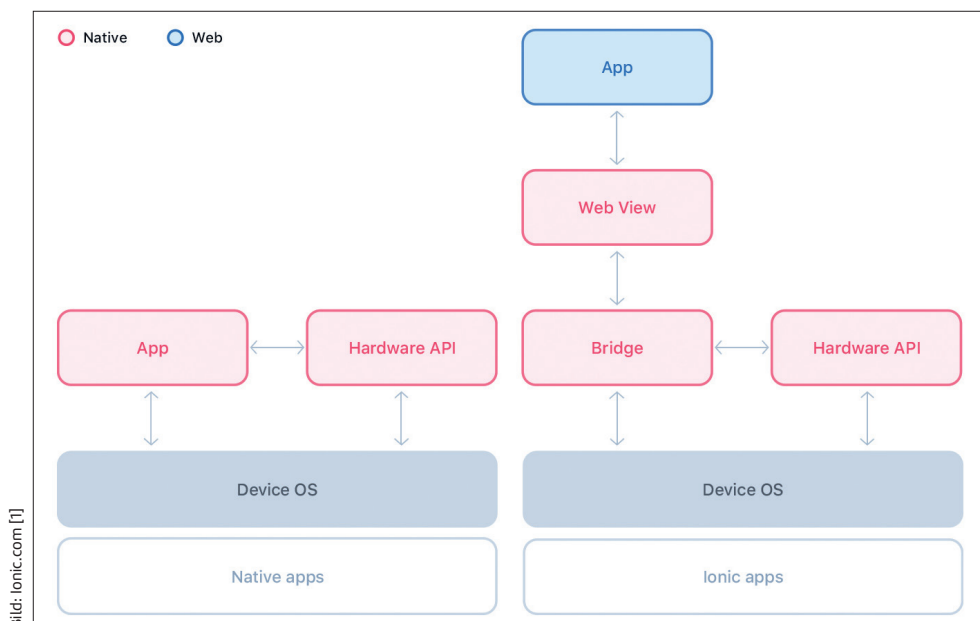
Das Bild der Webentwicklung hat sich in den letzten zehn Jahren stark gewandelt. Wo früher JavaScript und Co. oft belächelt wurden und mangels klarer Struktur schnell unwartbare Anwendungen entstanden, sind heute Frameworks wie Angular, React oder Vue hochgradig organisiert. Build- und Test-Tools (zum Beispiel Jest oder Cypress) sorgen für stabile Pipelines, während TypeScript und modulare Strukturen Übersichtlichkeit bewahren. In vielen Punkten gelten moderne Webtechnologien sogar als fortschrittlicher als klassische Desktop- oder Server-Frameworks. Darauf aufbauend ist es für Entwicklerteams einfacher, anspruchsvolle Anwendungen im Browserumfeld zu erstellen, ohne auf bewährte Software-Engineering-Prinzipien zu verzichten (vergleiche Bild 1).

Entsprechend gestalten sich Schulungen und Wissenstransfer reibungsloser. Neue Teammitglieder, die bereits Erfahrung in Webtechnologien haben, finden sich leichter ein. Komplexe Architekturentscheidungen und seltene plattform-spezifische Sprachen entfallen. Dadurch sinken Abhängigkeiten von einzelnen Schlüsselpersonen. Auch die langfristige Wartung einer Anwendung ist planbarer. Bugfixes und Security-Updates betreffen meist nur eine Codebasis, anstatt

pro Plattform einzeln aktualisiert zu werden. Das verringert die Gefahr, dass eine Plattform hinterherhinkt. Continuous Integration/Delivery-Pipelines sind effizienter, wenn nur ein zentrales Projekt gepackt und verteilt wird. Firmen, die auf hybride Konzepte setzen, berichten in Studien von signifikant gesunkenen Betriebskosten und kürzeren Feedback-Zyklen, weil sie rasch auf Kundenwünsche reagieren können.

In welchen Konstellationen das Ganze Sinn ergibt

Eine klassische Webanwendung, die lediglich im Browser läuft, deckt viele Anwendungsfälle ab, in denen Nutzer vor allem Lesefunktionen und einfache Formulare benötigen. Solange kein erweiterter Hardwarezugriff nötig ist und ein stabiles Online-Szenario gewährleistet werden kann, genügt oft eine rein responsive Website oder eine Single Page Application (SPA). Dadurch sparen Unternehmen den Aufwand, mehrere Plattformen nativ zu bedienen. Doch sobald komple-



Aufbau einer Hybrid-App in Vergleich zur Native App (Bild 1)

xere Interaktion gefragt ist, stoßen reine Web-Apps bisweilen an Grenzen. Beispielsweise sind Kamera- oder NFC-Zugriffe nur eingeschränkt über den Browser möglich.

Hier kommen Hybrid-Apps mit Ionic und Capacitor ins Spiel. Sie bieten einen Web-Technologie-Stack, aber erlauben den nahtlosen Zugriff auf GPS, Kamera, Barcode-Scanner oder NFC-Schnittstellen. Das senkt Entwicklungs- und Betriebskosten, weil kein getrenntes Android- oder iOS-Projekt notwendig ist. Gerade Unternehmen, in denen Mitarbeitende verteilt arbeiten, profitieren von dieser Flexibilität. Wer etwa Außendiensttechniker, Lagerpersonal und Büromitarbeiter hat, die mit verschiedenen Endgeräten auf denselben Datenbestand zugreifen, kann eine gemeinsame Codebasis pflegen und dennoch Kamerafunktionen für Statusfotos oder Barcode-Scanning einsetzen.

Hybrid-Apps bieten sich auch für Branchen an, in denen Software rasch aktualisiert werden muss. Neue Funktionen landen prompt in allen Clients. Zudem lassen sich die nativen Integrationen erweitern, ohne dass ständig mehrere Plattformteams koordiniert werden müssen. Bei aufwendigen Grafikberechnungen gilt es zu prüfen, ob GPU-intensives Rendering tatsächlich ein natives Framework erfordert. Modernes WebGL kann auch im Browser beachtliche 2D- oder 3D-Effekte erzielen, was in Hybrid-Apps ohne gesonderte Implementierung lauffähig bleibt. Wer offlinefähig sein möchte, kann die App als Progressive Web App (PWA) ausliefern und zusätzlich über Capacitor eine installationstaugliche Anwendung erstellen.

Insgesamt lohnt sich der Einsatz moderner Hybrid-Apps überall dort, wo verteilte Mitarbeiter oder Kunden ein einheitliches Bedienerlebnis der Applikation erwarten und gleichzeitig Hardwarefunktionen wie Kamera, GPS oder NFC benötigt werden. Unternehmen, die auf schnelle Iterationen und reduzierten Wartungsaufwand setzen, profitieren dabei vom geringen Overhead und der hohen Anpassbarkeit dieser Architektur.

Ionic mit Angular, React oder Vue.js: Am Ende ist es dennoch JavaScript

Ionic bietet die Möglichkeit, auf unterschiedlichen Frameworks aufzusetzen, was Teams mit verschiedenen Erfahrungs- und Technologiebereichen entgegenkommt. Grundlegend handelt es sich bei Ionic um ein Set von UI-Komponenten und Dienstprogrammen, die Webstandards wie HTML, CSS und JavaScript nutzen. Das ermöglicht die Gestaltung ansprechender Oberflächen, die sich automatisch an verschiedene Plattformen angleichen. Ein großer Pluspunkt besteht darin, dass Ionic bewusst Framework-agnostisch arbeitet: Ob Angular, React oder Vue.js – Entwickler können auf denselben Grundkomponenten aufbauen und müssen ihr gewohntes Ökosystem nicht verlassen.

Gerade für Desktop-Entwickler, die aus .NET oder anderen stark typisierten Umgebungen kommen, bietet sich oft Angular als Unterbau an. Der Grund liegt in seiner klaren Architektur, dem strukturierten Modular-System und der engen Integration mit TypeScript. Viele schätzen auch den inkludierten Router, Services und Dependency Injection, was eine gewisse Nähe zu Enterprise-Konzepten vermittelt. Angular tendiert zu einem Opinionated-Framework-Charakter, was für Entwickler nützlich ist, die klare Vorgaben und ein einheitliches Projektgerüst bevorzugen.

Entwickler, die sich mehr in der JavaScript- oder PHP-Welt bewegen, setzen hingegen oft auf React oder Vue.js. React besticht durch seine minimalistische Grundidee und die große Freiheit beim Zusammensetzen von Bibliotheken. Wer dynamisch kleinere Projekte aufbaut oder sich an das Ökosystem großer Web-Communities anlehnt, findet hier schnelle Ergebnisse. Vue.js wird ebenso geschätzt, weil es eine niedrige Einstiegshürde besitzt und viele Konzepte klar verständlich aufbereitet. So entsteht eine enge Bindung zur reinen JavaScript-Welt, in der man HTML-Templates und CSS flexibel kombinieren kann.

Diese Vielfalt hilft bei der Personalsuche und beim Einstieg in Hybrid-Anwendungen, weil bereits vorhandene Fähigkeiten genutzt werden. Ein Team, das sich mit Angular CLI und TypeScript auskennt, kann rasch produktive Ionic-Apps entwickeln. Ein anderes Team mit Fokus auf React oder Vue.js muss kein komplett neues Paradigma lernen, sondern bindet lediglich die Ionic-Komponenten in das eigene, vertraute Projektsetup ein. Unabhängig davon, welches Framework letztendlich gewählt wird: Ionic steuert umfassende UI-Bausteine, Navigationsmuster und einheitliches Styling bei, sodass die Anwendung auf jeder Plattform vertraut wirkt. Das schont Ressourcen, weil keine separaten Frameworks pro Endgerät gepflegt werden müssen.

Capacitor: ein Wrapper für moderne Cross-Plattform-Anwendungen

Capacitor [2] bildet die zentrale Brücke zwischen Webtechnologien und nativen Plattformfunktionen. Das Projekt stammt von den Machern von Ionic und verfolgt den Ansatz, jede Plattform mit ihrem eigenen nativen Teil zu unterstützen, während Entwickler die Schnittstellen über TypeScript beziehungsweise JavaScript nutzen. So entsteht ein plattformagnostisches System, das Anpassungen in Java oder Swift vermeidet und stattdessen auf gemeinsame Funktionsaufrufe setzt. Die offizielle Dokumentation [3] liefert dazu umfassende Informationen.

Ein Beispiel ist die Standortabfrage, die mithilfe des Geolocation-Plug-ins in wenigen Zeilen Code realisiert wird. Für den Browser greift Capacitor dabei auf das standardisierte HTML5-Geolocation-API zurück, während auf iOS und Android die jeweiligen nativen Standortdienste genutzt werden.

```
import { Geolocation } from '@capacitor/geolocation';

async function getCurrentLocation() {
  const coordinates =
    await Geolocation.getCurrentPosition();
  console.log('Latitude:', coordinates.coords.latitude);
  console.log('Longitude:',
    coordinates.coords.longitude);
}
```

Der oben gezeigte Code zieht sich die Koordinaten direkt aus den jeweiligen Plattform-APIs, ohne dass im Projekt zwischen Browser, iOS oder Android unterschieden werden muss. Bei Desktop-Browsern kommt die HTML5-Geolocation zum Einsatz, die vom Nutzer die Berechtigung abfragt und dann Positionsdaten zurückgibt. Auf iOS oder Android kümmert sich Capacitor darum, die jeweiligen nativen Funktionen anzusprechen und Zugriffsrechte einzuholen. Das erspart Entwicklern das Anlegen verschiedener Codepfade oder das Erlernen plattformspezifischer Sprachen. Die Aufrufe bleiben in TypeScript und lassen sich damit nahtlos in existierende Webframeworks einbinden.

Ähnlich funktioniert das auch mit weiteren Plug-ins wie Kamera, Push-Benachrichtigungen, NFC oder Barcode-Scanning. Während diese Features im Browser teilweise nur ►

eingeschränkt verfügbar sind, stellt Capacitor sicher, dass die nativen Funktionen bestmöglich genutzt werden. So kann man eine App entwickeln, die unterwegs per Smartphone Kamera und GPS verwendet und zu Hause im Desktop-Browser zumindest grundlegende Funktionen bereitstellt. All das ist zentral in einem einzigen Repository gepflegt, was Änderungen stark vereinfacht.

Der plattformagnostische Ansatz erlaubt somit ein einheitliches Architekturdesign: Webentwickler bleiben im wohlvertrauten JavaScript- oder TypeScript-Umfeld, greifen auf HTML5- und CSS3-Fähigkeiten zurück und profitieren trotzdem von nativen Plattform-Features. Für Fälle, in denen kein fertiges Plug-in bereitsteht, existiert eine gut dokumentierte Anleitung, um eigene Erweiterungen zu bauen. Derartig flexibel können Unternehmen eine Anwendung erstellen, die unterschiedliche Endgeräte abdeckt, schnelle Iterationen erlaubt und doch nur eine zentrale Codebasis benötigt. So sparen Teams Aufwand bei der Wartung und bieten Nutzern zugleich ein passendes, reaktionsschnelles Erlebnis auf jedem Gerät.

Die Diskussion um natives Look and Feel und Barrierefreiheit

Ionic statet Anwendungen automatisch mit angepassten Styles für iOS und Android aus. Das Framework erkennt, auf welcher Plattform die App läuft, und passt etwa die Darstellung von Navigationsleisten und Buttons an. So entsteht ein Look and Feel, das dem typischen Nutzererlebnis der jeweiligen Betriebssysteme nahekommt. Ein Dark- und Light-Theme ist ebenfalls integriert, was viele Anwender mittlerweile als Selbstverständlichkeit erwarten. Wer jedoch ein eigenes Erscheinungsbild bevorzugt, kann jede Komponente von Grund auf anpassen.

Einen wesentlichen Vorteil beim Design liefert HTML in Kombination mit CSS. Diese Kombination stellt das derzeit flexibelste System für die schnelle Entwicklung und Auszeichnung von UIs dar: Jedes Element kann bis auf die tiefste Ebene komplett umgestaltet werden, ohne sich an starre Vorgaben halten zu müssen. Während native Frameworks oft nur bestimmte Stilparameter zulassen oder auf vordefinierte Themes beschränkt sind, ermöglichen HTML und CSS eine freie Definition sämtlicher Oberflächenmerkmale. Das reicht von komplexen Layouts über fein abgestimmte Animationen bis hin zu Markup, das exakt an unternehmensinterne Designvorgaben angepasst werden kann.

Ein Beispiel hierfür wäre eine interne Anwendung, die nicht das typische iOS- oder Android-Layout aufgreift, sondern ein eigenständiges Corporate Design im Fokus hat. Statt einer Material-Design-Optik oder typischer iOS-Buttons sieht jedes UI-Element vom Header bis hin zu Kontextmenüs unternehmensspezifisch aus. Ionic liefert nur die flexible Grundstruktur, während Entwickler mithilfe von HTML und CSS alle Freiheiten bei Farben, Formen und Übergängen haben.

Barrierefreiheit (Accessibility) ist ein weiterer Aspekt, in dem sich HTML-Technologien auszeichnen. ARIA-Attribute, die Nutzern mit Screenreadern wichtige Hinweise geben, gehören zum Standard-Webrepertoire. Ionic generiert solche Informationen in vielen Fällen automatisch, sodass Apps be-

reits ohne Zusatzaufwand barrierefreier werden. Für Organisationen, die auf EU-Regularien zur Barrierefreiheit reagieren, ist das eine deutliche Arbeitserleichterung. Während man in manchen nativen Frameworks zusätzliches Setup benötigt, stehen die Web-Basisfunktionen hier direkt bereit.

Oft wird auch behauptet, Hybrid-Apps fühlten sich nicht so „echt“ an wie native Apps. Diese Kritik stammt jedoch, wie bereits beim Thema Performance erwähnt, aus einer Zeit in der Browserumgebungen noch schwach waren. Inzwischen ermöglichen leistungsfähige WebView-Engines nahezu durchgängig hardwarebeschleunigte Grafik, reaktive Übergänge und schnell ansprechende Touch-Gesten. Hybrid-Apps vereinen damit die Vorteile nativer Plattformanpassung mit den Freiheitsgraden eines vollständig HTML/CSS-basierten UI.

Web versus Cross-Platform-Native: Ein ewiger Streit?

Einige Entwickler greifen anstelle einer reinen nativen Entwicklung dann eher zu Frameworks wie React Native, Xamarin, Blazor/MAUI oder Flutter, weil sie sich davon eine native Performance versprechen. Diese Frameworks bieten zwar ebenfalls eine Cross-Platform-Experience, unterscheiden sich jedoch grundlegend von dem rein webbasierten Ansatz wie bei Ionic/Capacitor.

React Native führt eine JavaScript-Logik aus, nutzt jedoch keine klassische WebView. Stattdessen stellt das Framework eine Brücke (Bridge) bereit, die jeden UI-Befehl aus dem JavaScript-Kontext entgegennimmt und in echte native Komponenten auf Android und iOS umsetzt. Das Resultat ist eine Anwendung, die auf den Endgeräten zu großen Teilen „nativ aussieht“, weil Bedienelemente wie Buttons oder Listen im jeweiligen Betriebssystem-Framework gezeichnet werden. Das geht mit jeweiligen Vor- und Nachteilen einher.

Flutter geht einen Schritt weiter: Die Anwendungen werden mithilfe der Dart-Sprache geschrieben. Dieser Code wird weitgehend in Code für Android oder iOS kompiliert, wodurch keine klassische WebView zum Einsatz kommt und auch keine plattformspezifischen UI-Komponenten verwendet werden. Stattdessen stellt Flutter seine eigene Rendering-Engine (Skia) bereit, die sämtliche UI-Elemente selbst zeichnet. Dies ermöglicht ein sehr einheitliches Look and Feel, unabhängig von der Plattform. Für GPU-intensive Szenarien kann das vorteilhaft sein, da Flutter die Grafikpipeline vollständig kontrolliert. Allerdings entsteht bei tiefergehenden Fehlern oder Systeminkompatibilitäten eine komplexe Fehlersuche, weil sich Entwickler sowohl mit Dart- als auch mit den Plattformbesonderheiten auseinandersetzen müssen.

Xamarin (der Vorgänger von .NET MAUI) setzt stärker auf eine .NET-Basis: Der C#- oder XAML-Code wird in einer Umgebung ausgeführt, die teils just in time (Android) oder ahead of time (iOS), jedoch immer auf Basis von Objective-C/Swift (iOS) oder Java/Kotlin (Android) kompiliert. Native UI-Komponenten können über Xamarin.Forms angesprochen werden, was einerseits Performancevorteile verspricht. Andererseits existiert ein zusätzlicher Abstraktionslayer, der native übersetzte Funktionen aufruft und potenziell Debugging-Prozesse erschwert. Vor allem in großen Projekten kann das

zu verzögerten Fehleranalysen führen, wenn Abstürze nur in der generierten Java- oder Objective-C-Struktur sichtbar werden.

Bei Blazor WebAssembly lässt man eine komplette .NET-Laufzeit im Browser laufen, sodass C#-Code dort ausgeführt wird. Für eine Hybrid-Lösung wie .NET MAUI verwendet man native Projekte, die wiederum Blazor-Elemente einbinden. Das klingt verlockend für Teams mit .NET-Erfahrung, führt jedoch zu einer weiteren Abstraktionsebene: HTML und CSS werden genutzt, aber im Hintergrund laufen .NET-Interpreter oder eine Zwischenschicht über WASM. Die Kommunikation zwischen diesen Schichten funktioniert ebenfalls über JavaScript und SignalR/WebSockets. Außerdem entstehen Fragen zu langfristiger Wartung, wenn Microsofts Frame-

bugging auf der Ebene exakt desselben (interpretierten) Codes geschieht, den man geschrieben hat – ohne Kompilierung/Translation in eine andere native Sprache. Wer in den Entwicklertools einen Fehler sieht, erkennt sofort die zugehörige Codezeile in JavaScript oder im DOM.

Synergieeffekte durch Webtechnologien

Wer sich für einen rein webbasierten Ansatz wie Ionic/Capacitor entscheidet, profitiert davon, dass HTML, CSS und JavaScript mittlerweile zum Standardrepertoire der Softwareentwicklung zählen. Externe Dienstleister, neue Teammitglieder oder interne Weiterbildungen können meist auf vorhandenes Wissen aufbauen: Ob Frontend-Designer, UI/UX-Experten oder Full-Stack-Entwickler, sie alle kennen die gängigen

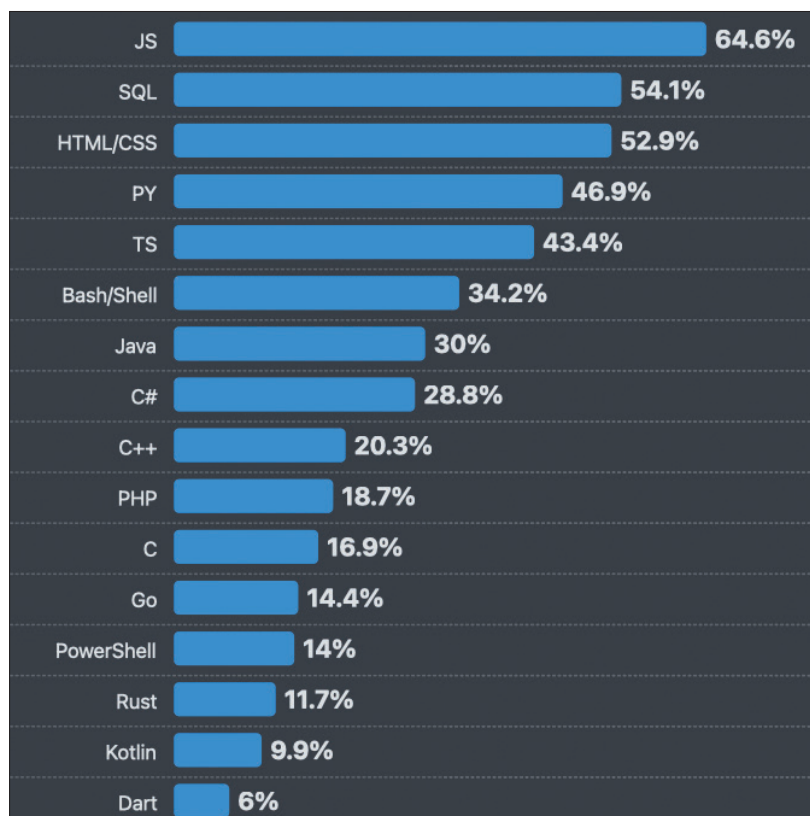
Webstandards. In vielen Unternehmen finden sich ohnehin Entwickler, die mit React, Angular, Vue oder reinen JavaScript-Lösungen vertraut sind. Der Aufwand für die Einarbeitung in eine Hybrid-App-Architektur fällt dadurch geringer aus, als wenn man zusätzlich eine kompakte, aber durchaus komplexe Zwischenschicht wie Flutter, Xamarin oder React Native einführen muss.

Gerade im Bereich UI/UX erleichtert dieser breitere Talentpool die Umsetzung, weil Webdesigner sich bestens mit konfigurierbaren Layouts, Responsive Design und semantischen Strukturen auskennen. Das Feedback- und Iterationsmodell, etwa bei Prototyping-Sprints, läuft oft schneller, wenn alle Beteiligten denselben Technologie-Stack verwenden. Externe Agenturen oder Freelancer, die HTML und CSS beherrschen, können kurzfristig eingebunden werden, ohne mühsam in eine spezielle Framework-Syntax eingewiesen zu werden. Zudem ist die Anzahl vorhandener Bibliotheken und Komponenten im JavaScript-Ökosystem gigantisch (vergleiche Bild 2): Ob Data Grids, Charts, Animationen oder Formvalidierungen – ein Großteil davon funktioniert „out of the box“ im Web.

Daneben genießen Teams, die auf WebStacks bauen, langfristige Stabilität: Browser-APIs werden rückwärtskompatibel oder über Polyfills abgesichert. Sogar bei wechselnden

Framework-Trends innerhalb der Webwelt kann ein großer Teil des Codes fortbestehen, weil zugrunde liegendes HTML, CSS und JavaScript selten grundlegend verändert werden. Wer also während eines Projekts von Angular zu Vue wechseln möchte, kann UI-Bausteine oft ohne Totalneubau übernehmen. Das ist bei Frameworks, die zu nativen Komponenten übersetzen, deutlich aufwendiger.

Ebenso spielt die Rekrutierung eine Rolle: Auf dem Arbeitsmarkt sind Fachkräfte mit Web-Know-how meist leichter zu finden als solche, die auf .NET, Dart, Swift oder Kotlin spezialisiert sind. Für ein Unternehmen kann dies bedeuten, dass Projekte schneller wachsen, neue Mitarbeiter ►



Stackoverflow Developer Survey 2024: Populärste Programmiersprachen von professionellen Entwicklern und Entwicklerinnen (Bild 2)

work-Politik wechselt oder bestimmte Pakete auslaufen. Bei MAUI geht man einen Schritt weiter und führt wie bei Xamarin einen teilnativen Code aus, welcher wiederum auf einer plattformspezifischen .NET Runtime basiert. Sieht man sich hier diverse Architektur-Schaubilder zu den beiden Frameworks an, fällt auf, dass diese für Entwickler auf den ersten Blick sehr attraktiv sind, wenn diese im gewohnten (und guten) .NET-Umfeld bleiben möchten, jedoch einiges an technischem Overhead liefern.

Im Gegensatz dazu verlässt sich Ionic auf HTML, CSS und JavaScript (oder TypeScript) in einer WebView, welche im Zielbetriebssystem integriert ist. Das hat den Vorteil, dass De-

unkompliziert an Bord kommen und personelle Risiken bei Fluktuation sinken. Auch in puncto Wissensmanagement trägt der Webansatz Früchte: Dokumentation, Tutorials und Community-Ressourcen stehen in enormer Vielfalt zur Verfügung. Gerade bei knappen Timelines oder sich ändernden Anforderungen ist das ein klares Argument, um Entwicklungszyklen kurz zu halten und flexibel auf Kundenfeedback reagieren zu können.

Live-Updates der App

Mit Ionic und den AppFlow-Services lassen sich Anwendungen nahezu „on the fly“ aktualisieren. Technisch betrachtet erfolgt dies, indem das Verzeichnis, in dem HTML-, CSS- und JavaScript-Dateien liegen, einfach ausgetauscht wird. Auf diese Weise können Bugfixes oder kleinere Updates sofort an alle Endgeräte ausgeliefert werden – ein großer Vorteil insbesondere in kritischen Bereichen wie Banking oder eHealth, wo Sicherheit und schnelle Reaktionszeiten essenziell sind. Zwar befinden sich solche Aktualisierungen in einer Grauzone, etwa im Hinblick auf Apples App-Richtlinien, doch viele Unternehmen nutzen diesen Ansatz, um unmittelbar auf erkannte Fehler zu reagieren, ohne langwierige Freigabeprozesse im App Store durchlaufen zu müssen.

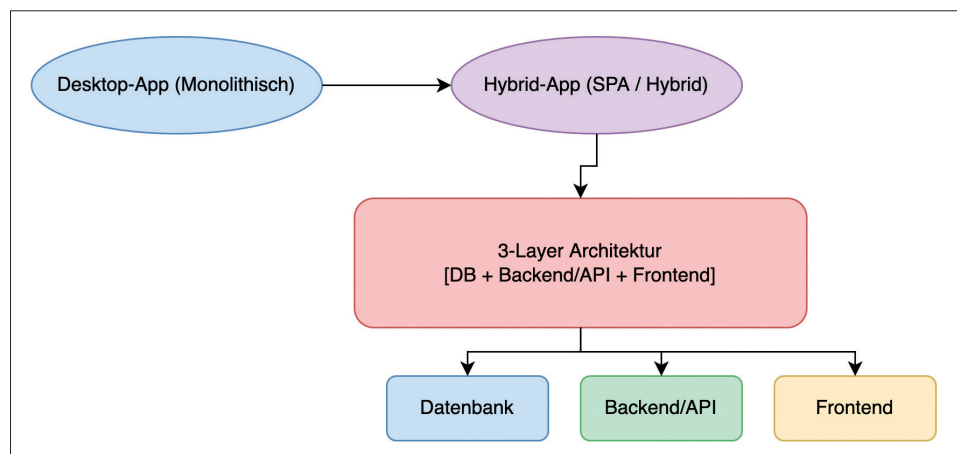
Daher bietet die Kombination aus Ionic und Capacitor einen weiteren Vorteil, den sie mit SPAs und Webapplikationen gemeinsam hat. Gegenüber herkömmlichen Desktop-Anwendungen besteht die Möglichkeit zu kontinuierlichen, schnellen Updates. Desktop-Apps erfordern oft einen erneuten Roll-out oder eine Neuinstallation, wobei zwingend ein Neustart nötig ist, um Änderungen wirksam zu machen. Diese Prozedur verzögert Reaktionszeiten und beeinträchtigt die Nutzererfahrung. Webbasierte Lösungen ermöglichen dagegen, dass Aktualisierungen nahtlos im Hintergrund durchgeführt werden – der gleiche Code, der geschrieben wurde, läuft unverändert in modernen WebViews. So profitieren Unternehmen von einer deutlich agileren Wartungs- und Update-Strategie, die sich in einem konsistenten und stets aktuellen Nutzererlebnis niederschlägt.

Die Migration von Desktop-Anwendungen ist die große Herausforderung

Der Übergang von klassischen Desktop-Anwendungen zu hybriden, webbasierten Lösungen bringt nicht nur gestalterische Herausforderungen mit sich, sondern erfordert auch eine grundlegende Neustrukturierung der Anwendungsarchitektur. Während monolithische Desktop-Apps oft sämtliche Geschäftslogik lokal abbildeten, ist bei SPAs und Hybrid-Apps eine klare Trennung in eine 3-Layer-Architektur not-

wendig: Datenbank, Backend/API und Frontend. Viele Prozesse, die früher direkt auf dem Desktop liefen, müssen nun in den Server verlagert werden. Das bedeutet, dass Funktionen, die in der alten Umgebung nahezu sofort reagierten, nun über Netzwerkanfragen abgewickelt werden. Dieser zusätzliche Kommunikationslayer kann zu erhöhten Latenzen und komplexeren Datenflüssen führen, die es sorgfältig zu steuern gilt (vergleiche Bild 3).

Ein konkretes Beispiel könnte eine Logistik- und Dispositions-Anwendung liefern, die ursprünglich alle relevanten Informationen eines Versandvorgangs – von Lagerbeständen über Routeninformationen bis hin zu Echtzeit-Statusupdates – in einer einzigen, umfassenden Desktop-Oberfläche darstellte. In der neuen Hybrid-Version müssen diese Daten



Migrationspfad einer monolithischen Desktop-App hin zu einer Hybrid-App (Bild 3)

in mehrere Ebenen oder Tabs aufgeteilt werden, um sie auf mobilen Geräten und Tablets übersichtlich darzustellen. Während auf dem Desktop eine zentrale Logik direkt ausgeführt wurde, übernimmt nun ein dediziertes Backend die Aufbereitung, Aggregation und Validierung der Daten. Dies erfordert umfangreiche API-Schnittstellen, Caching-Strategien und Synchronisationsmechanismen, um eine konsistente und performante Nutzererfahrung zu gewährleisten.

Der Mehraufwand, der mit der Verlagerung lokaler Logik in das Backend einhergeht, kann zu einer erhöhten Komplexität in der Gesamtarchitektur führen. Unternehmen müssen nun sicherstellen, dass das Backend stabil, skalierbar und sicher ist, um den reibungslosen Betrieb der Anwendung zu garantieren. Gleichzeitig steigt der Aufwand für die Abstimmung zwischen Frontend und Backend, da jede Änderung in der Geschäftslogik potenziell in mehreren Schichten berücksichtigt werden muss. Die Umstellung erfordert also nicht nur ein Umdenken in der UI-Gestaltung, sondern auch einen grundlegenden Architekturwechsel, der in vielen Fällen zusätzliche Investitionen in Infrastruktur und Entwicklungsressourcen verlangt.

Diese Herausforderungen sind für hochkomplexe, fachlich anspruchsvolle Anwendungen besonders kritisch. Eine fehlerhafte oder gar zu langsame Synchronisation mit der aus-

gelagerten Logik kann zu schlechter Nutzerakzeptanz führen, gerade wenn Anwender an die unmittelbare Reaktionsfähigkeit der alten Desktop-Lösung gewöhnt sind. Ein behutsam gewählter Migrationspfad sowie eine enge Abstimmung zwischen den beteiligten Teams sind daher unerlässlich, um die Vorteile moderner Hybrid-Apps optimal nutzen zu können, ohne dass dabei essenzielle Funktionen oder die Performance leiden.

Sind also Hybrid-Apps doch nichts für den Desktop?

Monolithische Desktop-Anwendungen lassen sich prinzipiell mithilfe von Electron [4] in Hybrid-Apps überführen. Electron nutzt wie Capacitor Webtechnologien wie HTML, CSS und JavaScript, um Desktop-Anwendungen zu erstellen – wie es beispielhaft bei Visual Studio Code eindrucksvoll seit Jahren eingesetzt wird. Im Prinzip ist Electron das Pendant zu Capacitor, jedoch für Desktop-Oberflächen. Der Ansatz, den gesamten bestehenden Code in eine WebView zu integrieren, ermöglicht es, bekannte Logik und UIs, die zum Beispiel auch mit Ionic entwickelt worden sind, beizubehalten und gleichzeitig von den Vorteilen plattformübergreifender Distribution zu profitieren.

Da wir hier wie bei Capacitor den vollen Umfang des Desktop-Betriebssystems in Anspruch nehmen können, sind damit auch monolithische Applikationen denkbar. Dies bedeutet aber auch die Gefahr, dass zentrale Geschäftslogik und UI-Komponenten weiterhin in einer einzigen, schwerfälligen Codebasis verankert sind. Während durch den Einsatz von Electron der direkte Zugriff auf den bestehenden Quellcode und das Debugging in den bekannten Entwicklertools ermöglicht wird, bleibt die inhärente Problematik einer monolithischen Architektur bestehen. Komplexe Anwendungen, bei denen alle Informationen und Funktionen auf einem Bildschirm vereint waren, müssen nun in mehrere Ebenen oder Tabs aufgeteilt werden.

Der Einsatz von Electron birgt also das Potenzial, bestehende Systeme schnell auf hybride Plattformen zu übertragen, ohne dass Entwickler ihre vertraute Systemarchitektur komplett neu aufbauen müssen. Gleichzeitig müssen aber die bestehenden monolithischen Schwächen kritisch hinterfragt werden und sollten aus meiner Erfahrung heraus im Zuge eines solchen Prozesses anderweitig aufgebaut werden. Nur so kann eine nachhaltige und zukunftssichere Lösung entstehen, die den Ansprüchen moderner, plattformübergreifender Anwendungen gerecht wird.

Ein strategischer Ausblick

Aus unserer Erfahrung sind Single Page Applications und Hybrid-Apps, die auf Webtechnologien basieren, die Zukunft der Frontend-Entwicklung. Zahlreiche Schulungen und Beratungen mit Kunden haben gezeigt, dass Projekte, die einst mit großer Euphorie mittels MAUI oder Flutter umgesetzt wurden, nach etwa zwei Jahren im Echtbetrieb mit Schwächen konfrontiert wurden, die den weiteren Einsatz infrage stellen. Im Vergleich dazu bieten PWAs oder reine Webapplikationen mit zum Beispiel Ionic eine stabile Basis, da sie

schlicht auf dem offenen Webstandard basieren – sie erlauben es uns als Entwickler, später nahtlos in mobile Apps zu migrieren, selbst wenn dies aktuell nicht explizit gefordert ist.

Die Anforderungen der Nutzer ändern sich kontinuierlich. Letztlich spielt es kaum eine Rolle, ob eine Anwendung auf .NET oder auf HTML, CSS und JavaScript basiert – entscheidend ist, dass sie moderne, intuitive und reaktionsschnelle Erlebnisse liefert, wie sie die Nutzer aus dem privaten Umfeld kennen. Dabei hängt der Erfolg einer Anwendung maßgeblich davon ab, wie praktikabel die UI/UX-Anforderungen umgesetzt werden. Ein Hybrid-Ansatz garantiert nicht automatisch ein optimales Nutzererlebnis, sondern erfordert eine sorgfältige, zielgerichtete Gestaltung, die komplexe Funktionen übersichtlich in mehreren Ebenen darstellt.

Der wesentliche Vorteil dieses Technologie-Stacks liegt in der Flexibilität für Entwicklungsteams und in der strategischen Ausrichtung der Entwicklung. Unternehmen profitieren von kürzeren Entwicklungszyklen, schnellerem Feedback und einer konsistenten plattformübergreifenden Nutzererfahrung. Dieser Ansatz ermöglicht es, auf bewährte Webtechnologien zurückzugreifen, deren umfangreiches Ökosystem und Community eine reibungslose Einarbeitung und kontinuierliche Weiterentwicklung sicherstellen – Voraussetzungen, die für den nachhaltigen Erfolg moderner Softwarelösungen unerlässlich sind.

In den letzten Jahren haben wir viele Teams bei der Umsetzung ihrer Softwareprojekte unterstützt und dabei auch viele klassische Applikationen mit WPF oder ASP.NET begleitet und umgesetzt. Sicher wird es immer Teams geben, die interne Tools entwickeln und mit diesen auf den bekannten Technologie-Stack schnellstmöglich eine praktikable Lösung bieten. Jedoch sehen wir auch, dass die Unternehmen, die davon abkommen, die typischen Teams sind, die einen entsprechenden kulturellen Wandel im Unternehmen vorantreiben. Sei es, um mehr als Dienstleister/Service-Anbieter für die Kunden zu fungieren oder um interessanter für die eigenen Mitarbeiter zu werden. Diese Teams sind es, die vor großen Refactoring- oder Neuentwicklungs-Projekten stehen und dabei einen genauen Blick auf die hier vorgestellten Technologien werfen sollten. ■

[1] Ionic-Case-Study,

www.dotnetpro.de/SL2504-05CrossPlattformSPA1

[2] Capacitor, <https://capacitorjs.com>

[3] Dokumentation zu Capacitor, <https://capacitorjs.com/docs>

[4] Electron, <https://www.electronjs.org>



Patrick Schnell

ist Geschäftsführer von schnell.digital, das auf Entwicklung und Architektur von Digitalisierungsprojekten und Beratung spezialisiert ist. Dabei nutzt er unterschiedlichste Technologien wie .NET, Angular, TypeScript und Node.js.

<https://schnell.digital>

dnpCode

A2504-05CrossPlattformSPA

SICHERHEITSLÜCKEN IN SPAS MIT OWASP ZAP

Keine Wellness für Sicherheitslücken

Mit OWASP ZAP lassen sich Webanwendungen automatisch auf Sicherheitslücken scannen. Das funktioniert auch bei Single Page Applications (SPAs).

Vor etwa einem Jahr haben wir in der dotnetpro [1][2] das Tool OWASP ZAP [3] vorgestellt, den Open Web Application Security Project Zed Attack Proxy. Dabei haben wir uns bei der Veranschaulichung seines Leistungsspektrums auf „herkömmliche“ Webanwendungen beschränkt.

Diesmal gehen wir einen Schritt weiter und schauen uns SPAs an, Single Page Applications. Das erfordert eine leicht angepasste Vorgehensweise, um potenziellen Sicherheitslücken auf die Spur zu kommen.

Zur Erinnerung: Im automatischen Modus versucht ZAP zunächst, alle URLs der Anwendung zu ermitteln. Dazu wird das HTML-Markup rekursiv nach Links und Formularen untersucht. Im zweiten Schritt wird die Liste der Endpunkt dann mit vorbereiteten Payloads attackiert – etwa typische XSS-

oder SQL-Injection-Angriffe. Das ist bei einer SPA etwas anders gelagert. Beispielsweise gestaltet sich hier das Cross-Site Scripting anders, denn Serverkommunikation funktioniert meist nur über ein API, dessen Rückgabe nicht direkt im Browser angezeigt wird. Dafür gibt es neue Bedrohungsszenarien, beispielsweise DOM-basiertes XSS, bei dem der Angriff ohne Einbindung eines Backends vollständig im Client stattfindet.

Viel einschneidender jedoch: Das Crawlen der Webanwendung ist schwieriger. Eine typische SPA lädt einmalig die Oberfläche, danach kommen APIs zum Zuge. Deren URLs befinden sich in der Regel nicht im HTML, sondern im JavaScript-Code. ZAP hat hierfür eine Lösung gefunden, die wir im Folgenden schildern werden.

Vorbereitungen

Um die Funktionsweise von ZAP bei Single Page Applications nachvollziehen zu können, benötigen wir zunächst eine Anwendung. Hier könnten wir natürlich zu einer beliebigen Blazor-App greifen, aber so richtig lohnt sich ein Einsatz erst bei einer Website, die ein nennenswertes Ausmaß an Sicherheitslücken aufweist.

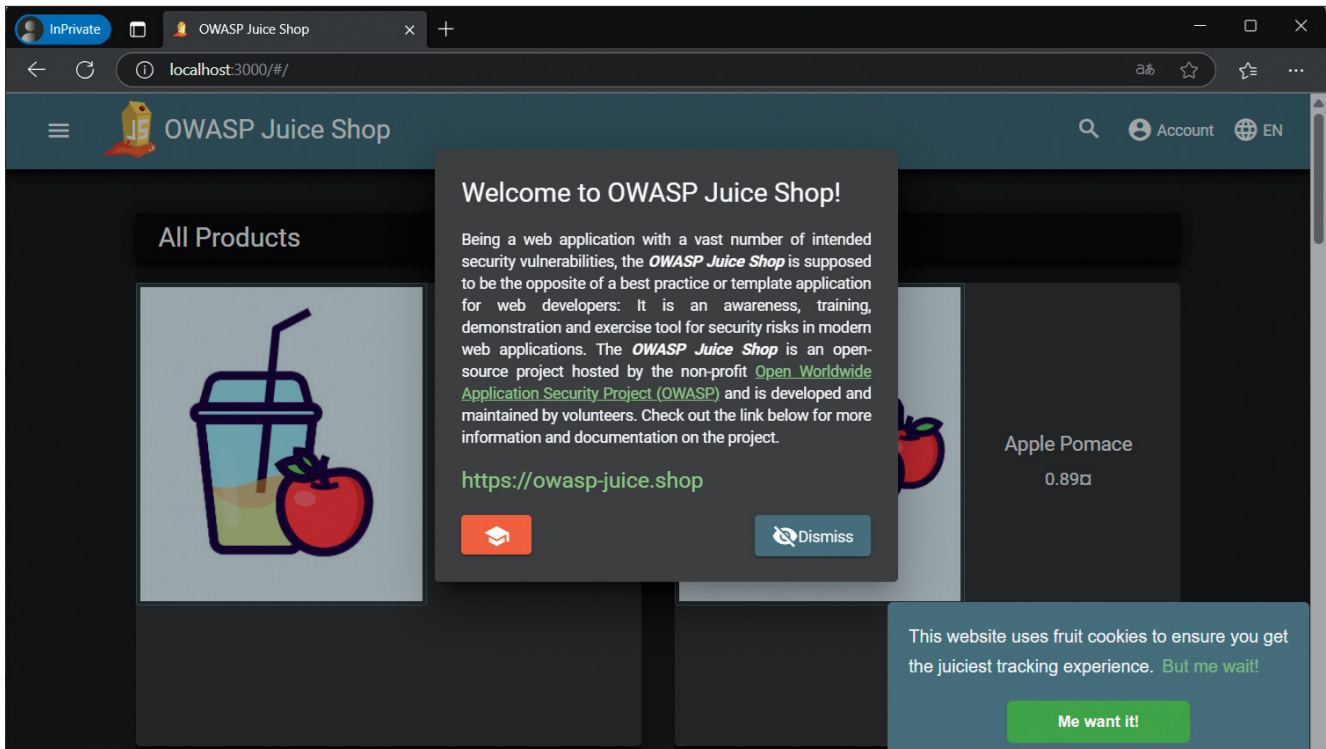
Im Rahmen der OWASP gibt hierzu ein eigenes Projekt, den Juice Shop [4]. Das ist praktischerweise eine Single Page Application auf JavaScript-Basis, zwar nicht mit Blazor umgesetzt, aber es geht ja um die prinzipielle Funktionsweise. Vom Juice Shop selbst gibt es sogar eine Live-Instanz [5], die regelmäßig wieder zurückgesetzt wird (das Hosten einer unsicheren An-

```
>docker pull bkimminich/juice-shop
Using default tag: latest
latest: Pulling from bkimminich/juice-shop
1c56d6035a42: Pull complete
e33bce57de28: Pull complete
473d8557b1b2: Pull complete
b6824ed73363: Pull complete
7c12895b777b: Pull complete
33e068de2649: Pull complete
5664b15f108b: Pull complete
27be814a09eb: Pull complete
4aa0ea1413d3: Pull complete
9ef7d74bdfdf: Pull complete
9112d77ee5b1: Pull complete
83f8d4690e1f: Pull complete
a4ba90834fb4: Pull complete
df368711b362: Pull complete
e89169bec965: Pull complete
7f3501c931c2: Pull complete
88934a1bc18c: Pull complete
e5035db4cc0a: Pull complete
716cf78d355f: Pull complete
baab02ec530b: Pull complete
9086d21455f5: Pull complete
Digest: sha256:4846e9496e6afe03d5dd56fee1193050e31a56a50fc9bd9bae3afaf4052d4b8
Status: Downloaded newer image for bkimminich/juice-shop:latest
docker.io/bkimminich/juice-shop:latest

What's next:
View a summary of image vulnerabilities and recommendations → docker scout quickview bkimminich/juice-shop

>docker run --rm -p 3000:3000 bkimminich/juice-shop
info: Detected Node.js version v20.17.0 (OK)
info: Detected OS linux (OK)
info: Detected CPU x64 (OK)
info: Configuration default validated (OK)
info: Entity models 19 of 19 are initialized (OK)
info: Required file server.js is present (OK)
info: Required file index.html is present (OK)
info: Required file styles.css is present (OK)
info: Required file main.js is present (OK)
info: Required file polyfills.js is present (OK)
info: Required file runtime.js is present (OK)
info: Required file vendor.js is present (OK)
info: Port 3000 is available (OK)
info: Chatbot training data botDefaultTrainingData.json validated (OK)
info: Domain https://www.alchemy.com/ is reachable (OK)
info: Server listening on port 3000
```

Ab vom Dock(er): Juice Shop wird in einem Container gestartet (Bild 1)



Saftige Sicherheitslücken: Der Juice Shop im Browser (Bild 2)

wendung ist ein gewisses Risiko). Allerdings ist es keine gute Idee, mit ZAP gegen diesen Server zu testen – Sie erzeugen damit potenziell große Last bei einer fremden Webanwendung. Besser ist es, selbst eine Instanz von Juice Shop zu hosten und diese dann ohne schlechtes Gewissen einem Audit zu unterziehen.

Die Dokumentation von Juice Shop bietet einige Optionen hierzu an [6]; wir entscheiden uns für den Einsatz von Docker, aber es gäbe auch die Möglichkeit, die Anwendung selbst zu kompilieren und laufen zu lassen (*npm* genügt). Die folgenden Kommandos sind erforderlich, sofern Docker lokal installiert ist (siehe Bild 1):

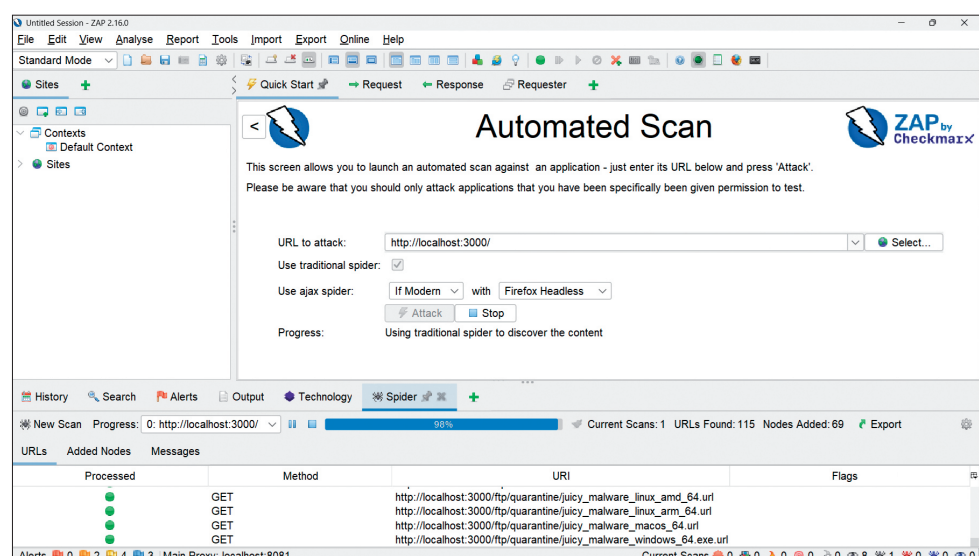
```
docker pull
  bkimminich/juice-shop
docker run --rm
  -p 127.0.0.1:3000:3000
  bkimminich/juice-shop
```

Die Angabe *3000:3000* deutet auf den verwendeten Browserport hin: Über *http://localhost:3000* gelangen Sie auf der Hostmaschine zur Anwendung (Bild 2).

Erste Angriffswelle

Die ersten Schritte unternimmt ZAP auch bei SPAs auf Wunsch ganz automatisch, wobei es natürlich auch ein wenig auf

die SPA ankommt, wie gut das funktioniert. Im *Automated Scan* muss *Ajax Spider* aktiviert werden. Das ist der Crawler, der für „Ajax-Anwendungen“ – die frühere Bezeichnung für Websites, die per JavaScript Daten vom Server holen – zuständig ist. Die Einstellung *If Modern* verlässt sich ein wenig auf die Erkennungslogik von ZAP: Ist es eine „moderne“ Anwendung, kommt Ajax Spider zum Einsatz. Dazu wird ein Webbrowser im Headless-Modus gestartet, also ohne sichtbare Benutzeroberfläche. In Bild 3 sehen Sie den kompletten Dialog. Nach dem Startschuss für den Scan lädt dann ZAP die Webanwendung und prüft, welche URLs im Hintergrund ►



Nichts für Arachnophobiker: „Ajax Spider“ crawlt Single Page Applications (Bild 3)

Processed	ID	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Header	Size Resp. Body
●	185	2/11/25, 9:29:31 AM	GET	http://localhost:3000/assets/public/favicon...	200	OK	15...	456 bytes	15,086 bytes
●	186	2/11/25, 9:29:32 AM	GET	http://localhost:3000/socket.io/?EIO=4&tr...	200	OK	16...	230 bytes	96 bytes
●	188	2/11/25, 9:29:32 AM	GET	http://localhost:3000/rest/admin/applicatio...	200	OK	17...	389 bytes	21,532 bytes
●	191	2/11/25, 9:29:32 AM	GET	http://localhost:3000/assets/i18n/en.json	200	OK	17...	475 bytes	32,311 bytes
●	193	2/11/25, 9:29:32 AM	GET	http://localhost:3000/rest/admin/applicatio...	200	OK	14...	384 bytes	20 bytes
●	194	2/11/25, 9:29:32 AM	GET	http://localhost:3000/rest/admin/applicatio...	304	Not Modif...	16...	306 bytes	0 bytes

Spinnenblick: Einige der von ZAP ermittelten URLs von Juice Shop (Bild 4)

abgefragt werden. Im unteren Bereich von Bild 3 ist zu sehen, wie die Liste Eintrag für Eintrag aufgebaut wird.

Während des Scans werden auch sukzessive die URLs erkannt, die die Anwendung nachlädt. Bild 4 zeigt einige Details während der Untersuchung des Juice Shop. Dort erkennen Sie unter anderem:

- das statische Favicon,
- die Initiierung einer Verbindung über Socket.IO (was später in eine WebSocket-Verbindung münden wird),
- diverse REST-Schnittstellen (Pfadbestandteil *rest*)
- und eine JSON-Datei mit Lokalisierungsressourcen.

Die so ermittelte URL-Liste ist hilfreicher, als es vielleicht im ersten Moment den Anschein hat. So stehen für das Audit vie-

APIs oder auch Tracking-Dienste. Auch diese Systeme können im Rahmen einer Sicherheitsüberprüfung mit herangezogen werden.

Doch zurück zum OWASP-Tool: Im nächsten Schritt schaltet ZAP endgültig in den Angriffsmodus. Nacheinander werden zwei weitere Tabs geöffnet: Bei *WebSockets* wird der ermittelte WebSockets-Endpunkt mit Requests beglückt; in *Active Scan* geht es schließlich den im ersten Schritt ermittelten Endpunkten an den Kragen, ähnlich wie beim Scan einer herkömmlichen Webanwendung (Bild 5). Kleines Schmankerl am Rande: ZAP sucht unter anderem nach *_framework/blazor.boot.json*, in der Hoffnung – wäre Juice Shop eine Blazor-App –, auf interessante Metadaten zu stoßen. Aber ZAP spekuliert auch darauf, dass andere relevante Dateien aus Ver-

ID	Req. Timestamp	Resp. Timestamp	Method	URL	Code	Reason	RTT
1,228	2/11/25, 9:35:10 AM	2/11/25, 9:35:10 AM	GET	http://localhost:3000/_framework/blazor.boot.json	200	OK	27 ms
1,229	2/11/25, 9:35:10 AM	2/11/25, 9:35:10 AM	GET	http://localhost:3000/.hg	200	OK	29 ms
1,231	2/11/25, 9:35:10 AM	2/11/25, 9:35:10 AM	GET	http://localhost:3000/.bzl	200	OK	22 ms
1,233	2/11/25, 9:35:10 AM	2/11/25, 9:35:10 AM	GET	http://localhost:3000/_darc	200	OK	28 ms
1,235	2/11/25, 9:35:10 AM	2/11/25, 9:35:10 AM	GET	http://localhost:3000/BitKeeper	200	OK	34 ms

Suche nach Reliquien: ZAP probiert diverse bekannte Dateinamen (Bild 5)

le API-Endpunkte zur Verfügung, die dann entsprechend auf Sicherheitsmängel überprüft werden können (beispielsweise angelehnt an die OWASP API Security Top Ten (vergleiche [7] und [8])). Die von der Anwendung verwendeten Ressourcen sind ebenfalls interessant, etwa hinsichtlich der Version eingesetzter Bibliotheken: Sind alte, unsichere Releases mit dabei? Und möglicherweise offenbaren sich weitere Server, die von der Anwendung verwendet werden, sei es für statische Daten oder für funktionale Aspekte wie etwa weitere

sehen auf dem Produktivsystem gelandet sind – *.hg* etwa kommt vom Git-Konkurrenten Mercurial.

Auf diese Art und Weise können wir bereits eine Reihe von mehr oder minder schwerwiegenden Problemen im Juice Shop identifizieren. Beim Test etwa wurden sechs Lücken vom Schweregrad *Medium*, achtmal *Low* und elfmal vom Level *Information* gefunden, aber kein einziges Mal ein *High Priority Alert*. Es wäre doch arg verwunderlich, wenn die App keine kritischen Sicherheitslücken aufweisen würde! Span-

Alerts (27) - SQL Injection - SQLite (2) - GET: http://localhost:3000/rest/products/s - GET: http://localhost:3000/api/Challenges - Content Security Policy (CSP) Header Not Se - Cross-Domain Misconfiguration (102) - Missing Anti-clickjacking Header (29) - Session ID in URL Rewrite (115) - Vulnerable JS Library	Other Info: The query time is controllable using parameter value [case randomblob(1000000) when not null then 1 else 1 end], which caused the request to take [134] milliseconds, parameter value [case randomblob(10000000) when not null then 1 else 1 end], which caused the request to take [849] milliseconds, when the original unmodified query with value [Score Board] took [96] milliseconds. Solution: Do not trust client side input, even if there is client side validation in place. In general, type check all data on the server side. If the application uses JDBC, use PreparedStatement or CallableStatement, with parameters passed by '?'
---	---

Langsam oder zu langsam? ZAP hat eine mögliche SQL Injection gefunden (Bild 6)

nenderweise ändern sich die Werte bei einem erneuten Durchlauf tatsächlich leicht, beispielsweise gibt es jetzt eine kritische Lücke: SQL Injection in zwei der API-Endpunkte. Ein solcher Scan ist also nicht notwendigerweise deterministisch, weil auch ZAP in der Überprüfung etwas variieren kann. Aber ist das nun schon das Ende der Fahnenstange?

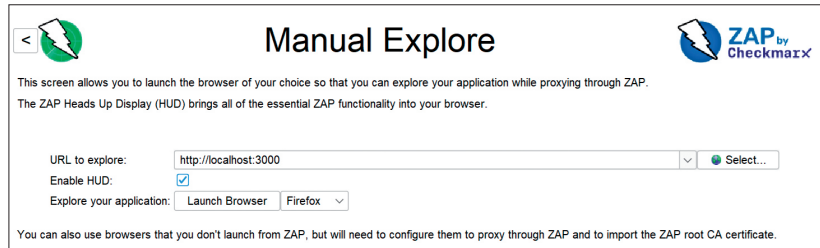
In der Tat entspricht unser Beispiel auch dem Vorgehen in der Praxis: Automatisierte Tools liefern zwar einige Anhaltspunkte, darunter aber auch viele Fehlmeldungen (*false positives*). Außerdem werden bei Weitem nicht alle Lücken aufgespürt. Wichtige Grundlagen für eine Sicherheitsüberprüfung sind deshalb die genaue Kenntnis und ein gutes Verständnis für Aufbau und Funktionsweise einer Webanwendung. Damit lässt sich gezielt nach Lücken fahnden. Dennoch kann uns auch ZAP durch manuelle Eingriffe weiterhin gute Dienste leisten.

Manuelle Intervention

Werfen wir zunächst einen Blick auf die vermeintlich erkannte SQL Injection. Zwei *HTTP-GET*-Requests lieferten Auffälligkeiten:

- `http://localhost:3000/rest/products/search?q=%27%28` führte zu einer SQL-Fehlermeldung in der HTTP-Rückgabe. Die URL-codierten Werte repräsentieren einen Apostroph und eine öffnende Klammer. Gerade ersteres Zeichen ist ein klassischer Bestandteil durchaus vieler SQL-Injection-Angriffe.
- `http://localhost:3000/api/Challenges/?name=case+randomblob%281000000%29+when+not+null+then+1+else+1+end+` führte zu einer langsameren Antwortzeit vom Server, im Test waren es 134 Millisekunden mit dem speziellen Querystring-Wert, 96 Millisekunden ohne (siehe Bild 6). Das im Request eingebettete `randomblob(1000000)` ist eine in SQLite verfügbare Funktion, die eine Zufallszeichenfolge ermittelt – durch den Parameter im Aufruf ist diese eine Million Bytes lang. Der Zeitunterschied liegt innerhalb der Messtoleranz, es ist also noch nicht klar, ob hier eine Sicherheitslücke vorliegt, aber eine Untersuchung lohnt sich natürlich trotzdem.

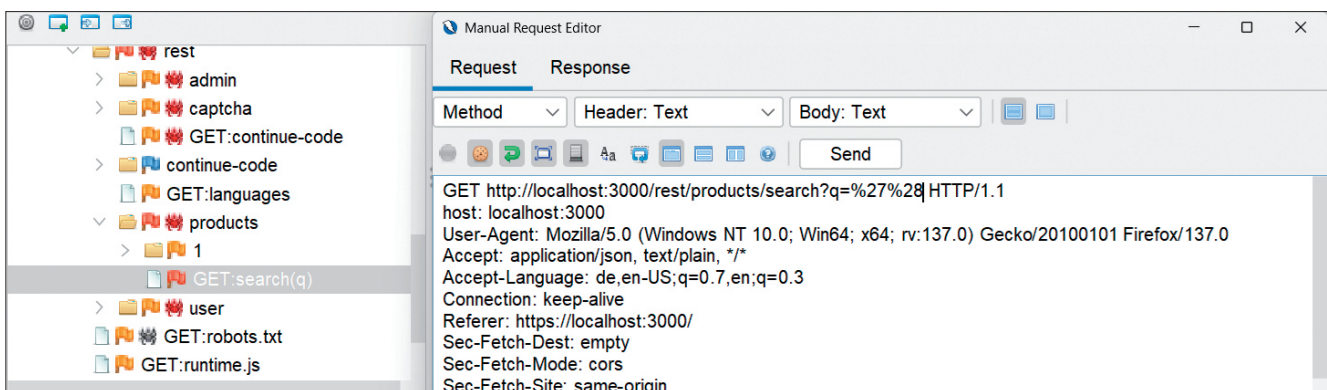
Zurück zu ZAP: Neben dem automatischen Angriffsmodus gibt es auch den Modus *Manual Explore* (Bild 7): Hier öffnet ZAP einen Webbrowser, in den aber ein ZAP-Dienst als Proxy eingebunden ist. In diesem Browser interagieren wir „von Hand“ mit der Webanwendung – und ZAP schneidet mit. Ziel dieses Vorgehens ist es unter anderem, die Liste der von der



Im Angriffsmodus: Der Browser hat einen Rückkanal zu ZAP (Bild 7)

Applikation verwendeten URLs zu erweitern. Manche Ressourcen oder Endpunkte werden erst nach bestimmten Aktionen eingesetzt, etwa Mausklicks oder Tastatureingaben, und sind so für ZAP „unsichtbar“. Auch hier gilt wieder das Mantra: Je besser wir eine Anwendung kennen, desto genauer können wir sie untersuchen.

Nach getaner „manueller Exploration“ der Single Page Application geht es wieder zurück zu ZAP, wo die Liste der HTTP-Anfragen im Optimalfall angewachsen ist. Für jeden dieser Requests lassen sich die zugehörigen Details ansehen. Besonders nützlich: Im Kontextmenü gibt es die Option *Open/Resend in Request Editor*. Hier können die HTTP-Anfragen bearbeitet und dann erneut gesendet werden. In der URL-Liste beispielsweise finden wir den Such-Endpunkt `/rest/products/search`, der bereits als auffällig markiert wurde. Die zuvor verwendete Angriffs-Payload, `'` (URL-codiert als `%27%28`), fügen wir ein (Bild 8) und schicken sie dann zurück an das API. Das Ergebnis aus Bild 9 spricht Bände: Wir erhalten Details über die SQL-Kommandos, mit denen die Anwendung arbeitet. SQL Injection liegt ziemlich sicher vor. Wir könnten diesen Befund nun entweder durch weitere Aufrufe an den Endpunkt oder aber auch durch einen Blick in den Quellcode, falls verfügbar, bekräftigen. ►



Kurskorrektur: Im Request Editor können wir die Payload verfeinern (Bild 8)

Zweite Angriffswelle

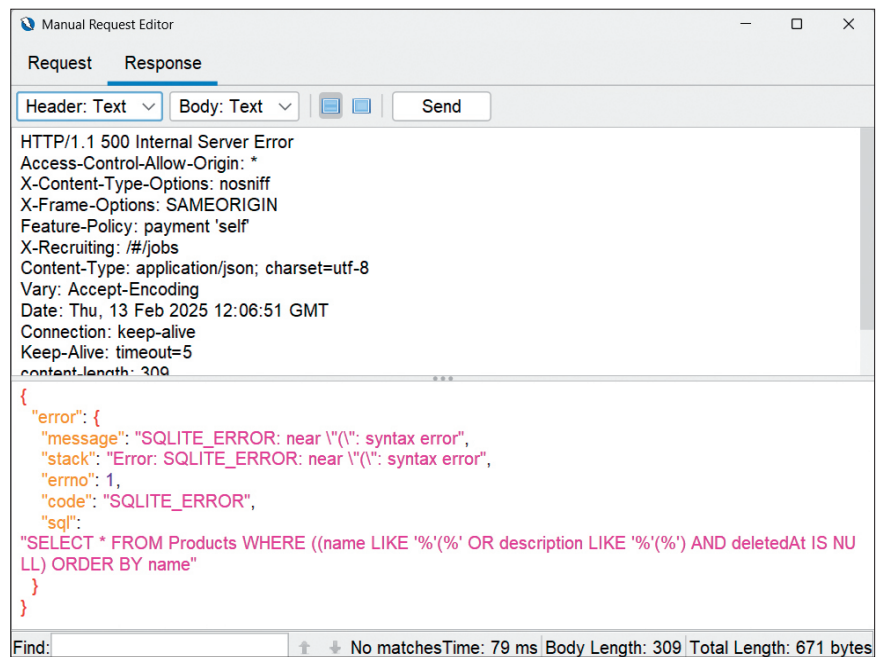
Bis dato waren wir als nicht eingeloggter User unterwegs. Zwar hat ZAP den Login-API-Endpunkt gefunden, wusste aber nicht damit umzugehen. Hier können wir manuell nachhelfen, was aber ein wenig Aufwand bedeutet. Wir starten erneut einen Browser im manuellen Modus und loggen uns dort in die Anwendung ein. Entweder nutzen wir die trivial zu findende SQL-Injection-Lücke im Login-Skript (diese zu entdecken ist Ihnen zur Übung überlassen), oder Sie registrieren zuvor unter `http://localhost:3000/#/register` einen neuen Nutzer. Dann loggen Sie sich über das Menü oben ein. Das genügt bereits, es geht zurück in ZAP.

Suchen Sie im Verlauf der aufgerufenen URLs (Tab *History*) den *HTTP-POST-Request* zum Endpunkt `/rest/user/login`. Im Kontextmenü finden Sie den Eintrag *Flag as Context/Default Context: JSON-based Auth Login Request* (siehe Bild 10).

Damit können Sie diesen Aufruf zu einem sogenannten Kontext hinzufügen – das ist im Wesentlichen eine Liste von URLs, die von ZAP mit einer bestimmten, gemeinsamen Konfiguration untersucht werden soll. Die gemeinsame Konfiguration ist, dass wir eingeloggt sind!

Mit etwas Glück ist im sich öffnenden Assistenten (zu sehen in Bild 11) bereits die JSON-Payload der HTTP-Anfrage mit enthalten. Falls nicht, fügen Sie diese ein. Ebenfalls können Sie weiter unten angeben, woran der Zustand „eingeloggt“ und „ausgeloggt“ diagnostiziert werden kann. Nach dem Login etwa gibt es einen Warenkorb, vor dem Login geht der *Account*-Link auf den Login-URL.

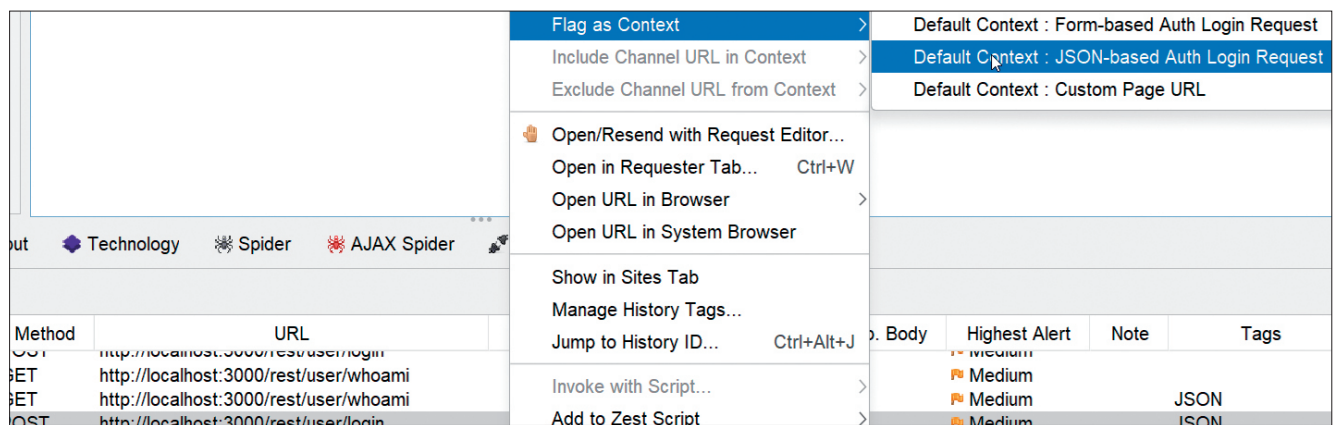
Jetzt wird es kurzzeitig skurril: Schließen Sie den Assistenten und klicken Sie im Tab *Sites* doppelt auf *Default Context*. Sie landen in derselben Maske, aber jetzt wurde die vorher hart codierte Benutzernamen-Passwort-Kombination durch zwei Platzhalter ersetzt. Im Bereich *Users* können Sie jetzt in ZAP beliebige Nutzer hinterlegen, mit denen sich das Tool



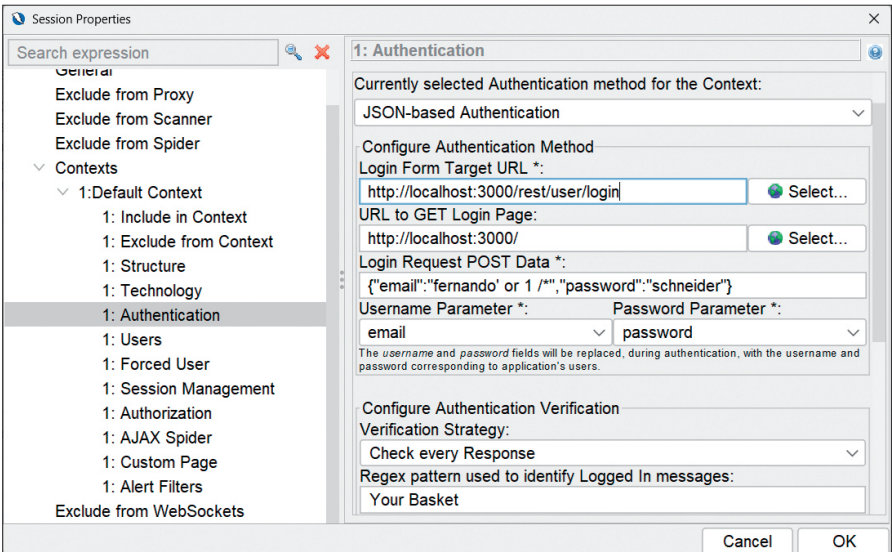
Positiver Test: Die HTTP-Response stützt die Vermutung, dass SQL Injection vorliegt (Bild 9)

dann einloggen kann. Außerdem können Sie weitere Aktionen durchführen, die den Zustand der Anwendung so verändern, dass weitere Endpunkte und Funktionalitäten zur Verfügung stehen. Beispiele hierfür sind etwa das Legen von Artikeln in den Warenkorb, dann nämlich können in diesem auch Artikel entfernt werden. So vervollständigen Sie sukzessive den für ZAP sichtbaren Teil der Anwendung. Im Rahmen des neu konfigurierten Kontexts darf jetzt wieder Ajax Spider aktiv werden und weitere URLs finden, die dann auf Sicherheitslücken untersucht werden können.

Es gibt noch einige weitere wichtige Aspekte, etwa wie nach dem Login die Information über den Login-Status transportiert wird (Spoiler: über den *Authorization*-Header, denn es wird mit JSON Web Tokens gearbeitet). Hierfür bietet sich die Scripting-Engine von ZAP an. Deren detaillierte Beschreibung liegt ein wenig außerhalb des Scopes für diesen Artikel, aber Andres Hermosilla hat hierzu eine gute, wenn auch



Kontextwechsel: Der Login-Request startet eine Authentifizierung via JSON (Bild 10)

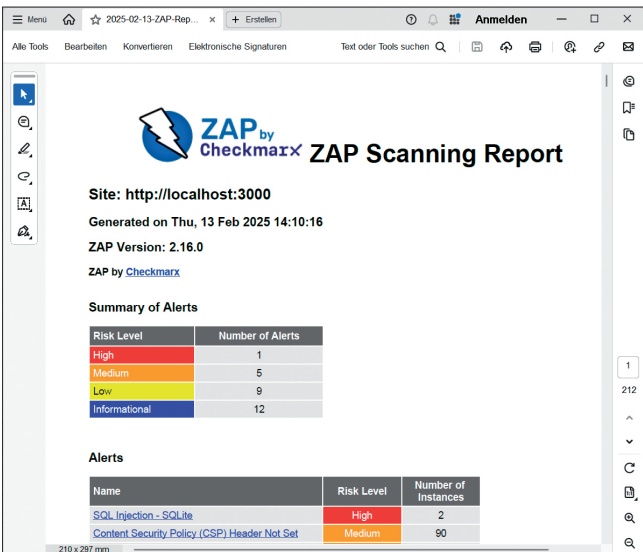


Feinjustierung: Einstellungen für den ZAP-Context (Bild 11)

nicht mehr komplett aktuelle Beschreibung veröffentlicht [9]. Auch die Dokumentation von ZAP hält rudimentäre Informationen über die Verwendung von Skripten bereit [10].

Fazit

Dieser kurze Einblick hat die prinzipielle Vorgehensweise beim Untersuchen einer Single Page Application mit OWASP ZAP geschildert. Wie fast jedes automatisierte Tool im Sicherheitsbereich ist auch ZAP keine vollautomatische Wunderwaffe. Bestimmte Aspekte eines Audits werden gut abgebildet, etwa eine – wenn auch häufig nicht komplette – Erfassung der zur Verfügung stehenden Endpunkte. Danach beginnt jedoch die eigentliche Arbeit: Die potenziellen Treffer gilt es zu analysieren und zu bewerten. Ist eine gefundene Schwachstelle wirklich eine oder doch nicht? Gibt es zusätzliche Sicherheitslücken, die sich beispielsweise durch das



Abschlusszeugnis: Auf Wunsch gibt es das Scanergebnis als PDF (oder in einem anderen Format) (Bild 12)

Anpassen von HTTP-Requests ermitteln lassen? Daraus entsteht dann ein möglicher Report, den übrigens ZAP auch auf Knopfdruck erstellen könnte (Bild 12) – die meisten Auditierenden haben jedoch eigene Templates und setzen im professionellen Umfeld eher auf andere Tools, wie beispielsweise die BurpSuite [11]. Dennoch ist OWASP ZAP ein gutes Mittel, um selbst einmal einen periodischen Blick auf Sicherheit einer Webanwendung zu werfen. Und auch wenn es abgedroschen klingt: Sicherheit ist ein Marathon und kein Sprint. Ein dynamischer Scanner wie ZAP, man spricht auch von DAST (Dynamic Application Security Testing), ist ein fester Bestandteil vieler Sicherheitsstrategien. Und das OWASP-Tool ist, im Unterschied zu vielen anderen Produkten, auch noch kostenfrei und Open Source.

- [1] Christian Wenz, Sicherheitslücken weggezappt, dotnetpro 4/2024, Seite 54 ff., www.dotnetpro.de/A2404ZAP
- [2] Christian Wenz, Erst einloggen, dann attackieren, dotnetpro 5/2024, Seite 54 ff., www.dotnetpro.de/A2405ZAP
- [3] OWASP ZAP, <https://www.zaproxy.org>
- [4] Projekt-Homepage von Juice Shop, www.dotnetpro.de/SL2504-05SPAzap1
- [5] Live-Instanz von Juice Shop, www.dotnetpro.de/SL2504-05SPAzap2
- [6] Dokumentation zum Starten von Juice Shop, www.dotnetpro.de/SL2504-05SPAzap3
- [7] Christian Wenz, No API risks, more fun, dotnetpro 1/2024, Seite 44 ff., www.dotnetpro.de/A2401OWASP
- [8] Christian Wenz, Alarmstufe Rot für APIs, dotnetpro 2/2024, Seite 44 ff., www.dotnetpro.de/A2402OWASP
- [9] Andres Hermosilla, Die Script-Engine von ZAP, www.dotnetpro.de/SL2504-05SPAzap5
- [10] Juice Shop, ZAP und Authentifizierung, www.dotnetpro.de/SL2504-05SPAzap4
- [11] BurpSuite, ein professionelles kommerzielles Tool für Sicherheitstests, <https://portswigger.net/burp>



Christian Wenz

ist Softwarearchitekt, Berater und Spezialist für die Themen Performance und Web Security. Seit 2004 ist er Microsoft MVP. Seine neuesten Bücher sind „ASP.NET Core Security“ (erschienen bei Manning) und „C# und .NET 8“ (Hanser). www.christianwenz.de

dnpCode A2504-05SPAzap

SINGLE-SPA – MICROFRONTENDS

Weißer Schimmel

Ein kleiner Erfahrungsbericht aus der Praxis zum Einsatz unterschiedlicher Frontend-Frameworks unter der Regie von Single-SPA.

Yep“. So lautet die lapidare Feststellung auf die Frage, ob der Name Single-SPA nicht ein wenig redundant sei. Single-SPA [1] ist ein JavaScript-Router für Frontend-Microservices und ermöglicht den Einsatz unterschiedlicher Frontend-Frameworks (Vue, React, Angular und weitere) für einzelne Microfrontends, die mittels Single-SPA gewissermaßen zu einer größeren Anwendung zusammengesteckt werden können.

Die Idee

Microfrontends [2] dürften, wie auch Microservices, sicherlich jedem ein Begriff sein. In mehrere Teams aufgeteilt, können die Entwickler unabhängig voneinander an den Teilen einer großen Anwendung in ihren jeweils bevorzugten Frameworks arbeiten.

Jedes einzelne Team ist sowohl für sein Frontend als auch sein Backend zuständig. Eine sinnvolle und konsequente Trennung der Belange wird nicht nur durch die Aufteilung der Anwendung in einzelne serverseitige Dienste erreicht, sondern auch an der Oberfläche durch ebendiese Microfrontends. Ein Beispiel für eine mögliche Aufteilung der Teams ist in **Bild 1** dargestellt.

Root Application		
Team A	Team B	Team C
Products	Payment	Users
Frontend Vue3	Frontend React	Frontend Angular
Backend C#	Backend PHP	Backend Java

Mögliche Aufteilung der Teams auf die Unteranwendungen einer Applikation (**Bild 1**)

Mal eben eine Beispielanwendung aufsetzen

Auch wenn „mal eben“ in etwa 99 Prozent der Fälle schiefgeht, so hat es für den Autor doch bei einem kleinen Beispiel erstaunlich reibungslos funktioniert. Ein aktuelles Node.js vorausgesetzt, lassen sich mit dem folgenden Befehl fix die benötigten Bibliotheken herunterladen:

```
npm install single-spa
```

Weiter geht es mit:

```
npx create-single-spa
```

Möglicherweise werden ein paar weitere Pakete installiert, die im Weiteren gebraucht werden. Die Konsole gibt ausreichend Auskunft, was passiert. Eine Auswahl zeigt, was genau nun kreiert werden soll, die entsprechenden Fragen werden wie folgt beantwortet:

```
? Directory for new project: root
? Select type to generate: single-spa root config
? Which package manager do you want to use? npm
? Will this project use TypeScript? Yes
? Would you like to use single-spa Layout Engine: Yes
? Organization name (can use letters, numbers, dash or
  underscore) irgendein-griffiger-name
```

Der *Organization name* wird später verwendet, um die einzelnen Pakete zu identifizieren. Dazu kann der *Organization name* in der *package.json* dem eigentlichen Namen vorangestellt werden:

```
{
  "name":
    "@irgendein-griffiger-name/ein-tolles-microfrontend",
  ...
}
```

Es werden nun vermutlich eine Reihe Fehlermeldungen erscheinen, die ignoriert werden können. Die *package.json* liegt eben nicht im aktuellen Ordner, sondern eine Ebene tiefer – wenn Sie dem Beispiel eben gefolgt sind, im Ordner *root*. Also gilt es nun, im *root*-Ordner *npm install* auszuführen.

Ein wenig freien Festplattenplatz sollte man mitbringen; es werden etwa 1100 Pakete installiert. Wenn das erledigt ist, startet *npm run start* die Rahmenanwendung, in die später die Microfrontends eingehängt werden. Und wenn „mal eben“ bei Ihnen genauso geklappt hat wie beim Autor, können Sie zu <http://localhost:9000> navigieren und das erste Zwischenergebnis begutachten (**Bild 2**).

Wo kommt das jetzt her?

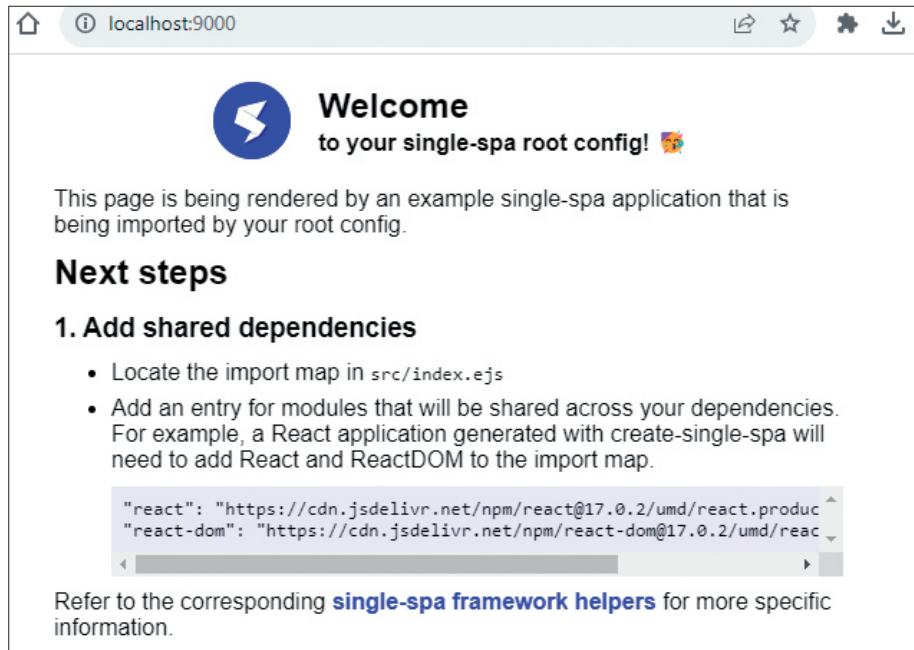
Im GitHub-Repository unter [4] finden Sie eine ganze Reihe von Beispielen zu polyglotten Microfrontends. Werfen Sie gerne einen Blick in dieses Repository. In der Root-Anwen-

derung gibt es eine `index.ejs`-Datei, die eine Importmap bereitstellt. Dieser werden die Microfrontends bekannt gegeben, sodass zu jeder Unteranwendung navigiert werden kann. Das kann dann zum Beispiel so aussehen:

```
<script>
  System.import(
    '@irgendein-griffiger-name/
      root-config');
  System.import(
    '@irgendein-griffiger-name/
      ein-tolles-microfrontend');
</script>
```

Für die Entwicklung kann diese Importmap überschrieben werden, sodass nur das Modul, das gerade bearbeitet wird, auf dem Entwicklungsserver läuft; alle anderen Microfrontends werden dann zum Beispiel von einem gemeinsam genutzten Test- oder Entwicklungsserver geladen.

Um ein Modul „on the fly“ einzuhängen, müssen Sie lediglich einen Schlüssel namens `devtools` mit dem Wert `true` in den Local Storage Ihres Browsers einfügen. Dann können Sie über eine kleine Benutzeroberfläche den URL zu einem Microfrontend überschreiben. Auch das wird in dem eben erwähnten Repository erläutert.



Die Standardseite nach dem Erstellen der Root-Anwendung (Bild 2)

Dann brauen wir mal ein Süppchen

So weit, so gut. Der Rahmen, in den wir die einzelnen Anwendungen einhängen können, steht. Jetzt gilt es, das Ganze mit etwas Inhalt zu füllen. Zunächst soll eine Vue3-App eines der Microfrontends sein. Diese wird mit folgendem Befehl aus dem Boden gestampft:

```
npm create vite@latest [3]
```

● Listing 1: Modifizieren der main.js

```
import './assets/main.css'
import { h, createApp } from 'vue'
// h zum import hinzufügen
import App from './App.vue'
import router from './router'
import singleSpaVue from 'single-spa-vue'
// hinzufügen

// aus der 'alten' main.js
// const app = createApp(App)

// app.use(router)

// app.mount('#app')

// ab hier 'neue' main.js
const vueLifecycles = singleSpaVue({
  createApp,
  appOptions: {
    render() {
      return h(App, {
        // single-spa props are available on the
        // "this" object. Forward them to your
        // component as needed.
        // https://single-spa.js.org/docs/building-
        // applications#lifecycle-props
        name: this.name
        // mountParcel: this.mountParcel,
        // singleSpa: this.singleSpa
      })
    },
    handleInstance: (app) => {
      app.use(router)
    }
  })

export const bootstrap = vueLifecycles.bootstrap
export const mount = vueLifecycles.mount
export const unmount = vueLifecycles.unmount
```

● **Tabelle 1: Sorgfältig empirisch evaluierte und zugleich vollkommen subjektive Einschätzung des Autors**

Pro Single-SPA	Contra Single-SPA
Große Unabhängigkeit in der Entwicklung von Modulen	Abhängigkeiten von Basismodulen sind nicht unbedingt sichtbar
Große Sprachvielfalt möglich	Synergien lassen sich schlechter nutzen
Gut geeignet für große Teams	Relativ großer Overhead für kleine Teams
Wer Unfug baut, baut erst mal nur für sein Team Unfug	Wer Unfug baut, baut Unfug für alle
Code-Review: Bessere Konzentration auf die eigentlichen Änderungen	Code-Review: Auswirkungen auf andere Module nicht sofort ersichtlich
Skalierbarkeit	Kommunikation der Module untereinander komplizierter
Gemeinsame Komponenten eher resistent gegen individuelle Erfordernisse	Gemeinsame Komponenten schwieriger zu realisieren
Deployment: Einzelne Teile der Anwendung können ausgetauscht werden	Deployment: Vereinfacht, aber es muss immer die komplette Anwendung ausgeliefert werden

Hiermit wird das wohl den meisten der Leser bekannte Grundgerüst einer Vue3-Anwendung erstellt, die, wenn man den Anweisungen an der Konsole folgt, unter <http://localhost:5173> auch sofort gestartet werden kann.

Jetzt muss diese erste Anwendung noch etwas modifiziert werden, um in die Root-Anwendung eingehängt werden zu können. Das Paket `single-spa-vue` wird mit

```
npm install single-spa-vue
```

heruntergeladen und die `main.js` entsprechend [Listing 1](#) modifiziert. Die drei ausschlaggebenden Lifecycle-Funktionen müssen vorhanden sein: `bootstrap`, `mount` und `unmount`.

Eine weitere Zutat

Hier tritt nun der nach Meinung des Autors größte Vorteil von Single-SPA in Erscheinung: Es ist ziemlich unerheblich, welches Team nun welche Vorlieben hat. Ob React, Angular, Vue oder was auch immer, all diese unterschiedlichen Anwendungen lassen sich relativ einfach zu einer großen Gesamtanwendung zusammenstecken, kein Entwickler muss sein Lieblings-Framework aufgeben. Lediglich der Einstiegspunkt einer jeden Unteranwendung muss die eben erwähnten Lifecycle-Methoden implementieren.

Ein gemeinsames Stylesheet kann in der Root-Anwendung hinterlegt werden. Auch hierzu sei noch einmal auf das unter [4] genannte GitHub-Repository verwiesen.

Aus dem Nähkästchen

Es ist nicht immer Gold, was glänzt. Eine kleine Firma in der Nähe des niedersächsischen Niemandslands, in dem der Autor beheimatet ist, hatte sich zur Aufgabe gestellt, ein neues Produkt auf die Beine zu stellen. Dass im Backend Microservices verwendet werden sollten, war gesetzt. Passend dazu sollte das Frontend nun auch in Microfrontends aufgeteilt werden. Die Voraussetzungen dafür waren gut: Jeder war von der Idee von Single-SPA überzeugt, und einige aus der

Gruppe der Entwickler (nennen wir sie Team A) hatten sogar schon ein Modul fertiggestellt, und zwar in Angular. Andere aus der Entwicklergruppe waren neugierig auf das derzeit noch taufrische Vue3 und bildeten Team V. Somit waren gute Anfangsbedingungen gegeben, die Vorteile von Single-SPA auch nutzen zu können. Die Rahmenanwendung (Single-SPA/Root) wurde aufgesetzt, die neuen Vue3-Module eingehängt, und auch das Angular-Modul von Team A machte wenig Probleme und alles konnte zur allgemeinen Zufriedenheit zusammengesteckt werden. Alle waren glücklich.

Alle? Nein, nicht alle, ein kleiner Entwickler in einer großen Stadt in der Nähe des bereits erwähnten niedersächsischen Niemandslandes hatte Bauchschmerzen. Team A war ebenfalls auf Vue3 neugierig geworden; das Composition-API bot tolle Möglichkeiten. Nach und nach liefen immer mehr Entwickler zu Team V über. Eine sorgfältige Analyse zeigte, dass das Modul von Team A mit vertretbarem Aufwand auf Vue3 umgebaut werden konnte, was dann auch geschah. Ein großer Vorteil von Single-SPA wurde nicht mehr genutzt. Dass Team A nun dem A den Rücken kehrte und ganz auf V schwor, eröffnete letztendlich die Möglichkeit, modulübergreifende Komponenten wie zum Beispiel Eingabefelder oder Listen als Vue-Komponenten in wiederverwertbarer Weise zu implementieren und benutzen. Die Bauchschmerzen wurden geringer.

Zurück zum (modularen) Monolithen

Mittlerweile waren keine Voraussetzungen mehr vorhanden, weiter an Single-SPA festzuhalten. In wenigen Tagen konnten die Microfrontends zu einem modularen Monolithen (oder auch Modulithen) [5][6] zusammengeführt werden. Nebenbei muss erwähnt werden, dass auch die Backend-Microservices alle in C# gebaut waren. Hier stand ebenfalls zur Diskussion, diese in einen Modulithen zu überführen, was dann auch vollzogen wurde.

Im Ergebnis waren alle zufrieden. Die Entwicklungsprozesse waren verschlankt. Das Deployment war einfacher, es

gab allgemein weniger Bauchschmerzen. Neue Features konnten wesentlich einfacher und schneller implementiert werden. Jeder konnte sich einfach und schnell einen Überblick über die gesamte Architektur verschaffen.

Fazit

Werfen Sie einen Blick auf **Tabelle 1** für eine subjektive Einschätzung des Autors. Single-SPA hat aber auf jeden Fall seine Berechtigung, wenn folgende Bedingungen gegeben sind:

- sehr großes Entwicklerteam,
- es wird in vielen unterschiedlichen Programmiersprachen gesprochen (TypeScript, JavaScript, PHP, Java, C#, ...),
- es werden unterschiedliche Frontend-Frameworks benutzt (React, Angular, Vue, ...),
- einzelne Anwendungsteile sind sehr unabhängig voneinander (Logik, Aussehen), zum Beispiel eine Chatbox, die neben der eigentlichen Hauptanwendung läuft.

Sind diese Bedingungen nicht gegeben, ist auf jeden Fall zu überlegen, ob Modulithen die bessere Lösung sind. In eher kleineren Teams mit wenigen gesprochenen Sprachen und einer geringen Anzahl an benutzten Frameworks fällt der gesamte Overhead von Microfrontends weg, man kann sich viel besser gegenseitig helfen und vielleicht auch die eine oder andere Lösung von der Kollegin oder vom Kollegen „abgucken“.

Jedenfalls ist besagte kleine Firma in der Nähe des niedersächsischen Niemandslands, in dem der Autor beheimatet ist, bis heute glücklich mit ihren modularen Monolithen. ■

[1] Single-SPA, <https://single-spa.js.org/>

[2] Martin Fowler, Micro Frontends, www.dotnetpro.de/SL2504-05SSPA1

[3] ViteConf 2024 Talks, The Build Tool for the Web, <https://vitejs.dev>

[4] polyglot-microfrontends auf GitHub, www.dotnetpro.de/SL2504-05SSPA2

[5] André Kappes, Modulare Monolithe statt Microservices, www.dotnetpro.de/SL2504-05SSPA3

[6] Florian Rappl, Monolith vs. Micro Frontend, www.dotnetpro.de/SL2504-05SSPA4



Christian Sender

ist bei der sycat IMS GmbH in Hannover als Softwareentwickler im Frontend und Backend unterwegs. Sie erreichen ihn unter c.sender@sycat.com.

dnPCode

A2504-05SSPA

DEV ACADEMY

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

3 TAGE
12.-14.05.2025

Architektur für .NET-Anwendungen

TRAINING!

Was wird behandelt

- Prinzipien für Objektorientierung
- Metriken
- Komponenten
- Inversion of Control
- Multi-Layer-und Multi-Tier-Architektur
- Referenzarchitektur



David Tielke
Trainer

EF CORE MIT MICROSOFT SQL SERVER



Performance-Falle

Asynchrone Abfragen großer Datenmengen beschleunigen.

Das Entity Framework hat sich in der .NET-Welt für die Arbeit mit relationalen Datenbanken etabliert. Provider für EF Core gibt es mittlerweile für alle gängigen Datenbanksysteme. In der .NET-Welt besonders beliebt ist Microsofts haus-eigene Datenbank, der SQL Server (MSSQL). Auch für asynchrone Abfragen mittels *async* und *await* hat sich das Entity Framework bewährt. Allerdings gibt es gerade bei der beliebten Konstellation von EF Core und Microsoft SQL in Kombination mit *async* und *await* immer wieder Performance-Probleme. Deutlich machen soll dies folgendes Minibeispiel:

```
public class ProblemDbContext : DbContext
{
    public DbSet<ProblemTable> Problem { get; set; }
    protected override void OnConfiguring(
        DbContextOptionsBuilder optionsBuilder)
    {
        optionsBuilder.UseSqlServer(connectionString:
            @"Server=(localdb)\MSSQLLocalDb;
            Initial Catalog=ProblemDb;")
    }
}

public class ProblemTable
{
    public Guid ID { get; set; }
    public string LargeString { get; set; }
    public byte[] LargeBinary { get; set; }
}
```

Es wird dafür ein einfacher *DbContext* mit einer einzigen Tabelle erzeugt. Das Performance-Problem, um das es hier geht, macht sich insbesondere bei Tabellen bemerkbar, die umfangreiche Binärdaten enthalten. Im Minibeispiel ist das die *ProblemTable*, die neben der Zeichenkette *LargeString* auch noch ein großes Byte-Feld *byte[] LargeBinary* enthält.

Es folgen ein paar Beispieldaten, mit denen die Tabelle gefüllt wird: Es sind zehn Einträge, jeweils mit einem zufällig mit 10 MByte gefüllten Byte-Feld plus einem 1024 Zeichen langen String. Es geht also nicht um atemberaubend unhandliche Datenmengen:

```
await using var ctx = new ProblemDbContext();
ctx.Database.EnsureCreated();
var buffer = new byte[1024 * 1024 * 10];
Random.Shared.NextBytes(buffer);
ctx.Problem.AddRange(Enumerable.Range(0, 10)
    .Select(x:int => new ProblemTable()
```

```
{
    LargeBinary = buffer,
    LargeString = new string(c: 't', count: 1024)
}));
await ctx.SaveChangesAsync();
```

Bei der Anlage des Beispiels gibt es mit diesem Code noch kein Problem. Richtig schmerzhaft wird allerdings die Abfrage von Daten aus diesem Minibeispiel.

Jetzt wird es langsam

Der „schmerzhaft“ Code gliedert sich in drei Teilabschnitte: In Phase 1 wird eine neue Instanz gebaut und der Kontext mit *await* abgefragt. Damit ist der Kontext bereit, das Modell von EF Core wurde erzeugt und EF Core selbst ist „warmgelaufen“. Phase 2 ruft danach einfach den ersten Datensatz ab, und zwar asynchron (*FirstOrDefaultAsync*), wohingegen Phase 3 den gleichen Datensatz synchron abrufen (*FirstOrDefault*).

```
// Phase 1
await using var initialContextToWarmUp =
    new ProblemDbContext();
await initialContextToWarmUp.Problem
    .FirstOrDefaultAsync();

// Phase 2
var asyncWatch = Stopwatch.StartNew();
await using var asyncContext = new ProblemDbContext();
await asyncContext.Problem.FirstOrDefaultAsync();
Console.WriteLine(asyncWatch.Elapsed);

// Phase 3
var synWatch = Stopwatch.StartNew();
await using var syncContext = new ProblemDbContext();
syncContext.Problem.FirstOrDefault();
Console.WriteLine(syncWatch.Elapsed);
```

Man könnte meinen, dass beide Abfragevarianten gleich schnell oder zumindest ähnlich schnell sein sollten. Bild 1 zeigt die Laufzeiten in der traurigen Realität: Die asynchrone Abfrage dauert rund 40-mal so lang (6,3101 Sek.) wie die syn-

Katastrophale Werte für
die asynchrone Abfrage
(oben) – unten die Werte
der synchronen Abfrage
(Bild 1)

00:00:06.3101399
00:00:00.1549630

Method	Mean	Error	StdDev	Gen 0	Gen 1	Gen 2	Allocated
Async	3,170.53 ms	222.636 ms	132.487 ms	635000.0000	634000.0000	633000.0000	13155.71 MB
Sync	37.88 ms	2.139 ms	1.415 ms	333.3333	333.3333	333.3333	20 MB

Die Community berichtet: Es wird schlimmer, je genauer man hinsieht (Bild 2)

chrone Abfrage (0,1549 Sek.). Das ist nicht nur „ein bisschen mehr“, sondern ein sagenhaft großer Laufzeitunterschied.

Woran liegt das?

Microsoft ist dieser Umstand längst bekannt. Schon seit Juni 2020 gibt es auf GitHub einen Problembericht (Issue) [1], der mittlerweile seit fast fünf Jahren auf einen Fix wartet.

Das EF-Core-Team sieht sich dabei als nicht zuständig an, weil der eigentliche Fehler im Treiber von Microsofts SQL Server liegt. Der Problembericht von EF Core referenziert nämlich ein Issue im Treiber, das ebenfalls seit Juni 2020 auf eine Lösung wartet [2]. Dort wurde das Problem bereits tiefgreifend analysiert, wie zum Beispiel der kanadische Entwickler Davoud Eshtehari unter [2] schreibt.

In Bild 2 sehen Sie einige Ergebnisse einer Messung mit BenchmarkDotNet, die nicht nur belegen, dass die Laufzeit unterirdisch schlecht ist, sondern auch, dass zugleich der Speicherbedarf atemberaubend groß ausfällt.

Mit anderen Worten: Alle Entwickler, die EF Core und Microsoft SQL Server in Verbindung mit *async* und *await* nutzen, haben dasselbe Problem bei allen Tabellen, die große Binärdaten enthalten. Und seien wir ehrlich: Das sind einige Projekte.

Kein Fix in Sicht?

Microsoft ist sich des Problems zwar offenbar bewusst, aber die Mühlen in Seattle mahlen recht langsam. Immerhin gibt es bereits eine „Teillösung“. Dafür verlangt EF Core mindestens die Version 5.1.6 des *Microsoft.Data.SqlClient*-Treibers. Besser wechselt man gleich zur neuesten Treiberversion 6.0.1, die vor rund zwei Monaten veröffentlicht wurde. Nutzt man diese, so ändert sich die Laufzeit des Mini-Beispiels ohne eine weitere Anpassung dramatisch. Anstatt mehr als sechs Sekunden benötigt der asynchrone Code jetzt nur noch rund 1,5 Sekunden.

Noch etwas mehr optimieren?

Es gibt allerdings noch eine weitere Optimierung, um den asynchronen Weg schneller zu machen. Das Grundproblem, das im Microsoft-Treiber steckt, ist offenbar die Paketverarbeitung des SQL-Server-Protokolls. Und hier gibt es eine Stellenschraube für Optimierungen. Dafür gibt man dem *ConnectionString* über den Parameter *Packet Size* mit, dass man gern größere, dafür aber weniger Pakete erhalten möchte:

```
optionsBuilder.UseSqlServer(connectionString:
    @Server=(localdb)\MSSQLLocalDb;
    Packet Size=32766;
    Initial Catalog=ProblemDb;)
```

Die asynchrone Abfrage

benötigt 0,33 Sekunden und dauert damit dreimal so lang wie die synchrone Variante (Bild 3)

00:00:00.3348248
00:00:00.1103132

Mit dieser Vorgabe liefert die Datenbank das Ergebnis noch einmal ein gutes Stück schneller als zuvor. Anstelle der ursprünglichen sechs Sekunden benötigt die asynchrone Abfrage lediglich 0,33 Sekunden und ist damit nur noch um den Faktor 3 langsamer als die synchrone Variante (Bild 3).

Fazit

Geliebtes Microsoft, so gern ich das .NET-Ökosystem auch habe: Ein solcher Fehler darf doch nicht passieren! Vor allem die lange Dauer bis zum Fix des Problems ist enttäuschend. Zwar kommuniziert Microsoft dies transparent in einem Video vom November 2024 [3], aber das ist dann doch zu wenig. Schließlich leidet die Performance, indem man Microsoft-Technologien und -Produkte einsetzt. Das Problem besteht nämlich nur, wenn man mit dem SQL Server arbeitet; Entwickler, die zum Beispiel auf PostgreSQL setzen, kennen das geschilderte Performance-Problem nicht, denn der PostgreSQL-Treiber arbeitet genau so, wie er es soll.

Wer also die Kombination aus SQL Server, EF Core und asynchronen Abfragen nutzen muss, sollte die vorgestellten Tipps berücksichtigen, mit denen sich asynchrone Abfragen deutlich beschleunigen lassen – auch wenn sie immer noch spürbar langsamer als synchrone Abfragen sind. ■

[1] GitHub, Issue #21147 zu EF Core,

www.dotnetpro.de/SL2504-05NETirol1

[2] GitHub, Reading large data (binary, text) asynchronously is extremely slow, www.dotnetpro.de/SL2504-05NETirol2

[3] Video: Navigating Async Challenges in EF Core, www.dotnetpro.de/SL2504-05NETirol3



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 hat.

christian@software.tirol

dnpCode

A2504-05NETirol

AZURE INTEGRATION SERVICES

Keine Innovation ohne Integration

Eine effiziente Integration von Daten, Prozessen und Systemen will gut überlegt sein. Die Azure Integration Services von Microsoft unterstützen Sie bei der Planung.

Innovation und Anpassungsfähigkeit sind entscheidende Faktoren für den Erfolg von Unternehmen in einer sich ständig wandelnden Technologielandschaft. Die Fähigkeit, neue Trends schnell zu adaptieren und effektiv in bestehende Prozesse zu integrieren, bestimmt, wie wettbewerbsfähig eine Organisation bleibt. Doch viele Unternehmen unterschätzen eine essenzielle Komponente auf dem Weg zur Innovation: Integration. Ohne nahtlose Integration bestehender Systeme und neuer Lösungen bleiben viele Innovationen Stückwerk oder scheitern ganz. Ein besonderes Hindernis stellen Datensilos dar – isolierte Systeme, die den freien Fluss von Informationen verhindern und damit Innovation und Effizienz bremsen.

Ich möchte Ihnen in diesem Artikel mögliche Integrationsansätze in der Theorie vorstellen, am Ende aber einen Praxisbezug herstellen und Ihnen zeigen, wie Azure Integration Services eine Möglichkeit zur Umsetzung bereitstellt. Los gehts!

Datensilos aufbrechen – für eine ganzheitliche Integrationsstrategie

In vielen Unternehmen sind Daten noch immer in isolierten Systemen gefangen – getrennt nach Abteilungen oder An-

wendungen. Eine erfolgreiche Integrationsstrategie ist entscheidend, um diese Datensilos aufzulösen und eine nahtlose, unternehmensweite Nutzung von Informationen zu ermöglichen. Dafür müssen Systeme und Anwendungen optimal vernetzt und harmonisiert werden.

Moderne Integrationsarchitekturen vereinen Daten- und Integrationsplattform miteinander, um eine einheitliche und zugängliche Datenbasis zu schaffen. Doch eine effektive Integration erfordert mehr als nur Technologie: Einheitliche Prozesse, Standards und Schnittstellen sind essenziell, um Daten konsistent zu erfassen, zu verarbeiten, bereitzustellen und zu verstehen. Dieses Vorgehen verhindert, dass Daten doppelt gehalten werden, verbessert die Datenqualität und stellt einen reibungslosen Informationsfluss zwischen allen Anwendungen und Personen sicher.

Eine erfolgreiche Integrationsstrategie erfordert zudem eine enge Zusammenarbeit zwischen IT, Fachabteilungen und Management. Nur wenn alle Akteure an einem Strang ziehen, kann eine ganzheitliche integrierte Daten-, Prozess- und Anwendungslandschaft entstehen. Offene Schnittstellen (APIs), Cloud-basierte Plattformen und eine durchdachte

● **Tabelle 1: Integrationsansätze**

Ansatz	Beschreibung	Vorteile	Nachteile
Punkt-zu-Punkt	Direkte Verbindungen zwischen zwei Systemen oder Anwendungen	Einfach zu implementieren und geeignet für Schnittstellen, die „über Jahre“ gleich bleiben	Schwer skalierbar, erhöht die Komplexität bei vielen Verbindungen
Hub and Spoke	Ein zentraler Hub (Integrationsplattform) verbindet mehrere Systeme	Vereinfacht die Verwaltung, reduziert die Anzahl der Verbindungen	Hub kann zum Flaschenhals werden, Single Point of Failure
Enterprise Service Bus (ESB)	Middleware, die die Kommunikation zwischen verschiedenen Anwendungen erleichtert	Flexibel, unterstützt heterogene Systeme, fördert lose Kopplung	Komplexere Implementierung, mögliche Performance-Herausforderungen
API-driven	Nutzung von APIs, um Daten und Dienste zwischen Anwendungen auszutauschen	Standardisierte Schnittstellen, fördert Interoperabilität, erleichtert Integration mit externen Partnern	API-Sicherheit und -Management sind kritisch, die Performance kann variieren
Event-driven	Systeme kommunizieren durch das Versenden und Empfangen von Ereignissen	Echtzeit-Verarbeitung, hohe Reaktionsfähigkeit, fördert lose Kopplung	Komplexere Fehlersuche, das Ereignismanagement kann herausfordernd sein

Data Governance helfen, Systeme nachhaltig zu integrieren und wertvolle Erkenntnisse aus den vorhandenen Daten zu gewinnen.

Integration als Innovationsmotor

In einer Zeit, in der Microservices, Cloud Computing und künstliche Intelligenz die Entwicklung bestimmen, geht es nicht nur um neue Features. Über den Erfolg entscheidet vielmehr die Fähigkeit, neue Technologien in bestehende Landschaften zu integrieren. Unternehmen, die ihre Legacy-Systeme nicht mit modernen APIs oder Cloud-Diensten verknüpfen, laufen Gefahr, den Anschluss zu verlieren.

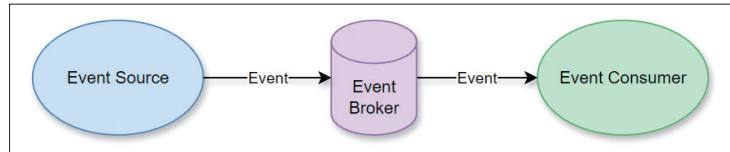
Richten Sie zur Veranschaulichung Ihren Blick auf das folgende Beispiel: Ein Unternehmen möchte maschinelles Lernen zur Datenanalyse nutzen. Allerdings sind die bestehenden Datenbanken monolithisch aufgebaut und bieten keine flexiblen Schnittstellen. Die fehlende Integration verhindert hier direkt den Innovationsfortschritt. Zudem erschweren diese Datensilos den Zugriff auf wichtige Informationen, sodass datengetriebene Entscheidungen verzögert oder sogar unmöglich gemacht werden. Um dieses zu verhindern, ist die Integration als Innovationsmotor unerlässlich. Doch bevor die konkrete Umsetzung mit den Azure Integration Services beginnt, habe ich Ihnen in **Tabelle 1** in der Praxis gängige Integrationsansätze gegenübergestellt.

Alle genannten Integrationsansätze sind valide, aber in Kundenprojekten kristallisieren sich die beiden letztgenannten Ansätze – API-driven und Event-driven – als „De-facto-Standard“ für moderne Integrationsplattformen heraus, auf die ich mich im weiteren Verlauf dieses Artikels konzentrieren werde. Kurzer Spoiler: In der Praxis werden diese Ansätze miteinander vermischt, um für jeden Use Case die optimale Lösung heranzuziehen. Es gibt hier also kein Entweder-oder, sondern ein Sowohl-als-auch. Lassen Sie uns die beiden Ansätze im Folgenden näher betrachten.

API-driven Integration

API-driven ist ein Integrationsansatz, bei dem Anwendungen und Systeme über Programmierschnittstellen (APIs) miteinander verbunden werden. APIs bieten standardisierte Schnittstellen, über die Daten und Funktionen ausgetauscht werden können. Dieser Ansatz bietet folgende Vorteile:

- **Standardisierung und Interoperabilität:** APIs bieten eine einheitliche Methode, um auf Daten und Dienste zuzugreifen. Dies erleichtert die Integration unterschiedlicher Systeme und fördert die Interoperabilität zwischen Anwendungen.
- **Flexibilität und Skalierbarkeit:** Durch die Verwendung von APIs können Unternehmen ihre Integrationen flexibel anpassen und skalieren. Neue Dienste oder Anwendungen können schnell integriert werden, ohne bestehende Systeme zu beeinträchtigen.
- **Schnellere Entwicklungszyklen:** Mit API-driven Integration können Entwickler schneller neue Funktionen und Anwendungen bereitstellen, da sie auf bereits vorhandene APIs zugreifen können, anstatt alles von Grund auf neu zu entwickeln.



Komponenten der Event-driven Integration (Bild 1)

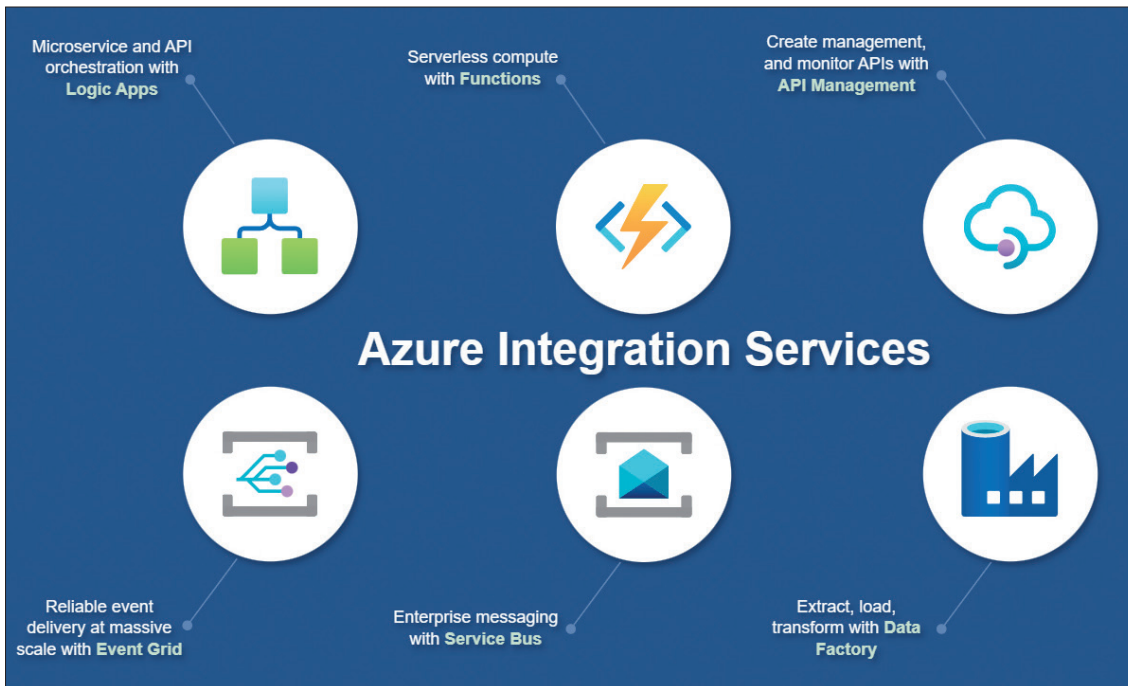
- **Sicherheit:** APIs ermöglichen eine fein abgestufte Zugriffskontrolle und Authentifizierung, wodurch die Sicherheit der Integrationen erhöht wird. Dies ist besonders wichtig, wenn Daten zwischen verschiedenen Systemen und Plattformen ausgetauscht werden.
- **Einfachere Wartung und Verwaltung:** Die Verwaltung von Integrationen über APIs ist einfacher, da Änderungen an einem Dienst oder einem API nicht zwangsläufig Änderungen an allen verbundenen Systemen erfordern. Dies reduziert den Wartungsaufwand und erleichtert die Fehlerbehebung.

Doch wo Licht ist, ist auch Schatten. Eine der größten Herausforderung ist der sogenannte API Sprawl [1], bei dem die APIs an vielen verschiedenen Orten verwaltet werden, was die große Gefahr mit sich bringt, die Übersicht über diese zu verlieren. Wichtig und entscheidend für den Erfolg von API-driven Integration ist deshalb die unternehmensweite Verwaltung aller APIs an einem zentralen Ort. Diese Verwaltung wird als API-Management bezeichnet, auf die wir im folgenden Abschnitt eingehen.

API-Management als zentraler Baustein

Die meisten Unternehmen verfügen über eine historisch gewachsene IT-Infrastruktur, die eine Mischung aus On-Premises-Lösungen, monolithischen Anwendungen und modernen Cloud-Diensten umfasst. Dadurch existieren bereits viele APIs, und es stellt sich die Frage, wie diese APIs organisiert werden. Eine gut durchdachte API-Management-Strategie ist essenziell, um eine nahtlose und effiziente Integration zwischen Line-of-Business-Systemen (LoB) und Anwendungen zu gewährleisten. Dabei spielt es keine Rolle, ob es sich um Eigenentwicklungen oder um Drittanbieter-Anwendungen handelt.

Ein modernes API-Management sorgt für eine optimierte Kommunikation zwischen Systemen und erhöht die Sicherheit durch einheitliche Governance und standardisierte Authentifizierungs- und Autorisierungsmechanismen. API-Management ist ein Prozess, bei dem APIs erstellt, veröffentlicht, gesichert, überwacht und verwaltet werden. Es bietet eine zentrale Plattform, um APIs effizient zu steuern und deren Nutzung zu optimieren. Vereinfacht gesagt bietet API-Management eine Katalogisierung aller im Unternehmen befindlichen APIs an einem zentralen Ort, an dem Entwickler diese finden und nutzen können, einschließlich einer Berechtigungssteuerung. Als Entwickler abstrahieren wir gern, weil es das ist, was wir sehr gut können. Im Grunde wird beim API-Management nichts anderes getan: Wir abstrahieren wiederkehrende Logik wie Authentifizierung in eine höhere Schicht und brauchen uns in unserem Code nicht darum zu kümmern. Klingt gut, oder? Aber es wird noch besser – mit der Event-driven Integration. ►



Azure Integration Services
im Überblick
(Bild 2)

Event-driven Integration

Event-driven Integration ist ein Ansatz, bei dem Systeme durch das Senden und Empfangen von Events kommunizieren. Ein Event ist die Benachrichtigung über eine unwiderrufliche Änderung des Zustands, die von einer Anwendung oder einem System erkannt wird und potenziell eine Reaktion auslöst. Ein konkretes Beispiel ist der Eingang einer neuen Bestellung über einen Webshop. Der Event könnte *OrderReceived-Event* heißen und löst einen nachgelagerten Prozess wie etwa das Reservieren der bestellten Artikel sowie die Zahlungsabwicklung aus. Mit Event-driven werden reaktive Systeme erstellt, die nicht mit *PULL* funktionieren, sondern vielmehr mit *PUSH*. Dadurch entfällt im Übrigen auch das ständige Nachfragen in einer Datenbank, ob eine neue Bestellung eingetroffen ist. Dank Event-driven wird ein System benachrichtigt, wenn etwas passiert ist – und zwar genau nur dann.

Event-driven Integration hat drei wesentliche Komponenten, die in **Bild 1** schematisch dargestellt sind. Deren Beschreibung finden Sie in **Tabelle 2**.

Event-driven Integration bietet die folgenden Vorteile:

- **Echtzeit-Reaktion:** Event-driven Integration ermöglicht es Systemen, in Echtzeit auf Änderungen zu reagieren und sofortige Maßnahmen zu ergreifen.
- **Lose Kopplung:** Durch die Verwendung von Events als Kommunikationsmechanismus sind Systeme weniger stark miteinander verknüpft, was die Flexibilität und Wartbarkeit erhöht.
- **Skalierbarkeit:** Durch den Event-driven-Ansatz kann leichter skaliert werden, da Events unabhängig voneinander verarbeitet werden können und die Last auf verschiedene Komponenten verteilt werden kann.

Den diskutierten Vorteilen stehen drei wesentliche Herausforderungen gegenüber:

- **Komplexität:** Die Verwaltung und Fehlersuche in einem Event-driven-System kann komplexer sein, insbesondere wenn viele Events und Event Consumer beteiligt sind.
- **Latenz:** In einigen Fällen kann die Verarbeitung zu Verzögerungen führen, insbesondere wenn der Event Broker stark ausgelastet ist.
- **Event Sourcing:** Die Verwaltung von Events, einschließlich Speicherung, Wiederholung und Fehlerbehandlung, erfordert sorgfältige Planung und Implementierung.

So weit zu den grundlegenden Konzepten. Ich habe mich bemüht, die dahinterliegenden Ansätze so kompakt wie möglich darzustellen, Ihnen aber zugleich so viel Hintergrundwissen wie nötig an die Hand zu geben, damit die Zusammen-

• Tabelle 2: Beschreibung der Komponenten

Komponente	Beschreibung
Events und Event Sources	Events werden von Event Sources generiert, wenn ein bestimmter Zustand erreicht oder eine Aktion durchgeführt wird. Beispiele für Events sind das Hinzufügen eines neuen Kunden, der Abschluss einer Bestellung oder das Ändern des Lagerbestands.
Event Broker	Events werden von einem Event Broker erfasst und verarbeitet. Dies kann Echtzeitverarbeitung oder Batch-Verarbeitung umfassen, je nach den Anforderungen der Anwendung.
Event Consumer	Event Consumer sind Anwendungen oder Systeme, die auf bestimmte Events reagieren und entsprechende Aktionen ausführen. Beispielsweise könnte ein Inventarsystem auf einen Event reagieren, der eine Bestellung signalisiert, indem es den Lagerbestand aktualisiert.

hänge erkennbar werden. Wenden wir uns nun der Umsetzung der Konzepte zu, um die Datensilos abzureißen, die verhindern, dass Mehrwerte für Unternehmen geschaffen werden. Der Werkzeugkasten, den ich Ihnen hierfür vorstellen möchte, sind die Azure Integration Services.

Azure Integration Services

Azure Integration Services ist eine Platform-as-a-Service (PaaS) von Microsoft und zielt darauf ab, die Integration von Anwendungen, Daten und Prozessen zu vereinfachen. Diese PaaS ermöglicht es Unternehmen, ihre IT-Infrastruktur effizienter zu gestalten, indem sie verschiedene Dienste nahtlos miteinander verbindet. Bild 2 zeigt eine Übersicht der Bausteine, aus denen sich die Azure Integration Services (im Folgenden AIS) zusammensetzen. AIS ist das Nachfolgeprodukt von BizTalk Server (BTS), dessen End of Life 2027 bevorsteht.

Im Gegensatz zu BTS, der on-Prem installiert und betrieben wird, sind die AIS ein reines Cloud-Angebot. Ich habe AIS als Nachfolgeprodukt bezeichnet, was in der Definition nicht ganz richtig ist. Bei AIS handelt es sich um eine Bündelung von vorhandenen Azure-Services, die Sie auch einzeln verwenden können. Aus meiner Sicht ist das Fluch und Segen zugleich. Fluch aus dem Grund, da keine einheitliche Benutzerführung über alle Bausteine existiert und jeder Baustein für sich betrachtet werden muss. Auf der anderen Seite ist das aber wiederum ein Segen, da Sie sich per Mix and Match die für Ihre Anforderungen passenden Bausteine selbst „zusammenklicken“ können.

Azure Logic Apps

Azure Logic Apps [2] ermöglicht die Automatisierung von Workflows und Geschäftsprozessen. Mit einer benutzerfreundlichen grafischen Oberfläche (siehe Bild 3) können Anwender komplexe Workflows in Form von No-Code/Low-Code erstellen, um die verschiedenen Cloud- und On-Premises-Dienste zu integrieren. Dies umfasst auch die Integration von Drittanbieter-Diensten. Dazu stehen weit mehr als 1000 Konnektoren (unter anderem HTTP, Salesforce, Dynamics, M365, SAP und weitere) zur Verfügung. Die Anzahl der Konnektoren wächst täglich.

Azure Logic Apps sind in zwei SKUs (Consumption und Standard) verfügbar. Logic Apps Consumption basieren auf einem Serverless-Architekturmodell und bieten eine nutzungabhängige Abrechnung. Das bedeutet, dass Kosten nur dann entstehen, wenn eine Logic App ausgeführt wird. Mit dieser SKU haben Sie zwei wesentliche Vorteile:

- **Kostenoptimiert:** Abrechnung erfolgt pro Ausführung, ideal für unregelmäßige oder selten auszuführende Workflows.
- **Einfache Skalierung:** Automatische Skalierung ohne manuelle Konfigurationsanforderungen.

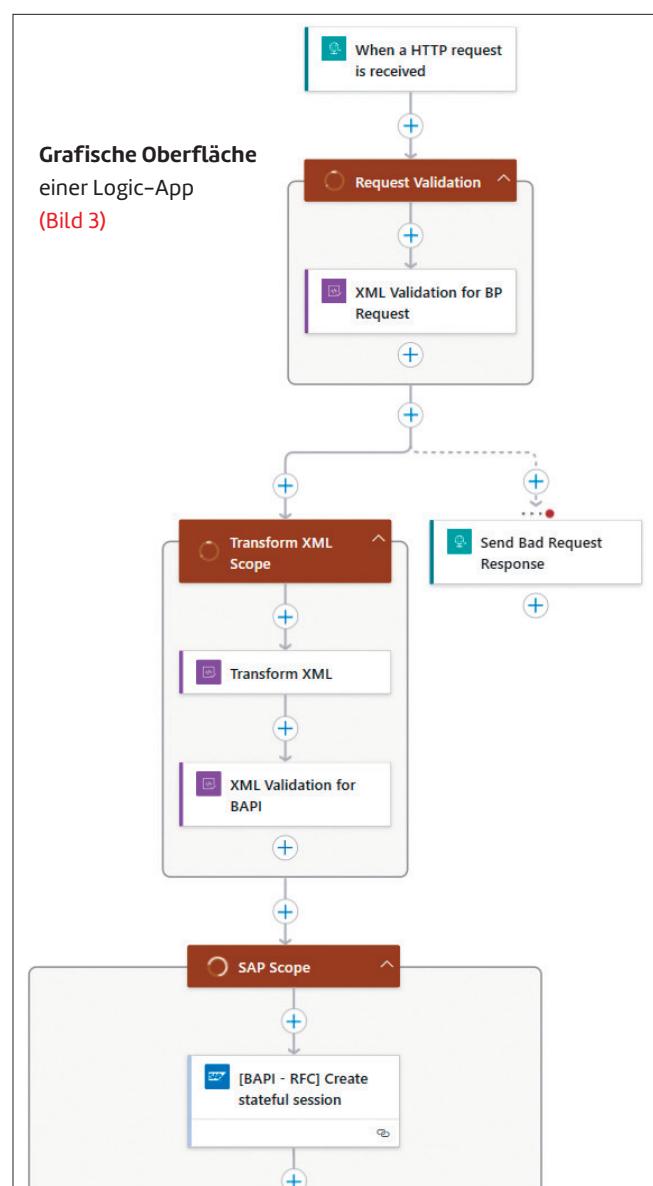
Dem stehen folgende Nachteile gegenüber:

- **Eingeschränkte Kontrolle:** Logic Apps Consumption werden in einer Multi-Tenant-Umgebung ausgeführt, die komplett von Microsoft verwaltet wird.
- **Eingeschränkte Laufzeitumgebung:** Nur Serverless-Modell möglich, wodurch einige Enterprise-Anforderungen wie

etwa Isolated Network Boundaries und andere nicht oder nur sehr schwer umsetzbar sind.

Damit auch Enterprise-Anforderungen mit Logic Apps umgesetzt werden können, hat Microsoft die Standard-SKU eingeführt, die eine dedizierte Laufzeitumgebung zur Verfügung stellt. Diese Laufzeitumgebung erlaubt es sogar, in anderen Clouds oder im eigenen Rechenzentrum betrieben werden zu können. Dies ist besonders im Hinblick auf die Erfüllung von Compliance- oder Gesetzgebungsvorgaben (Datenschutz) ein sehr wichtiger Punkt. Daraus ergeben sich folgende Vorteile bei der Nutzung der Standard-SKU:

- **Flexiblere Hosting-Optionen:** Wie bereits erwähnt, wird der Betrieb in einer Single-Tenant-Umgebung angeboten, was auch eine bessere Performance bedeutet.
- **Stateful und Stateless Workflows:** Nutzer können zwischen zustandsbehafteten oder zustandslosen Workflows wählen.
- **VNET-Integration:** Unterstützung für virtuelle Netzwerke, um Isolated Network Boundaries zu gewährleisten. ►



Listing 1: Eine HelloWorld-Function, die ausgeführt wird, sobald eine HTTP-Anfrage gesendet wird

```

public class HelloWorldFunction
{
    private readonly ILogger<HelloWorldFunction>
        _logger;

    public HelloWorldFunction(
        ILogger<HelloWorldFunction> logger)
    {
        _logger = logger;
    }
}

```

```

[Function("HelloWorldFunction")]
public IActionResult Run([HttpTrigger(
    AuthorizationLevel.Function, "get", "post")]
    HttpRequest req)
{
    _logger.LogInformation("C# HTTP trigger function
        processed a request.");
    return new OkObjectResult(
        "Welcome to Azure Functions!");
}
}

```

Sie bezahlen die Flexibilität mit einer komplexeren Kostenstruktur und einer manuellen Skalierung. In der Standard-SKU bezahlen Sie nicht mehr per Ausführung, sondern für die bereitgestellte und reservierte Rechenleistung, die einen monatlichen Basispreis pro Instanz kostet – egal ob Sie einen Workflow ausführen oder nicht. Instanz ist ein gutes Stichwort: Sie können Skalierungsregeln definieren, die Sie zum Beispiel auf Speicher und CPU anwenden können. Ist ein Schwellwert erreicht, wird eine weitere Instanz automatisch bereitgestellt. Ab diesem Zeitpunkt bezahlen Sie den monatlichen Basispreis für die neue Anzahl der Instanzen. Für unternehmenskritische Workflows mit hohen Performance- und Netzwerk-anforderungen führt kein Weg an der Standard-SKU vorbei.

Azure Functions

Es gibt Umstände, da ist No-Code/Low-Code zu komplex. In diesem Fall sind wenige Zeilen Code effizienter, um ein Geschäftsproblem zu lösen. Logic Apps lässt sich sehr einfach mit Code erweitern, der in Azure Functions [3] bereitgestellt wird.

Sie können Code ausführen, ohne sich um die Bereitstellung und Verwaltung der Infrastruktur kümmern zu müssen, was eine hohe Skalierbarkeit und Kosteneffizienz ermöglicht. Unterstützt werden mehrere Programmiersprachen, darunter C#, JavaScript, Python, PowerShell, Java und TypeScript. **Listing 1** zeigt ein Beispiel einer *HelloWorld*-Function.

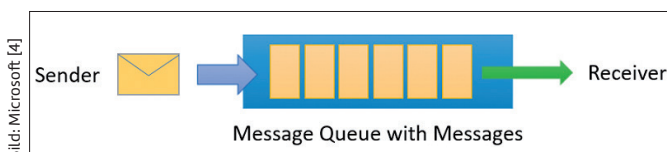
Für die Bereitstellung von Azure Functions gibt es unter anderem folgende Hosting-Optionen: den Consumption Plan, den Premium Plan und den Dedicated Plan. Der Consumption Plan basiert auf einem serverlosen Modell, bei dem Abrechnung und Skalierung vollständig von Azure verwaltet werden. Dabei wird Code nur ausgeführt, wenn Events ausgelöst werden. Allerdings gibt es Einschränkungen wie Kaltstarts, eine maxima-

le Laufzeit von standardmäßig fünf Minuten und begrenzte Netzwerkoptionen ohne direkte VNet-Integration (ähnlich den fehlenden Enterprise-Features bei Logic Apps Consumption).

Der Premium Plan adressiert einige der Nachteile des Consumption Plans, indem er vorgewärmte Instanzen bereitstellt, Kaltstarts eliminiert und eine erweiterte Netzwerkintegration ermöglicht. Funktionen können ohne Zeitbegrenzung laufen, und die Leistung ist insgesamt höher. Allerdings bezahlen Sie hier für die reservierten Instanzen einen monatlichen Basispreis, und es ist ein gewisser manueller Skalierungsaufwand erforderlich. Dass dies sehr ähnlich klingt wie bei den Logic Apps Standard, ist kein Zufall, denn Logic Apps Standard basieren auf der Azure Functions Runtime. Zwischen dem Premium Plan und dem Consumption Plan gibt es noch den Flex Consumption Plan, der kosteneffizienter ist und gleichzeitig einige Features des Premium Plans wie das Verhindern von Kaltstarts mitbringt.

Für Entwickler bietet Azure Functions eine Vielzahl von Einsatzmöglichkeiten. Zum Beispiel lassen sich APIs schnell und einfach als serverlose Funktionen bereitstellen oder Backend-Prozesse automatisieren. Funktionen können durch verschiedene Trigger ausgelöst werden, beispielsweise HTTP-Anfragen, Datenbankänderungen oder Nachrichten in Warteschlangen. Wer Workflows nicht grafisch, sondern per Code erstellen möchte, dem bieten die Azure Functions eine Erweiterung namens Durable Functions; sie ermöglicht die Erstellung von zustandsbehafteten, lange laufenden Workflows. Mit Durable Functions können Sie komplexe Orchestrierungs- und Steuerungsabläufe umsetzen, indem sie mehrere Azure Functions miteinander koordinieren. Diese Orchestrierung erfolgt über verschiedene Muster, darunter Function Chaining, Fan-Out/Fan-In und Async HTTP APIs. Ein typischer Anwendungsfall für Durable Functions ist die Automatisierung von Geschäftsprozessen, die mehrere Schritte umfassen, oder das Handling von wiederholten Verzögerungen, wie sie beispielsweise in Genehmigungsworkflows auftreten.

An dieser Stelle können wir zusammenfassen, dass Logic Apps ein Design-first-Werkzeug ist und Azure Functions ein Code-first-Werkzeug, um jeweils Geschäftsprozesse zu erstellen und zu automatisieren.



Einfache Warteschlange mit einem Receiver (Bild 4)

Azure Service Bus

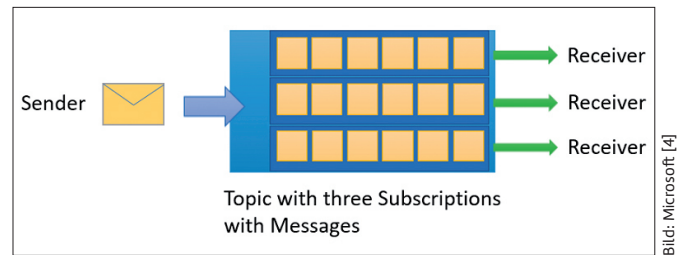
Azure Service Bus [4] dient als Message Broker, um Daten sicher und asynchron zwischen verschiedenen Systemen zu übertragen, was den Messaging-Dienst ideal für Microservices-Architekturen und Enterprise-Integration macht. Dabei bietet Azure Service Bus zwei zentrale Messaging-Modelle: Queues für eine Punkt-zu-Punkt-Kommunikation, bei der Nachrichten in einer Warteschlange gespeichert und von einem einzelnen Empfänger verarbeitet werden (siehe Bild 4), sowie Topics & Subscriptions, die ein Publish-Subscribe-Muster ermöglichen, bei dem eine Nachricht an mehrere Abonnenten gesendet werden kann (siehe Bild 5).

Zusätzlich zu diesen grundlegenden Funktionen bietet Azure Service Bus erweiterte Features wie FIFO-Verarbeitung (First in, first out), Duplicate Detection, Scheduled Messages, Dead-Lettering für fehlerhafte Nachrichten sowie Message Sessions, um zusammenhängende Nachrichten zu gruppieren. Je nach Anforderungen gibt es drei Preis- und Leistungsstufen: Basic, das einfache Warteschlangen ohne erweiterte Funktionen wie Topics & Subscriptions bereitstellt; Standard, das erweiterte Messaging-Funktionen wie Topic-basierte Kommunikation unterstützt; sowie Premium, das dedizierte Ressourcen für höhere Leistung, bessere Skalierbarkeit und erweiterte Sicherheitsoptionen wie Virtual-Netzwerk-Integration (VNet) bietet.

Azure Service Bus kann in verschiedenen Szenarien eingesetzt werden, darunter asynchrone Verarbeitung, um Anwendungen zu entkoppeln und Engpässe zu vermeiden, Event-driven Integration, die eine Verteilung von Ereignissen an mehrere Systeme ermöglicht, Microservices-Kommunikation, um eine zuverlässige und skalierbare Interprozesskommunikation sicherzustellen, sowie die Integration mit Azure-Diensten wie Azure Functions, Logic Apps und Event Grid für automatisierte Workflows. Service Bus ist quasi das WhatsApp für Ihre Anwendungen und Prozesse, nur ohne blauen Haken (einen schönen Gruß an Gregor Biswanger an dieser Stelle).

Azure Event Grid

Azure Event Grid [5] ist ein vollständig verwalteter Event Broker, der die Entwicklung von Event-driven Integration erleichtert. Er ermöglicht eine zuverlässige Übermittlung von Events zwischen Quellen (Event Sources/Event Publisher) und Zielen (Event Consumer/Eventhandler) und eignet sich besonders für die Echtzeitkommunikation in verteilten Systemen, Microservices-Architekturen und serverlosen Anwendungen. Im Gegensatz zum Azure Service Bus, der für Nach-



Publish & Subscribe mit einem Sender und mehreren Receivern (Bild 5)

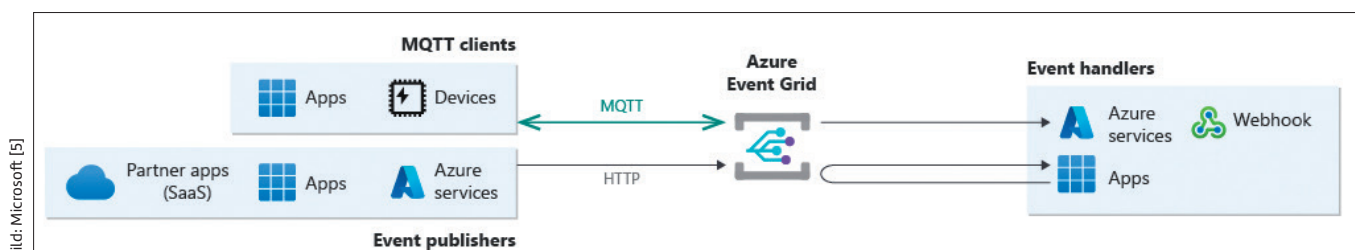
richtenwarteschlangen und asynchrone Verarbeitung konzipiert ist, basiert Event Grid auf dem Konzept der Event-Weiterleitung und verteilt Ereignisse direkt an mehrere Empfänger. Event Grid kann auch als MQTT-Broker eingesetzt werden. Das ist gerade im Zusammenhang mit IoT ein sehr interessanter Anwendungsfall.

In Bild 6 sehen sie die Funktionen des Event Grid. Azure Event Grid arbeitet nach dem Publish-Subscribe-Modell, wobei verschiedene Azure-Dienste und externe Systeme als Event Sources und/oder Event Consumer fungieren können, darunter Azure Functions, Logic Apps, Azure Service Bus und benutzerdefinierte Anwendungen mit Webhooks. Durch dieses Modell wird ein effizientes und skalierbares Event Processing ermöglicht, da mehrere Abonnenten gleichzeitig über relevante Ereignisse informiert werden können. Zentrale Vorteile von Azure Event Grid sind die geringe Latenz und hohe Skalierbarkeit, was es ideal für Anwendungen und Prozesse macht, die in Echtzeit auf Änderungen reagieren müssen.

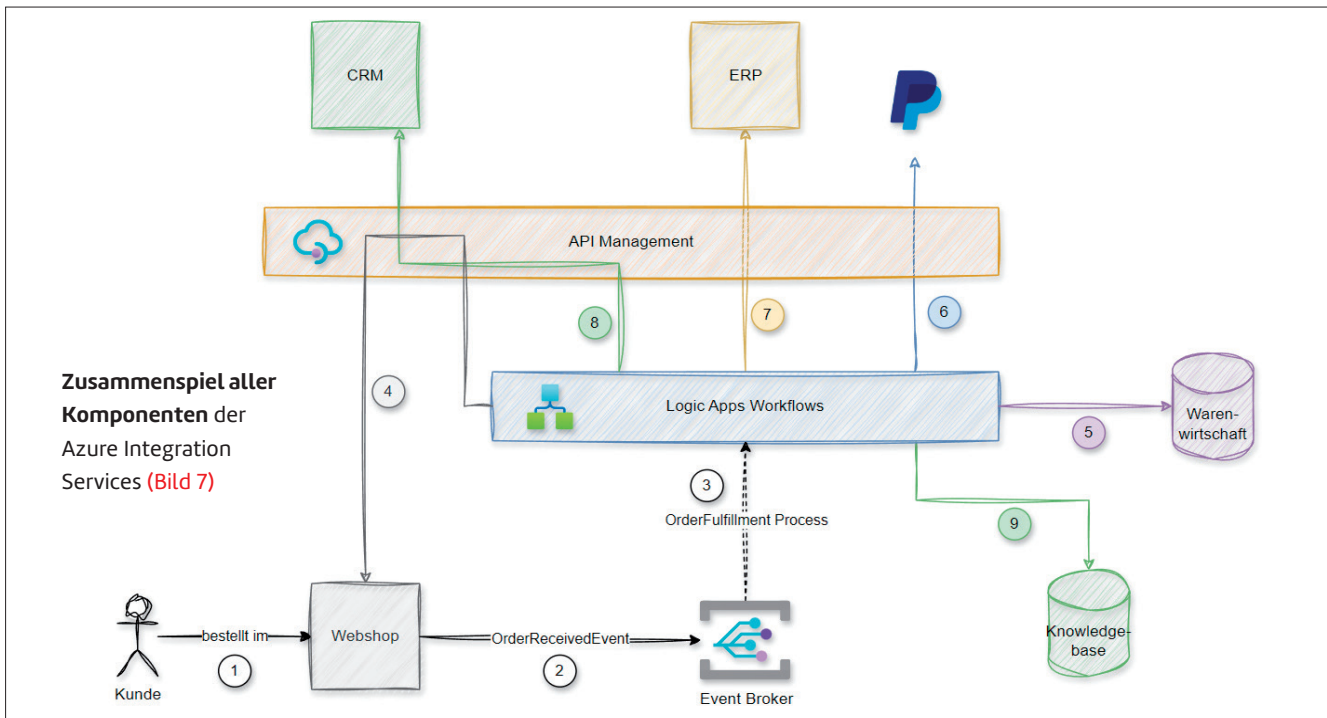
Hinsichtlich der Zuverlässigkeit und Sicherheit bietet Azure Event Grid eine garantierte Zustellung von Events, indem es eine At-least-once-Zustellung sicherstellt. Falls ein Event nicht erfolgreich zugestellt wird, erfolgt ein automatisches Retry mit exponentiellem Back-off, sodass kritische Ereignisse nicht verloren gehen. Zudem ermöglicht Event Grid die Integration mit Virtual Networks (VNet) für eine sichere Kommunikation. Azure Event Grid ist in verschiedenen Preisstufen verfügbar, wobei die Abrechnung basierend auf der Anzahl der gesendeten Events erfolgt.

Azure API Management

Azure API Management [6] ist ein Service, der Unternehmen dabei unterstützt, APIs sicher, effizient und skalierbar bereitzustellen, zu überwachen und zu verwalten. Wie schon weiter oben im Artikel beschrieben, kommt APIs im Hinblick auf ►



Schematische Darstellung des Azure Event Grid beim Einsatz der Protokolle MQTT und HTTP (Bild 6)



Integration eine entscheidende Rolle zu, indem sie verschiedene Systeme, Microservices und Anwendungen miteinander verbinden. Azure API Management bietet eine zentrale Plattform, um APIs sowohl für interne als auch externe Nutzer bereitzustellen und deren Nutzung zu kontrollieren.

Ein wesentliches Merkmal von Azure API Management ist das API-Gateway, das als zentrale Schnittstelle für alle API-Aufrufe fungiert. Es übernimmt wichtige Funktionen wie Authentifizierung, Autorisierung, Bandbreitenbegrenzung (Rate Limiting), Caching und Protokollierung. Dadurch wird sichergestellt, dass APIs nicht nur leistungsfähig und sicher, sondern auch vor Missbrauch geschützt sind. Sie können Richtlinien (Policies) definieren, um API-Anfragen zu transformieren, Anfragen zu validieren oder Antwortzeiten zu optimieren. Ein weiteres zentrales Feature ist das Developer Portal, das es Entwicklern ermöglicht, APIs zu entdecken, zu testen und zu nutzen. Dieses Portal bietet eine interaktive API-Dokumentation, Beispielanfragen und eine Testumgebung, wodurch die Integration von APIs vereinfacht wird. Das Portal kann an die eigenen Bedürfnisse angepasst werden, um eine konsistente CI und Benutzererfahrung bereitzustellen.

Azure API Management unterstützt verschiedene Sicherheitsmechanismen, darunter OAuth 2.0, OpenID Connect und API-Key-basierte Authentifizierung. Zudem ermöglicht die Integration mit Entra ID eine sichere und zentrale Verwaltung von Benutzerzugriffen. Durch diese Mechanismen können Unternehmen sicherstellen, dass nur autorisierte Nutzer auf ihre APIs zugreifen. Einige der Sicherheitsfunktionen und andere Features stehen Ihnen nur in bestimmten Preismodellen beziehungsweise Tiers zur Verfügung, die sich auf verschiedene Anforderungen anpassen lassen. Die Stufen reichen von Developer Tier (für Testumgebungen) über Basic- und Standard Tier (für kleine bis mittlere Workloads) bis hin

zum Premium Tier (für globale und hochverfügbare API-Infrastrukturen mit Multi-Region-Unterstützung und Virtual Network Integration).

Azure Data Factory

Azure Data Factory [7] ist ein Service, der für die Orchestrierung und Automatisierung von Datenpipelines entwickelt wurde. Er ermöglicht die Integration, Transformation und Bewegung von Daten aus verschiedenen Quellen in Data Warehouses, Data Lakes oder andere Zielsysteme. Azure Data Factory (ADF) ist besonders für ETL- (Extract, Transform, Load) und ELT-Prozesse (Extract, Load, Transform) in Big-Data- und Analyse-Architekturen geeignet. Ein zentrales Konzept von Azure Data Factory ist die Pipeline, eine logische Abfolge von Aktivitäten zur Datenverarbeitung. Diese Pipelines bestehen aus verschiedenen Komponenten wie Datasets (Datenquellen und -ziele), Linked Services (Verbindungsinformationen zu externen Systemen) und Activities (Datenbewegung und -transformation). ADF bietet eine große Auswahl an Konnektoren, die die Anbindung an relationale Datenbanken, NoSQL-Datenbanken, Cloud-Speicher, SaaS-Anwendungen und On-Premises-Systeme erleichtern.

Für die Datenverarbeitung unterstützt Azure Data Factory verschiedene Methoden. Die Copy Activity ermöglicht es, Daten effizient zwischen Quellen und Zielen zu verschieben, während Mapping Data Flows eine visuelle Umgebung für Transformationen bietet, ohne dass umfangreiche Code-Kenntnisse erforderlich sind. Alternativ kann Azure Data Factory mit Azure Databricks, Azure Synapse Analytics oder SQL Server Integration Services (SSIS) integriert werden, um komplexere Datenverarbeitungsaufgaben zu bewältigen. Ein weiteres wichtiges Feature ist die Trigger-basierte Automatisierung, die Pipelines basierend auf Zeitplänen, Events oder manuellen Auslösern

startet. Dies ermöglicht eine effiziente und skalierbare Verarbeitung großer Datenmengen in Echtzeit oder im Batch-Modus. ADF ist in verschiedenen Preismodellen verfügbar, wobei die Abrechnung basierend auf der Nutzung von Pipeline-Ausführungen, Data Flows und Datenbewegung erfolgt.

Vorteile der Azure Integration Services

- **Skalierbarkeit:** Azure Integration Services bietet hohe Skalierbarkeit und Flexibilität, um den wachsenden Anforderungen moderner Unternehmen gerecht zu werden.
- **Kosteneffizienz:** Durch die Nutzung von Cloud-Ressourcen können Unternehmen ihre IT-Kosten optimieren und nur für tatsächlich genutzte Dienste bezahlen.
- **Sicherheit:** Mit integrierten Sicherheitsfunktionen wie Authentifizierung, Autorisierung und Verschlüsselung können Unternehmen sicherstellen, dass ihre Daten und Anwendungen geschützt sind.
- **Einfache Verwaltung:** Die zentrale Verwaltung und Überwachung von Integrationen erleichtert die Administration und Fehlerbehebung.

Beispiel

Lassen Sie uns die Puzzleteile nun zusammenstecken und die vorgestellten Integrationsansätze mit den Azure Integration Services in einem stark vereinfachten, nicht vollständigen Beispiel anwenden. In **Bild 7** sehen Sie ein gängiges Szenario. Ein Kunde bestellt Artikel über einen Webshop – Schritt (1) im Schaubild. Der Webshop informiert über den Event *OrderReceivedEvent* die am Prozess beteiligten Systeme über die Zustandsänderung. Dazu wird der Event dem Event Broker übergeben (2). Der Event Broker informiert nun die Systeme, die sich für diesen Event interessieren (Abonnenten des Events) (3). Im vorliegenden Beispiel wird der *OrderFulfillment Process* gestartet, der als Logic-Apps-Workflow abgebildet ist.

Mithilfe der Logic Apps lassen sich Systeme über Konnektoren anbinden und orchestrieren – ideal geeignet für unseren Anwendungsfall. Im Event werden Metadaten mitgeliefert, jedoch nicht die Details der Bestellung. Diese Details werden in Schritt (4) in **Bild 7** direkt vom Webshop abgeholt. Aufrufe auf den Webshop werden nicht direkt durchgeführt, sondern über das API-Gateway des API-Managements, das alle APIs in diesem Szenario beinhaltet und somit als zentrale Instanz verwaltet. Sobald die Logic App die Bestelldetails gelesen hat, folgt in Schritt (5) die Reservierung der Artikel im Warenwirtschaftssystem. Das Warenwirtschaftssystem kann direkt über den SQL-Konnektor angebunden werden.

Die Abwicklung der Zahlung erfolgt über PayPal in Schritt (6) in **Bild 7** über die Aufrufe im API-Management. Ist die Zahlungsabwicklung erfolgreich, wird der Auftrag im ERP angelegt (7) und die Bestellhistorie des Kunden im CRM gepflegt (8). Auch diese Aufrufe werden über das API-Management zugänglich gemacht. Zwischen allen Systemen muss in der Regel eine Transformation der Daten in unterschiedliche Datenformate vorgenommen werden, damit die Zielsysteme die Daten auch verstehen. In Logic Apps können Sie dies über einen grafischen Mapper oder über eingebaute XML- beziehungsweise JSON-Operationen durchführen.

Sie haben nun eine Vorstellung davon, wie die Azure Integration Services bei der Umsetzung solcher Integrationen eingesetzt werden können. Noch etwas wird durch das Szenario klar: Sie müssen nicht alle Bausteine der Azure Integration Services einsetzen, sondern können dediziert die Bausteine der Suite wählen, die Sie für die Umsetzung Ihrer Anforderungen benötigen.

Ihnen wird sicherlich aufgefallen sein, dass wir Schritt (9) in **Bild 7** noch nicht betrachtet haben. Diesen Schritt habe ich mir absichtlich für den Schluss aufgehoben. Da Sie in diesem Szenario in der Logic App viele relevante Informationen für eine Bestellung haben, können Sie diese Daten nutzen und sie zum Beispiel in einem Vector Store, der Knowledge Base, speichern und für die Anreicherung von Large Language Models (LLMs) zur Verfügung stellen. Damit schaffen Sie die Möglichkeit, Ihre Anwendungen mit innovativen Technologien wie künstlicher Intelligenz auszustatten. So habe ich es Ihnen ja zu Beginn des Artikels versprochen.

Fazit

Innovation ohne Integration ist wie ein Auto ohne Räder – vielversprechend, aber nicht einsatzfähig. Unternehmen, die ihre digitalen Initiativen erfolgreich vorantreiben wollen, müssen von Anfang an auf eine durchdachte Integrationsstrategie setzen. Wer innovative Technologien nutzt, aber bestehende Prozesse und Systeme ignoriert, riskiert Insellösungen und verpasste Chancen. Erst durch eine reibungslose Integration entfalten Innovationen ihr volles Potenzial. Microsoft bietet mit den Azure Integration Services eine leistungsstarke Suite von Tools, die Unternehmen dabei unterstützen, ihre Anwendungen, Daten und Prozesse nahtlos zu integrieren. ■

- [1] *API Sprawl*, www.dotnetpro.de/SL2504-05AzureIntegrationServices1
- [2] *Microsoft Learn, Was ist Azure Logic Apps?*, www.dotnetpro.de/SL2504-05AzureIntegrationServices2
- [3] *Microsoft Learn, Übersicht zu Azure Functions*, www.dotnetpro.de/SL2504-05AzureIntegrationServices3
- [4] *Microsoft Learn, Was ist Azure Service Bus?*, www.dotnetpro.de/SL2504-05AzureIntegrationServices4
- [5] *Microsoft Learn, Was ist Azure Event Grid?*, www.dotnetpro.de/SL2504-05AzureIntegrationServices5
- [6] *Microsoft Learn, Was ist Azure API Management?*, www.dotnetpro.de/SL2504-05AzureIntegrationServices6
- [7] *Microsoft Learn, Was ist Azure Data Factory?*, www.dotnetpro.de/SL2504-05AzureIntegrationServices7



Sebastian Meyer

ist Lead Architect bei der QUIBIQ Hamburg GmbH und Microsoft MVP. Er berät seit vielen Jahren Unternehmen bei der Architektur und Umsetzung von Integrationsplattformen, ist Sprecher, Trainer und Autor.

www.linkedin.com/in/sebastianmeyerit

dnpCode

A2504-05AzureIntegrationServices

QUANTENPOWER AUS DER CLOUD

Was Azure Quantum möglich macht

So lässt sich die revolutionäre Quantentechnologie heute schon nutzen.

Stellen wir uns eine Welt vor, in der Medikamente nicht mehr durch jahrelange Laborversuche, sondern innerhalb weniger Stunden am Computer entwickelt werden. Eine Welt, in der Verkehrsflüsse in Echtzeit so optimiert werden, dass es keine Staus mehr gibt. Eine Welt, in der Verschlüsselungen, die heute als sicher gelten, sich in Sekunden knacken lassen – oder in der sie durch Quantenmechanismen sogar absolut unüberwindbar werden.

Was heute noch wie Science-Fiction klingt, könnte mit Quantencomputern Realität werden. Doch während Unternehmen und Wissenschaftler fieberhaft an der Quantenhardware der Zukunft arbeiten, stellt sich eine zentrale Frage: Wie kann man diese revolutionäre Technologie heute schon in der Softwareentwicklung nutzen? Die Antwort darauf liegt in der Cloud – und Microsofts Azure Quantum [1] spielt dabei eine Schlüsselrolle.

Quantencomputing als Gamechanger

Quantencomputer sind nicht einfach schnellere Versionen klassischer Computer – sie funktionieren grundlegend anders. Während herkömmliche Computer mit Bits arbeiten, die entweder den Zustand 0 oder 1 aufweisen, nutzen Quantencomputer Qubits, die dank Superposition mehrere Zustände gleichzeitig annehmen können. Dies erlaubt es Quantenrechnern, viele Berechnungen parallel durchzuführen, anstatt sie Schritt für Schritt abzuarbeiten.

Ein weiteres revolutionäres Konzept des Quanten-Computing ist die Quantenverschränkung. Hierbei werden Qubits so miteinander gekoppelt, dass die Änderung eines Qubits sofort Auswirkungen auf das andere hat – unabhängig von deren räumlicher Entfernung. Diese Eigenschaften ermöglichen eine bisher unerreichte Rechenleistung und revolutionäre Ansätze für Optimierung, Simulation und Kryptografie.

Die potenziellen Anwendungsbereiche der Quantencomputer sind gewaltig:

- **Kryptografie:** Klassische Verschlüsselungsmethoden könnten gebrochen, aber auch unknackbare Quantenkryptografie-Verfahren ermöglicht werden.
- **Material- und Medikamentenentwicklung:** Quantencomputer simulieren Moleküle auf atomarer Ebene, was neue Materialien und Medikamente schneller verfügbar macht.
- **Optimierungsprobleme:** Logistik, Finanzmärkte oder Verkehrssteuerung profitieren von extrem schnellen Berechnungen optimaler Lösungen.
- **Künstliche Intelligenz:** Maschinelles Lernen könnte durch Quantenalgorithmen eine völlig neue Dimension erreichen.

Trotz dieser bahnbrechenden Möglichkeiten bleibt eine zentrale Herausforderung: Der Betrieb eines eigenen Quantencomputers ist extrem komplex und teuer. Und das ist genau der Punkt, an dem Quantencomputing in der Cloud ins Spiel kommen kann.

● **Tabelle 1: Quantencomputing-Anbieter in der Cloud**

Plattform	Unterstützte Hardware	Besonderheiten
Azure Quantum	IonQ (Ionenfallen), Quantinuum, Rigetti (supraleitende Qubits), Microsoft (Majorana-Qubits – topologische Qubits in Entwicklung)	Integration in Microsoft Azure, hybride Quanten-Klassik-Algorithmen
IBM Quantum	IBM-eigene supraleitende Qubits	Open-Source-Framework Qiskit, breite Entwickler-Community
Google Quantum AI	Sycamore-Prozessor (supraleitende Qubits)	Fokus auf Quantenhardware-Entwicklung
Amazon Braket	IonQ (Ionenfallen), IQM und Rigetti (supraleitende Qubits), QuEra (Neutral-Atom)	Multi-Hardware-Plattform mit AWS-Integration
Leap – D-Wave Quantum Cloud	D-Wave-Quantencomputer (Quantum Annealing)	Spezialisiert auf Optimierungsprobleme, erster kommerzieller Quantencomputer
Quandela Cloud	Photonik-Quantencomputer (Photonic Qubits)	Europäischer Anbieter, Fokus auf optische Quantenrechner

Quantencomputing in der Cloud

Quantencomputer benötigen spezielle Umgebungen: Supraleitende Qubits müssen bis nahe an den absoluten Nullpunkt (-273,15 °C) gekühlt werden, während Ionenfallen-Quantencomputer hochpräzise Laser benötigen. Die Einrichtung und die Wartung solcher Systeme erfordern nicht nur spezialisierte Hardware, sondern auch tiefgehendes Expertenwissen. Darum ist das Quantencomputing in der Cloud die beste und meist einzige tragbare Nutzungsoption. Durch die Integration von Quantencomputern in bestehende Cloud-Plattformen erhalten Entwickler, Wissenschaftler und Unternehmen Zugang zu modernster Quantenhardware, ohne eigene Maschinen betreiben zu müssen. Dies bietet entscheidende Vorteile:

- **Skalierbarkeit:** Je nach Bedarf kann die passende Rechenleistung flexibel genutzt werden.
- **Kosteneffizienz:** Keine Investitionen in teure Hardware – die Abrechnung erfolgt nur für die tatsächlich genutzten Ressourcen.
- **Hybride Algorithmen:** Kombination aus klassischer und Quantenhardware für optimale Lösungen.
- **Schnelle Updates:** Direkter Zugriff auf die neuesten technologischen Fortschritte ohne Hardware-Upgrades vor Ort.

Microsoft hat diese Vorteile früh erkannt und mit Azure Quantum eine Plattform geschaffen, die den Zugang zu verschiedenen Quantenarchitekturen über die Cloud ermöglicht. Doch Microsoft ist nicht der einzige Anbieter in diesem Bereich. Neben Azure Quantum gibt es auch andere Cloud-Plattformen, die Zugriffsmöglichkeiten auf Quantencomputer bereitstellen (siehe [Tabelle 1](#)).

Während namhafte Anbieter wie IBM [2] und Google [3] sowie aufstrebende wie D-Wave [4] und Quandella [5] eigene Quantencomputer entwickeln und betreiben, setzen Microsoft Azure Quantum und Amazon Braket [6] auf ein Ökosystem mit verschiedenen Hardware-Partnern. Außerdem führt Microsoft eigene Forschung an topologischen Qubits durch. Diese könnten in Zukunft besonders robust gegenüber Fehlern sein – ein entscheidender Vorteil für skalierbare Quantenrechner.

Azure Quantum unterstützt verschiedene Quantenprozessoren (QPU-Architekturen), darunter:

- **Supraleitende Qubits (Rigetti, Quantinuum):** Weit verbreitet; die hohe Fehlerrate erfordert stärkere Fehlerkorrektur.
- **Ionenfallen (IonQ):** Langlebige Qubits mit hoher Präzision, aber langsame Gatterzeiten.
- **Topologische Qubits (Microsoft):** Noch in Entwicklung, versprechen eine höhere Stabilität und geringere Fehleranfälligkeit [7].

Detaillierter wurde das Quantencomputing als Schwerpunktthema bereits in der Februar-Ausgabe der dotnetpro unter [8] und [9] dargestellt. Interessierte Leser können dort und in den dort verlinkten Quellen weitere Informationen nachschlagen.

Was ist Azure Quantum?

Azure Quantum [1] ist der Quantencomputing-Clouddienst auf der Azure-Cloud-Plattform, der einen offenen, flexiblen und zukunftssicheren Weg zum Quantencomputing bietet. Er

verfolgt einen hybriden Ansatz, der sich an unterschiedliche Anwendungsfälle und Entwickleranforderungen anpasst. Dabei umfasst Azure Quantum eine breite Palette an Quantencomputing-Lösungen:

- Quantenhardware von branchenführenden Anbietern wie IonQ, Quantinuum und Rigetti,
- Quantensoftware für die Entwicklung und Optimierung von Quantenalgorithmen,
- Quantendienste zur Simulation, Ressourcenschätzung und für hybride Quantenklassik-Berechnungen.

Mit Azure Quantum können Entwickler und Unternehmen Quantenprogramme auf echter Quantenhardware ausführen, Quantenalgorithmen simulieren und den Ressourcenbedarf zukünftiger skalierter Quantencomputer berechnen. Ein entscheidender Vorteil von Azure Quantum ist die tiefe Integration in das Microsoft-Ökosystem, wodurch Nutzer hybride Workflows mit klassischen Cloud-Computing-Ressourcen kombinieren können. Dadurch lassen sich Quanten- und klassische Berechnungen effizient verbinden – ein entscheidender Schritt auf dem Weg zur praktischen Nutzung von Quantencomputing.

Azure Quantum bietet eine Vielzahl von Funktionen, die es zu einer der leistungsstärksten und flexibelsten Quantenplattformen machen. Die wichtigsten Features sind diverse Quantencomputer-Architekturen, Simulationen und Ressourcenschätzung, hybride Workflows sowie das Azure Quantum Development Kit (QDK).

Diverse Quantencomputer-Architekturen

Azure Quantum unterstützt mehrere Quantencomputer-Technologien, die für unterschiedliche Anwendungsfälle optimiert sind. Nutzer können zwischen folgenden Anbietern (Quantum-Architekturen) wählen:

- **IonQ (Ionenfallen-Quantencomputer):** Hohe Stabilität und geringe Dekohärenz, jedoch langsame Gatteroperationen.
- **Quantinuum (supraleitende Qubits):** Supraleitende Qubits mit fortschrittlicher Fehlerkorrektur.
- **Rigetti (supraleitende Qubits):** Schnellere Berechnungen, aber höhere Fehleranfälligkeit.

Diese Vielfalt erlaubt es Entwicklern, die am besten geeignete Hardware für ihre spezifischen Quantenalgorithmen auszuwählen und zu testen.

Simulationen und Ressourcenschätzung

Nicht alle Quantenalgorithmen müssen sofort auf realer Hardware ausgeführt werden. Mit Azure Quantum können Nutzer ihre Quantenprogramme zunächst in klassischen Simulationsumgebungen testen. Diese bieten eine kontrollierte Umgebung, um Algorithmen zu optimieren, bevor sie auf echte Qubits angewendet werden.

Zusätzlich stellt Microsoft den Azure Quantum Resource Estimator [10] bereit. Dieses Tool ermöglicht es, den Rechen- und Energiebedarf zukünftiger Quantencomputer für spezifische Probleme abzuschätzen. Besonders für Unternehmen und Forschungseinrichtungen ist dies ein wichtiges Feature, ►

um die Skalierbarkeit und Effizienz ihrer Quantenlösungen frühzeitig zu bewerten und darauf basierend effiziente Lösungsarchitekturen aufbauen.

Hybride Workflows

Da heutige Quantencomputer noch nicht leistungsfähig genug sind, um alle Berechnungen alleine durchzuführen, bietet Azure Quantum die Möglichkeit, klassische und Quanten-Berechnungen zu kombinieren. Dieser hybride Ansatz [11] nutzt leistungsstarke klassische Computer für Vorverarbeitungsschritte und delegiert nur spezifische Rechenoperationen an Quantencomputer.

Azure Quantum verfolgt eine zukunftsorientierte Vision für hybrides Quantencomputing, bei der einige Architekturen bereits betriebsbereit sind, während andere – wie zum Beispiel Microsofts topologische Qubits – aktiv entwickelt werden. Durch die tiefgreifende Integration in Azure-Dienste wie zum Beispiel Azure Functions, Azure Machine Learning und andere können Unternehmen ihre bestehenden Cloud-Workflows nahtlos mit Quantenalgorithmen erweitern.

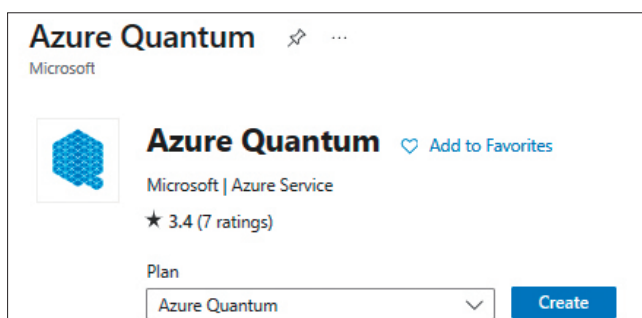
Dieser Ansatz ermöglicht es, praktische Anwendungen zu entwickeln, die von den Stärken beider Rechenmodelle profitieren und gleichzeitig auf zukünftige technologische Fortschritte vorbereitet sind.

Azure Quantum Development Kit

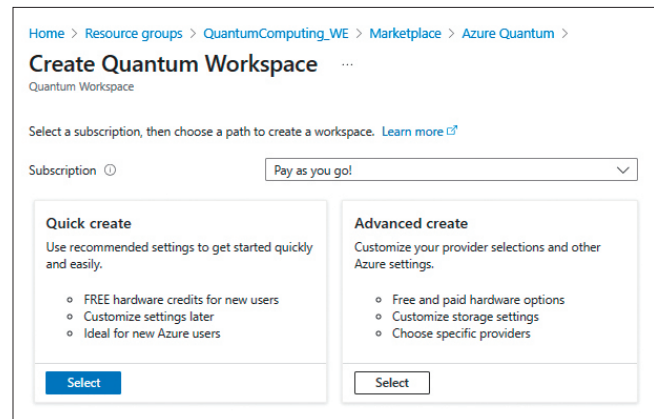
Das Azure Quantum Development Kit (QDK) [12] ist eine leistungsstarke lokale Entwicklungsumgebung, die als Erweiterung für Visual Studio Code verfügbar ist [13]. In VS-Code-Web (Quantum) [14] hingegen ist Azure QDK bereits vorinstalliert. Das Herzstück des QDK ist die Programmiersprache Q#, die speziell für Quantencomputing entwickelt wurde.

Azure QDK richtet sich an Entwickler, die professionelle Quantenprogramme in Q# schreiben und auf echter Quantenhardware ausführen möchten. Die Hauptfunktionen von Azure QDK sind:

- Unterstützung für VS Code, Visual Studio und Python für maximale Flexibilität bei der Entwicklung.
- Erstellung und Debugging von Q#-Programmen in einer vollständigen Entwicklungsumgebung.
- Syntaxhervorhebung, intelligente Fehleranalyse und Autovervollständigung.
- Jupyter-Notebook-Unterstützung, um interaktive Quantenalgorithmen zu entwickeln.



Azure Quantum Service erstellen (Bild 1)



Quantum Workspace erstellen – Schritt 1 (Bild 2)

- Bibliotheken für Optimierung, Simulation und Fehlerkorrektur, die Entwicklern helfen, komplexe Algorithmen effizient umzusetzen.
- Visualisierung von Quantengattern und Schaltungen.
- Nahtlose Integration mit Azure Quantum, sodass Programme direkt aus der Entwicklungsumgebung in der Cloud ausgeführt werden können (direkte Verbindung zu Azure-Quantum-Arbeitsbereichen).
- Ressourcenschätzung, um den Qubit-Bedarf zukünftiger Quantenalgorithmen abzuschätzen.

Dank dieser umfangreichen Toolkette können Entwickler effizient an Quantenanwendungen arbeiten, ohne tief in die Hardware-Details einsteigen zu müssen.

Azure-Quantum-Arbeitsbereich erstellen

Um Quantenprogramme auf Azure Quantum auszuführen, muss zunächst ein Azure-Quantum-Arbeitsbereich erstellt werden. Dieser Arbeitsbereich ist die zentrale Verwaltungsinstanz für Quantenprojekte und bietet Zugriff auf Quantenhardware, Jupyter-Notebooks, Job-Management und vieles mehr. Sie erstellen ihn mit folgenden Schritten:

- Melden Sie sich mit den Anmeldeinformationen Ihres Azure-Abonnements im Microsoft Azure Portal an [15].
- Wählen Sie *Ressource erstellen (Create a resource)* und suchen Sie nach *Azure Quantum*. In den Ergebnissen sollte eine Azure-Quantum-Kachel erscheinen. Wählen Sie diese aus und klicken Sie auf *Erstellen (Create)*, um das Konfigurationsformular zu öffnen (siehe Bild 1).
- Wählen Sie ein Azure-Abonnement, dem der neue Arbeitsbereich zugeordnet werden soll (siehe Bild 2).
- Nun stehen Ihnen zwei Optionen zur Verfügung (siehe Bild 2): *Schnellerstellung (Quick create)* – empfohlen für einen schnellen Einstieg) oder *Erweiterte Erstellung (Advanced create)* – für eine individuelle Konfiguration). Um die Schnellerstellung zu verwenden, müssen Sie Besitzer des Abonnements sein, das im vorherigen Schritt ausgewählt wurde.
- Bei Schnellerstellung geben Sie Arbeitsbereichsnamen (*Workspace name*) und Region an und klicken anschließend auf *Erstellen (Create)*, siehe Bild 3).

Arbeitsgruppen-Name sowie Storage-Account werden bei der Schnellerstellung eines neuen Arbeitsbereichs automa-

Home > Resource groups > QuantumComputing_WE > Marketplace > Azure Quantum >

Create Quantum Workspace

Quantum Workspace

Enter a workspace name, then click create. It takes about 5 minutes for workspace creation to complete. [Learn more](#)

Subscription: Pay as you go!

Resource group: AzureQuantum

Workspace name: quantum-workspace-1

Region: West Europe

Storage Account: (New) aq...

Providers included in this workspace

You can make changes after creating your workspace.

Name	Provider type	Plan	Pricing
IonQ IonQ	Quantum Computing	Azure Quantum Credits	FREE
Quantinuum Quantinuum	Quantum Computing	Azure Quantum Credits	FREE
Rigetti Quantum Rigetti Computing	Quantum Computing	Azure Quantum Credits	FREE

By clicking create you accept each included provider's terms and conditions and privacy policy and agree Microsoft can share your account details

Create Back

Quantum Workspace erstellen – Schritt 2 (Bild 3)

tisch zugewiesen. Außerdem bekommt man pro Anbieter (IonQ, Quantinuum und Rigetti) 500 US-Dollar kostenloses Budget, um auf den „Quantum-Geschmack“ der echten Quantencomputer kommen zu können!

Die Bereitstellung kann einige Minuten dauern. Status- und Bereitstellungsdetails werden im Azure Portal aktualisiert. Falls während der Bereitstellung Probleme auftreten, bietet Microsoft eine FAQ-Seite zu häufigen Problemen. Bei der erweiterten Erstellung eines neuen Arbeitsbereichs kann man zusätzlich noch Ressourcen-Gruppe und Storage-Konto sowie alle vorhandenen oder ausgewählten Anbieter als kostenfreie (mit 500 US-Dollar Kontingent) oder Abo-Option angeben. Alle diese Schritte lassen sich auch via CLI mit Quantum-cmdlets [16] erledigen, zum Beispiel so:

```
$object = New-AzQuantumProviderObject -Id "ionq"
-Sku "pay-as-you-go "
```

```
New-AzQuantumWorkspace -Name Az_Quantum_WS
-ResourceGroupName Az_Quantum_RG -Location eastus
-IdentityType 'SystemAssigned' -Provider $object
-StorageAccount "/subscriptions/{subId}/
resourceGroups/Az_Quantum_RG/providers/
Microsoft.Storage/storageAccounts/azqntsa"
```

Nach der Erstellung und Auswahl eines neuen Quanten-Arbeitsbereichs im Azure Portal kann er ähnlich wie auf **Bild 4** aussehen. Nun steht Ihnen nichts mehr im Weg, mit Azure Quantum in die faszinierte Welt des Quantencomputing durchstarten zu können!

Azure Quantum – welche Option ist die beste?

Azure Quantum bietet drei unterschiedliche Schnittstellen, um Quantencomputing und Quantenprogrammierung zu erkunden:

- **Azure Quantum Website** – webbasierte Plattform für interaktives Lernen und Programmieren mit Copilot-Unterstützung.
- **Azure Quantum Development Kit (QDK)** für Visual Studio Code – lokale Entwicklungsumgebung für Q#-Programme.
- **Azure Quantum Portal** – webbasierte Implementierung und/oder Verwaltungsschnittstelle für Quantum Azure Arbeitsbereiche, Jupyter-Notebooks mit Microsoft Q# [17], IBM Qiskit [18] und Google Cirq [18], Quanten-Aufträge (Jobs) und Anbieter (Quantum Hardware-Provider).

Jede dieser Optionen hat spezifische Stärken und richtet sich an unterschiedliche Nutzeranforderungen. Die nachfolgenden ausführlicheren Beschreibungen helfen bei der Wahl der passenden Umgebung. ▶

quantum-workspace-...-01

Quantum Workspace

Search

Delete Refresh Help Feedback

Azure Quantum Credits will no longer be available after June 1st, 2025.

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Operations

Notebooks

Job management

Providers

Credits and Quotas

Access Keys

Monitoring

Alerts

Automation

CLI / PS

Tasks

Help

Essentials

Resource group (move): AzureQuantum

Status: Succeeded

Location: West Europe

Subscription (move): Pay as you go!

Subscription ID: ...

Tags (edit): Add tags

Getting started Local setup guide

Resource ID: /subscriptions/bx...

Storage account: gqa5...

Online Jupyter notebooks

VS Code (Web)

Local development

Quantum Workspace ist startklar (Bild 4)

● **Tabelle 2: Variantenvergleich von Azure Quantum**

Funktion	Azure Quantum Website	Azure QDK (VS Code)	Azure Quantum Portal
Quantenprogrammierung	Ja, mit Copilot und Katas	Ja, mit Q# und Python	Ja, mit Jupyter-Notebooks
Code-Generierung mit KI	Ja, Copilot für Q#-Code	Nein	Nein
Installation erforderlich	Nein	Ja, VS Code und QDK	Nein
Simulation von Quantenalgorithmen	Ja, über In-Memory Simulator oder Quantinuum H-Series Emulator	Ja, über lokale oder Azure-Provider-Simulatoren	Ja, über Azure-Provider-Simulatoren
Ausführung auf echter Quantenhardware	Nein	Ja, über Anbieter im Arbeitsbereich	Ja, über Anbieter im Arbeitsbereich
Quantenauftragsverwaltung	Nein	Ja	Ja
Zugriff auf Quantum-Guthaben	Nein	Ja, mit Azure-Arbeitsbereich	Ja, 500 US-Dollar pro Anbieter (Provider)
Geeignet für	Einsteiger und Lernende	Entwickler und Forscher	Unternehmen und Arbeitsgruppen

Die Azure Quantum Website [1] ist der einfachste Einstieg in die Welt des Quantencomputing. In dieser webbasierten Umgebung können Nutzer lernen und ohne Installation erste Quantenprogramme schreiben, simulieren und mit Copilot-KI-Unterstützung erstellen. Wählt man auf der Azure Quantum Website via *Tools*-Menü die Webseite Azure Quantum Coding [20] aus, kann man folgende nützliche Features nutzen:

- Copilot in Azure Quantum – generiert Q#-Code basierend auf natürlichen Spracheingaben und erklärt Quantenkonzepte in einer interaktiven Umgebung.
- Ausführung der Q#-Programme direkt im Webbrowser.
- Diverse Programmbeispiele.

Ferner findet man dort auch Azure Quantum Katas [21] – ein Tutorial mit Unterstützung durch Copiloten in Azure Quantum zum eigenständigen Lernen. Man kann die Option mit der Azure Quantum Website nutzen, wenn man Folgendes tun möchte:

- schnell in Quantenprogrammierung einsteigen,
- Quantencomputing mit interaktiven Tutorials lernen,
- erste Programme im Browser simulieren, ohne Hardware oder eine Entwicklungsumgebung einzurichten.

Das Azure Quantum Development Kit (QDK) wurde oben bereits beschrieben. Wann sollte man aber auf Azure QDK zurückgreifen? Hier sind einige Gründe dafür:

- Wenn man lokal mit Q# arbeiten möchte.
- Wenn man ein professionelles Quantenentwicklungs-Setup benötigt.
- Wenn man Ressourcenschätzungen für skalierte Quantenalgorithmen durchführen möchte.
- Wenn man Programme direkt an Azure-Quantum-Arbeitsbereiche für bestimmte Quantenhardware übermitteln soll.

Das Azure Quantum Portal [15] (siehe **Bild 4**) ist die zentrale Verwaltungsschnittstelle für Azure-Quantum-Arbeitsbereiche, Quantenaufträge (Jobs) und Anbieter (Provider). Es ermöglicht Nutzern, ihre Quantenprogramme auf echter Quantenhardware oder in Simulationsumgebungen auszuführen.

Eine Simulationsumgebung ist hilfreich, wenn ein Quantenprogramm ausprobiert/getestet und/oder debuggt werden muss, und verursacht in der Regel gar keine oder geringe Nutzungskosten.

Hauptfunktionen im Azure Quantum Portal sind:

- Erstellung und Verwaltung von Azure-Quantum-Arbeitsbereichen.
- Auswahl und Verwaltung von Quantencomputing-Anbietern – IonQ, Quantinuum, Rigetti.
- Speicherung und Verwaltung eingehender Quantenjobs.
- Überwachung des Azure-Quantum-Guthabens und Nutzungsstatistiken.
- Direkte Ausführung von Q#-, Qiskit- und Cirq-Programmen in einem Jupyter-Notebook.

Die Nutzung des Azure Quantum Portals wäre die beste Option, wenn man Folgendes tun möchte:

- einen Arbeitsbereich (Quantum Workspace) für Azure Quantum einrichten,
- die Quantenprogramme auf echter Quantenhardware ausführen,
- die Quantenaufträge (Jobs) und Guthaben verwalten,
- eine Cloud-basierte Entwicklungsumgebung mit Azure Jupyter-Notebooks nutzen.

In der **Tabelle 2** ist ein qualitativer Vergleich für drei Anwendungsfälle von Azure Quantum dargestellt. Die meisten Quantencomputing-Entwickler und -Forscher nutzen eine Kombination aller drei Variationen, um das Beste aus Azure Quantum herauszuholen.

Quantenprogrammierung mit Copilot in Azure Quantum

Mit Copilot in Azure Quantum Coding [21] stellt Microsoft eine KI-gestützte Entwicklungsumgebung bereit, die den Einstieg in die Quantenprogrammierung erheblich erleichtert. Copilot kombiniert modernste KI-Technologien mit der Leistungsfähigkeit von Quantencomputing, um Entwicklern interaktive Unterstützung bei der Erstellung, Optimierung und

Simulation von Quantenprogrammen zu bieten. Copilot in Azure Quantum basiert auf Microsofts fortschrittlichen generativen KI-Modellen und ermöglicht es Nutzern, Q#-Code zu schreiben, zu verstehen und zu verbessern, indem sie einfache Texteingaben verwenden. Dies reduziert die Einstiegshürden für Entwickler, Wissenschaftler und Unternehmen, die sich mit Quantencomputing beschäftigen möchten. Copilot ermöglicht es Entwicklern, Quantenprogramme mit natürlicher Sprache zu beschreiben, woraufhin die KI automatisch den passenden Q#-Code generiert. Zum Beispiel kann man im Copilot-Chat angeben:

"Schreibe ein Q#-Programm, das ein Qubit in Superposition versetzt und misst."

Daraufhin schlägt Azure Quantum Copilot ein Q#-Programm vor, das mit dem Button *Copy code to editor* (siehe (1) in Bild 5) übernommen und ausgeführt werden kann (siehe (2) in Bild 5). Man kann den erstellten Code auch für die weitere Verarbeitung entweder in die Zwischenablage kopieren (zum Beispiel für VS Code lokal) oder in die webbasierte VS-Code-Variante [23] übertragen (siehe jeweils (3) und (4) in Bild 5).

Copilot kann darüber hinaus auch quantenmechanische Prinzipien erklären und Visualisierungen erstellen. Dazu ein weiteres Beispiel für die Eingabe in Copilot:

"Erkläre die Quantenverschränkung und zeige ein Beispiel in Q#."

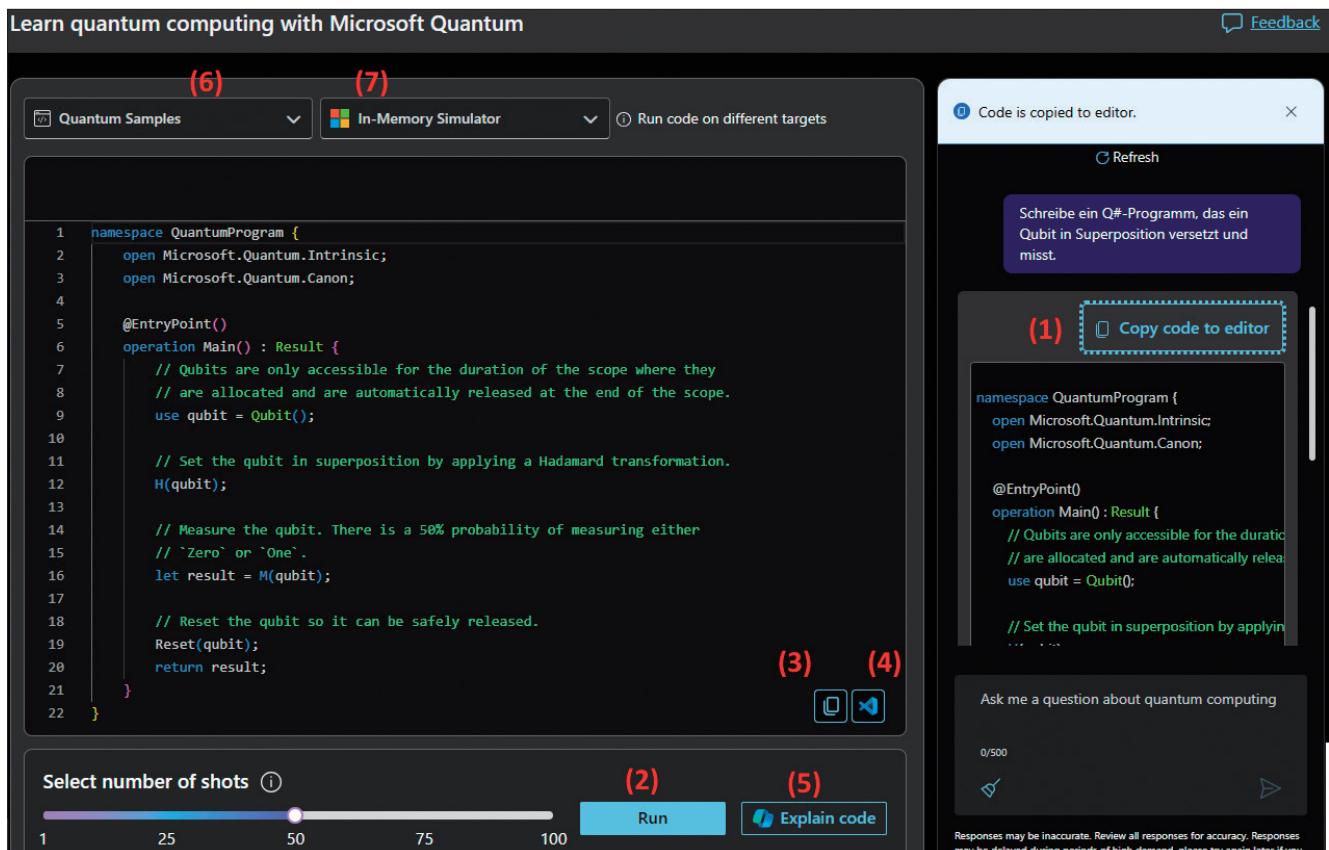
Daraufhin liefert Azure Quantum Copilot eine Erklärung der Quantenbits-Verschränkung sowie ein Beispielprogramm für zwei verschränkte Qubits. Diese Funktion erleichtert das Verständnis komplexer Quantenkonzepte, auch ohne tiefgehende theoretische Vorkenntnisse.

Außerdem kann Azure Quantum Copilot verschiedene Code-Probleme identifizieren, Fehler analysieren und Lösungen vorschlagen. Diese Funktion ist besonders nützlich, da Quantenprogramme oft schwierig zu debuggen sind (siehe (5) in Bild 5).

Typische Anwendungsfälle für Quantenprogramm-Analyse wären:

- **Optimierung der Code-Effizienz** – Copilot schlägt alternative Algorithmen vor, um Quantenoperationen effizienter auszuführen.
- **Korrektur von Syntaxfehlern** – Copilot erkennt Syntaxprobleme und liefert direkt die korrekte Version.
- **Optimierung der Ressourcennutzung** – Copilot schlägt alternative Algorithmen vor, um Ressourcen der Quanten-Hardware optimaler zu nutzen.

Für die Nutzer besteht auch Möglichkeit, weitere Programm-Beispiele (Quantum Samples) sowie einen Quantensimulator auszuwählen (siehe jeweils (6) und (7) in Bild 5). Um ein Programm von der Azure Quantum Website auf echter Quantenhardware auszuführen, muss man es entweder in VS Code mit Azure QDK oder als Jupyter-Notebook in den Quanten-Arbeitsbereich auf dem Azure Portal übertragen. ►



Copilot in der Azure Quantum Website (Bild 5)

Quantenprogrammierung mit Visual Studio Code

Quantenprogrammierung mit Visual Studio Code (VS Code) ist bereits sehr gut im dotnetpro-Artikel „Q# für Einsteiger“ [22] und in der Dokumentation zu Azure Quantum [23] beschrieben. Daher soll hier nur die Übermittlung von Aufträgen (Jobs) an Quantenhardware erläutert werden.

Nachdem die Extension Azure Quantum QDK in VS Code installiert und der Azure-Quantum-Arbeitsbereich eingerichtet ist (siehe oben den Abschnitt „Azure-Quantum-Arbeitsbereich erstellen“), ist es an der Zeit, eine Verbindung herzustellen und einen Quantenauftrag zu übermitteln. Egal ob man mit Q# oder Python arbeitet: Azure Quantum bietet eine flexible Umgebung für die Entwicklung und Ausführung von Quantenprogrammen. Die Verbindung zu Azure Quantum kann auf zwei Wegen erfolgen: über VS Code oder mit Python. Für die Verbindung mit VS Code durchläuft man folgende Schritte:

- In Visual Studio Code die Befehlspalette mit [Strg]+[Umschalt]+[P] (Windows/Linux) oder [Cmd]+[Umschalt]+[P] (macOS) aufrufen.
- Nach Q#: *Connect to an Azure Quantum workspace* suchen.
- *Azure Account* oder *Connection string* auswählen und die Verbindungsdetails (siehe im Azure Portal) des jeweiligen Quantum-Arbeitsbereichs eingeben.

Wer lieber mit Python arbeitet, kann eine Verbindung mit wenigen Zeilen Code herstellen:

```
from azure.quantum import Workspace
```

```
workspace = Workspace(
    resource_id="DEINE-RESOURCE-ID",
    location="DEIN-STANDORT"
)
```

Die Ressourcen-ID und den Standort eines Arbeitsbereichs findet man im Azure Portal unter *Übersicht* des jeweiligen Quantum-Arbeitsbereichs. Sobald die Verbindung steht, kann man ein erstes Quantenprogramm an Azure Quantum übermitteln:

- In *Quantum Workspaces* im VS-Code-Explorer findet man den jeweiligen Anbieter mit der gewünschten Quantenhardware und betätigt den *Submit*-Button (siehe etwa *rigetti-sim.qvm* in **Bild 6**).
- Danach gibt man den Job-Namen und Anzahl von Shots (Anzahl von Programmausführungen) an (siehe **Bild 6**).
- Nach der Ausführung kann man via *Jobs*-Verzeichnis (siehe **Bild 7**) die Ergebnisse des übermittelten Auftrags als JSON-Datei herunterladen (dieselbe Funktion steht auch via Azure Portal zur Verfügung).

Falls man lieber mit Python arbeitet, kann man den Auftrag so übermitteln:

```
from azure.quantum.qiskit import AzureQuantumProvider
```

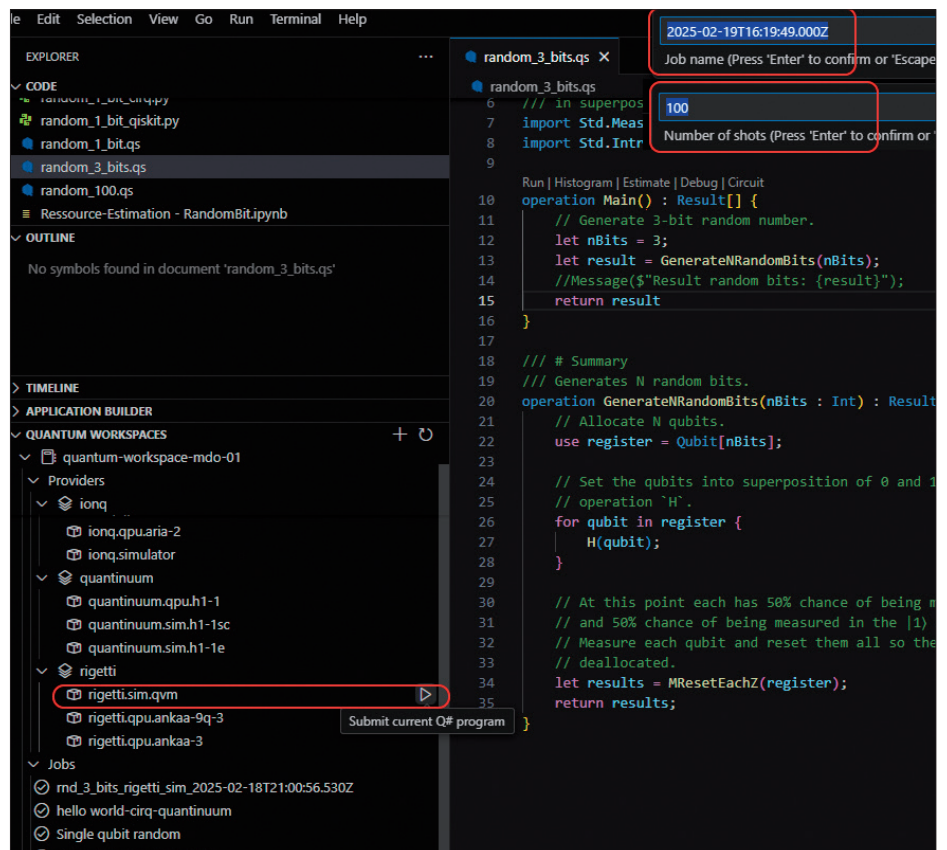
```
provider = AzureQuantumProvider(resource_id=
    "DEINE-RESOURCE-ID", location="DEIN-STANDORT")
backend = provider.get_backend("ionq.simulator")
job = backend.run(circuit)
job_monitor(job)
result = job.result()
print(result.get_counts())
```

Sobald der Auftrag erfolgreich ausgeführt ist, kann man die Ergebnisse direkt auswerten. Der Status eines Auftrags lässt sich sowohl via *Jobs*-Verzeichnis in VS Code und/oder dem Quantum-Arbeitsbereich im Azure Portal überwachen.

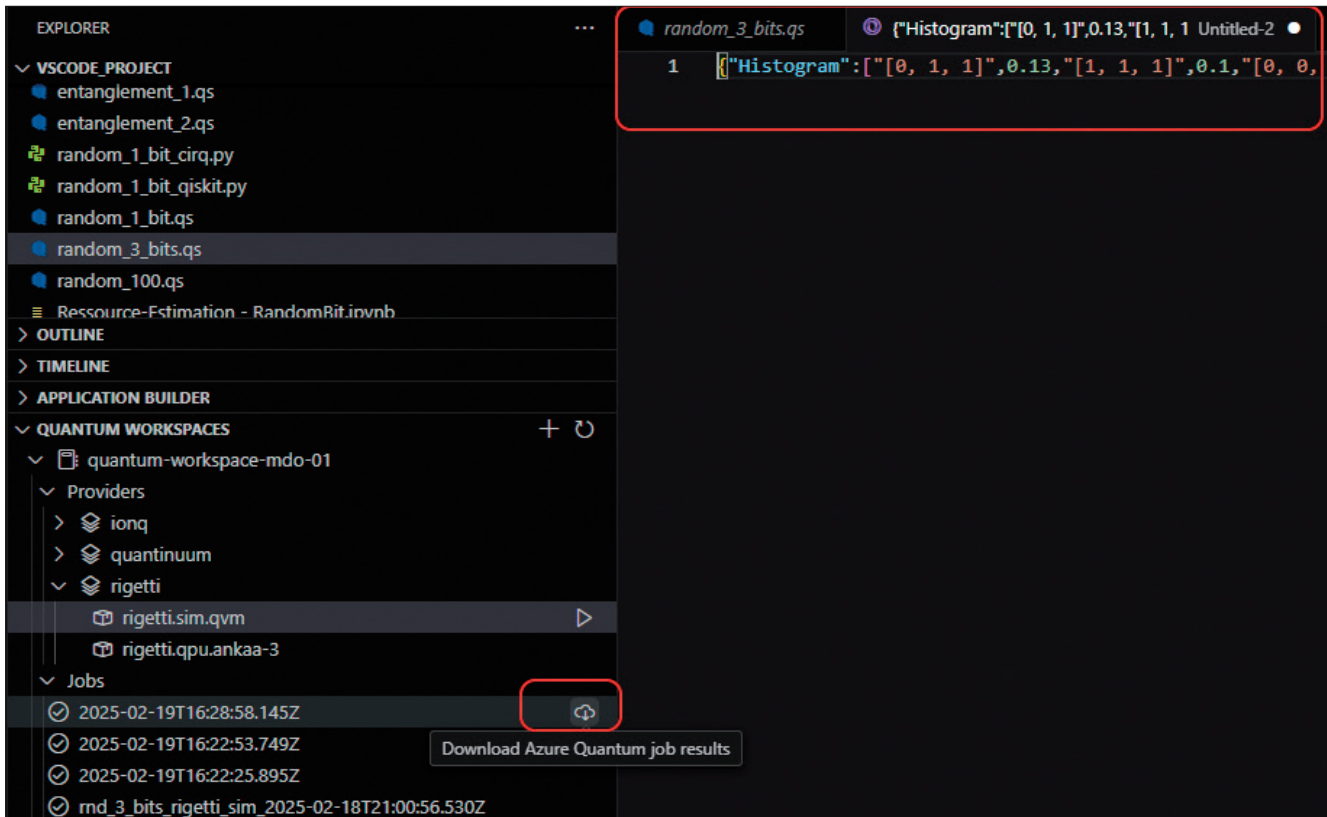
Fazit

Quantencomputing ist eine disruptive Technologie mit dem Potenzial, zahlreiche Industrien zu transformieren. Obwohl manche Quantenhardware aktuell noch in der Entwicklungsphase steckt, ermöglicht Cloud-Computing bereits heute den Zugriff auf leistungsfähige Quantencomputer.

Microsofts Azure Quantum nimmt hier eine Schlüsselrolle ein: Die Plattform kombiniert bestehende QPU-Technologien mit innovativen Forschungsansätzen und hybriden Lösungen, die Quanten- und klassische Computer optimal verbinden. Besonders vielversprechend ist Microsofts Entwicklung topologischer Qubits, die langfristig die Stabilität und Skalierbarkeit von Quantencomputern erheblich verbessern



Quantum-Job in VSCode – Übermittlung (**Bild 6**)



Quantum-Job in VSCode – Ergebnisse (Bild 7)

könnten. Während Letztere heute noch weitgehend „in den Kinderschuhen“ stecken, ist damit klar: Die Zukunft ist bereits in greifbarer Nähe – und sie beginnt in der Cloud. ■

- [1] Azure Quantum, <https://quantum.microsoft.com/>
- [2] IBM Quantum, <https://quantum.ibm.com/>
- [3] Google Quantum AI, <https://quantumai.google>
- [4] Leap – D-Wave Quantum Cloud-Service, www.dotnetpro.de/SL2504-05AzureQuantum1
- [5] Quandela Cloud, <https://cloud.quandela.com/>
- [6] Amazon Braket, www.dotnetpro.de/SL2504-05AzureQuantum2
- [7] Azure Quantum Insights, Topological Qubits, www.dotnetpro.de/SL2504-05AzureQuantum3
- [8] Mykola Dobrochynskyy, Wegweisende Symbiose, dotnetpro 2/2025, Seite 20 ff., www.dotnetpro.de/A2502QuantumCloud
- [9] Alexandra Waldherr, Neue Denkweisen, dotnetpro 2/2025, Seite 10 ff., www.dotnetpro.de/A2502Quantum
- [10] Azure Quantum Insights, Resource Estimation, www.dotnetpro.de/SL2504-05AzureQuantum4
- [11] Microsoft Learn, Was ist Hybrid Quantum Computing?, www.dotnetpro.de/SL2504-05AzureQuantum5
- [12] Azure Quantum Development Kit (QDK), <https://github.com/microsoft/qsharp>
- [13] Microsoft Learn, Einrichten des Quantum Development Kit, www.dotnetpro.de/SL2504-05AzureQuantum6
- [14] Visual Studio Code Web (Quantum), <https://vscode.dev/quantum>

[15] Microsoft Azure Portal, <https://portal.azure.com/>

[16] Microsoft Learn, Microsoft Azure PowerShell: Quantum cmdlets, www.dotnetpro.de/SL2504-05AzureQuantum7

[17] Microsoft Azure Quantum, Q# Programming Language, www.dotnetpro.de/SL2504-05AzureQuantum8

[18] IBM Quantum, Qiskit SDK, <https://www.ibm.com/quantum/qiskit>

[19] Google Quantum AI, Cirq Framework, <https://quantumai.google/cirq>

[20] Azure Quantum Coding, www.dotnetpro.de/SL2504-05AzureQuantum9

[21] Azure Quantum Katas, www.dotnetpro.de/SL2504-05AzureQuantum10

[22] Daniel Basler, Q# für Einsteiger, dotnetpro 2/2025, Seite 38 ff., www.dotnetpro.de/A2502QSharp

[23] Dokumentation zu Azure Quantum, www.dotnetpro.de/SL2504-05AzureQuantum11



Mykola Dobrochynskyy

ist Senior Architekt bei der Porsche AG. Seine Schwerpunkte sind Softwarearchitektur, modellgetriebene Softwareentwicklung, generative Programmierung, KI sowie Cloud- und dienstorientierte Softwarearchitekturen. www.xing.com/profile/Mykola_Dobrochynskyy

dnpCode

A2504-05AzureQuantum



DATA ACCESS

Die Datenzugriffsberatung mit DR. HOLGER SCHWICHTENBERG

DAS NEUE NUGET-PAKET MICROSOFT.EXTENSIONS.CACHING.HYBRID

Hybrider Zwischenspeicher

Microsoft hat einen neuen hybriden Cache für .NET veröffentlicht, der das parallele Caching im RAM und einen persistenten Speicher unterstützt.

Schon vor .NET 9.0 existierten Cache-Lösungen in der .NET-Klassenbibliothek. Seit .NET Framework 4.0 gab es die Klasse `System.Runtime.Caching.MemoryCache` (für das Zwischenspeichern im RAM). Diese Klasse steht auch im modernen .NET seit .NET Core 2.0 zur Verfügung.

Seit .NET Core 1.0 gibt es zusätzlich zwei Schnittstellen mit Implementierungen:

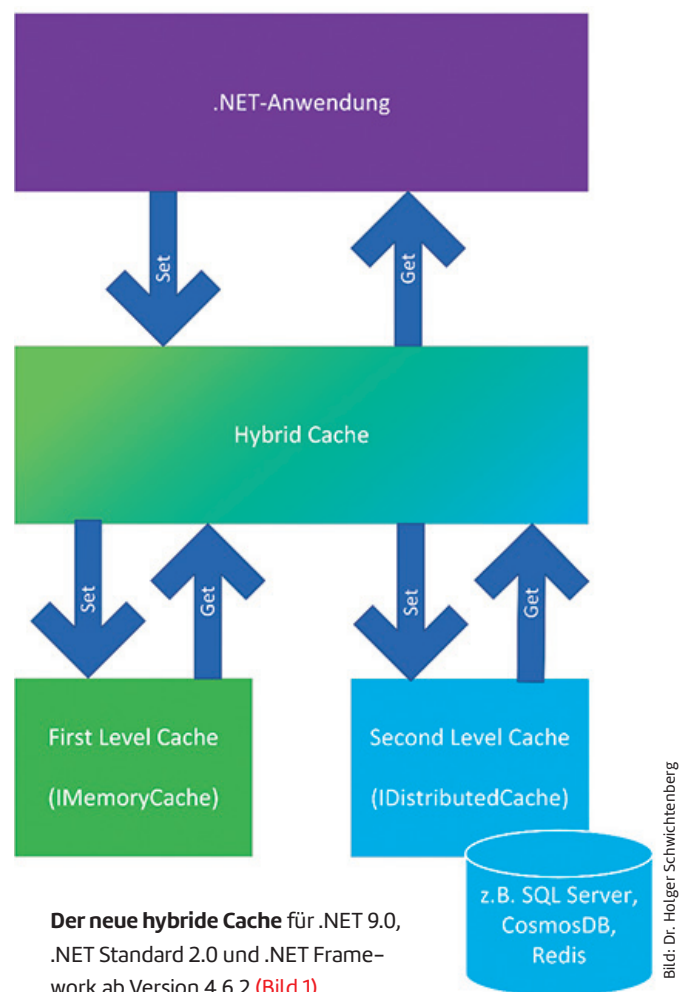
- `Microsoft.Extensions.Caching.Memory.IMemoryCache`: wird im Dependency-Injection-Container aufgenommen mit `AddMemoryCache()` und ist automatisch enthalten bei `AddMvc()`, `AddRazorPages()` und anderen ASP.NET-Core-Dependency-Injection-Methoden.
- `Microsoft.Extensions.Caching.Distributed.IDistributedCache`: wird im Dependency-Injection-Container aufgenommen mit `AddDistributedXYCache()`, je nach Implementierung.

Für die Schnittstelle `IMemoryCache` gibt es die Implementierung `Microsoft.Extensions.Caching.Memory.MemoryCache` im gleichnamigen NuGet-Paket.

Diverse Implementierungen für `IDistributedCache`

Die Schnittstelle `IDistributedCache` ist eine Abstraktion für diverse Zwischenspeicher. Es existieren einige Implementierungen der Schnittstelle `IDistributedCache` von Microsoft selbst:

- `Microsoft.Extensions.Caching.Redis.RedisCache` (Caching in Redis)
- `Microsoft.Extensions.Caching.StackExchangeRedis.RedisCache` (Caching in Redis)
- `Microsoft.Extensions.Caching.SqlServer.SqlServerCache` (Caching im Microsoft SQL Server)
- `Microsoft.Extensions.Caching.Cosmos` (Caching in Cosmos-DB)
- `Microsoft.Extensions.Caching.Distributed.MemoryDistributedCache` (wobei dieser Cache nicht verteilt, sondern lokal arbeitet)



Außerdem gibt es Implementierungen von Amazon Web Services (AWS) in der Klasse `AWS.DistributedCacheProvider.AWS.DistributedCacheProvider.DynamoDBDistributedCache` aus dem NuGet-Paket `AWS.AspNetCore.DistributedCache`.



Provider und in der Klasse `Alachisoft.NCache.Caching.Distributed.NCacheDistributedCache` aus der Community (NCache [1]).

Sowohl `Microsoft.Extensions.Caching.Memory.IMemoryCache` als auch `Microsoft.Extensions.Caching.Distributed.IDistributedCache` und deren Implementierungen gibt es auch für das klassische .NET Framework.

Unzufriedenheit mit IDistributedCache

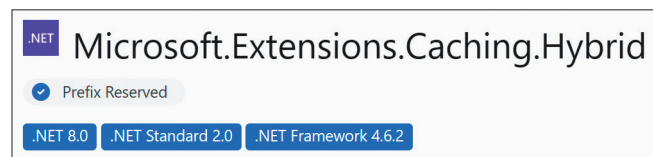
Microsoft äußerte sich im Januar 2024 auf GitHub unzufrieden mit der Schnittstelle `IDistributedCache`: „The distributed cache in asp.net (i.e. `IDistributedCache`) is not particularly developed; it is inconvenient to use, lacks many desirable features, and is inefficient.“ [2]. So gibt es zum Beispiel in `IDistributedCache` keinen Schutz vor einem „Ansturm“ (englisch: Stampede), bei dem mehrere gleichzeitige Anforderungen für denselben nicht zwischengespeicherten Wert zu einer gleichzeitigen Backend-Last für dieselben Daten führen. Und auch um die Serialisierung der Daten in eine Bytefolge mussten sich Entwicklerinnen und Entwickler selbst kümmern.

Der neue hybride Cache

Der neue hybride Cache in der Klasse `Microsoft.Extensions.Caching.Hybrid.HybridCache` ist keine Implementierung der Schnittstelle `IDistributedCache`, sondern eine Abstraktion oberhalb der Implementierungen von `IDistributedCache` und vereint einen First-Level-Cache im RAM des lokalen Systems (mit `IMemoryCache`) mit einem beliebigen Second-Level-Cache, der mit `IDistributedCache` arbeitet (siehe Bild 1). Daher der Name „hybrider Cache“. Beide Cache-Arten sind abschaltbar.

NuGet-Paket Microsoft.Extensions.Caching.Hybrid

Den neuen hybriden Cache liefert Microsoft im NuGet-Paket `Microsoft.Extensions.Caching.Hybrid` nicht nur für .NET 9.0, sondern auch für ältere Laufzeitumgebungen (.NET Framework 4.6.2 und .NET Standard 2.0) (Bild 2).



Die unterstützten Plattformen für das NuGet-Paket `Microsoft.Extensions.Caching.Hybrid` (Bild 2)

Microsoft hat sich am 12. September 2024 entschlossen, den Hybrid Cache aus dem Kern von .NET 9.0 zu entfernen und zu einem späteren Zeitpunkt als eigenständiges NuGet-Paket `Microsoft.Extensions.Caching.Hybrid` zu liefern, und erklärt dazu: „HybridCache has relocated to dotnet/extensions/dev; it does not ship in .NET 9 RC1, as a few missing and necessary features are still in development; however, we expect to ship either alongside or very-shortly-after .NET 9! (‘extensions’ has a different release train, that allows additional changes beyond the limit usually reserved for in-box packages; HybridCache has always been described as out-of-box – i.e. a NuGet package – so: there is no reason for us to limit ourselves by the runtime restrictions)“ [2].

Zum RTM-Termin von .NET 9.0 am 12. November 2024 hatte das Paket den Stand „Preview 9“. Am 14. Januar 2025 kam Version 9.1 des Pakets, aber ebenfalls als Preview gekennzeichnet (vergleiche Bild 3).

Einfache Methode GetOrCreateAsync()

Anstelle der bisher notwendigen Kombination der Operationen `GetAsync()` und `SetAsync()` bietet der neue hybride Cache eine vereinfachte Methode `GetOrCreateAsync()` an, die einen Wert aus dem Cache liefert oder (wenn der Wert nicht im Cache ist) eine Factory-Funktion aufruft, die den Wert liefert und im Cache speichert. `SetAsync()` gibt es weiterhin; `GetAsync()` existiert nicht mehr.

Zum Löschen von Einträgen aus dem Cache kann man `RemoveAsync()` (Löschen eines konkreten Eintrags anhand des Schlüssels) oder `RemoveByTagAsync()` (Löschen aller ►

✖ EXTEXP0018

'Microsoft.Extensions.DependencyInjection.HybridCacheServiceExtensions.AddHybridCache(Microsoft.Extensions.DependencyInjection.IServiceCollection, System.Action<Microsoft.Extensions.Caching.Hybrid.HybridCacheOptions>)' is for evaluation purposes only and is subject to change or removal in future updates. Suppress this diagnostic to proceed. (<https://aka.ms/dotnet-extensions-warnings/EXTEXP0018>)

Diese Warnmeldung kommt, wenn man das Paket `Microsoft.Extensions.Caching.Hybrid` in seiner Preview-Version verwendet (Bild 3)

Listing 1: Einsatz des hybriden Cache mit SQL Server als Second-Level-Cache

```

using System.Diagnostics;
using System.Text;
using System.Text.Json;
using BL;
using ITVisions;
using Microsoft.Extensions.Caching.Distributed;
using Microsoft.Extensions.Caching.Hybrid;
using Microsoft.Extensions.DependencyInjection;

namespace NET9_Console.FCL90;

/// <summary>
/// Hybrider Cache: RAM + SQL Server (
/// Microsoft.Extensions.Caching.Hybrid.HybridCache)
/// </summary>
class FCL9_HybridCache
{
    private const int FirstLevelCache_Duration = 3;
    private const int SecondLevelCache_Duration = 10;

    public async Task Run()
    {
        CUI.Demo("Hybrid Cache: RAM + SQL Server");
        Console.WriteLine("FirstLevelCache_Duration = " +
            FirstLevelCache_Duration + "sec");
        Console.WriteLine("SecondLevelCache_Duration = " +
            SecondLevelCache_Duration + "sec");

        // DI-Container konfigurieren
        var services = new Microsoft.Extensions.
            DependencyInjection.ServiceCollection();
        //services.AddScoped<PersonManagerOld>();
        services.AddScoped<PersonManager>();

        // Hybrid Cache konfigurieren
        services.AddHybridCache(options =>
            // optionale Einstellungen
        {
            options.MaximumPayloadBytes = 1024 * 1024;
            options.MaximumKeyLength = 1024;
            options.DefaultEntryOptions =
                new HybridCacheEntryOptions
                {
                    LocalCacheExpiration = TimeSpan.FromSeconds(
                        FirstLevelCache_Duration),
                    // Lebensdauer im First-Level-Cache
                    Expiration = TimeSpan.FromSeconds(
                        SecondLevelCache_Duration),
                    // Lebensdauer im Second-Level-Cache
                    Flags = HybridCacheEntryFlags.DisableCompression
                    // Standardmäßig wird die Komprimierung
                    // aktiviert
                };
        });

        // Second-Level-Cache konfigurieren
        services.AddDistributedSqlServerCache(options =>
        {
            options.ConnectionString =
                "Data Source=E61;Initial Catalog=NET_Cache;" +
                "Integrated Security=True;" +
                "Connect Timeout=30;Encrypt=True;" +
                "Trust Server Certificate=True;" +
                "Application Intent=ReadWrite;" +
                "Multi Subnet Failover=False";
            options.SchemaName = "dbo";
            options.TableName = "Cache1";
        });

        // Zweiten Second-Level-Cache konfigurieren
        // (nur zum Test, was dann passiert => es wird nur
        // dieser zweite verwendet!)
        services.AddDistributedSqlServerCache(options =>
        {
            options.ConnectionString =
                "Data Source=E61;Initial Catalog=NET_Cache;" +
                "Integrated Security=True;" +
                "Connect Timeout=30;Encrypt=True;" +
                "Trust Server Certificate=True;" +
                "Application Intent=ReadWrite;" +
                "Multi Subnet Failover=False";
            options.SchemaName = "dbo";
            options.TableName = "Cache2";
        });

        // DI-Container nutzen
        var provider = services.BuildServiceProvider();
        //var pm = provider.GetService<PersonManagerOld>();
        var pm = provider.GetService<PersonManager>();

        //await pm.InitCache();

        // 10x Personen-Suche (mit Pausen)
        for (int i = 0; i < 100; i++)
    }
}

```

Einträge anhand eines Tags, wobei die Vergabe von Tags optional ist) aufrufen. Es ist wichtig zu beachten, dass beim Entfernen eines Eintrags dieser sowohl aus dem primären als auch aus dem sekundären Cache entfernt wird. Den hybriden

Cache verwendet man, indem man im Dependency-Injection-Container *AddHybridCache()* aufruft, optional mit Vorgabewerten für die Zwischenspeicherdauer und die Speichermenge im RAM.

```

{
    Stopwatch stopwatch = new Stopwatch();
    CUI.Cyan("Suche Person...");
    stopwatch.Start();
    var pSet = await pm.SearchPerson("Schwichtenberg");
    stopwatch.Stop();

    CUI.Print($"Gefundene Personen: {pSet.Count} |
        Dauer: {stopwatch.Elapsed.TotalMilliseconds}
        ms", stopwatch.Elapsed.TotalMilliseconds
        < 1000 ? ConsoleColor.Green :
        ConsoleColor.Yellow);
    foreach (var p in pSet)
    {
        CUI.LI(p);
    }
    await Task.Delay(1000);
}

public class PersonManager(HybridCache cache)
{
    /// <summary>
    /// Legt ein Suchergebnis exemplarisch in den Cache
    /// </summary>
    public async Task InitCache()
    {
        CUI.Cyan("Initialisiere Cache...");
        await cache.SetAsync(
            "SearchPerson:Schwichtenberg", await new
            Backend().SearchPersonOnServer(
                "Schwichtenberg"));
    }

    /// <summary>
    /// Personensuche mit Caching
    /// </summary>
    public async Task<List<BO.Person>> SearchPerson(
        string name, CancellationToken token = default)
    {
        // Individuelle Einstellungen für diesen
        // Cache-Eintrag (optional)
        var eo = new HybridCacheEntryOptions
        {
            LocalCacheExpiration = TimeSpan.FromSeconds(
                FirstLevelCache_Duration),
            // Lebensdauer im First-Level-Cache

            Expiration = TimeSpan.FromSeconds(
                SecondLevelCache_Duration),
            // Lebensdauer im Second-Level-Cache
        };

        return await cache.GetOrCreateAsync(
            $"SearchPerson:{name}",
            // eindeutiger Schlüssel für diese Datenmenge
            async cancel => await new Backend().
                SearchPersonOnServer(name, cancel),
            // Factory-Funktion, die die Daten lädt
            options: eo,
            // optional eigene HybridCacheEntryOptions
            // für diesen Eintrag, wenn die globalen
            // Werte nicht gelten sollen
            cancellationToken token,
            // optionales Cancellation Token
            tags: ["SearchResults"] // optionales Tag
        );
    }
}

class Backend
{
    static int id = 0;

    /// <summary>
    /// Simulation des Backends
    /// </summary>
    public async Task<List<BO.Person>>
        SearchPersonOnServer(string name,
            CancellationToken cancel = default)
    {
        id++;
        CUI.Yellow(
            "--> CACHE MISS! Lade Daten aus Backend...");
        var pSet = new PersonManager().GetPersonSet(
            42, 20000, false);
        await Task.Delay(500);
        // Simuliere zusätzliche Latenz
        // wähle per Zufallszahl eine Person aus
        var p = pSet[new Random(DateTime.Now.Millisecond).
            Next(0, pSet.Count)];
        return [p];
    }
}

```

Hybrid Cache im Praxiseinsatz

Listing 1 zeigt den hybriden Cache mit der schon zuvor bestehenden SQL-Server-Cache-Implementierung im NuGet-Paket `Microsoft.Extensions.Caching.SqlServer` als Second-

Level-Cache. Eine explizite Verbindung zwischen hybridem Cache und Second-Level-Cache ist nicht notwendig. Der hybride Cache sucht per Dependency Injection nach einer Implementierung von `IDistributedCache`. Wenn er eine ►

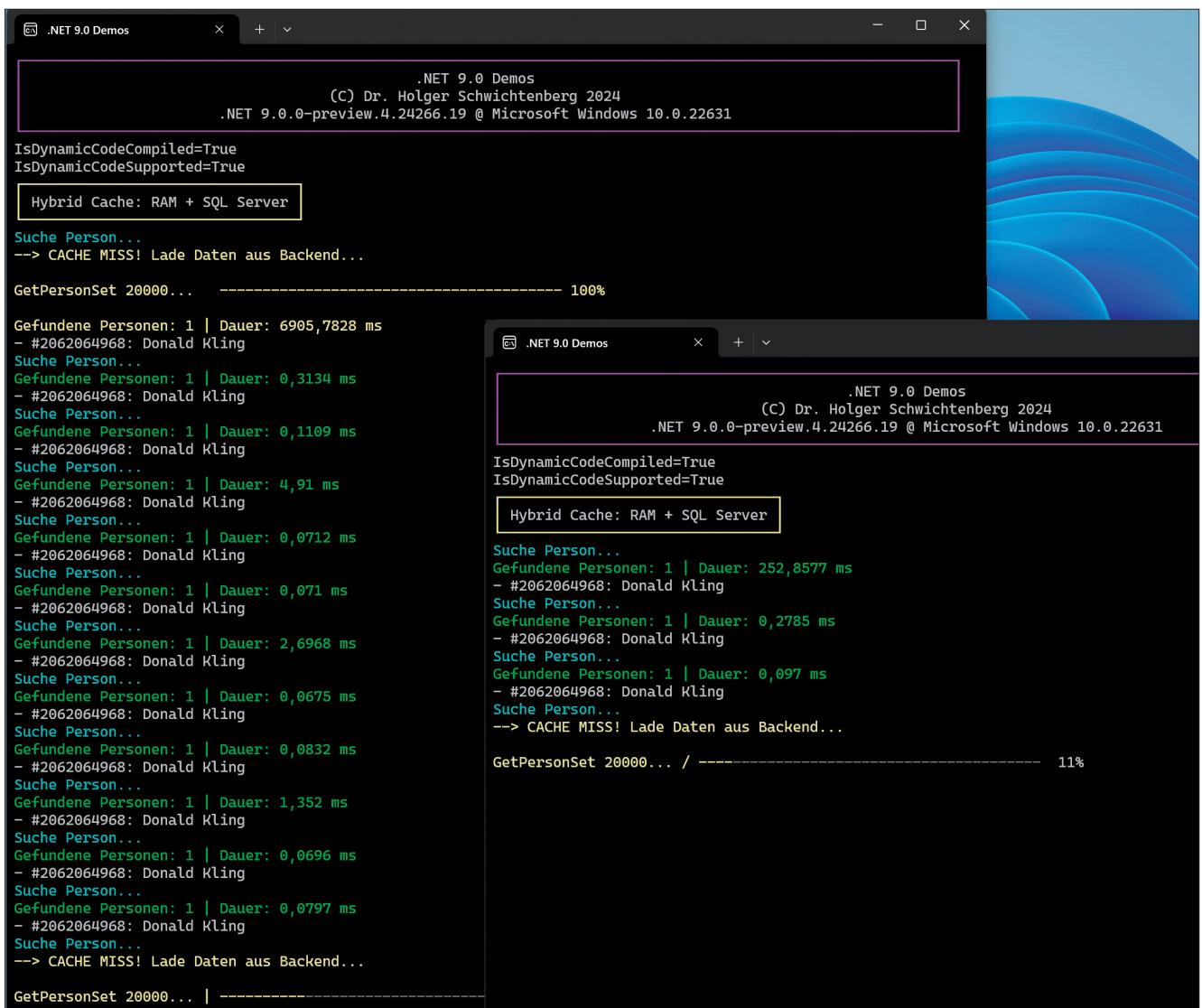
Listing 2: PersonManager ohne den hybriden Cache

```
public class PersonManagerOld(IDistributedCache cache)
{
    public async Task<List<Person>> SearchPerson
        (string name, CancellationToken token = default)
    {
        var key = $"SearchPerson:{name}"; // Schlüssel
        var bytes = await cache.GetAsync(key, token);
        // Ist der Schlüssel im Cache?
        List<Person> pSet;

        if (bytes is null) // Nein (Cache Miss)
        {
            // hole Daten aus Backend
            pSet = await new
                Backend().SearchPersonOnServer(name, token);
            // Serialisieren und Cachen

            string json = JsonSerializer.Serialize(pSet);
            await cache.SetAsync(
                key, Encoding.UTF8.GetBytes(json), token);
        }
        else
        {
            // Ja (Cache Hit) --> Deserialisieren!
            pSet = JsonSerializer.Deserialize<List<Person>>(
                Encoding.UTF8.GetString(bytes));
        }

        return pSet;
    }
}
```



Doppelter Start der Hybrid-Cache-Demo (Bild 4)

SQLQuery1.sql - D1...Cache (ITV\HS (61))

```
1 /***** Script for SelectTopNRows command from SSMS *****/
2 SELECT TOP (1000) [Id]
3     , [Value]
4     , [ExpiresAtTime]
5     , [SlidingExpirationInSeconds]
6     , [AbsoluteExpiration]
7 FROM [NET_Cache].[dbo].[Cache]
8
9 /***** Script for SelectTopNRows command from SSMS *****/
10 SELECT TOP (1000) [Id]
11     , [Value]
12     , [ExpiresAtTime]
13     , [SlidingExpirationInSeconds]
14     , [AbsoluteExpiration]
15 FROM [NET_Cache].[dbo].[Cache2]
```

100 %

Results Messages

Id	Value	ExpiresAtTime	SlidingExpirationInSeconds	AbsoluteExpiration
1	SearchPerson:Schwichtenberg	0x5B7B224944223A3132332C2246756C6C4E616D6522...	2024-05-31 17:47:21.0340935 +00:00	NULL

Zeile mit dem zwischengespeicherten Wert im SQL Server. Die erste Tabelle „Cache“ ist leer. Das Caching findet nur in der zuletzt hinzugefügten Tabelle „Cache2“ statt (Bild 5)

Implementierung findet, sendet er bei `SetAsync()` beziehungsweise `GetOrCreateAsync()` alle neuen Daten automatisch an den Second-Level-Cache. Wenn der hybride Cache in mehreren Prozessen (zum Beispiel einer Application-Server-Farm) eingesetzt wird, könnte es also den Fall geben, dass ein Wert im lokalen RAM nicht vorhanden ist, wohl aber im Second-Level-Cache, und dann von dort genommen wird. Durch Flags sind sowohl der First- als auch der Second-Level-Cache fallweise abschaltbar.

Wenn der hybride Cache keine Implementierung von `IDistributedCache` findet, speichert er nur im RAM.

Spannend war der Test, wenn man mehrere Implementierungen von `IDistributedCache` im Dependency-Injection-Container registriert, denn zu dieser Variante steht nichts in der Dokumentation [3]. In diesem Fall verwendet der hybride Cache die als letzte registrierte Implementierung von `IDistributedCache`.

Der neue hybride Cache serialisiert nicht nur Byte-Arrays, sondern alle Datentypen, wobei Zeichenketten und Byte-Array direkt serialisiert werden, alle anderen Datentypen über JSON via `System.Text.Json`. Entwicklerinnen und Entwickler können mit `WithSerializer()` hinter `AddHybridCache()` andere Serialisierungsimplementierungen (zum Beispiel XML oder Protobuffers) registrieren.

Zudem bietet Hybrid Cache einen Schutz vor Ansturm (Stampede): Hybrid Cache verhindert mehrere fast gleichzeitige Abrufe des gleichen Schlüssels und sorgt dafür, dass die Factory-Funktion nur einmal aufgerufen wird. Wenn bereits ein Factory-Funktionsaufruf für einen Schlüssel läuft, werden weitere Aufrufe zu diesem Schlüssel in eine Warteschlange gestellt, bis die Factory-Funktion beendet ist und der Wert im Cache vorliegt.

In Summe wird die Implementierung von Caching durch den hybriden Cache wesentlich prägnanter, was man beim Vergleich der Methode `SearchPerson()` in Listing 1 (hybrider

Cache) und der gleichnamigen Methode in Listing 2 (gleiche Implementierung mit `IDistributedCache`) sofort merkt.

Ausgabe des Beispiels

Wenn man die Anwendung startet, sieht man, dass

- zu Beginn die Daten aus dem Backend geladen werden, was rund sieben Sekunden dauert,
- danach für jeweils zwei Sekunden die Daten sehr schnell bereitstehen (< eine Millisekunde), weil sie aus dem RAM kommen,
- und alle drei Sekunden ein Zugriff auf den Second-Level-Cache erfolgt (circa 2,5 bis 5 Millisekunden).

Wenn man die Anwendung zweimal nacheinander startet, sieht man, dass der zweite Prozess die Daten nicht mehr aus dem Backend laden muss, sondern sie im SQL-Server-Cache findet (was aber inklusive Verbindungsaufbau 250 Millisekunden beim ersten Mal dauert), siehe Bild 4 und Bild 5. ■

[1] NuGet-Paket NCache,
www.dotnetpro.de/SL2504-05DataAccess1

[2] IDistributedCache updates in .NET 9, Issue 53255,
www.dotnetpro.de/SL2504-05DataAccess2

[3] Microsoft Learn, Overview of caching in ASP.NET Core,
www.dotnetpro.de/SL2504-05DataAccess3

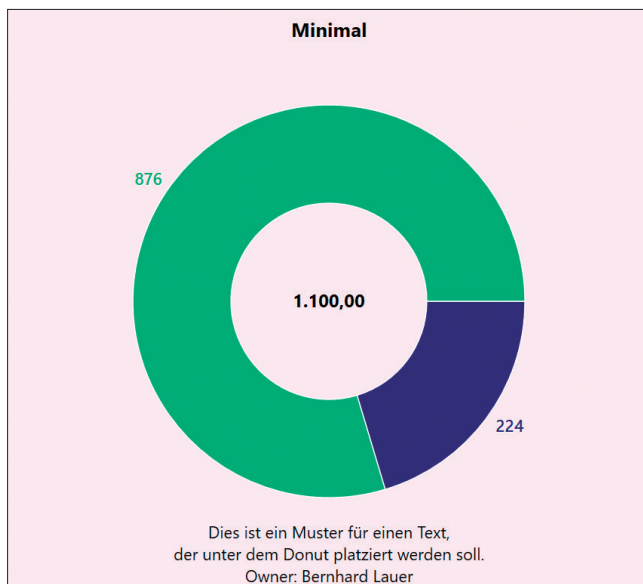
Dr. Holger Schwichtenberg
gehört zu den bekanntesten .NET- und Webexperten in Deutschland. Von Microsoft ist er seit 21 Jahren als MVP ausgezeichnet. Mit dem 53-köpfigen Expertenteam bei www.IT-Visions.de bietet er Beratungen, Schulungen und Softwareentwicklung an.
www.dotnet-doktor.de

CHART-KOMPONENTEN IN VISUAL BASIC

Donut im Eigenbau

Einfach nutzen oder individuell ausbauen.

Eine meiner Aufgaben bestand kürzlich darin, ein in Excel umgesetztes Finanzprogramm auf .NET zu portieren und dabei zu modernisieren. Umgesetzt wurde die Aufgabe in Visual Basic .NET und mit der Windows Presentation Foundation (WPF), weil das .NET-Programm künftig ausschließlich auf Windows-Desktop-Rechnern laufen soll. Der Umbau gestaltete sich nicht besonders schwierig und war in kurzer Zeit erledigt. Mit einer Ausnahme: Die Excel-Anwendung überzeugte vor allem mit ihrem Dashboard, welches die aktuellen Daten übersichtlich in Form von Charts zusammengefasst hat. Um das Dashboard nachzubauen, waren unabdingbar Chart-Komponenten für .NET erforderlich.



Minimaler Donut-Chart (Bild 1)

Die in vielen kostenpflichtigen Komponentensammlungen enthaltenen Charts kamen wegen der Kosten nicht infrage. Eine kurze Recherche förderte die .NET-Chartbibliothek LiveCharts2 [1] zutage, die unter der kostenlosen MIT-Lizenz läuft und optional ein kostenpflichtiges Paket zur Verbesserung der Leistung und Erweiterung der Funktionen umfasst. So wunderbar die animierten Beispiele auf der Webseite der Bibliothek auch waren, ein Blick in die Dokumentation zeigte, dass diese Lösung wohl etwas zu umfangreich für das Vorhaben ist. So reifte der Entschluss, die Zeit nicht in die Einarbeitung zu investieren, sondern mit VB.NET und WPF etwas Eigenes aufzubauen. Die Ergebnisse – vorerst nur die Donut- und Kreisdiagramme – werden hier vorgestellt.

Ziel der Arbeit war ein besonders einfach zu nutzendes System, das zugleich flexible Anpassungen erlaubt, aber längst nicht alle Möglichkeiten eines vollständigen Profichartsystems umfasst. Wichtiger war, dass es übersichtlich bleibt und leicht weiterzuentwickeln ist. Mit dem Stempel „Yagni“ (You ain't gonna need it) versehen wurden neben den Animationen zum Beispiel die Möglichkeit, einzelne Kreissegmente aus dem Chart herauszuziehen, oder die automatische Vermeidung von Kollisionen der Segmentbeschriftungen. Letzteres wurde einfach damit abgefangen, dass Anwender den Kreis bei Bedarf per Mausklick drehen und eventuell ineinanderlaufende Texte damit entflechten können.

Um einen blDonut – so der Name des Chart-Elements – zu erzeugen, genügt der folgende minimale Aufruf:

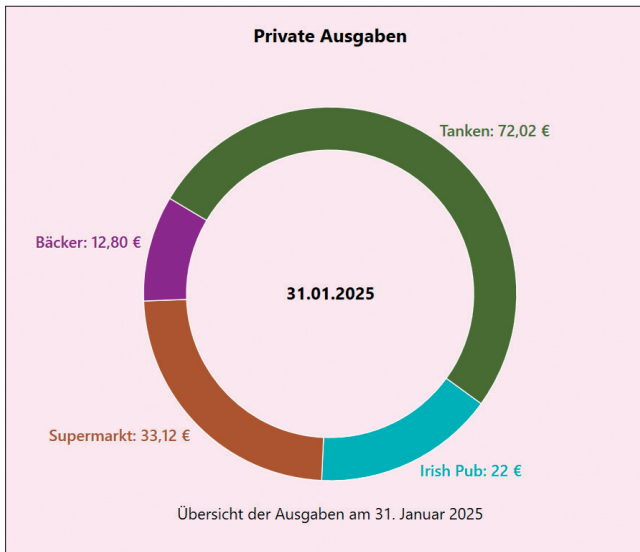
```
blDonut(canvas, "Minimal", {224, 876})
```

Dabei ist *canvas* der Name der im XAML-Code angelegten Leinwand, auf die der Donut gezeichnet werden soll, *Minimal* ist die Überschrift des Charts, und das Array {224, 876} liefert die beiden Werte, für die jeweils ein Kreissegment dargestellt werden soll. Alle übrigen für den Aufbau des Donuts erforderlichen Werte setzt blDonut anhand seiner Vorgaben selbst ein. In Bild 1 sehen Sie das Ergebnis.

Die Canvas, auf welcher der Donut gezeichnet wird, gibt der WPF-XAML-Code vor, beispielsweise so:

```
<Window x:Class="MainWindow"
...
Title="blDonut" Width="1600" Height="1000">
<Grid>
    <StackPanel x:Name="stpChart"
        VerticalAlignment="top">
        <Canvas x:Name="canvas" Margin="20"
            Width="1040" Height="980"
            Background="LavenderBlush"
        />
    ...
</StackPanel>
...
</Window>
```

Damit ist dem Programm die Leinwand für die Grafik bekannt. Den minimalen Aufruf von blDonut von oben kann man jetzt einfach ins Ereignis *MainWindow_Loaded(...)* setzen, und die Grafik wird beim Programmstart gezeichnet. Die im *StackPanel stpChart* angelegte Canvas hat im Beispiel die Maße 1040 x 980 Pixel. Diese Werte für Breite und Höhe nutzt



Donut mit individuellen Farben und selbst gewähltem Innendurchmesser (Bild 2)

blDonut, um passende Maße für den Chart zu ermitteln. Außerdem sind Standardfarben hinterlegt, sodass man auf Anhieb eine ansprechende Grafik erhält. Der Hintergrund *LavenderBlush* wird im XAML-Code gesetzt und kann folglich selbst gewählt werden, auch wenn man weiterhin die Standardfarben nutzen möchte.

Die übergebenen Werte werden jeweils neben der Mitte des zugehörigen Kreissegments in der Segmentfarbe als Beschriftung angezeigt, und die Summe der Werte steht, wie in Bild 1 zu sehen ist, im „Auge“ des Donuts, sofern dort genügend Platz für den Wert ist.

Flexibel ausgebaut

Das minimale API ist allerdings nur für den schnellen Donut zwischendurch gedacht. Die Methode *blDonut* bietet darüber hinaus eine Menge Möglichkeiten, den Chart an spezielle Anforderungen anzupassen. Welche Optionen zur Verfügung stehen, zeigt Listing 1. Alle darin als optional angegebenen Parameter kann man beim Aufruf der Methode übergeben und damit das Standardverhalten ändern.

Der folgende Beispielaufruf hat vier Segmente, die Überschrift lautet *Private Ausgaben*, in der Mitte steht anstelle der Summe der Werte das Datum, bis zu dem das Geld ausgegeben wurde. Vor den Werten steht, wofür das Geld verwendet wurde, und dahinter das Euro-Zeichen. Zudem sorgt der Aufruf für von der Standardpalette abweichende Farben und wählt einen größeren Innenradius, wodurch der Donut dünner gezeichnet wird. Abschließend wird bestimmt, dass der Chart um 36 Grad im Uhrzeigersinn gedreht wird.

```
blDonut(canvas,
    headline:"Private Ausgaben",
    Farben:={"DarkTurquoise", "Sienna", "Purple",
            "DarkOliveGreen"},
    vorWert:={"Irish Pub: ", "Supermarkt: ",
            "Bäcker: ", "Tanken: "},
    werte:={22.0, 33.12, 12.8, 72.02},
    nachWert:={" €"},
    textMitte:"31.01.2025",
    CU:"Übersicht der Ausgaben am 31. Januar 2025",
    innerRadius:=200,
    stAngle:=36)
```

```
werte:={22.0, 33.12, 12.8, 72.02},
nachWert:={" €"},
textMitte:"31.01.2025",
CU:"Übersicht der Ausgaben am 31. Januar 2025",
innerRadius:=200,
stAngle:=36)
```

Die optionalen Parameter wurden hier mit ihrem Namen angegeben (Beispiel: *Farben:={"DarkTurquoise", ...}*), sodass die Reihenfolge nicht beachtet werden muss. Den durch diesen Aufruf erzeugten Chart sehen Sie in Bild 2. Die Abkürzung CU steht übrigens für Chart-Unterschrift. Beim Abschätzen, um wie viel man den Startwinkel (*stAngle*) verändern sollte (*Standard = 0*), hilft der Debug-Modus, den Sie weiter unten kennenlernen.

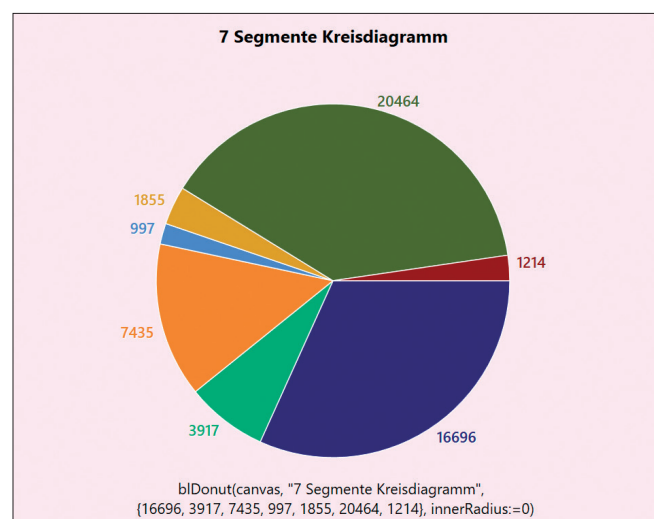
Absolute Werte, wie hier für den Innendurchmesser, sollte man mit etwas Fingerspitzengefühl wählen. Bestimmt man aus Versehen zum Beispiel einen inneren Radius, der größer ist als der hier automatisch berechnete äußere Radius, führt das zwar zu keinem Absturz, aber teilweise zu seltsamen Ergebnissen, bei denen die Beschriftungen innerhalb des Donuts liegen.

Sonderfall Kreisdiagramm

Um anstelle eines Donuts einen Kreis zeichnen zu lassen, genügt es, den Innenradius auf 0 zu setzen. blDonut überprüft, ob der in die Mitte zu schreibende Text dort auch Platz hat. Ist das nicht der Fall, wird er automatisch weggelassen. Hier der Aufruf für das einfache Kreisdiagramm, das Sie in Bild 3 sehen:

```
blDonut(canvas,
    "7 Segmente Kreisdiagramm",
    {16696, 3917, 7435, 997, 1855, 20464, 1214},
    innerRadius:=0)
```

Die Nutzung und der Zweck der in Bild 2 und Bild 3 benutzten optionalen Parameter dürfte klar sein. Die weiteren optionalen Parameter sind *FontSize* für die Schriftgröße der Segment-



Mit einem Innendurchmesser von 0 wird der Donut zum Kreisdiagramm (Bild 3)

werte, *TopMargin* für einen größeren oder kleineren Abstand vom oberen Rand der Canvas und *outerRadius* für die Größe des Donut-Kreises insgesamt.

Bild 4 zeigt ein Beispiel für einen schlanken Donut mit dem Maximalausbau von neun Segmenten und einer zweizeiligen Überschrift, welche einfach durch einen in den String eingefügten Zeilenumbruch erzeugt wird. Der Code für den Aufruf sieht so aus:

```
Dim Headline As String = "9 Segmente," &
vbCrLf & "großer Innenradius"
Dim Werte() As Decimal = {2400, 9030, 220, 4620,
1340, 1880, 16000, 1450, 999}
Dim vorWert() As String = {"1 --> ", "2 --> ",
"3: --> ", "4: --> ", "5: --> ", "6: --> ",
"7: --> ", "8: --> ", "9: --> "}
```

```
b1Donut(canvas, Headline, Werte,
vorWert:=vorWert, FontSize:=20,
outerRadius:=220, innerRadius:=190,
stAngle:=45, CU:=" ")
```

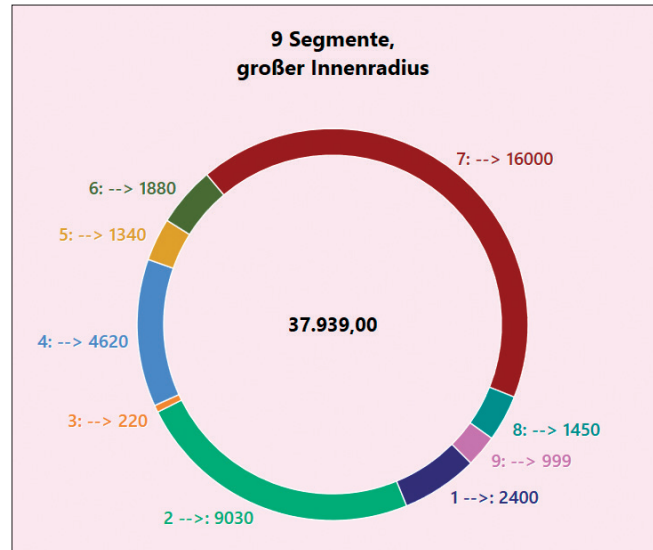
Das Zentrieren der beiden Zeilen der Überschrift auf der Canvas übernimmt *b1Donut* ohne weiteres Zutun. Außerdem erlaubt *b1Donut* es noch, den Parameter *DebugMode* auf *True* zu setzen:

```
b1Donut(... DebugMode:=True)
```

Dadurch werden zum eigentlichen Chart noch Begrenzungslinien für die Canvas und Linien durch den Mittelpunkt des Kreises hinzugefügt. Die Segmentbeschriftungen erhalten einen rechteckigen Rahmen, und es gibt eine Linie zum Kreispunkt in der Mitte des aktuellen Segments zum oberen linken

● Listing 1: Signatur der Methode *b1Donut()*

```
Sub b1Donut(canvas As Canvas,
headline As String,
werte() As Decimal,
Optional Farben() As String = Nothing,
Optional vorWert() As String = Nothing,
Optional nachWert() As String = Nothing,
Optional textMitte As String = "",
Optional FontSize As Integer = 0,
Optional stAngle As Integer = 0,
Optional TopMargin As Integer = 0,
Optional outerRadius As Integer = 0,
Optional innerRadius As Integer = -1,
Optional CU As String = "",
Optional DebugMode As Boolean = False
)
Const MaxSegmente = 9
...
```



Dünner Ring mit den als Maximum festgelegten 9 Werten (**Bild 4**)

Startpunkt des Segmenttextes. Die Chart-Unterschrift meldet außerdem alle Werte zum Chart. In **Bild 5** sehen Sie ein Beispiel, dessen Aufrufdaten in der folgenden Zeile stehen:

```
b1Donut(canvas, "Prozentwerte, Debug-Modus",
{70, 17, 9, 4}, vorWert:={"FestVerz. ",
"Aktien ", "Liquide ", "Bitcoin "},
nachWert:={"%"}, textMitte:="Plan 2026",
DebugMode:=True, outerRadius:=240,
innerRadius:=180)
```

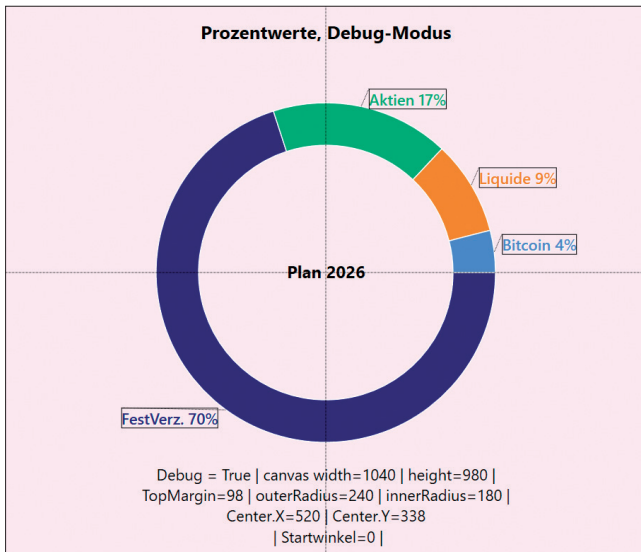
Blick in den Maschinenraum

Beim Einsatz von *b1Donut* kommt man nur mit der gerade vorgestellten, gleichnamigen Methode in Berührung. Diese Methode ist in ein Modul namens *ChartModul* ausgelagert. Im Modul wird zunächst die Struktur *DonutSkelett* angelegt, die neben den schon bekannten Daten auch die Anzahl der Segmente als Integer-Wert aufnimmt, siehe **Listing 2**. Diese Struktur wird instanziiert mit *Dim don As DonutSkelett* und im weiteren Verlauf an alle Methoden durchgereicht, die Zugriff auf diese Daten benötigen.

Die Methode *b1Donut()*, deren Signatur sie schon in **Listing 1** gesehen haben, ermittelt die Anzahl der Segmente aus der Anzahl der übergebenen Werte und überträgt die Eingabeparameter nach einer Fehlerprüfung in eine Variable vom Typ *DonutSkelett*. Für fehlende Daten werden Standardwerte eingesetzt, beispielsweise das Array mit den Standardfarben.

```
If Farben Is Nothing Then Farben = {
"DarkBlue", "MediumSeaGreen",
"DarkOrange", "CornflowerBlue", "Goldenrod",
"DarkOliveGreen", "DarkRed", "DarkCyan",
"Orchid", "Black"}
```

Zu den Werten, die im Aufruf von *b1Donut* nicht angegeben werden müssen, gehört auch die Größe des Charts. Diese wird anhand der folgenden Relationen automatisch gesetzt:



Debug-Modus mit Größenangaben in der Chart-Unterschrift (Bild 5)

```
Dim don As DonutSkelett
...
With don
    ...
    .outerRadius = .canvas.Width \ 4,
    .innerRadius = .outerRadius \ 2,
    .TopMargin = .canvas.Height \ 10,
    .fontSize = Math.Min(20, .outerRadius \ 10)
    ...
End With
```

Zum Abschluss ruft blDonut noch die *Init*-Methode des DonutSkeletts auf, welche den Mittelpunkt des Kreises ermittelt, dessen y-Koordinate abhängig von den angegebenen Größenangaben ist.

```
Structure DonutSkelett
    ...
    Sub Init()
        Center.X = canvas.Width \ 2
        Center.Y = TopMargin + outerRadius
    End Sub
End Structure
```

Dann übergibt blDonut das DonutSkelett an die Methode *MakeDonut*, die das Zeichnen des Charts übernimmt.

```
MakeDonut(don)
```

Die Kernmethoden

Die Methode *MakeDonut* hat dank der Struktur *DonutSkelett* eine sehr einfache Signatur:

```
Sub MakeDonut(d As DonutSkelett)
```

Alle noch anfallenden Aufgaben werden von dieser Methode erledigt:

- In einer *For*-Schleife werden die Kreissegmente berechnet, gezeichnet, mit Eventhandlern für Mausklicks versehen, denen ebenfalls das DonutSkelett übergeben wird, und auch das Einsetzen der Segmenttexte wird hier erledigt, im Debug-Modus inklusive Rahmen und Linie von Kreis zum Textblock.
- Der Text für das „Auge“ des Donuts wird eingesetzt, sofern dort genug Platz dafür vorhanden ist.
- Im Debug-Modus werden die Daten für die spezielle Chart-Unterschrift zusammengestellt.
- Die Chart-Unterschrift wird platziert.

Die *For*-Schleife läuft von 0 bis zur Anzahl der Segmente, die sich aus der Anzahl der darzustellenden Werte ergibt und schon im DonutSkelett gespeichert ist. Das erste Kreissegment startet beim im Skelett hinterlegten Startwinkel (*stAngle*, Standard = 0) und läuft bis zum Endwinkel *ea*. Dieser wird ermittelt über den Anteil, den der aktuelle Wert an der Gesamtsumme aller Werte (*total*) hat. Die Berechnung innerhalb der Schleife sieht somit so aus:

```
' Endwinkel ea des Segments berechnen
ea = d.werte(i) / total * 360 + stAngle
```

Das genügt, um mit der Funktion *CreateDonutSegment* ein Segment vom Datentyp *Path* zu berechnen: ►

Listing 2: ChartModul mit Struktur DonutSkelett

```
Module ChartModul
    Structure DonutSkelett
        Dim canvas As Canvas
        Dim AnzSegmente As Integer
        Dim TopMargin As Integer
        Dim Headline As String
        Dim outerRadius As Integer
        Dim innerRadius As Integer
        Dim Center As Windows.Point
        Dim startAngle As Integer
        Dim werte() As Double
        Dim Farben() As String
        Dim vorWert() As String
        Dim nachWert() As String
        Dim textMitte As String
        Dim fontSize As Integer
        Dim CU As String ' Chart-Unterschrift
        Dim DebugMode As Boolean

        Sub Init()
            Center.X = canvas.Width \ 2
            Center.Y = TopMargin + outerRadius
        End Sub
    End Structure
    ...
End Module
```


Listing 3: CreateDonutSegment

```
Private Function CreateDonutSegment(
    d As DonutSkelett,
    startAngle As Double,
    endAngle As Double) As Geometry

    Dim startOuter As System.Windows.Point =
        GetPoint(d.Center, d.outerRadius,
            startAngle)
    Dim endOuter As System.Windows.Point =
        GetPoint(d.Center, d.outerRadius, endAngle)
    Dim startInner As System.Windows.Point =
        GetPoint(d.Center, d.innerRadius,
            startAngle)
    Dim endInner As System.Windows.Point =
        GetPoint(d.Center, d.innerRadius, endAngle)

    Dim pathFigure As New PathFigure() With {
        .StartPoint = startOuter
    }
    pathFigure.Segments.Add(
        New ArcSegment(endOuter,
            New System.Windows.Size(
                d.outerRadius, d.outerRadius), 0,
                endAngle - startAngle > 180,
                SweepDirection.Clockwise, True))
    pathFigure.Segments.Add(
        New LineSegment(endInner, True))
    pathFigure.Segments.Add(
        New ArcSegment(startInner,
            New System.Windows.Size(
                d.innerRadius, d.innerRadius), 0,
                endAngle - startAngle > 180,
                SweepDirection.Counterclockwise, True))
    pathFigure.Segments.Add(
        New LineSegment(startOuter, True))

    Dim pathGeometry As New PathGeometry()
    pathGeometry.Figures.Add(pathFigure)

    Return pathGeometry
End Function
```

```
segment(i).Data =
    CreateDonutSegment(d, stAngle, ea)
```

Die Funktion *CreateDonutSegment* (Listing 3) ermittelt vier spezifische Punkte: Zwei befinden sich am äußeren Rand des Donuts, nämlich *startOuter* und *endOuter*, und zwei weitere am inneren Rand, nämlich *startInner* und *endInner*.

Die Berechnung durch die Funktion *Point*, deren Code Sie in Listing 4 finden, erfolgt anhand des Mittelpunkts des Donuts (*d.Center*) sowie des äußeren und inneren Radius (*d.outerRadius* und *d.innerRadius*). Außerdem fließen noch die Start- und Endwinkel (*startAngle* und *endAngle*) in das Funktionsergebnis ein.

Listing 4: Die Methode GetPoint

```
Private Function GetPoint(
    Center As Windows.Point,
    radius As Double,
    angle As Double) As System.Windows.Point

    Dim radian As Double = angle * Math.PI / 180

    Return New System.Windows.Point(
        Center.X + radius * Math.Cos(radian),
        Center.Y + radius * Math.Sin(radian))
End Function
```

Anschließend erstellt die Funktion eine *PathFigure*, die ihren Ursprung bei *startOuter* hat. Die nächsten Schritte beinhalten das Hinzufügen der Segmente:

- Ein Bogen (*ArcSegment*) verbindet den *startOuter*- mit dem *endOuter*-Punkt.
- Eine Linie (*LineSegment*) führt vom *endOuter*-Punkt zum *endInner*-Punkt.
- Ein weiterer Bogen (*ArcSegment*) verläuft vom *endInner*-Punkt zum *startInner*-Punkt.
- Eine abschließende Linie (*LineSegment*) führt vom *startInner*-Punkt zurück zum *startOuter*-Punkt, um das Segment zu schließen.

Zum Schluss wird eine *PathGeometry* angelegt, die *PathFigure* hinzugefügt und das fertige Objekt an die aufrufende Methode übermittelt.

Zurück in *MakeDonut* wird das Segment mit Farbe gefüllt, eine äußere Linie (*Stroke*) bestimmt, das Segment platziert und zur Leinwand hinzugefügt:

```
Dim bc As New BrushConverter
...
segment(i).Data =
    CreateDonutSegment(d, stAngle, ea)
segment(i).Fill =
    CType(bc.ConvertFromString(
        d.Farben(i)), SolidColorBrush)

segment(i).Stroke = Brushes.White
segment(i).StrokeThickness = 1.5
```

Listing 5: Höhe und Breite eines TextBlocks ermitteln

```
Private Function TextHöheUndBreite(  
    t As TextBlock,  
    pixelsPerDip As Double) As TextWH  
  
    ' TextWH ist eine Struktur mit den Double-  
    ' Werten width und height  
    ,  
  
    ' Public Structure TextWH  
    '     Dim width As Double  
    '     Dim height As Double  
    ' End Structure  
  
    Dim twh As New TextWH  
    Dim formattedText As New FormattedText(  
        t.Text,  
        System.Globalization.CultureInfo.CurrentCulture,  
        System.Windows.FlowDirection.LeftToRight,  
        New Typeface(t.FontFamily, t.FontStyle,  
        t.FontWeight, t.FontStretch),  
        t.FontSize, Brushes.Black, pixelsPerDip)  
  
    twh.width = formattedText.Width  
    twh.height = formattedText.Height  
  
    Return twh  
End Function
```

```
d.canvas.SetLeft(segment(i), 0)  
d.canvas.SetTop(segment(i), d.TopMargin)  
d.canvas.Children.Add(segment(i))  
...
```

Der Eventhandler für einen Klick mit der linken Maustaste soll den Startwinkel um sechs Grad weiterdrehen und dann *MakeDonut* erneut aufrufen. Das klappt, indem man dem Eventhandler das Donut-Skelett durchreicht:

```
AddHandler segment(i).MouseLeftButtonDown,  
    Sub(sender As Object, e As MouseButtonEventArgs)  
        Segment_MouseLeftButtonDown(sender, e, d)  
    End Sub  
...  
  
Private Sub Segment_MouseLeftButtonDown(  
    sender As Object,  
    e As MouseButtonEventArgs, d As DonutSkelett)  
    d.startAngle = (d.startAngle + 6) Mod 360  
    MakeDonut(d)  
End Sub
```

Der Segmenttext wird als WPF-*TextBlock* auf der Canvas platziert. Anzugeben ist dafür die linke obere Ecke des

Blocks. Dabei helfen die Funktionen *TextHöheUndBreite* sowie *StartPunktText*. Wie Höhe und Breite eines Blocks ermittelt werden, sehen Sie in Listing 5. Der Aufruf dieser Funktion berücksichtigt die Auflösung des Monitors und sieht folgendermaßen aus:

```
twh = TextHöheUndBreite(t(i),  
    VisualTreeHelper.GetDpi(  
        Application.Current.MainWindow).PixelsPerDip)
```

Bedeutend aufwendiger ist es, den Startpunkt für den Segmenttext zu finden. Er soll neben dem Kreis und in der Mitte des zugehörigen Segments stehen. Da der Text immer von links nach rechts läuft, ist es dafür erforderlich zu berücksichtigen, ob der Text links oder rechts vom Kreis sowie ob er oben oder unten stehen soll. Listing 6 zeigt einen Auszug für den Fall, dass der Text rechts unten (Winkel 0 bis 90 Grad) stehen soll. *tAng* ist dabei der Winkel – beginnt das Segment bei 0 Grad und ist der Endwinkel bei 60 Grad, wäre der gesuchte ►

Listing 6: Startpunkt für den Segmenttext ermitteln

```
Private Function StartPunktText(  
    tAng As Integer,  
    d As DonutSkelett,  
    twh As TextWH,  
    FontSize As Integer) As Windows.Point  
  
    Dim Kpt As System.Windows.Point =  
        GetPoint(d.Center, d.outerRadius, tAng)  
    Dim t0 As System.Windows.Point  
    Dim Abstand As Integer = FontSize / 2  
  
    ...  
  
    If tAng >= 0 And tAng <= 90 Then  
        dx = Abstand  
        dy = Abstand - twh.height / 2  
        t0.X = Kpt.X + dx  
        t0.Y = d.TopMargin + Kpt.Y + dy  
  
        If d.DebugMode Then  
            Linie(d.canvas,  
                New Windows.Point(t0.X, t0.Y),  
                New Windows.Point(Kpt.X, Kpt.Y +  
                    d.TopMargin))  
        End If  
  
    ElseIf ...  
        ...  
    End If  
    ...  
    Return t0  
End Function
```

Listing 7: Chart-Unterschrift auf Canvas setzen

```

Sub CU_auf_Canvas_setzen(d As DonutSkelett)
    ' Erstelle den Textblock
    Dim tb As New TextBlock()
    With tb
        .Text = d.CU
        .Foreground = Brushes.Black
        .FontSize = d.fontSize
        .FontWeight = FontWeights.Normal
        .HorizontalAlignment =
            System.Windows.HorizontalAlignment.Center
        .VerticalAlignment =
            System.Windows.VerticalAlignment.Center
        .TextAlignment = TextAlignment.Center
    End With

    Dim startPoint As New Windows.Point

    Dim twh As TextWH
    twh =
        TextHöheUndBreite(tb,
            VisualTreeHelper.GetDpi(
                Application.Current.MainWindow)
                .PixelsPerDip)

    startPoint.X = d.Center.X - twh.width \ 2
    startPoint.Y = d.Center.Y + d.outerRadius +
        d.TopMargin + 30

    d.canvas.SetLeft(tb, startPoint.X)
    d.canvas.SetTop(tb, startPoint.Y)
    d.canvas.Children.Add(tb)
End Sub

```

Winkel $tAng$ 30 Grad. Kpt ist der Bezugspunkt für den Segmenttext auf dem Kreis (für den Winkel $tAng$), der Abstand sorgt dafür, dass der Text nicht direkt am Kreis beginnt oder endet, $t0$ ist das Ergebnis der Funktion. Im DebugModus wird eine Linie vom Kreispunkt zum Startpunkt des Textes gezogen (Bild 6).

Gesetzt wird der Segmenttext in der Methode *MakeDonut* über die folgenden Zeilen – $t(i)$ repräsentiert hier den zum Segment i gehörenden *TextBlock*:

```

d.canvas.SetLeft(t(i), t0.X)
d.canvas.SetTop(t(i), t0.Y)
d.canvas.Children.Add(t(i))

```

Jetzt fehlt noch der Text für das „Auge“ des Donuts, wofür die Länge des *TextBlocks* wie oben über die Funktion *TextHöheUndBreite* gemessen und mit dem Innendurchmesser ($d.innerRadius * 2 - 4$) verglichen wird. Zwei Pixel rechts und links vom Text soll der Mindestabstand betragen.

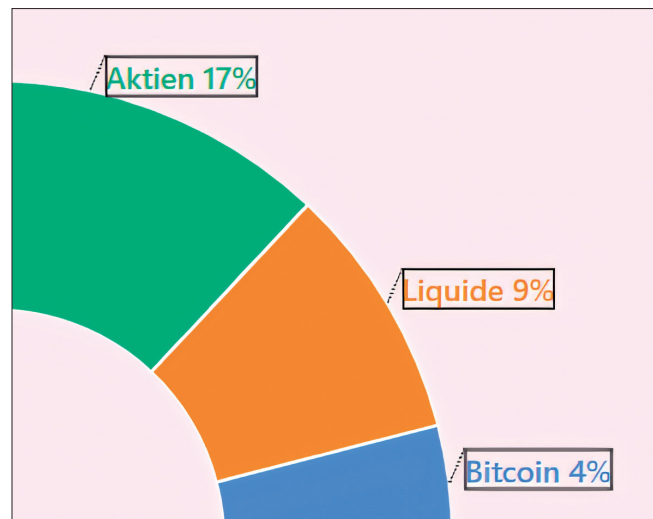
Während man den einzelnen Text in der Mitte des Donuts noch leicht selbst zentrieren kann, wäre das im Fall einer mehrzeiligen Überschrift oder für die Chart-Unterschrift schwierig. Da hilft aber folgende *TextBlock*-Eigenschaft:

```
TextBlock.TextAlignment = TextAlignment.Center
```

Sie zentriert jede einzelne Zeile des Textblocks ohne weiteres Zutun. Das Einsetzen der Chart-Unterschrift erledigt die Funktion *CU_auf_Canvas_setzen*, der das *DonutSkelett* übergeben wird und deren Code in Listing 7 zu finden ist.

Fazit

Das Chart-Modul *blDonut* ist sehr einfach zu nutzen und dabei flexibel, weil sich die Geometrie des Charts am auf der Canvas vorhandenen Platz orientiert, gegebenenfalls aber



Im Debug-Modus wird eine Linie vom Kreispunkt „Kpt“ zur linken oberen Ecke des eingerahmten Beschriftungstextes gezogen (Bild 6)

auch strikt vorgegeben werden kann und zudem noch etliche optionale Feineinstellungen vorgesehen sind. ■

[1] LiveCharts2, <https://livecharts.dev>



Bernhard Lauer

beschäftigt sich seit Jahrzehnten mit IT-Themen und bereitet diese als Autor und Redakteur auf. Programmieren gelernt hat er mit dem C64 und Basic. Er hat über die Anfänge von Java, JavaScript, HTML und .NET berichtet und sich zuletzt mit Python befasst.

dnpCode

A2504-05VBDonuts

M:CON CONGRESS CENTER
ROSENGARTEN | MANNHEIM
30. JUNI - 3. JULI 2025

PROGRAMM DEVELOPER WEEK '25

DWX

DAS EVENT FÜR AI, CLOUD, WEB UND .NET

Software Design Principles | Software Architecture | Software Quality | Software Testing | Software Security | Doing Agile/Agile in Practice | DevOps | CI / CD
Git, GitHub | .NET Fundamentals | .NET Backend (Development)
.NET Frontend (Development) | .NET Doing | Web Frontend | Web Backend
Web Hybrid | Web Design | Web Languages | Cloud Development
Container | Data & Analytics | Databases | GenAI Fundamentals
GenAI Tools & Solutions | Personal Growth | From Me to We | Sustainability
Robotics | IoT & Industry 4.0

developer-week.de | #DWX | Find us on   

Präsentiert von: 

WARUM SICH WEITERBILDUNG UND DIE TEILNAHME AN DER DWX LOHNT



„FORTBILDUNG WIRD IMMER MEHR ZUM WETTBEWERBSVORTEIL“

Kommentar der Conference-Chairs Fernando Schneider und Tilman Börner

Keine Zeit für Fortbildung, weil die Kolleg:innen fehlen?

Diese Einstellung ist unserer Meinung nach falsch.

Denn Fachkräftemangel, Künstliche Intelligenz, täglich neue Frameworks & Tools und vieles mehr lassen aus unserer Sicht nur eine Schlussfolgerung zu: Sich regelmäßig upzudaten und so auf dem Laufenden zu bleiben – technisch wie auch methodisch – ist erste Entwickler:innenpflicht. Nehmen Sie sich also die Zeit, um Ihre Säge zu schärfen, damit Sie die nächsten hundert Softwarebäume viel schneller umschneiden können.

Der Fortbildung hat sich die Developer Week seit jeher verpflichtet.

Ihr umfangreiches Programm aus Ganz- und Halbtags-Workshops sowie Sessions bietet alles, was das Entwickler:innenherz begehrt.

Gepaart mit der Möglichkeit, sich an einem Ort eine Woche lang mit Gleichgesinnten und Expert:innen in lockerer Atmosphäre intensiv austauschen zu können, liefert sie Ihnen wichtige Anregungen, Ideen und Orientierung, um die Herausforderungen am Arbeitsplatz zu meistern.

Starke Argumente für eine Teilnahme an der DWX.

Sehen Sie auch so?

Dann freuen wir uns zusammen mit den Expert:innen auf das Gespräch mit Ihnen auf der Konferenz. Seien Sie herzlich willkommen.

EXPO DER DWX

JETZT
PARTNER
ENTDECKEN

Ausgewählte Partner und Sponsoren stellen sich der DWX-Community binnen drei Tagen vor.

adesso

adesso ist einer der führenden unabhängigen IT-Dienstleister im deutschsprachigen Raum und konzentriert sich mit Beratung sowie individueller Softwareentwicklung auf die Kerngeschäftsprozesse von Unternehmen und öffentlichen Verwaltungen. Die Strategie von adesso beruht auf drei Säulen: einem umfassenden Branchen-Know-how der Mitarbeitenden, einer breiten, herstellerneutralen Technologiekompetenz und erprobten Methoden bei der Umsetzung von Softwareprojekten. Das Ergebnis sind IT-Lösungen, mit denen Unternehmen wettbewerbsfähiger werden. Zu unseren Kernbranchen zählen Automotive, Banken und Finanzdienstleistungen, Gesundheitswesen, Life Science, Lotterrie, Energieversorgung, Manufacturing Industry, Messegesellschaften, Öffentliche Verwaltung, Verkehrsbetriebe, Versicherungen/Rückversicherungen und Handel. adesso wurde 1997 gegründet. Die adesso Group ist mit über 9.000 Mitarbeitenden und einem Jahresumsatz 2022 von 900,25 Mio Euro eines der größten deutschen IT-Dienstleistungsunternehmen.

TEXTCONTROL

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 handhabbaren Textverarbeitung kombiniert - voll programmier- und integrierbar in einer .NET-Anwendung.

fecher.

Seit der Firmengründung im Jahr 1987 hat sich fecher vom Softwareentwickler für Individualprojekte und Hersteller von Tools und Klassenbibliotheken für Legacy-Software zum ausgewiesenen Spezialisten für Anwendungsmodernisierung entwickelt. Effektive Tools, die im Laufe von über 500 erfolgreichen Portierungsprojekten kontinuierlich weiterentwickelt wurden, ermöglichen dabei ein hohes Maß an Automatisierung. So lassen sich unternehmenskritische Altanwendungen und Datenbanken zuverlässig, effizient und schlüsselfertig migrieren.

Unsere Services umfassen:

- Automatisierte Migration von Legacy-Anwendungen nach .NET (u. a. Visual Basic 6, Gupta Team Developer, Microsoft Access) - inklusive Datenbankmigration
- Web-Enabling von bestehenden .NET-Desktopanwendungen (WPF oder WinForms) mit Wisej.NET
- Wisej.NET-Services als offizieller Partner: Webentwicklung, Projektbegleitung, Training und Beratung rund um Wisej.NET
- UI/UX-Redesign im Rahmen von Modernisierungsprojekten

oose.

Das selbstorganisierte Schulungs- und Beratungshaus in Hamburg.

Seit über 20 Jahren bieten wir exzellente Seminare, Workshops, Beratung und Projektunterstützung für Software & Systems Engineering, New Work und Innovation. Wir begleiten Veränderungen, führen neue Methoden ein, vermitteln Wissen - und befähigen Menschen dazu, eigene Lösungen zu finden. Ob vor Ort bei unseren Kunden und Kundinnen, bei uns auf dem oose campus in Hamburg oder online.

exeta

18 Standorte. Über 1200 Kolleg:innen. 1 Team. Jeder Einzelne ist unsere größte Stärke. Gemeinsam begleiten wir Transformationen, verändern Märkte und Mindsets. Egal, ob in Berlin oder Bratislava, uns vereint unsere Leidenschaft für Technologie, unser Teamspirit und der Drang, die Welt nachhaltig ein Stück zum Besseren zu drehen. Unser Portfolio ist in zwei Bereiche unterteilt die eng miteinander verbunden sind: Beratung - „Strategie & Transformation“ - Technologie - „Tech & Digital Services“. Dies bieten wir in den Branchen: Automotive, Financial Services, Energy sowie Public Sektor an.

ROHDE & SCHWARZ

Make ideas real



Bei Rohde & Schwarz arbeiten wir seit mehr als 90 Jahren entlang der gesamten Wertschöpfungskette: wir entwickeln, produzieren und vermarkten Ideen, Systeme und Produkte für eine sichere und vernetzte Welt. Als deutsches Familienunternehmen setzen wir auf Unabhängigkeit, Flexibilität und technologische Relevanz, um unsere Kunden die besten Lösungen zu bieten. Unsere über 14.400 Mitarbeitenden sind dabei der Schlüssel zum Erfolg! Geschäftsbereiche: Test and measurement | Networks and cybersecurity | Broadcast and media technology | Aerospace, Defense, Security

Fast Reports

Reporting must be Fast!

Fast Reports entwickelt seit 1998 Bibliotheken und Werkzeuge für die Erstellung von Berichten und Dokumenten aus Datenbanken. Fast-Reports-Komponenten wurden so Teil von Hunderttausenden von Softwarelösungen auf verschiedenen Programmiersprachen und für verschiedene Plattformen wie .NET Core, RAD Studio, Visual Studio, Lazarus, Mono, und andere. Fast-Reports-Produkte sind in 32 Sprachen lokalisiert und bedienen Kunden auf der ganzen Welt, einschließlich der Fortune-500-Unternehmen.

M&M software

M&M Software ist internationaler Software- und Digitalisierungspartner und begleitet Unternehmen bei der digitalen Transformation ihrer Organisationen, Produkte und Geschäftsmodelle. Wir identifizieren Potenziale, generieren Ideen, leiten Strategien ab und entwickeln maßgeschneiderte Softwarelösungen für die digitale Welt. M&M Software bietet mehr als 35 Jahre Erfahrung und hochmodernes technisches Know-how. Unsere fast 300 hochmotivierten Mitarbeiter an vier Standorten weltweit sind ein wesentlicher Erfolgsfaktor unserer Projekte.

Bundesverband green.software

CODELUTION

envite

DEV ACADEMY

UHR-ZEIT/ PAUSEN	Workshop 1 .NET FRONTEND	Workshop 2 WEB FRONTEND	Workshop 3 SOFTSKILLS	Workshop 4 AI	Workshop 5 ARCHITECTURE
09.00 - 17.30	Christian Giesswein 	Manfred Steyer  Michael Egger-Zikes 	Stefan Mintert 	Christian Liebel  Marco Frodl 	David Tielke 
PAUSEN: 10.30-11.00 15.00-15.30					
Mittag: 12.30-13.30	BLAZOR - ANWENDUNGEN MIT DEM UI FRAMEWORK VON MICROSOFT - EGAL OB WEB, DESKTOP ODER MOBILE	MODERNE ANGULAR ARCHITEKTUREN - NX, MICRO FRONTENDS, DDD UND SIGNALS	AGILE TEAM LEADERSHIP 2025	AI-PRAXIS: INTELLIGENTE BUSINESS-APPS MIT ANGULAR, LANGGRAPH UND RAG	DAS EINMALEINS GUTER SOFTWARE-ARCHITEKTUR

VORMITTAG	Workshop A WEB	Workshop B CONTAINER 
09.00 - 12.30 PAUSE: 10.30-11.00	 Peter Kröner	 Rebai Hamida
	ASYNC/AWAIT 2.0: NATIVE STREAMS UND ASYNCHRONE ITERABLES FÜR JAVASCRIPT UND TYPESCRIPT	ARCHITECTING .NET MICROSERVICES IN A DOCKER ECOSYSTEM
Mittag: 12.30-13.30		
NACHMITTAG	Workshop F CLOUD 	Workshop G ARCHITECTURE
13.30 - 17.30 PAUSE: 15.00-15.30	 Julia Kordick	 Henning Schwentner
	A (SHORT) GUIDE TO PLATFORM ENGINEERING EXCELLENCE WITH AZURE AND GITHUB	DOMAIN-DRIVEN DESIGN

Die Programmdetails sind zu finden unter:
developer-week.de/programm

Workshop 6 CI / CD	Workshop 7 DATABASES	Workshop 8 AI	Workshop 9 .NET BACKEND 	Workshop 10 SOFTWARE QUALITY
Marc Müller 	Dr. Holger Schwichtenberg 	Jörg Neumann 	Loek Duys  Alex Thissen 	Lutz Marquardt 
MASTERING CI/CD PIPELINES WITH AZURE DEVOPS	OR-MAPPING MIT EF CORE UND C#	AI-DRIVEN SOFTWARE DEVELOPMENT MIT GPT UND CO.	CLOUD ASPIRATIONS: BUILDING CLOUD-NATI- VE APPLICATIONS WITH .NET ASPIRE	CLEAN PULL REQUESTS UND METHODISCHE CODE REVIEWS

Workshop C AI	Workshop D GIT, GITHUB 	Workshop E QUALITY 
Gregor Biswanger 	Lukas Durovsky 	Leo Visser 
MICROSOFT SEMANTIC KERNEL - GENERATIVE AI MIT ASP.NET CORE	GITHUB COPILOT: THE SMARTEST WAY TO CODE?	BOOST YOUR AZURE FUNCTIONS WITH API MANAGEMENT

Workshop H AI	Workshop I .NET	Workshop J .NET DOING 
Dr. Damir Dobric 	André Krämer 	Thorsten Kansy 
ARCHITECTING AND DEVELOPMENT OF CUSTOM COPILOTS	DEIN WEG ZUM CROSS-PLATTFORM-APP- ENTWICKLER MIT .NET MAUI UND .NET 9	KONFIGURATION, DEPENDENCY INJECTION & LOGGEN MIT .NET RICHTIG NUTZEN

UHRZEIT	SOFTWARE DESIGN PRINCIPLES	SOFTWARE TESTING	DEVOPS	.NET FUNDAMENTALS	.NET BACKEND
09.00 - 11.00	DWX Pre-Con				
10.00 - 11.00	Konferenz-Einlass, Networking, Besuch der Fachausstellung				
11.00 - 12.00	Opening Keynote				
12.00 - 13.30	Mittagspause und Besuch der Fachausstellung				
13.30 - 14.20	SOLID vs. CUPID - Gegner oder Verbündete? <i>Stephan Roth</i>	Lass mich in Ruhe mit deinen Standards, oder: Es ist nicht alles Testen, was glänzt <i>Georg Haupt</i>	Unlocking the Power of AI: Transforming Software Development <i>Michael Kaufmann</i> 	Ist C# nun langweilig - oder spannender denn je? <i>Christian Giesswein</i>	Telemetrie und Distributed Tracing mit OpenTelemetry für ASP.NET Core Micro-services <i>Marc Müller</i>
	Raumwechsel				
14.30 - 15.30	Timeless Craftsmanship: Universelle Prinzipien für langlebige Software <i>David Sautter</i>	Chiseled to perfection: Einführung in modernes Unit Testing <i>Stefan Dirschnabel</i>	You build it, but you ain't gonna run it... - DevOps-Praktiken für die Arbeit mit IT-Dienstleistern <i>Konstantin Diener</i>	Throw exceptions... out of your codebase <i>Guillaume Faas</i> 	Microsoft Semantic Kernel - Einstieg und Best Practices für Generative AI mit ASP.NET Core <i>Gregor Biswanger</i>
15.30 - 16.00	Kaffeepause und Besuch der Fachausstellung				
16.00 - 16.50	The Root of All Evil: Shared Mutable State and How to Get Rid of It <i>Marcus Crestani</i> 	15 years of insights from a TDD practitioner <i>Dennis Doomen</i> 	A (short) guide to Platform Engineering Excellence with Azure and GitHub <i>Julia Kordick</i> 	From Task.Run to Task.WhenAll: The Good, The Bad, and The Async <i>Steven Giesel</i> 	Minimal API 2.0: Advanced Patterns and Best Practices for Lightweight Web Services <i>Kajetan Duszyński</i> 
	Raumwechsel				
17.00 - 17.50	Over-Engineering verhindern <i>David Tielke</i>	TestArchitect - Why everybody needs (at least) one <i>Marco Achtziger</i> 	GitHub Copilot mit Nachbrenner: Erweiterungen & Power-User Funktionen für smarteres Coden <i>Nico Orschel, Harald Binkle</i> 	The hidden life of method calls <i>Kevin Gosse</i> 	Modern service-to-service communication with gRPC <i>Roland König</i>
	Raumwechsel				
18.00 - 19.00	Können wir Entwickler:innen-Produktivität messen? <i>Eberhard Wolff</i>	Evaluiert von Gen-AI-Applikationen - Ein Bericht aus der Praxis <i>Toni Fetzner</i>	Warum Transformationsoffices KI-Transformationen erfolgreich steuern <i>Michael Schemel</i>	An IOI of .NET internals <i>Nico Vermeir</i>	Performance-Tuning bei Entity Framework Core <i>Dr. Holger Schwichtenberg</i>
19.30 - 22.00	Abendveranstaltung				

Die Programmdetails sind zu finden unter:
developer-week.de/programm

ALLE WEITEREN INFORMATIONEN
UND ANMELDUNG AUF:
DEVELOPER-WEEK.DE

WEB HYBRID	CLOUD DEVELOPMENT	DATA BASES	PERSONAL GROWTH	ROBOTICS	UHRZEIT
DWX Pre-Con					09.00 - 11.00
Konferenz-Einlass, Networking, Besuch der Fachausstellung					10.00 - 11.00
Opening Keynote					11.00 - 12.00
Mittagspause und Besuch der Fachausstellung					12.00 - 13.30
Close the (feature) gap: how the web is rivalling native apps <i>Francesco Leardini</i> 	Building Cloud-based Enterprise Applications that leverage GenAI <i>Dr. Damir Dobric</i>	Stored Procedures vs Views vs AdHoc-Abfragen <i>Thorsten Kansy</i>	The mindful Software Developer <i>Stefan Mintert</i>	Roboter 2.0: Wenn Künstliche Intelligenz auf Stahl trifft <i>Thomas Tomow</i>	13.30 - 14.20
Raumwechsel					
Cross-platform apps using Rust <i>Marcel Koch</i>	Boost your Azure Functions with API Management <i>Leo Visser</i> 	Memory Grant in Azure SQL <i>Torsten Strauß</i> 	Hard skills, soft skills, and human skills <i>Pablo Bouzada</i> 	Offline und sicher: So baut man Voice Interfaces für Roboter <i>Oliver Guhr</i>	14.30 - 15.30
Kaffeepause und Besuch der Fachausstellung					15.30 - 16.00
Built-in AI und WebNN: wie das Web zur GenAI-Plattform wird <i>Christian Liebel</i>	Developing Event-Driven and Cloud-Native Apps in .NET using Azure Functions on Azure Container Apps <i>Jonah Andersson</i> 	Beyond EF Core Migrations <i>Leonhard Fischl</i>	From Senior Engineer to Tech Lead: Close the gap and grow in the role <i>Alfonso Graziano</i> 	Die ethischen Herausforderungen der künstlichen Intelligenz <i>Carsten Fröhlich</i>	16.00 - 16.50
Raumwechsel					
Apps für alle: Accessibility als Mobile Entwickler <i>Giulia Maier</i>	Azure Durable Functions IOI: Serverless Workflow Orchestration <i>Florian Lenz</i>	Vektordatenbanken verstehen und optimal einsetzen: Was hinter dem Hype der semantischen Suche steckt <i>Gregor Biswanger</i>	Cognitive Diversity for Innovation: The Critical Role of Neurodiversity in Development <i>Adina Cristoceca</i> 	Die Zukunft der Robotik im Automotive-Bereich <i>Dhruvin Doshi, Malte Springer</i>	17.00 - 17.50
Raumwechsel					
Nahtlose KI in Blazor MAUI - komplett lokal <i>Jochen Kluger</i>	.NET Aspire in der Cloud <i>Angefragt</i>	SQL Server - Partitioning for Beginners <i>Uwe Ricken</i>	AI Writes Code. What Makes You Special? This Will: The No-BS Guide to Future-Proof Your Career <i>Pierluigi Meloni</i> 	Robotik in der Automatisierungstechnik <i>Angefragt</i>	18.00 - 19.00
Abendveranstaltung					19.30 - 22.00

UHRZEIT	SOFTWARE ARCHITECTURE	SOFTWARE SECURITY	CI / CD	.NET DOING	WEB FRONTEND
09.00 - 10.00	Konferenz-Einlass, Networking, Besuch der Fachausstellung / DWX Pre-Con				
10.00 - 10.50	Opening Keynote				
11.00 - 12.00	10 Fehler, die die Architektur zerstören <i>David Tielke</i>	What is Linux kernel keystore and why you should use it in your next application <i>Ignat Korchagin</i> 	Azure DevOps: Deployments mit Pipelines YAML <i>Neno Loje</i>	What you can learn from an open-source project with 450 million downloads <i>Dennis Doomen</i> 	Decoding the Impact <i>Daniela Kubesch</i> 
12.00 - 13.30	Mittagspause und Besuch der Fachausstellung				
13.30 - 14.20	Langlebige Software-architekturen - ohne Schulden die Jahrzehnte überdauern <i>Carola Lilienthal</i>	Social Engineering: Hacking Humans <i>Pawel Sucholbiak</i> 	Moderne CI/CD-Prozesse mit GitHub Actions <i>Nico Orschel</i>	WinForms trifft WPF: Der Hybrid-Weg zur modernen Desktop-Anwendung <i>Peter Dannenberg</i>	Privacy-first in-browser Generative AI web apps: offline-ready, future-proof, standards-based <i>Maxim Salnikov</i> 
	Raumwechsel				
14.30 - 15.30	Where message queues fit in modern architecture <i>Lovisa Johansson</i> 	Web-Security up-to-date: Die OWASP Top Ten 2025 <i>Christian Wenz</i>	Advanced Pull Request - Custom Checks and Policies <i>Marc Müller</i>	From Hell to Heaven: Porting Doom to MAUI <i>Nico Vermeir</i>	Hello (Virtual) World <i>Keren Kenzi</i> 
15.30 - 16.00	Kaffeepause und Besuch der Fachausstellung				
16.00 - 16.50	Vom Monolith zum Modulith - Eine Fallstudie aus der Halbleiterindustrie <i>Hendrik Lösch</i>	Sichere Softwareentwicklung: Mehr als nur Threat Modeling und Penetration Testing <i>Sven Rieger</i>	Reusability in IaC - The art of module development in Terraform <i>Stefan Rapp</i>	Everything a .NET developer needs to know about configuration and secret management <i>Sander ten Brinke</i> 	From the crypt to the code: Web security explored through horror movies <i>Ramona Schwering</i> 
	Raumwechsel				
17.00 - 18.00	arc42: Was, Wie, Woher, Wohin? <i>Gernot Starke, Ralf D. Müller</i>	Software Security Angefragt	Stay Secure and Updated: Strategien für kontinuierliches Dependency-Management <i>Felix Burkhard</i>	Blazor Static Angefragt	Enterprise Apps mit Blazor Server & EF Core entwickeln <i>Thomas Claudius Huber</i>
	Raumwechsel				
18.10 - 19.00	Night Session				
19.30 - 22.00	Meetups, Specials & more				

Die Programmdetails sind zu finden unter:
developer-week.de/programm

ALLE WEITEREN INFORMATIONEN
UND ANMELDUNG AUF:
DEVELOPER-WEEK.DE

WEB DESIGN	CONTAINER	GENAI FUNDAMENTALS	FROM ME TO WE	IOT / INDUSTRY 4.0	UHRZEIT
Konferenz-Einlass, Networking, Besuch der Fachausstellung / DWX Pre-Con					09.00 - 10.00
Opening Keynote					10.00 - 10.50
Applied Psychology: Psychology-based UI improvements <i>Keren Kenzi</i> 	Warum so langsam? Continuous Profiling mit Grafana Pyroscope <i>Michael Kotten</i>	GenAI Unpacked: Beyond Basic <i>Dr. Damir Dobric</i>	Dealing with difficult people: Wie geht man mit schwierigen Menschen im Team um? <i>Stefan Mintert</i>	Azure IoT Operations: Zukunftssichere IoT- Plattform für moderne Anwendungen <i>Florian Bader</i>	11.00 - 12.00
Mittagspause und Besuch der Fachausstellung					12.00 - 13.30
Gamification: Nutzer- zentrierte Software, die begeistert - ohne Spielerei <i>Roman Rackwitz</i>	Advanced scaling and resiliency patterns with Karpenter <i>Alex König</i>	RAG-tastisch: Der neue Standard in der intelligenten Informationssuche <i>Thomas Tomow</i>	Ein neues Beobachten, Denken und Handeln: Als Team wachsen mit der Systemischen Perspektive <i>Christoph Jung, Vera Hofheinz</i>	Industrie 4.0 und Selbstorganisation: Produktionssysteme von der Natur inspiriert <i>Martin Krockert</i>	13.30 - 14.20
Raumwechsel					
Von Null auf Wow: Wie Bolt.new das Web-Development auf den Kopf stellt <i>Ulrike Stirnweiß, Nina Hauer</i>	Let's talk about layers <i>Rick Rackow</i> 	In geheimer Mission: AI Agents entwickeln <i>Jörg Neumann</i>	Those „business people“ - they just don't get me! How to talk to non-IT leaders <i>Hanna Engel</i> 	Interaktive Digitale Zwillinge: Reale Objekte mit virtuellen Erleb- nissen verbinden <i>Sebastian Springer</i>	14.30 - 15.30
Kaffeepause und Besuch der Fachausstellung					15.30 - 16.00
Accessibility Overlays Through A Scientific Lens <i>Daniela Kubesch</i> 	Devcontainer: Die gesamte Entwicklungsumgebung im Griff <i>Alexander Rampf</i>	Serverless AI mit WebAssembly und Spin <i>Thorsten Hans</i>	Die goldenen Regeln - Wie vermiese ich Neulingen den Einstieg in mein Projekt? <i>Andreas Monschau</i>	If .NET brewed beer... <i>Shaun Lawrence</i> 	16.00 - 16.50
Raumwechsel					
Unlocking intuitive design: How brain science and UX principles boost usability <i>Vicky Pirker</i> 	Azure-Container Angefragt	Coding in Zeiten des AI Act Angefragt	Alles wär gut, wenn dieser sch*** Kollege nicht wäre <i>Tom Wendel</i>	Developer-friendly IoT: Stressfreies lokales Debuggen mit Azure IoT Edge <i>Arthur Grot, Markus Bergt</i>	17.00 - 18.00
Raumwechsel					
Night Session					18.10 - 19.00
Meetups, Specials & more					19.30 - 22.00

UHRZEIT	SOFTWARE QUALITY	DOING AGILE/ AGILE IN PRACTICE	GIT, GITHUB	.NET FRONTEND	WEB BACKEND
09.00 - 10.00	DWX Closing Panel KI ohne K.O. Wie du Im Zeitalter der KI als Softwareentwickler:in bestehst. Michael Schemel, Director IT/Technology & AI Executive, UnternehmerTUM GmbH, München Christian Weyer, CTO, Thinktecture AG, Würzburg <i>(weitere Teilnehmer angefragt)</i> Moderator: Tilman Börner, Developer Advocate and AI Evangelist, Ebner Media Group, München				
10.10 - 11.00	High-quality software development <i>David Tielke</i>	Vom Machen zum Sein: Wie Doing Agile den Weg zu Being Agile ebnet <i>Ina Einemann</i>	git mal anders: Smarte Workflows mit Jujutsu und GitButler <i>Marko Beelmann</i>	.NET 9.0 Desktop Development: Was ist neu in WPF, Windows Forms und WinUI 3 <i>Thomas Claudius Huber</i>	Breaking the Chains: API Versioning Strategies for a Changing World <i>Lukas Durovsky</i> 
11.00 - 11.30	Kaffeepause und Besuch der Fachausstellung				
11.30 - 12.30	Endlich gute Anforderungen <i>Gernot Starke</i>	XP Rediscovered: Alte Schätze für die moderne agile Softwareentwicklung <i>Ulf Mewe</i>	Mastering Git <i>Soumaya Erradi</i> 	Blazor - Der Weg zur eigenen UI Component Library <i>Christian Giesswein</i>	Von Blazor zu WASI: Praktische Einblicke in WebAssembly mit .NET und C# <i>Oliver Sturm</i>
12.30 - 14.00	Mittagspause und Besuch der Fachausstellung				
14.00 - 14.50	Technische Schulden dem Management erklären <i>Carola Lilienthal</i>	How to Ask Better Questions <i>Amber Vandenburg</i> 	GitHub Copilot: Einmal genutzt, will man nicht mehr zurück <i>Neno Loje</i>	Let's turn your website into a hybrid .NET MAUI app <i>Mark Allibone</i> 	Level Up Your Security: OpenID Connect/OAuth Update <i>Roland Guijt</i> 
	Raumwechsel				
15.00 - 16.00	Fachmodellverschmutzung ist schlecht - was kann ich dagegen tun? <i>Henning Schwentner</i>	Brauchen Organisationen ein anderes Betriebssystem, damit Agilität funktioniert? <i>Thomas Schissler, Neno Loje</i>	Deep-Dive into Git <i>Mike Kaufmann</i>	.NET MAUI, Uno Platform und Avalonia im Vergleich <i>André Krämer</i>	Processing One Billion Rows in PHP <i>Florian Engelhardt</i> 
	Raumwechsel				
16.30 - 17.30	Richtig gute Tests schreiben - die Best Practices für bessere (JUnit-) Tests <i>Falk Sippach</i>	Der Wert von Zielen und die Qualität einer Softwareentwicklung <i>Uwe Vigenschow</i>	GitHub Copilot <i>Angefragt</i>	Modern cross-platform applications with .NET and Avalonia UI <i>Roland König</i>	Die Schönheit von Rust ... für Backend-Entwickler! <i>Thorsten Hans</i>
17.40	Das war die DWX 2025				

Die Programmdetails sind zu finden unter:
developer-week.de/programm

ALLE WEITEREN INFORMATIONEN
UND ANMELDUNG AUF:
DEVELOPER-WEEK.DE

WEB LANGUAGES	DATA & ANALYTICS	GENAI TOOLS & SOLUTIONS	SUSTAINABILITY	UHRZEIT
DWX Closing Panel KI ohne K.O. Wie du Im Zeitalter der KI als Softwareentwickler:in bestehst. Michael Schemel, Director IT/Technology & AI Executive, UnternehmerTUM GmbH, München Christian Weyer, CTO, Thinktecture AG, Würzburg <i>(weitere Teilnehmer angefragt)</i> Moderator: Tilman Börner, Developer Advocate and AI Evangelist, Ebner Media Group, München				09.00 - 10.00
ECMAScript Decorators: Deep Dive <i>Peter Kröner</i>	Microsoft Fabric Data Activator: Real-Time Automation Made Simples <i>Andreas Laeubli</i> 	LLMs & Embeddings in Action: Real-World Patterns für Ihre Anwendungen <i>Christian Weyer</i>	Ein Reiseführer zur ressour- censchonenden Software <i>Hendrik Lösch</i>	10.10 - 11.00
Kaffeepause und Besuch der Fachausstellung				11.00 - 11.30
Mein Browser kann bitte was?! Wundertüte JavaScript-APIs <i>Christian Wenz</i>	Data Aware Architectures als Basis für KI & Analytics <i>Matthias Niehoff</i>	GitHub Copilot & GitHub Copilot Chat - Top Anwen- dungsfälle aus der Praxis <i>Marc Rufer</i>	Green Frontend - Energieeffi- zienz von Web-Anwendungen erfassen und optimieren <i>David Kopp, Jan Kirchner</i>	11.30 - 12.30
Mittagspause und Besuch der Fachausstellung				12.30 - 14.00
Cross Site Scripting ist Schnee von gestern, oder? <i>Martina Kraus</i>	APIs für Deine Datenbank: REST, GraphQL und mehr <i>Constantin Klein</i>	„Talk to your data“: Signifikant bessere LLM-RAG-Lösungen durch Real-World Tipps <i>Sebastian Gingter</i>	CO2-Emissionen von Software messen <i>Aydin Mir Mohammadi</i>	14.00 - 14.50
Raumwechsel				
It's all greek to me! - Lokalisie- rung in Web-Apps elegant und einfach <i>Harald Binkle</i>	Best Practises für Datenvisu- alisierung: Übersichtlichkeit, Informationsgehalt und Barrierefreiheit <i>Ina Humpert, Julia Norget</i>	Semantic Flow: Komplexe KI-Prozesse im Griff mit Semantic Kernel <i>Gregor Biswanger</i>	Software sustainability certifications - Blue Angel (Blauer Engel) for software <i>Geerd-Dietger Hoffmann</i> 	15.00 - 16.00
Raumwechsel				
Best of the Worst - Die Hitpa- rade der Top 5 schlimmsten JavaScript-Features <i>Peter Kröner</i>	Azure Integration Services <i>Sebastian Meyer</i>	Real Time AI <i>Angefragt</i>	Shaping the future with Carbon-Aware Process Execution <i>Maximilian Behrendt, Luc Weinbrecht</i> 	16.30 - 17.30
Das war die DWX 2025				17.40

INTERVIEW MIT CHRISTIAN WEYER

„Sieh die KI als Juniorentwickler“

CTO Christian Weyer fühlt sich jung wie schon lange nicht mehr. Woran das liegt und warum er keine Angst um seinen Job hat, erzählt er im dotnetpro-Interview.

Beginnen wir mit einer Binsenweisheit: Die künstliche Intelligenz respektive Large Language Models halten überall Einzug. Die Softwareentwicklung ist aber in besonderem Maße betroffen. Als Vertreter der vielseitigen Gründe seien nur zwei genannt: massive Textorientierung des Codes mit strenger Grammatik und Tonnen an Dokumentation für APIs und Frameworks, die es zu wissen gilt. dotnetpro hat Christian Weyer befragt, welchen konkreten Einfluss die KI auf die Softwareentwicklung im Jahr zwei nach ChatGPT hat.

dotnetpro: Christian, bitte erkläre kurz, was deine Firma Thinktecture [1] macht.

Christian Weyer: Thinktecture gibt es seit 2004, ursprünglich gegründet von Ingo Rammer und mir. Wir machen tieftechnisches Consulting für Softwareentwickler und Softwarearchitekten – immer mit dem Blick auf die neuesten Technologien, um mit den Kunden gemeinsam ihre Projekte voranzubringen. Die Kunden sind meist Independent Software Vendors, die ihre eigenen, meistens Branchenlösungen bauen. Früher hat man die in einem Karton auf einer CD vertrieben, heut-

viele Mitarbeiter brauche, oder ist es etwas Schlechtes? Das müssten wir dann ein bisschen größer aufspannen, was quasi der gesamtgesellschaftliche Impact von KI ist.

Schneiden wir es etwas kleiner. Würdest du davon ausgehen, dass du künftig weniger Mitarbeiter brauchst, um die gleiche Menge an Projekten zu stemmen?

Christian: Ja. Aktuell sind wir zwölf fest angestellte technische Experten. Wir wachsen vielleicht noch um ein paar mehr. Das war aber auch schon vor dem ganzen KI-Hype unser Ziel. Ganz einfach aus dem Grund, weil wir hoch spezialisierte Experten brauchen, die es entweder schon fertig irgendwo gibt – und die gut zu unserem Mindset passen –, oder sie bringen die notwendige Wissbegierigkeit mit, und wir formen sie zum technischen Consultant.

Wir hatten mal überlegt, das skalierbar zu machen. Das hätte aber viele Umbaumaßnahmen bei uns bedeutet. Und die Art und Weise, wie wir mit unseren Kunden arbeiten, hätte sich auch geändert.

Wir haben uns immer spezielle Themen gesucht, die wir uns dann in aller Tiefe erschlossen, angewendet und vermittelt haben. Nach Remoting und WCF in den Anfangsjahren waren wir die SPA- und Angular-Experten. Zudem haben wir sehr viel Security und Identity gemacht und sehr viel Azure und Cloud Native.

Künftig werden beispielsweise Neuerungen von Angular 24 auf Angular 25 aber immer unwichtiger werden. Trotzdem wird es noch Tiefenexperten brauchen, wenn es um robuste, wirklich sichere und performante Software geht. Aber ein Großteil unserer Aktivität wird durch KI-basierte Ansätze unterstützt werden.

Darf man das so interpretieren, dass auch deine eigene Produktivität größer wird, weil du jetzt einfach nicht mehr so viel tippen musst? Weil du schneller irgendeine Funktion aus der KI geliefert bekommst.

Christian: Ich fühle mich informationstechnisch so jung wie schon lange nicht mehr. Nach 26 Jahren im Beruf war nichts mehr da, was mich so richtig gepackt hätte. Ob Rust, Java, .NET, ob Angular, Vue oder React, Azure, AWS oder GCP: alles irgendwie das Gleiche. Und dann kam Cline [2], ehemals ClaudeDev, als Plug-in für Visual Studio Code. Zusammen

„Dass sich die Rolle des Entwicklers und dass sich die Rolle der Softwareentwicklung ändern wird und ändern muss, ist vollkommen klar.“

zutage sind das meistens SaaS-Lösungen. Aber eben auch die Enterprise-Entwickler, die dann Inhouse-Lösungen bauen.

Hast du Angst, dass du künftig, sagen wir mal in zwei bis drei Jahren, weniger Angestellte haben wirst, weil die KI einiges übernimmt?

Christian: Das ist eine philosophisch angehauchte Frage: Ist es etwas Gutes, dass ich in Zukunft vielleicht nicht mehr so



Christian Weyer

ist Mitgründer und CTO der Thinktecture AG, Microsoft MVP für AI, Google GDE für Web AI und steht in direktem Austausch mit den Produktgruppen in Redmond und Mountain View. Er konzentriert sich auf die praxisnahe Umsetzung von Generative AI in modernen Softwarelösungen. Mit über 25 Jahren Erfahrung im Software Engineering unterstützt er ISVs und Unternehmen dabei, LLMs und andere KI-Technologien effizient in ihre Anwendungen zu integrieren. Als leidenschaftlicher Sprecher teilt er sein Wissen weltweit auf Konferenzen und Veranstaltungen – wer ihn schon erlebt hat, kann das bestätigen. Wer ihn noch nicht auf der Bühne gesehen hat, sollte das dringend nachholen – zum Beispiel auf der Developer Week 2025.

mit einem Kunden haben wir ein GenAI-basiertes Projekt mit Cline gebaut. Das Projekt ist in Dogfooding-Manier entstanden. Es war unfassbar. Für das Proof-of-Concept-Projekt (POC) hatten wir drei Wochen veranschlagt, aber mithilfe von Cline war ich nach viereinhalb Tagen fertig. Das Problem war dann vielmehr, die Zeit für die Abstimmung mit dem Kunden zu finden.

Das hat so gut funktioniert, weil ich viel Erfahrung habe, weil ich Softwarearchitektur kann, weil ich Software Engineering kann, weil ich schon alle möglichen Herausforderungen gesehen habe. Unterstützt durch die GenAI-Tools konnte ich mehr machen.

Du warst also produktiver?

Christian: Ja, auf jeden Fall. Produktiver zu werden fühlt sich einfach gut an. Das Perverse ist ja, dass du beinahe nichts mehr tippst – vielleicht noch deine Prompts und deine Requirements, aber du kannst es genauso gut mit Spracheingabe machen.

Aber neben der Produktivitätserhöhung war ein wichtiger Aspekt das Mehr an Kreativität. Ich habe Dinge ausprobiert, die ich vorher nicht mal mit spitzen Fingern angefasst hätte.

Nenne bitte ein Beispiel, was du ausprobiert hast.

Christian: Extensive Frontend-Arbeit. Ich habe zwar jahrelang SPA und Angular gemacht, aber ich war nie der Designer. UI und UX – um Himmels Willen. Das noch gepaart mit CSS und dessen Ablegern, das muss ich nicht haben. Und dann habe ich dem Kunden im POC-Projekt ein Frontend gebaut, das, gewürzt mit seiner Corporate Identity, genau das war, was er haben wollte. Oder Dinge, die kein Must-have sind, sondern die man normalerweise später angeht. Von diesen Non-Goals habe ich auch welche eingebaut. Ganz einfach, weil ich viel kreativer sein konnte. Das hat mich total geflasht.

Man merkt deine Begeisterung ...

Christian: Ja, aber wir sind ja immer noch ganz am Anfang dieser Entwicklung.

Hast du neben Cline noch andere KI-Tools für das Projekt verwendet?

Christian: Normalerweise bleibe ich mittlerweile komplett innerhalb der IDE, weil ich mit dem Ding diskutieren kann, als wäre ich innerhalb des Browsers in ChatGPT oder Claude AI.

Was hältst du von Tools wie Bolt.new [3]?

Christian: Ich habe das Tool noch nicht selbst ausprobiert. Aber was ich bislang gemacht habe, war quasi Architektur und Coding innerhalb der IDE. Das geht mit Cline oder Cursor [4] oder wie sie alle heißen. In diesen Tools hast du einen Modus, um zu planen. Dann verwenden sie ein Reasoning Model dahinter, zum Beispiel o1 oder o3 von OpenAI oder R1 von DeepSeek, die sich auf Reasoning verstehen. Oder du sagst Act, dann ist es

quasi Code-Generierung – mithilfe von GPT-4o oder Claude Sonnet als Coding Assistance.

Worauf du anspielst, sind Prototyping Tools und Prototyping Assistance, mit denen du mehr oder weniger End-to-End-Anwendungen bauen kannst. Dazu gehören v0 [5], Bolt.new oder Lovable [6]. Damit kannst du basierend auf einem Requirements-schwangeren Prompt eine Anwendung bauen, also von Frontend mit UI/UX-Aspekten bis hin zum API-Layer mit Backend-Implementierung. Das ist natürlich superinteressant. Ob du vorne Angular oder React hast – und das mag jetzt für den ein oder anderen wirklich markerschütternd sein –, ist im Prinzip egal.

Über die Jahre hinweg werden diese Basistechnologien weiter Commodity werden, und wir müssen uns mit anderen Herausforderungen auseinandersetzen, zum Beispiel mit den Tools, die du erwähnt hast.

Welchen Einfluss werden diese Tools haben?

Christian: Am Ende des Tages werden wir einen großen Schritt brauchen, über den wir wahrscheinlich schon seit 20 Jahren reden, nämlich die Abstraktionsebene zu erhöhen. Wir sprechen ja immer noch über C#-Typen und über Java-APIs, ►

über Frameworks und darüber, ob ich jetzt Angular Signals verwende oder irgendwelche anderen Technologien. Damit wird die Softwarewelt aber nicht produktiver und effizienter. Das heißt, wir brauchen irgendwie eine andere Abstraktionsebene, sprich all das, was seit circa 20 Jahren rund um Low Code und No Code kreucht und fleucht. Wenn die zwei Themen, nämlich Generative-AI-assisted Coding und Low Code oder vielleicht sogar No Code, wirklich einmal zusammenkommen, dann wird es explodieren. Ich vermute, dass irgendeiner von den Großen bereits daran arbeitet. Das hebt die Abstraktionsebene, die wir eigentlich brauchen, damit wir uns nicht mehr mit Details befassen müssen.

Angesichts dieser Prognose: Hast du dann nicht Angst um deinen Job oder um die Jobs der vielen anderen Entwickler?

Christian: Dass sich die Rolle des Entwicklers und dass sich die Rolle der Softwareentwicklung ändern wird und ändern muss, ist vollkommen klar. Das heißt aber nicht, dass mein Job weg ist. Ich glaube nicht, dass jeder Hinz und Kunz oder jeder in der Requirements-Abteilung seine eigene Anwendung baut. Dazu gehört ein bisschen mehr, wie zum Beispiel Unternehmensrahmenbedingungen, funktionale und nicht-funktionale Aspekte.

Das heißt also, dass man vieles AI-assistiert wird erstellen können, aber nicht alles. Das kann jedoch in zwei Jahren ganz anders aussehen. Wenn wir in zwei Jahren wieder darüber sprechen, kann es sein, dass ich sage: „Tilman, ich glaube, ich muss meine Rente vorziehen.“ Vielleicht gibt es dann nichts mehr zu tun. *(grinst)*

Hört sich schlüssig an. Allerdings scheint sich der Punkt, an dem Entwickler überflüssig werden, nach hinten geschoben zu haben. Nach dem Benchmark SWE-Lancer [7], der 1400 echte Freelance-Aufgaben aus Upwork [8] verwendet, sind die aktuellen LLMs nicht in der Lage, vernünftige Lösungen zu finden [9].

Christian: Das ist, glaube ich, nur ein temporärer Effekt. Die Modelle werden besser. Allein wenn man sich ansieht, was mit DeepSeek passiert ist, dass man keine Multimillionen-Data-Center braucht, um ein LLM zu bauen.

Die Basismodelle werden besser werden, die Patterns oben drüber werden noch besser. Die Erfahrungen mit der Art und Weise, wie wir den Input gestalten, wird besser, und dann ein Schritt hin zu Multiagenten, die quasi eine Teamstruktur simulieren und abbilden können.

Welchen Einfluss hat die KI auf die technischen Schulden?

Christian: Ich weiß nicht, ob es die KI ist, die den Effekt mit sich bringt, oder das Heben der Abstraktion. Die technischen Schulden, die ich in den letzten 25 Jahren gesehen habe, waren Framework-Themen beziehungsweise Architekturfehlentscheidungen. Die passieren immer, weil es nicht die eine Architektur gibt. Die Tools werden wahrscheinlich erst einmal mit ein paar Templates starten, die dann mehr oder weniger gut passen. Aber die tatsächlichen technischen Schulden kommen oftmals durch Frameworks rein, also durch Lower-Level-Frameworks, weil man sie nicht verstanden hat, weil man sie

nicht richtig einsetzt oder weil man sie nicht upgraden kann oder upgraden darf. Irgendwie hängt man dann hinterher, weil es nur um Features, Features, Features geht, und unten kommt man nicht mehr nach, die Schulden zu tilgen.

Ich kann mir vorstellen, dass es bestimmte KI-basierte Methodiken gibt, die technische Schulden aufdecken und Migrationspfade aus diesem Schlamassel heraus aufzeigen.

Aber ich glaube auch, dass ein besseres Vorgehen wäre, die Abstraktion zu erhöhen, auf der wir Software denken und Software entwickeln. Es muss sich wirklich keiner mehr die Frage stellen, ob er jetzt Angular Signals oder Observables oder was auch immer verwendet. Für uns Techies ist das zwar cool, aber das bringt die Software nicht weiter und hilft nicht denen, die die Software brauchen.

Was sagst du zu Tests beziehungsweise Unit-Tests und KI?

Christian: Das finde ich ein spannendes Thema. Ich bin jetzt nicht so der Verfechter von Unit-Tests. Für mich sind tatsächlich die End-to-End-Tests viel wichtiger. Am Ende des Tages muss es End-to-End funktionieren, und so viel Business-Logik gibt es oftmals nicht, sondern es sind viel mehr CRUD-Operationen. Dafür sind die Tools extrem gut. Hier kann ich meine Rahmenbedingungen formulieren: Das ist der Input und

„In jedem Projekt, das ich bislang gesehen habe, war die Dokumentation sofort veraltet, sobald der Entwickler auf ‚Save‘ gedrückt hat.“

das muss der Output sein, und dann erzeugt das Tool die Tests. Das ist natürlich schon extrem sexy.

Viele haben sich immer geschaut, Tests selbst zu schreiben und sie dann – und jetzt kommt der wichtige Teil – auch nachzuziehen und upzudaten. Denn irgendwann wurden sie rot, und dann hat man das Feature abgeschaltet. Aber die KI-basierten Tools können die Tests automatisch nachziehen oder darauf hinweisen: „Du hast jetzt die Architektur geändert, und die Tests sind rot. Hier sind meine Vorschläge, wie du das fixen kannst.“

Brownfield und KI: Ist KI Segen oder Fluch?

Christian: *(lacht)* Wir haben Legacy-Migrationsprojekte bei drei Kunden: Einmal eine uralte datenbankzentrierte Technologie und Anwendung, einmal von einem Framework A zu Framework B und einmal ein Projekt, bei dem das Backend



in Java geschrieben ist, aber nie ein Java-Backend-Entwickler verfügbar ist. Bei allen drei Projekten haben wir KI-Tools verwendet – entweder innerhalb der Entwicklungsumgebung oder indem wir ein kleines Kommandozeilen-Tool geschrieben haben, das das OpenAI-SDK nutzt, um Migrationsaspekte beziehungsweise Integrationsaspekte komplett zu automatisieren. Wir haben damit niemanden gebraucht, der uns die Architektur erklärt. Das Tool hat uns das alles erklärt, was übrigens ein Super-Power-Feature der KI ist. Ob das jetzt nur APIs sind oder ein Datenbankschema ist oder eben eine komplette Code-Basis – das ist Magie.

Bei „erklären lassen“ fällt einem sofort die technische Dokumentation ein.

Christian: Ein wichtiger Punkt. In jedem Projekt, das ich bislang gesehen habe, war die Dokumentation sofort veraltet, sobald der Entwickler auf „Save“ gedrückt hat. Und keiner hat sie mehr nachgezogen. Das kann man jetzt komplett neu denken. Man könnte einen eigenen AI-Agenten bauen, der die Dokumentation automatisch nachzieht und mir dann nur noch Pushes gibt, um Feedback einzuholen. Oder noch radikaler: Für was brauche ich Entwicklerdokumentation? Ich frage einfach



das System, und es gibt mir immer den aktuellen Blick darauf. Das ist schon richtig toll. Und das funktioniert mit einem komplexen, Microservice-basierten Ansatz End-to-End mit vielen fachlichen, nichtfachlichen, synchronen, asynchronen Aspekten, in puncto Security, verschiedene Frontend-Technologien und, und, und. Das ist der Wahnsinn.

Gibt es Nachfrage von euren Kunden, die KI in ihre Anwendungen einzubauen?

Christian: Ja. Wir sind tief in generativen KI-Technologien mit Embeddings, Small und Large Language Models, on Premises, in der Cloud und so weiter, um den Kunden zu zeigen, was jenseits von Hype und Bullshit Bingo tatsächlich funktioniert und was nicht funktioniert. Kunden, die neu zu uns kommen, wollen immer etwas mit GenAI haben und nichts mehr mit Standardtechnologien, Architektur oder Frameworks.

Da trübt dein WCF-Auge, oder?

Christian: Na, das ist aber schon lange her. Da waren alle Haare noch schwarz. Aber wir müssen in die Zukunft schauen, neue Ansätze verwenden und neue Technologien, um zu evaluieren, was sinnvoll für das jeweilige Projekt oder Produkt ist. Wie baut man es dann ein? Denn streng genommen sind wir immer im Brownfield unterwegs. Greenfield existiert quasi nur für Demos, Konferenzen und Workshops. Aber im echten Leben ist es nun mal Brownfield. Selbst wenn wir nur auf der existierenden Datenbank aufsetzen, ist es ja schon etwas Existierendes.

Die Kunden wollen dann die Unwissenheit und die Unsicherheit wegbekommen, wollen also wissen, was geht, was nicht geht und wie es mit diesen Halluzinationen ist.

Ziel ist dann herauszufinden, wie mögliche Use Cases aussehen können, also das, was über einen Chatbot über der Dokumentation hinausgeht. Aber man kann natürlich viel mehr machen, wenn man an das Potenzial von Language Models denkt. Menschliche Sprache quasi zum First Class Citizen zu machen oder über agentenbasierte Denkweisen eine Teilautomatisierung von Arbeitsabläufen hinzubekommen.

Momentan liegen Large Language Models sehr im Trend. Sind wir dadurch blind für andere KI-Entwicklungen?

Christian: Ja, Punkt. Na klar, das Spotlight der Scheinwerfer ist komplett auf LLMs, also auf diesen Transformer-basierten Architekturen. Wenn du aber schaust, was die Experten so veröffentlichen, wie zum Beispiel Yann LeCun [10], Chief AI Scientist von Meta, der sagt, es ist zwar alles geil, was hier gerade passiert, aber Transformer sind Kindergarten. Die werden uns nicht wirklich weiterbringen. Am Ende des Tages sind die auch nicht AI, sondern super clevere, statistische, mathematische Modelle, die im Trainingsprozess mit unglaublich vielen Daten gespeist wurden.

Die forschen natürlich an anderen Ideen, die darüber hinausgehen, und das fliegt dadurch aktuell komplett unterm Radar. Das ist aber auch ganz okay, weil wir uns über die Large Language Models und über die Embedding-Modelle ganz gut dieser Problematik und Thematik nähern können, ohne dass es gleich komplett abgedreht wird.

Wir können uns die ersten Jahre ein bisschen warm machen mit künstlicher Intelligenz. Menschliche Sprache rein, menschliche Sprache raus oder strukturierte Informationen rein, strukturierte Informationen raus. Von daher finde ich das gar nicht so schlecht, dass das Spotlight darauf liegt – aber es ist auf jeden Fall nicht das Ende der Fahnenstange.

Welche Tipps würdest du Entwicklern geben, die durch die KI vielleicht etwas verunsichert sind?

Christian: Ich würde das empfehlen, was ich auch meinem Sohn gesagt habe. Der ist jetzt achtzehneinhalb und hat mit dem Studium für Angewandte KI an einer Online-Uni angefangen. Da ist das Curriculum noch relativ klassisch, aber natürlich passiert sehr viel über Teams und über Online-Tools und so weiter.

Ich habe ihm geraten: Versuch mal, sowohl die klassischen Challenges so zu machen, wie sie es dir sagen, und dann aber auch GenAI-assisted, also innerhalb der IDE, um ein Gespür dafür zu bekommen, was sinnvoll und was nicht sinnvoll ►

ist. Wo hast du wirklich mehr Power und wo weniger? Wo bist du wirklich produktiv, wo weniger? Wo bist du kreativer? Das heißt, mein Tipp ist tatsächlich: Schaut euch diese integrierten KI-Entwicklungstools an, und ja – ihr könnt euch langsam von dem Dinosaurier Visual Studio verabschieden. Denn da wird wahrscheinlich absehbar nichts Innovatives passieren. Der ganze Mojo passiert eher in Visual Studio Code.

Entweder gibt es Plug-ins innerhalb des Editors, oder sie forken Visual Studio Code, um dann ihre eigene IDE zu machen, wie zum Beispiel bei Cursor.

Das heißt, guckt euch so ein Tool wie Cursor oder Cline an und baut etwas damit. Entweder ein privates Projekt oder einen Proof of Concept innerhalb der Firma. Behandelt die KI aber nicht als allwissende, mächtige Instanz, sondern als Junior Developer, dem du alles genau erklären musst und dessen Output du immer überprüfen musst. Mit dem du aber Pair Programming und Pair Architecting machen kannst. Der vor allem nie schlecht gelaunt, immer verfügbar, aber leider dement ist. Denn wenn du eine neue Sitzung beginnst, ist das Kontextfenster weg. Dann gibt es natürlich entsprechende Verständnisprobleme.



Aber du kannst immer mit diesem Junior arbeiten, und dieser Junior hat zusätzlich Zugriff auf das ganze Wissen, das du ihm zur Verfügung stellst. Das ist ein ganz wichtiger Punkt.

Ich finde diese Large-Language-Model-basierten Ansätze vor allem deswegen so sensationell, weil sie Konzepte und Sprache verstehen, und nicht notwendigerweise, weil sie Wissen haben. Denn dieses Wissen ist vielleicht überholt. Zum Beispiel musst du an einem Projekt mit der neuesten Angular-Version arbeiten, aber der Knowledge Cut Off des Large Language Models war vielleicht vor eineinhalb Jahren. Dann erzählt dir das Modell irgendeinen Blödsinn. Du sagst beispielsweise: „Gib mir mal etwas mit Angular Signals“, und dann halluziniert dir das Modell das Blaue vom Himmel herunter. Deshalb ist es wichtig, dass du Tools verwendest, in denen du der KI Leitplanken geben kannst. Sprich, du kannst in Tools wie Cursor die aktuellste Angular-Dokumentation oder deine Coding Guidelines dazupacken. Das hilft, den Kontextbezug zu formalisieren und zu konkretisieren und

ebendiesen Hang zu Halluzinationen fast zu eliminieren. Nimm also ein Tool, aber sieh es als Junior, der dir hilft, schneller, besser, kreativer und produktiver zu werden.

Der Blick in die Glaskugel: Wie wird dein Beruf, wie wird dein Job, dein Alltag in zwei Jahren aussehen?

Christian: Ich muss dich enttäuschen. Das kann ich echt nicht sagen. Vor zweieinviertel Jahren wusste ich nicht, dass ich heute hier sitze und genau das erzähle, was ich jetzt erzähle. Wir leben in einer Zeit, die man so noch nicht erlebt hat. Es ist eine Art industrielle Revolution, die gerade passiert.

Von daher wage ich keine Projektion nach vorne, und vor allem will ich mich lieber überraschen lassen. Wöchentlich kommt so viel Interessantes, Spannendes, Sensationelles raus, dass man sich diese Art der Freude der Weiterentwicklung auch nicht nehmen lassen sollte.

Ein wichtiger Punkt ist vielleicht noch: Ich arbeite nicht an den Basistechnologien und an den Forschungsergebnissen ganz unten, sondern wir setzen auf dem API-Layer auf. Daher lese ich natürlich immer wieder mal die Weekly Top Papers aus der Forschung oder lasse sie mir von der KI zusam-

„Schaut euch diese integrierten KI-Entwicklungstools an, und ja – ihr könnt euch langsam von dem Dinosaurier Visual Studio verabschieden.“

menfassen. Und ich schaue natürlich auch links und rechts über den Tellerrand. Ich kann dir nicht sagen, was passieren wird, aber was wir machen können: Wir verabreden uns für den 25. Februar 2027 zu einem Folgeinterview. ■

Interview: Tilman Börner. Fotos: Aus einem Video-Call.

[1] Thinktecture, www.thinktecture.com

[2] Cline, <https://cline.bot>

[3] Bolt.new, <https://bolt.new>

[4] Cursor, www.cursor.com

[5] v0, <https://v0.dev>

[6] Lovable, <https://lovable.dev>

[7] SWE-Lancer, www.dotnetpro.de/SL2504-05Interview1

[8] Upwork, www.upwork.com

[9] OpenAI Researchers Find That Even the Best AI Is Unable To Solve the Majority of Coding Problems, www.dotnetpro.de/SL2504-05Interview2

[10] Yann LeCun, <http://yann.lecun.com>

DEV ACADEMY

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



Wie integriere ich Künstliche Intelligenz in meinen Coding-Alltag

KÜNSTLICHE INTELLIGENZ (AI)

ChatGPT als Unterstützung für Entwickler (2 Tage)

KI-System-Design und Code-Generation mit Generativen
Großen Sprachmodellen (ChatGPT & Co.) (2 Tage)

Einführung in Neuronale Netze und Deep Learning (2 Tage)

Mastering GitHub Copilot - Prompt Engineering & Einblick
in zukünftige Entwicklungen (1 Tage)

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

Einstieg in Python (3 Tage)

KI-Grundlagen für Product Owner (1 Tage)

**REMOTE TRAININGS
MÖGLICH!**



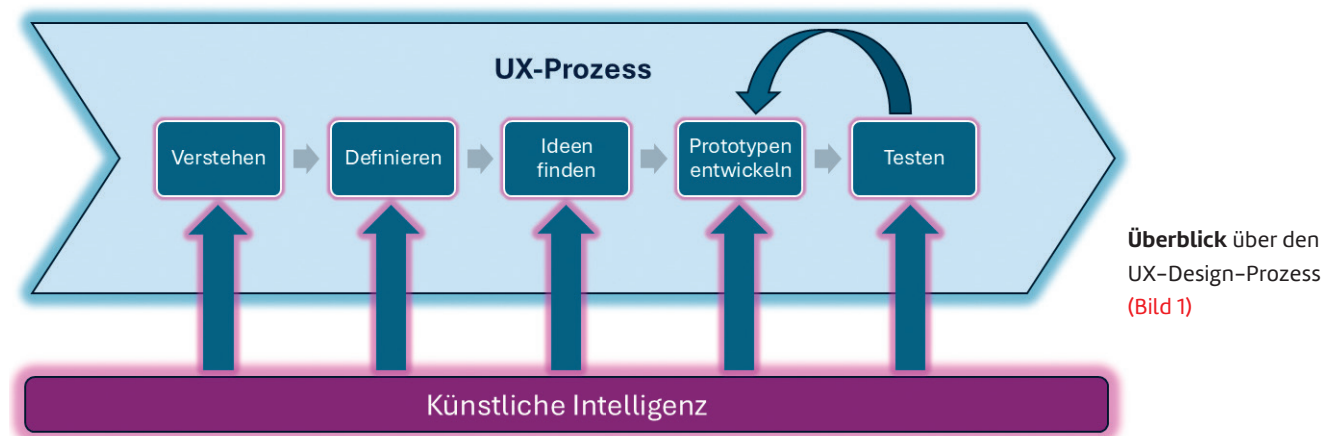
Weitere Informationen auf developer-academy.de
Preise auf Anfrage!

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

KI ALS PERSÖNLICHER ASSISTENT BEI DER UX-GESTALTUNG

Magische Hilfe für das UI

Wie KI den UX-Design-Prozess revolutionieren kann und warum sie für Softwareentwickler:innen, die auch in Designaufgaben involviert sind, ein unverzichtbarer Helfer wird.



Neben ihrer Hauptaufgabe – dem Programmieren – sind es in vielen Unternehmen die Softwareentwickler:innen, die sich auch um die Gestaltung von Benutzeroberflächen (UI) kümmern müssen. Hier bietet künstliche Intelligenz (KI) eine umfassende Hilfestellung. Anhand von spezialisierten Tools und Algorithmen können Entwickler:innen Defizite im Gestaltungsprozess kompensieren und gleichzeitig ihre Effizienz steigern.

Der Artikel gibt einen Überblick über die Möglichkeiten, die KI im Bereich User Research, UX-Konzeption, -Gestaltung und -Testing bietet. Dabei wird nicht nur aufgezeigt, wie KI dabei helfen kann, Designentscheidungen datenbasiert zu treffen, sondern auch, wie sich mit ihr der kreative Prozess unterstützen lässt. Von automatisierten Wireframing-Tools bis hin zu generativen Designansätzen eröffnet KI völlig neue Potenziale, die es Entwickler:innen ermöglichen, benutzerfreundliche, ansprechende User Interfaces schneller und präziser zu gestalten.

Zudem werden Herausforderungen beleuchtet, die sich aus der Integration von KI in bestehende Design- und Entwicklungsprozesse ergeben. Welche Kompetenzen müssen Entwickler:innen aufbauen, um KI effektiv einzusetzen? Dieser Artikel zeigt konkrete Tools und praxisnahe Anwendungsfälle auf, um Softwareentwickler:innen zu inspirieren und aufzuzeigen, wie KI sie in ihrer Doppelrolle als Programmierer:in und Designer:in entlasten kann.

KI als Kreativpartner

Die Gestaltung von Benutzererlebnissen verlangt nach Kreativität, Empathie und einer genauen Kenntnis der Nutzer-

bedürfnisse. Doch kreative Prozesse können mühsam und zeitintensiv sein. In solchen Situationen erweist sich KI als wertvolle Unterstützung.

KI-gestützte Tools haben sich von rein technischen Werkzeugen zu echten kreativen Begleitern entwickelt. Sie sind in der Lage, Designprobleme aus einem unvoreingenommenen Blickwinkel zu betrachten, unkonventionelle Ansätze vorzuschlagen und auf Basis großer Datenmengen neue Möglichkeiten aufzuzeigen. Dabei liegt der Fokus nicht darauf, die menschliche Kreativität zu ersetzen, sondern sie gezielt zu ergänzen. Einige der spannendsten Einsatzbereiche von KI in der UX-Gestaltung sind:

- **Generative Designs:** KI kann aus einer einfachen Idee oder einem groben Konzept mehrere Designvorschläge generieren. Ob es sich um Layouts, Farbschemata oder Typografien handelt – die Tools liefern Inspiration in Sekundenschnelle.
- **Simulieren und Testen von User Journeys:** Durch die Analyse von Nutzerverhalten ist KI in der Lage, Vorschläge zu machen, wie Interaktionen effizienter und benutzerfreundlicher gestaltet werden können.
- **Automatische Content-Generierung:** KI hilft dabei, Platzhaltertexte oder -bilder durch relevante Inhalte zu ersetzen, die zum Designkontext passen.

Ein wesentlicher Vorteil der KI liegt in ihrer Fähigkeit, Muster und Trends aus riesigen Datenmengen zu erkennen und daraus Designansätze abzuleiten. Während Menschen oft durch ihre eigenen Erfahrungen und Vorlieben begrenzt sind, bringt KI eine frische Perspektive ein. Sie analysiert erfolgreiche Designmuster aus verschiedenen Branchen, iden-

tifiziert Gemeinsamkeiten und passt diese an die spezifischen Anforderungen eines Projekts an.

Mit KI können Designer:innen schneller von der Idee zur Umsetzung gelangen. Routineaufgaben wie das Erstellen von Wireframes oder das Layouten von Elementen werden automatisiert, sodass mehr Zeit für die Feinarbeit bleibt. Dabei ist KI vor allem ein Katalysator: Sie gibt Impulse und liefert Alternativen, die dann von den menschlichen Designer:innen weiterentwickelt werden.

Nachfolgend werden wir die praktischen Einsatzmöglichkeiten der KI im Rahmen des UX-Designs erläutern.

KI in verschiedenen Phasen des UX-Designs

Der UX-Prozess besteht aus fünf aufeinanderfolgenden Phasen (Bild 1). Die erste Phase, *Verstehen*, legt den Grundstein, indem die Bedürfnisse, Erwartungen und Probleme der Nutzer erfasst werden. In dem Zusammenhang spricht man von UX-Research. In der Phase *Definieren* werden die gesammelten Erkenntnisse strukturiert, um klare Ziele und Anforderungen zu formulieren. Darauf folgt die Phase *Ideen finden*, in der kreative Lösungen für die definierten Probleme entwickelt werden. In der vierten Phase, *Prototypen*, werden die besten Ideen in konkrete, greifbare Konzepte umgesetzt. Dies kann durch die Erstellung von Wireframes oder interaktiven Prototypen geschehen, die das spätere Produkt simulieren. In dieser Phase ist Iteration besonders wichtig, da Prototypen basierend auf Feedback fortlaufend verbessert werden. Die abschließende Phase, *Testen*, dient dazu, die Prototypen zu überprüfen. Durch Usability-Tests wird beobachtet, wie Nutzer mit dem Produkt interagieren, und Feedback wird gesammelt, um Schwachstellen zu identifizieren. Die gewonnenen Erkenntnisse fließen direkt in die Optimierung des Designs ein, sodass das Produkt kontinuierlich verbessert wird. Daher kommen wir zu folgendem Schluss: KI kann in nahezu jeder Phase des UX-Design-Prozesses eingesetzt werden.

KI im Rahmen von User Research

In der Anfangsphase beziehungsweise bei der Analyse geht es darum, die Bedürfnisse, Erwartungen und Verhaltensweisen der Benutzer zu verstehen.

Bereits in der Vorbereitung kann KI im Rahmen von User Research zur Datenanalyse eingesetzt werden. So kann man große Mengen an Nutzerdaten, beispielsweise Clickstream-Daten, Social Media oder Umfragen, analysieren, um Muster und Trends zu erkennen und so wertvolle Erkenntnisse über das Verhalten der Nutzer zu gewinnen.

Es gibt eine Vielzahl von KI-Tools, die in diesem Bereich eingesetzt werden können:

- **Hotjar [1]:** Ein Tool, das Heatmaps, Session Recording und Feedback-Umfragen bietet. Mit KI-unterstützter Analyse kann es Muster im Nutzerverhalten identifizieren und Empfehlungen zur Verbesserung der Benutzeroberfläche geben. Es kann außerdem zur Erkennung von Klickmustern, Scroll-Verhalten und Formularabschlüssen genutzt werden. Die KI hilft dabei, die „schwächeren“ Stellen einer Anwendung zu erkennen, an denen Nutzer oft Probleme haben oder abspringen.

- **Lyssna (UsabilityHub) [2]:** Die Plattform bietet UX-Designern ein schnelles und effizientes Feedback zu Designs, Prototypen und Benutzeroberflächen. Zu den wichtigsten Angeboten gehört die Analyse der Abfolge, in der Nutzer auf bestimmte Elemente klicken, um die Navigation und Usability zu optimieren. Auch der Vergleich von Design-Alternativen, um die bevorzugte Version zu ermitteln, sowie das Erstellen von gezielten Umfragen, um Nutzerfeedback zu spezifischen Themen oder Designs zu sammeln, sind wichtige Bestandteile davon. Das Tool ist besonders für schnelle Iterationen und datengestützte Designentscheidungen nützlich. Es ermöglicht UX-Designer:innen, potenzielle Probleme frühzeitig zu erkennen und benutzerfreundlichere Lösungen zu entwickeln.


Persona: Lisa M., 23 – BWL-Studentin

Persönliche Daten

- Name: Lisa M.
- Alter: 23 Jahre
- Studiengang: BWL (5. Semester, Universität)
- Lernziel: Klausurvorbereitung & Wissensfestigung
- Lernstil: Strukturiert, visuell, praxisnah

Hintergrund & Bedürfnisse

Lisa studiert BWL und arbeitet nebenbei als Werkstudentin. Sie möchte sich effizient auf Klausuren vorbereiten und sucht eine App, die ihr hilft, Wissen zu vertiefen, ohne viel Zeit zu verlieren. Ihr Fokus liegt auf interaktiven Lernmethoden und flexibler Nutzung.

Herausforderungen

- ✓ Effiziente Prüfungsvorbereitung
- ✓ Interaktive Lernmethoden (Lernkarten, Quizzes)
- ✓ Praxisnahe Übungen für bessere Anwendung
- ✓ Zeitmanagement durch mobile Nutzung

Nutzung der BWL Master-App

- Theorie Lernen – Kompakte, strukturierte Inhalte.
- Beispiele Rechnen – Übung mit praxisnahen Aufgaben.
- Lernkarten Erstellen – Schnelles Wiederholen von Konzepten.
- Abfragen & Klausuren Simulieren – Vorbereitung unter realistischen Bedingungen.

Zitate von Lisa

„Ich brauche eine App, die mir hilft, flexibel und effizient zu lernen.“

Persona mit ChatGPT erstellen (Bild 2)

- **FullStory [3]:** Dieses Werkzeug bietet eine detaillierte Analyse des Nutzerverhaltens. FullStory nutzt KI, um automatisch Muster im Nutzerverhalten zu erkennen, wie beispielsweise häufig wiederkehrende Probleme oder Interaktionen, die zu Abbrüchen führen. FullStory zeichnet jede Interaktion eines Nutzers mit einer Website oder App auf. UX-Designer können die Sitzungen ansehen, um zu verstehen, wie Nutzer navigieren, wo sie hängen bleiben oder frustriert werden.

Mithilfe von KI können Nutzerkommentare, Bewertungen und Forenbeiträge analysiert werden, um die Stimmung und die Bedürfnisse der Zielgruppe zu verstehen. KI-gesteuerte Chatbots oder virtuelle Assistenten lassen sich nutzen, um Umfragen unter Benutzern anzustellen und so wertvolle qualitative Daten zu sammeln. Zudem kann KI bei der Definition unterschiedlicher Benutzersegmente ebenso hilfreich sein ►

wie bei der Erstellung personalisierter Erfahrungen für verschiedene Gruppen.

KI in der Definitionsphase

Die Definitionsphase im UX-Design strukturiert die Erkenntnisse der Research-Phase und setzt klare Designziele. Dazu zählen die Zielgruppenanalyse mit Personas und User Needs, um Nutzerbedürfnisse zu verstehen. Mithilfe von User Journeys und Empathy Maps werden Interaktionen und Erwartungen visualisiert. Danach werden Problem Statements formuliert und Design-Ziele definiert. Zudem entsteht eine Value Proposition, die den Mehrwert des Produkts beschreibt, während erste Ideation-Ansätze mögliche Lösungen skizzieren.

Personas im UX-Design helfen, die Zielgruppe besser zu verstehen und Produkte gezielt auf deren Bedürfnisse auszurichten. Die Erstellung von Personas ist aufwendig, weil sie auf gründlicher Datenanalyse, Research und strategischer Ableitung basiert. Ein realitätsnahes Nutzerprofil erfordert mehr als nur Annahmen – es muss auf echten Bedürfnissen, Verhaltensweisen und Herausforderungen der Zielgruppe beruhen. Wie wäre es, wenn wir unserem KI-Assistenten diese Aufgabe anvertrauen. Welche KI-Tools sind in der Lage, uns dabei Hilfe zu leisten?

Bereits die kostenlose ChatGPT-Version ist in der Lage, den ersten Entwurf einer Persona zu erstellen (Bild 2). Die erstellte Persona enthält alle notwendigen Informationen, ist gut strukturiert und liefert wichtige Denkanstöße. Die Persona liegt jedoch in einer Textform vor und muss erst noch in die Form einer Karteikarte überführt werden.

Spezielle KI-Tools bieten noch mehr Möglichkeiten. So ist es mit dem Tool LeanScope AI [4] möglich, einbaufertige Personas ad hoc zu erstellen (Bild 3). Die erstellte Karteikarte enthält alle wichtigen Punkte und ist mit einem Bild versehen, was die Persona noch realistischer wirken lässt. Das Tool XPressia [5] bietet ebenfalls die Möglichkeit, Personas mit KI zu erstellen. Dies ist jedoch nur auf Englisch möglich, und Fotos sind erst in der Pro-Version verfügbar (Bild 4).

Nachdem die Nutzerforschung durchgeführt und Personas erstellt wurden, besteht der nächste Schritt darin, die gewonnenen Erkenntnisse in eine klare Problemstellung und in User Stories für das Design zu übersetzen.

Eine Problemstellung (Problem Statement) ist eine prägnante Beschreibung der Aufgabe oder des Problems, das ein UX-Designer lösen soll. Eine von ChatGPT gelieferte Variante des Problem Statements klingt wie folgt: „Lisa ist eine BWL-Studentin, die ein effizientes und flexibles Lernwerkzeug braucht, weil sie ihr Wissen vertiefen und sich auf Prüfungen vorbereiten möchte, während sie ihr Studium und einen Teilzeitjob gleichzeitig managt.“ Diese Problem Statement basiert auf folgender Vorlage, die wir im Prompt eingegeben haben: *[Name des Nutzers] ist ein/e [Nutzermerkmal/e], der/die [Nutzerbedürfnis] benötigt, weil [Erkenntnis].*

User Stories beschreiben die gewünschte Funktionalität eines Systems aus Sicht des Anwenders. An dieser Stelle ist unser Assistent ChatGPT schnell dabei (Bild 5). Es gelingt ihm, zahlreiche User Stories basierend auf verschiedenen Nutzerbedürfnissen zu erstellen. Aus den User Stories werden anschließend die Funktionen der Anwendung abgeleitet.

The screenshot displays the LeanScope AI interface for creating a persona. The main section shows a generated persona named **Beatrix Müller**, identified as a **BWL-Studierende** (BWL student). A placeholder image shows a woman studying. Below the image, a list of interests is provided: **BWL Master**, **Authentic, Professional, Style**, **Strategies for learning material**, **Creating realistic study plans**, **Creating your own digital flashcards**, **Creating to-do lists**, and **Prepare Quizzes**.

The interface is divided into several sections:

- MOTIVATIONS:**
 - Wissen gezielt aufbauen
 - Optimal auf Prüfungen vorbereiten
 - Effizient auf Klausuren vorbereiten
 - Wissen vertiefen
 - Effiziente Prüfungsvorbereitung durch strukturierte Lerninhalte.
- FRUSTRATIONS:**
 - Schwierigkeiten beim Verständnis komplexer betriebswirtschaftlicher Konzepte.
 - Mangelnde Motivation, regelmäßig zu lernen und die App zu nutzen.
 - Frustration über unzureichende Übungsaufgaben oder Beispiele für bestimmte Themen.
 - Technische Probleme oder Bugs in der App, die das Lernen stören.
 - Unzureichende Vorbereitung auf spezifische Prüfungsformate oder -inhalte.
- QUOTE:**
 - "Es ist frustrierend, dass ich trotz aller Ressourcen oft nicht weiß, wo ich anfangen soll."
- SOCIODEMOGRAPHIC INFO:**
 - Gender: Female
 - Status: ledig
 - Age: 24
 - Children: 2
 - Education: BWL Masterabschluss, 24 Jahre
 - Profession: BWL-Studierende
 - Work Environment: Universitäten, Fachhochschulen und duale Hochschulen
 - Nationality: german
 - IT-Skills: iPhone, Windows, PowerPoint, Excel, Online-Lernplattformen
 - Hobbies: Malen und Zeichnen, Lesen von Belletristik, Reisen und neue Kulturen entdecken, Musik hören und Konzerte besuchen
- PROFILE:** (This section is partially visible at the bottom of the image)

Persona mit LeanScope erstellen (Bild 3)

The screenshot shows the UXPressia interface for creating a user persona. The top navigation bar includes icons for UNDO, REDO, SHARE, COPY, PRESENT, EXPORT, COMMENTS, FONT, and HISTORY. On the left, a sidebar shows 'PERSONAS (2)' with a 'Not tagged' filter and two persona cards: 'Student' and 'Sophia, Business Student'. The main area displays the 'Sophia, Business Student' persona card with the following sections:

- NAME:** Sophia, Business Student
- MARKET SIZE:** 0 %
- TYPE:** Add Personality
- Goals:**
 - To pass exams and graduate with a good degree.
 - To understand complex business administration concepts.
 - To find a study method that fits her schedule.
- Quote:** "I need a study tool that adapts to my busy schedule and helps me grasp complex concepts without overwhelming me."
- Demography:**
 - Female, 20 years
 - United States
 - Single
 - Student
- Background:** Sophia is a second-year student at a major university, specializing in business administration. She has a part-time job and is active in several student organizations, which leaves her with limited time for studying. Sophia is tech-savvy and prefers using digital tools for learning over traditional textbooks.
- Motivations:**
 - Flexible learning that fits her busy schedule.
 - Interactive content that makes studying more engaging.
 - Tools that help her track her progress and identify areas of improvement.
 - Being able to study anywhere.
- Frustrations:**
 - Traditional textbooks are too bulky and not interactive.
 - Finding reliable resources for practice problems and quizzes.
- Needs:**
 - Access to a comprehensive study tool that covers all aspects of business administration.
 - A platform that offers both theory

Persona mit UXPressia (Bild 4)

KI bei der Ideenfindung

Sobald die Problemstellung klar definiert ist und die User Stories vorliegen, beginnt die Phase der Ideenfindung, in der verschiedene Lösungsansätze für ein definiertes Problem

entwickelt werden. Ziel ist es, durch Brainstorming, Design Thinking oder Methoden wie „How Might We?“ einsatzfähige Ideen zu generieren, die nutzerzentriert, innovativ und umsetzbar sind. Dabei können verschiedene Kreativtechniken sowie KI-gestützte Tools genutzt werden, um den Prozess zu unterstützen und eine breite Palette an Lösungsansätzen zu entwickeln.

Die „How Might We?“-Technik wird häufig genutzt, um Herausforderungen in Chancen für innovative Lösungen zu verwandeln. Die Grundidee dieses Ansatzes besteht darin, Probleme in offene, positiv formulierte Fragen zu überführen, die kreatives Denken anregen. Diese Fragen sind spezifisch, lösungsorientiert und fördern die Entwicklung vielfältiger Lösungsansätze.

Unser KI-Assistent liefert bereits die ersten Ansätze, die nachbearbeitet und erweitert werden können (Bild 6). Er geht dabei strukturiert vor und definiert im ersten Schritt die Problemstellung, nachfolgend formuliert das Tool dazu Fragen, die darauf abzielen, das Problem zu lösen, und anschließend leitet es Lösungsansätze ab.

Auch bei der Erstellung von User Flows bietet KI eine gute Unterstützung. Einen Ausschnitt davon zeigt Bild 7. Ein User Flow beschreibt dabei den Weg, den ein Nutzer durch eine App oder Website geht, um ein bestimmtes Ziel zu erreichen. Er zeigt die verschiedenen Schritte und Entscheidungen, die ein Nutzer trifft, sowie die Bildschirme oder Funktionen, die er durchläuft.

Hier sind einige User Stories für die BWL Master-App, basierend auf verschiedenen Nutzerbedürfnissen:

Lernen & Theorie verstehen

- Als BWL-Student möchte ich strukturierte Lerninhalte zu verschiedenen BWL-Fächern abrufen können, damit ich gezielt Wissen aufbauen kann.
- Als Student möchte ich kurze und prägnante Zusammenfassungen lesen, damit ich mich schnell in ein Thema einarbeiten kann.
- Als visuell lernender Nutzer möchte ich interaktive Grafiken und erklärende Videos nutzen können, um komplexe Zusammenhänge besser zu verstehen.

Üben & Wissen festigen

- Als Student möchte ich typische Rechenbeispiele aus der Praxis durchgehen, damit ich sicherer in der Anwendung der Theorie werde.
- Als Nutzer möchte ich Quizfragen mit direktem Feedback beantworten können, um mein Wissen zu testen.
- Als Student möchte ich eine Auswertung meiner Fehler erhalten, damit ich gezielt an meinen Schwächen arbeiten kann.

Lernkarten & Wiederholungen

- Als Student möchte ich digitale Lernkarten erstellen und speichern, damit ich gezielt Begriffe und Definitionen wiederholen kann.
- Als Nutzer möchte ich ein System, das mir automatisch Lernkarten häufiger zeigt, die ich falsch beantwortete, damit ich effizienter lernen kann.
- Als Nutzer möchte ich auf bereits vorhandene Lernkartensets zugreifen, um mir Zeit bei der Erstellung eigener Karten zu sparen.

User Stories mit ChatGPT erstellen (Bild 5)

Prototypen mit KI erstellen

KI-Tools sind auch bei der Erstellung von Prototypen hilfreich. Dabei sind verschiedene Einsatzmöglichkeiten denkbar. Es ist beispielsweise möglich, aus einfachen Textbeschreibungen automatisch Wireframes zu generieren, handgezeichnete Skizzen in digitale Prototypen umzuwandeln, Farben, Layouts und Typografie anzupassen und basierend auf Designrichtlinien Vorschläge zur Verbesserung der Benutzererfahrung zu machen.

Anhand von Figma AI [6] schauen wir uns einige konkrete Einsatzmöglichkeiten von KI an. Figma AI bietet Unterstützung für den gesamten UX-Design-Prozess, von der Idee bis zum fertigen Produkt.

FigJam ist ein digitales Whiteboard-Tool von Figma, das für kollaborative Brainstorming-Sessions, Workshops und Team-Meetings entwickelt wurde. Die KI-Funktion von FigJam macht es möglich, aus den Brainstorming-Sessions mit nur einem Click direkt Flow Charts zu generieren beziehungsweise aus einer einfachen Beschreibung ein Flowchart oder eine Mindmap zu erstellen. Diese Funktion ist für eine Strukturierung der künftigen Anwendung sehr hilfreich. Die KI kann auch aus einer Vielzahl von Sticky Notes oder Kommentaren eine kompakte Zusammenfassung erstellen, Notizen, Gedanken und Ideen clustern und diese logisch anordnen.

Eine weitere nützliche Funktion ist eine automatische Erstellung von Präsentationen. Die Designer kennen die Situation: Ein Produktvorschlag soll den Stakeholdern vorgestellt werden. Es kann sehr mühsam sein und viel Zeit kosten, eine entsprechende Produktpräsentation zu erstellen. Das übernimmt die KI von FigJam und liefert auf nur einen Click hin eine fertige Präsentation. Das erspart viel Zeit.

Hier ein Beispiel für die Anwendung der "How Might We" (HMW)-Methode zur Ideenfindung für die BWL Master-App:

Schritt 1: Problemstellung definieren

Studierende haben oft Schwierigkeiten, sich strukturiert auf Prüfungen vorzubereiten, da sie von der Vielzahl an Lernmaterialien überwältigt sind und nicht wissen, wie sie effektiv lernen sollen.

Schritt 2: Formulierung von HMW-Fragen

Basierend auf diesem Problem könnten wir folgende "How Might We"-Fragen stellen:

1. Wie könnten wir Studierenden helfen, ihr Lernen effizienter zu strukturieren?
2. Wie könnten wir das Lernen interaktiver und motivierender gestalten?
3. Wie könnten wir Studierenden ermöglichen, gezielt ihre Schwächen zu erkennen und daran zu arbeiten?
4. Wie könnten wir realistische Prüfungssituationen simulieren, um die Vorbereitung zu verbessern?
5. Wie könnten wir kollaboratives Lernen fördern, damit Studierende sich gegenseitig unterstützen können?

Schritt 3: Ideen ableiten

Für jede dieser Fragen könnten nun kreative Lösungen entwickelt werden:

1. **Individueller Lernplan** – Automatisch generierte Lernpläne basierend auf Prüfungsdatum und Lernfortschritt.
2. **Gamification-Elemente** – Punkte, Badges oder Challenges, um Motivation zu steigern.
3. **KI-gestützte Analyse** – Automatische Auswertung von Tests, um gezielt Lücken zu erkennen.
4. **Klausurmodus** – Prüfungen unter realistischen Bedingungen mit Zeitbegrenzung und Auswertung.

Ideenfindung mit dem KI-Assistenten ChatGPT (Bild 6)

Auch bei der Erstellung von Prototypen bietet die AI-Funktion von Figma eine Unterstützung. Die Funktion „First Draft“ liefert die ersten Ideen, wie eine künftige Anwendung aussehen könnte. Diese können beliebig geändert und angepasst werden (Bild 8).

Designer haben oft das folgende Problem: Um Prototypen realistisch wirken zu lassen, müssen diese mit Inhalt gefüllt

Hier ist ein **User Flow** für die **BWL Master-App**, der zeigt, wie ein Nutzer durch die App navigieren kann:

🔴 Einstieg & Anmeldung

1. **Startbildschirm** → Nutzer öffnet die App
2. **Anmeldung / Registrierung**
 - Login mit E-Mail, Google oder Apple
 - Optional: Gastmodus
3. **Onboarding & Interessenabfrage**
 - Auswahl der BWL-Fächer (z. B. Rechnungswesen, Marketing, VWL)
 - Angabe des Studienfortschritts (Erstsemester, Fortgeschritten, Prüfungsvorbereitung)
 - Lernziele festlegen (z. B. „Klausur bestehen“, „Wissen vertiefen“)

Unterstützung bei der
Erstellung des User
Flows (Bild 7)

📄 Hauptmenü – Überblick über die Funktionen

- **Lernen** → Theorie & Zusammenfassungen
- **Üben** → Rechenbeispiele & Aufgaben
- **Lernkarten** → Eigene & vorgefertigte Karteikarten
- **Quiz & Abfragen** → Interaktive Fragen zur Selbstkontrolle
- **Klausurmodus** → Simulation unter echten Bedingungen
- **Profil & Fortschritt** → Lernstatistik, Badges & Motivation

📖 1. Theorie lernen

1. Auswahl eines Fachs (z. B. Rechnungswesen)
2. Auswahl eines Kapitels (z. B. Bilanzierung)
3. Anzeige von Lernmaterialien
 - Textbasierte Zusammenfassungen
 - Infografiken & Formeln
 - Erklärvideos (falls vorhanden)
4. **Aktionen:**
 - ✓ „Merken“ für spätere Wiederholung
 - ✓ „Lernkarten erstellen“ aus wichtigem Inhalt

📊 2. Üben & Rechenbeispiele

1. Auswahl eines Fachs & Themas (z. B. Kostenrechnung)
2. Anzeige von Beispielaufgaben
 - Multiple-Choice, offene Fragen, Rechnungen
 - Schritt-für-Schritt-Lösung & Erklärungen
3. **Aktionen:**
 - ✓ Aufgabe als „leicht“ oder „schwierig“ markieren
 - ✓ Notizen hinzufügen
 - ✓ Ähnliche Aufgaben empfehlen lassen

📇 3. Lernkarten erstellen & nutzen

werden. Hierfür kommen ebenfalls zahlreiche KI-Funktionen von Figma infrage. So können Texte ebenso wie Tabellen durch Anweisungen erstellt, geändert, erweitert und umgeschrieben werden (Bild 9). KI ist auch in der Lage, die eingefügten Texte in verschiedene Sprachen zu übersetzen und dazu passende Bilder zu generieren. Sowohl einzelne Textabschnitte als auch fertige Designs können mit einem Click übersetzt werden.

Das KI-Feature „Create Prototype“ macht die Prototypen in wenigen Minuten interaktiv (Bild 10). Dieser Prozess kann ohne KI viel Zeit in Anspruch nehmen.

Prototypen mit KI evaluieren und testen

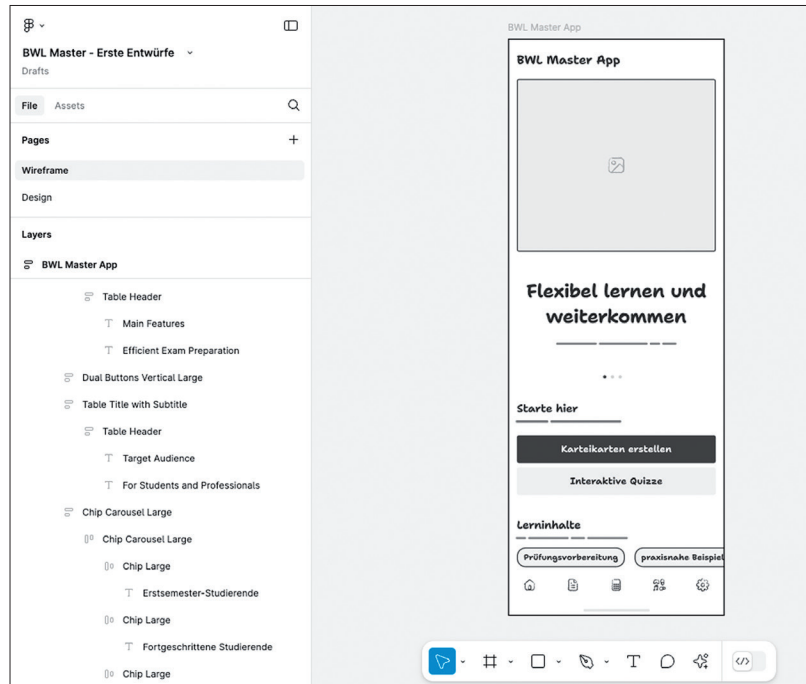
KI kann die Endnutzer:innen bei der Evaluierung von Prototypen nicht ersetzen, da Produkte letztendlich von Menschen mit individuellen Erfahrungen, Erwartungen und unterschiedlichem technischem Wissen genutzt werden. Daher bleiben Usability-Tests mit echten Nutzer:innen unerlässlich. Allerdings kann KI als wertvolles Hilfsmittel dienen, insbesondere zur Unterstützung bei der Bewertung von Designstandards und der Analyse von Nutzerinteraktionen.

Mit dem Tool Hotjar [1] ist es beispielsweise möglich, die Ideen mit KI-gestützten Umfragen zu validieren, Heatmaps zu erstellen und Sitzungsaufzeichnungen durchzuführen. KI-gestützte Onsite-Umfragen kann man so integrieren, dass sie während der Nutzerreise genau dann erscheinen, wenn sie gebraucht werden. Tools wie Attention Insight [7] und Neurons AI [8] nutzen Eye-Tracking-Studien und neurowissenschaftliche Daten, um Aufmerksamkeitsmuster vorherzusagen und verschiedene Designvarianten zu vergleichen. Maze [9] führt automatisierte UX-Tests durch, analysiert Klickpfade und generiert Heatmaps. Lookback [10] erfasst Live-Nutzertests mit Videoaufzeichnung und ermöglicht Interaktionsanalysen.

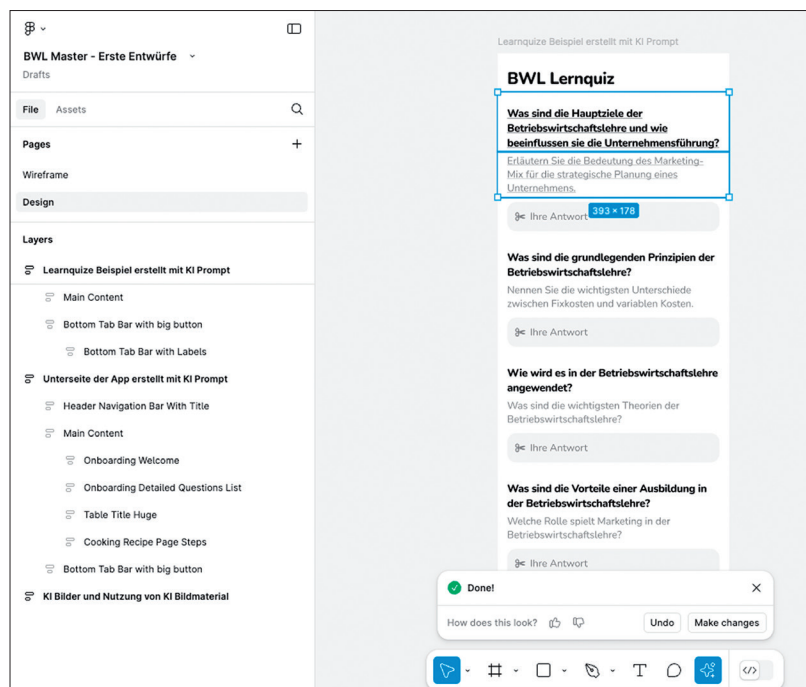
KI-gestützte Testing-Tools helfen dabei, UX-Designs effizienter zu analysieren, Nutzerverhalten vorherzusagen und Usability-Probleme frühzeitig zu erkennen. Sie ermöglichen eine schnellere, automatisierte und datengestützte Evaluierung von Prototypen und digitalen Produkten. Diese KI-gestützten Technologien können die Zusammenarbeit zwischen UX-Designer:innen und Nutzer:innen nicht ersetzen, aber gezielt eingesetzt die Effizienz steigern und den Designprozess optimieren.

Prompt Engineering

Die Interaktion mit KI erfolgt über Prompts. Die Gestaltung der Input-Prompts spielt eine zentrale Rolle, auch im Rahmen



Prototypen mit einem Klick in Figma Ai erstellen (Bild 8)



Inhalte mit KI in Figma AI generieren (Bild 9)

des UX-Designs mit KI-Assistenten, da die Qualität der Ergebnisse maßgeblich von der Güte der Eingaben abhängt. Prompt Engineering beschreibt den Prozess der gezielten Anpassung von Sprache und Format eines Prompts, um die KI optimal zur gewünschten Ausgabe zu führen. Präzise und klare Prompts wirken sich direkt auf die Qualität der Antworten oder Aktionen der KI aus.

Es ist wichtig, darauf vorbereitet zu sein, dass nicht jedes Ergebnis auf Anhieb optimal ausfallen wird. Häufig sind ►

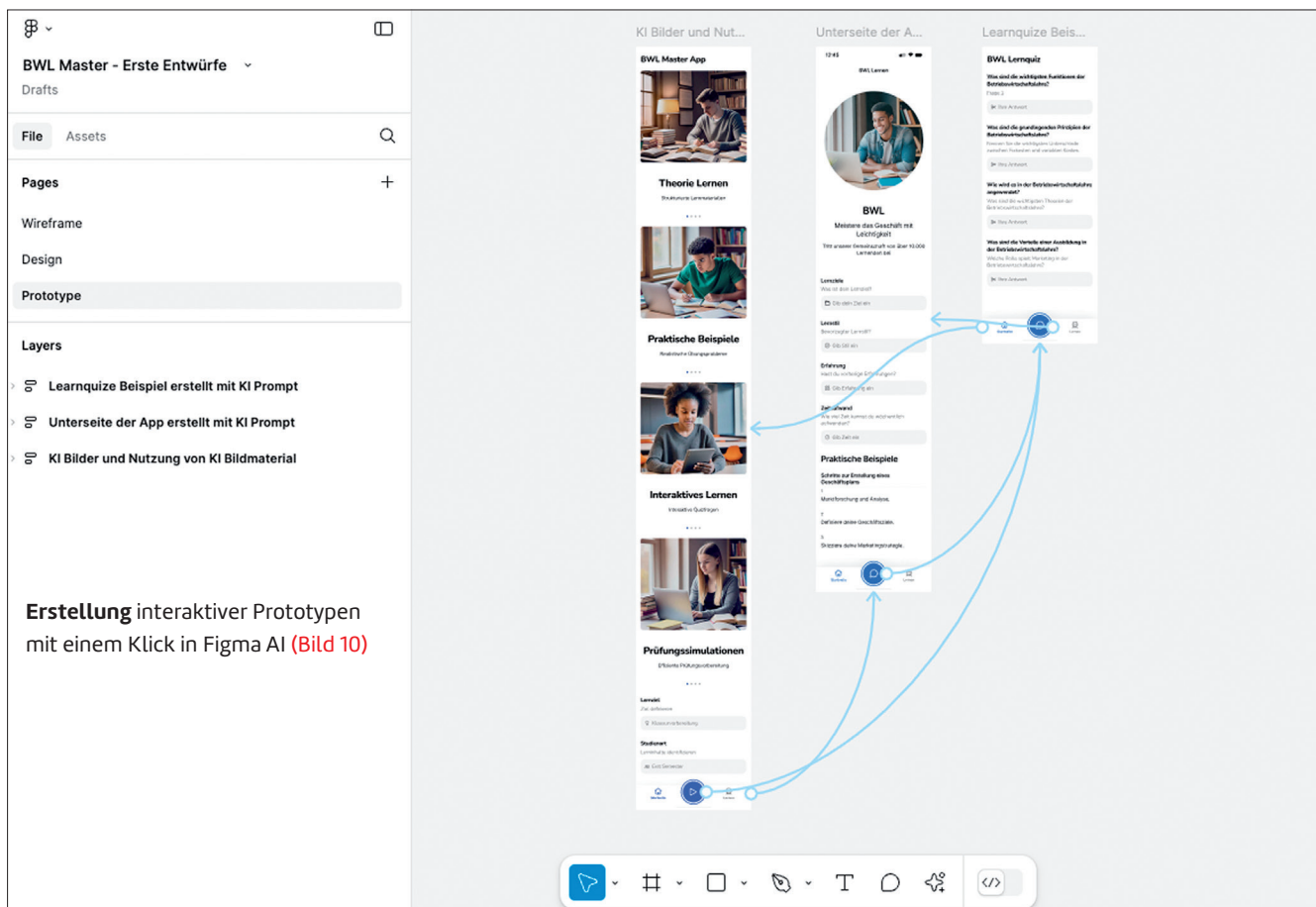
mehrere Anläufe oder Beispiele notwendig, um die gewünschte Qualität zu erreichen.

Es gibt verschiedene Arten von Prompts, die sich in ihrer Struktur und ihrem Einfluss auf die Antworten der KI unterscheiden. Bei Zero-Shot Prompts erhält die KI keinerlei spezifische Beispiele und generiert ihre Antworten ausschließlich auf Basis ihres allgemeinen Wissens. Dadurch sind die Ergebnisse meist eher allgemein gehalten. One-Shot Prompts hingegen geben der KI ein einzelnes Beispiel oder eine gezielte Anweisung mit, die als Orientierung dient. Dies ermöglicht es, die Antwort in eine bestimmte Richtung zu lenken. Noch spezifischer sind Few-Shot Prompts, die mehrere Beispiele oder Informationen enthalten. Dadurch kann die KI aus den vorgegebenen Mustern lernen und kontextuell präzisere sowie detailliertere Antworten liefern. Eine weitere Technik ist das sogenannte Chain-of-Thought Prompting. Hierbei wird die KI dazu angeleitet, ihren Denkprozess Schritt für Schritt darzulegen. Diese Methode ist besonders nützlich, um Einblicke in die Herleitung der KI-Ergebnisse zu gewinnen und ihre Entscheidungsprozesse besser nachvollziehen zu können. Im Rahmen des UX-Designs hat sich die Vorgehensweise „Vom Allgemeinen zum Speziellen“ durchgesetzt. Dabei fängt man mit allgemeinen Prompts an, die zunächst die Problemstellung grob erklären.

Beispiele für solche Prompts für die im Kasten **Beispiel-App BWL Master** beschriebene BWL-Master-App könnten wie folgt aussehen:

- **Zero-Shot Prompt:** „Wie kann eine Lern-App für BWL-Studierende benutzerfreundlich gestaltet werden?“ Die KI würde daraufhin allgemeine Prinzipien wie eine intuitive Navigation, klare Strukturierung der Inhalte oder Gamification-Elemente vorschlagen.
- **One-Shot Prompt:** „Wie kann der Prozess des Erstellens von digitalen Lernkarten in einer BWL-Lern-App möglichst intuitiv gestaltet werden? Berücksichtige ein Beispiel einer bestehenden App wie Anki.“ Dadurch würde die KI gezielt Designlösungen vorschlagen, die sich an bewährten Methoden orientieren.
- **Few-Shot Prompt:** „Hier sind drei verschiedene Layouts für den Quiz-Modus einer Lern-App. Analysiere ihre Vor- und Nachteile und schlage eine Verbesserung vor.“ Die KI könnte dann Best Practices ableiten und konkrete Optimierungsvorschläge liefern.
- **Chain-of-Thought Prompt:** „Beschreibe Schritt für Schritt, wie man den Onboarding-Prozess für eine BWL-Lern-App gestaltet, um Erstsemester-Studierenden einen optimalen Einstieg zu ermöglichen.“ Die KI würde dabei einzelne Phasen wie Begrüßungsbildschirm, Einführungstouren und erste Interaktionsmöglichkeiten detailliert durchdenken.

Wie bereits erwähnt, hängt die Qualität der KI-Ausgabe stark von der Gestaltung der Eingabe ab. Daher ist es essenziell, Prompts effektiv zu formulieren, um die gewünschten Ergebnisse zu erzielen. Es ist wichtig, klar anzugeben, in welchem



● Beispiel-App BWL Master

Nachfolgend wollen wir die Funktionen der KI in unterschiedlichen Phasen des User-Research-Prozesses anhand der Beispiel-App BWL Master testen.

Bei BWL Master handelt es sich um eine Lern- und Übungs-App für BWL-Studierende, die ihr Wissen gezielt aufbauen und sich optimal auf Prüfungen vorbereiten möchten. Die App bietet hierfür eine breite Auswahl an Themen und umfasst zahlreiche Funktionen:

- **Theorie lernen:** Strukturiert aufbereitete Lerninhalte mit verständlichen Erklärungen und praxisnahen Beispielen.
- **Beispiele rechnen:** Übung mit realistischen Aufgaben und ausführlichen Lösungen, um das Verständnis zu festigen.
- **Lernkarten erstellen:** Möglichkeit der Erstellung eigener digitaler Lernkarten beziehungsweise Nutzung vorgefertigter Sets, um Wissen einzuprägen.
- **Abfragen:** Eine Gelegenheit, das Wissen mit interaktiven Quizfragen zu überprüfen.
- **Klausuren simulieren:** Prüfungsvorbereitung unter realen Bedingungen.

Die Zielgruppe der BWL-Master-App sind hauptsächlich Studierende der Betriebswirtschaftslehre an Universitäten, Fachhochschulen und dualen Hochschulen, die sich effizient auf Klausuren vorbereiten und ihr Wissen vertiefen möchten:

- Erstsemester-Studierende, die sich schnell in die Grundlagen einarbeiten wollen.
- Fortgeschrittene Studierende, die gezielt für Prüfungen lernen und ihr Wissen festigen möchten.
- Duale Studenten, die Theorie und Praxis verbinden und ihr Wissen flexibel auffrischen müssen.
- Berufstätige in der Weiterbildung, die sich neben dem Job auf BWL-relevante Themen vorbereiten.
- Prüflinge in wirtschaftsbezogenen Zertifikatskursen, die sich strukturiert und zeitsparend auf ihre Prüfungen vorbereiten möchten.

Die App richtet sich an alle, die flexibel, digital und interaktiv lernen wollen – egal ob für regelmäßige Wiederholungen oder die intensive Klausurvorbereitung.

Format die Antwort gewünscht wird. Falls eine bestimmte Struktur erforderlich ist, sollte dies explizit formuliert werden, etwa: „Erstelle einen Vergleich als Tabelle mit drei Spalten und den Überschriften...“. Besonders bei komplexen Aufgaben ist Kontext unverzichtbar, um die KI gezielt zu führen. Beschreibungen der aktuellen Situation oder relevanter vorheriger Interaktionen können hilfreich sein. Beispiele bieten zusätzliche Orientierung. Oft benötigt es mehrere Anläufe, um die besten Ergebnisse zu erzielen. Die iterative Anpassung der Prompts verbessert die Qualität der Ausgaben schrittweise. Die gewünschte Antwortlänge kann explizit vorgegeben werden. Für detaillierte Antworten könnte der Prompt beispielsweise lauten: „Erkläre mir ausführlich...“, während für kürzere Antworten formuliert werden kann: „Fasse in ein paar Sätzen zusammen...“. Unterstützende Tools wie das Prompt-Perfect-Plug-in oder Vorlagen können den Prozess der Anweisungserstellung vereinfachen und neue kreative Ansätze inspirieren. Das Prompt-Perfect-Plug-in ist eine Erweiterung, die dazu dient, die Qualität der Antworten von ChatGPT zu steigern. Es wurde entwickelt, um die Eingaben zu bewerten und sie bei Bedarf in klarere, spezifischere und kontextbezogene Anfragen umzuformulieren. Diese Erweiterung erweist sich als besonders nützlich, wenn die ursprüngliche Eingabe mehrdeutig oder unklar ist, da sie dabei hilft, Missverständnisse zu vermeiden und die Präzision der Antworten von ChatGPT zu erhöhen. Wie alle anderen ChatGPT-Plug-ins kann Prompt Perfect nur in kostenpflichtigen Versionen von ChatGPT verwendet werden. Man findet Prompt Perfect im Plug-in-Bereich. Wenn man eine Eingabeaufforderung mit diesem Plug-ins verbessern möchte, fügt man einfach das Präfix „Perfekt“ gefolgt von der Eingabeaufforderung ein. Zum Beispiel: „Perfekt [Ihre Eingabeaufforderung hier]“.

Fazit und Ausblick

Künstliche Intelligenz ist mehr als nur ein Werkzeug – sie ist ein echter Kreativpartner in der UX-Gestaltung. Sie inspiriert, automatisiert Routineaufgaben und ermöglicht es, neue Designansätze zu entdecken, die vorher vielleicht nicht in Betracht gezogen worden wären. Dennoch bleibt die menschliche Kreativität unverzichtbar: Nur durch die Kombination aus KI-gestützter Inspiration und menschlicher Intuition entstehen Designs, die nicht nur funktional, sondern auch emotional ansprechend sind. ■

[1] Hotjar, <https://www.hotjar.com>

[2] Lyssna, <https://www.lyssna.com>

[3] FullStory, <https://www.fullstory.com>

[4] LeanScope, <https://www.leanscope.ai>

[5] Uxpressia, <https://uxpressia.com>

[6] Figma AI, <https://www.figma.com/ai/>

[7] Attention Insight, <https://attentioninsight.com>

[8] Neurons, <https://www.neuronsinc.com/neurons-ai>

[9] Maze, <https://maze.co>

[10] Lookback, <https://www.lookback.com>



Elena Bochkor

beschäftigt sich mit dem Entwurf und Design mobiler Anwendungen, Webseiten und digitaler Produkte. Ihr technologischer Fokus liegt auf C#, WPF, WinUI 3 und .NET MAUI, und sie bietet entsprechende Workshops für Entwickler an.

<https://larinet.com>

dnpCode

A2504-05UXundKI

AN DER WURZEL ANSETZEN

Die Schuld

Der Abbau von Schulden in der Softwareentwicklung ist schon schwierig genug – beginnt man damit jedoch nicht an der richtigen Stelle, wird das Problem nur noch schlimmer.

Das Betrachten und Ermitteln von Schulden in der Softwareentwicklung ist in meinen Beratungsprojekten das täglich Brot, und damit verbunden ist selbstverständlich auch das Erarbeiten einer Lösungsstrategie für ebendiese Schulden. Dabei ist es mir schon oft passiert, dass Teilnehmer in Meetings mich freundlich in meinem Sprachgebrauch korrigierten: „David, es heißt technische Schulden, nicht technische Schuld – es geht um das Problem und nicht darum, wer es verursacht hat.“ Für mich ist es immer ein großer Spaß, weil ich häufig lange warten muss, bis dieser kleine, aber feine Unterschied in meiner Wortwahl einem Teilnehmer auffällt. Es stimmt: Wenn wir Fehler in der Softwareentwicklung machen, bauen wir Schritt für Schritt einen nicht unerheblichen Berg von Schulden auf – also eine Last, die wir irgendwann „zurückzahlen“ müssen. Dennoch habe ich mit meinem kleinen Wortspiel recht: Wir reden auch von der Schuld, denn wenn wir die verursachende Person oder Situation nicht identifizieren, können wir das gesamte Problem nicht lösen.

Die Schuldfrage

Eine der häufigsten Aussagen im Hinblick auf die Zusammenarbeit von Teams ist, dass „Fingerpointing“ niemandem weiterhelfe und nur die Teamdynamik verschlechtere. Das sehe ich nicht so. Im Gegenteil halte ich diese Ansicht sogar für gefährlich – für die Softwarequalität und den langfristigen Erfolg des Projekts an sich. Zunächst müssen wir jedoch feststellen: Ich mache Fehler, sehr häufig sogar, wenn man die kleinen Fehler mit einberechnet. Genauso wie Sie häufig Fehler machen, oder Ihre Kollegen und Ihre Vorgesetzten. Jeder von uns macht also Fehler, und diese Fehler führen zu Problemen in unseren Softwareprojekten. Das ist ein Umstand, den wir nur schwerlich verhindern können und leider immer wieder falsch angehen. Ich kenne einige Unternehmen, in denen es als absolutes No-Go gilt, einen Fehler zu begehen. Und passiert einem dergleichen dennoch, versucht man alles, um diesen Fehler zu kaschieren, damit bloß kein Kollege oder gar Vorgesetzter ihn erkennt und ihn einem selbst zuordnet, weil man für Fehler später direkt oder indirekt bestraft wird.

Ich habe aber deutlich mehr Kunden, die dieselbe Problematik komplett anders angehen. Mache ich oder ein Kollege einen Fehler, wird dieser direkt angesprochen, analysiert und gemeinsam überlegt, wie erreicht werden kann, dass der Fehler nicht noch einmal passiert. Es wird also bei jedem Fehler individuell oder als gesamtes Team und/oder Abteilung daraus gelernt. Einer meiner Kunden prämiert in der Entwicklungsabteilung sogar jeden Monat den lehrreichsten ge-

machten Fehler mit einem Amazon-Gutschein über 50 Euro. In welchem Team würden Sie lieber arbeiten? In dem angst-erfüllten Team, oder in dem Team, das Fehler konstruktiv anspricht und gemeinsam behebt? Was glauben Sie, welches Team sich im Lauf der Zeit erfolgreicher weiterentwickelt?

Ich denke, bei beiden Fragen kann ich Ihre Antwort vorwegnehmen: Ein Team mit einer positiven Fehlerkultur hat eine bessere Arbeitsumgebung und entwickelt sich kontinuierlich weiter. Diese Aussagen hier sind natürlich primär auf allgemeine „Alltagsfehler“ in der Softwareentwicklung bezogen, aber ihnen kommt noch deutlich mehr Gewicht zu, wenn es um das Thema der entstandenen Schulden geht.

Die Ursache

In meinen Workshops nenne ich ein Problem dieser Art gern „Aspirinproblem“. Stellen Sie sich vor, ich habe einen mehr-tätigen Workshop bei Ihnen, und am ersten Tag wache ich morgens im Hotel mit unerträglichen Kopfschmerzen auf. Genau für solche Fälle habe ich immer Kopfschmerztabletten dabei. Damit lässt sich das Problem recht schnell in den Griff bekommen, und so kann der Workshop morgens zeitnah losgehen. Habe ich am nächsten Tag erneut Kopfschmerzen, so kann ich die Prozedur wiederholen, und am dritten Tag auch. Aber wäre es nicht sinnvoller, darüber nachzudenken, warum die Kopfschmerzen überhaupt auftreten? Ist vielleicht meine Haltung beim Einschlafen ungesund? Ernähre ich mich falsch? Konsumiere ich zu viel Alkohol am Vorabend? Liegt eine solche Ursache tatsächlich vor, kann ich natürlich trotzdem eine Tablette nach der anderen einwerfen – und bekämpfe damit kurzfristig das Problem. Langfristig löse ich es aber nicht.

Genau dieses „Aspirinproblem“ trifft in besonderem Maß auch bei den Schulden in der Softwareentwicklung zu. Sind diverse Schuldenberge im Projekt entstanden, so erfordert dies meist enorme Mengen Zeit in verschiedenen Bereichen, um sie wieder abzutragen. Das Problem an sich ist schon schwerwiegend genug. Aber bevor dieser langlaufende Prozess angestoßen werden sollte, steht als Erstes die Frage im Raum: „Wer oder was hat überhaupt Fehler gemacht, sodass diese technischen Schulden entstanden sind?“ Solange diese Fehlerquellen nicht behoben sind, werden die Verursachenden weiterhin fleißig immer neue Schulden erzeugen.

Die Dimension

Was hier ganz klar, logisch und nachvollziehbar klingen mag, ist in der Praxis oft ein großes Problem. Zuzugeben, einen Fehler begangen zu haben, der einen Sprint kaputt gemacht,

einen Kunden verärgert oder anderweitig Geld gekostet hat, bedeutet für viele bereits eine große Überwindung. Im Bereich der Schulden reden wir aber von völlig anderen Dimensionen.

Vor einigen Tagen hatte ich beispielsweise einen Teamleader, der vor rund zehn Jahren einige (offensichtlich) wichtige Schulungen in seinem Team nicht durchgeführt hat. Die dadurch entstandenen Wissensschulden haben zu einem schlechten Anwendungsdesign geführt und somit zu enormen strukturellen Schulden in Höhe von etwa 3,5 Millionen Euro im Jahr 2025. Über Fehler dieser Größenordnung offen und frei reden zu können, solche Fehler also grundlegend analysieren und entsprechende Maßnahmen ergreifen zu können, ist die absolute Grundvoraussetzung für den richtigen Umgang mit Schulden jeder Art in der Softwareentwicklung. Damit aber von Beginn an Klarheit herrscht: Die Schuld an sich ist manchmal ein individuelles Versagen, oftmals aber auch ein kollektives und noch öfter auch ein organisatorisches Versagen. Hier ist es enorm wichtig, eine Feedbackkultur zu pflegen, die sowohl individuelle Fehler begrüßt als auch solche, die aus dem Team oder der Organisation selbst heraus entstehen, und wo diese Feedbackkultur nur eingeschränkt oder gar nicht existiert, gilt es sie zu fördern und gegebenenfalls zu etablieren. Einer der Kernpfeiler der modernen agilen Softwareentwicklung ist das kontinuierliche Verbessern (englisch: Continuous improvement), und an dieser Stelle ist es ganz besonders wichtig.

Auf allen Ebenen – wirklich allen!

Diskutiert man diesen Umstand in Projekten, herrscht meist allgemeiner Konsens darüber, wie wichtig dieses Thema ist, und gerade Personen außerhalb der Softwareentwicklung ermutigen die Entwickler:innen oftmals, offen und transparent mit diesen Fehlern umzugehen. Dieselben außenstehenden Personen realisieren aber häufig nicht, wie weit sie mitunter selbst in die Problemlagen mit eingebunden sind. Hier nutze ich gern die 5-Why-Analyse: Zu einem existierenden Problem wird fünfmal die Frage nach dem „Warum?“ gestellt, mit dem Ergebnis, dann oft der Ursache auf die Schliche gekommen zu sein. Warum ist also der Quellcode so unfassbar komplex, dass er nicht mehr erweitert werden kann? Weil die Architektur schlecht ist. Warum ist die Architektur schlecht? Weil nur sehr wenig Wissen darüber im Team existiert. Warum existiert so wenig Wissen darüber? Weil von Führungsseite versäumt wurde, für entsprechende Weiterbildung zu sorgen. Warum wurde das versäumt? Weil die Führung immer mehr neue Funktionen wollte und daneben kaum Zeit blieb. Warum wurden immer nur Funktionen priorisiert? Weil der Führung nicht klar war oder ist, was in der Softwareentwicklung wichtig ist.

So oder ähnlich laufen viele Analysen zu Schuldenbergen in der Softwareentwicklung ab. Man kann natürlich ausgiebig darüber diskutieren, auf welcher Ebene der Fehler nun wie sehr verstärkt wurde, aber so gut wie alle Schulden entstehen auf der Ebene der Entscheider. Weil Prioritäten falsch gesetzt wurden, falsche Entscheidungen getroffen wurden oder weil – wie in unserem Beispiel – oft einfach nicht das



Wissen vorhanden war, wie Softwareentwicklung funktioniert und was tatsächlich wichtig ist.

Wie Sie sich vorstellen können, erzeugt das in der Praxis häufig große und manchmal sogar unlösbare Probleme. Vor allem diejenigen Führungskräfte, die durch ihren eigenen schlechten Umgang mit Fehlern für eine miserable Fehlerkultur im Unternehmen sorgen, übernehmen oft auch nicht die Verantwortung. Es ist dann meist nicht mehr weit bis zu dem Punkt, an dem auch die jeweiligen Mitarbeiter ihre Fehler nicht auf-

arbeiten, sondern sie lieber ignorieren. Vor einiger Zeit fasste ein Geschäftsführer einen Workshop treffend so zusammen: „Herr Tielke, aber in der Praxis erzeugt doch jedes Team verschiedenen Arten von Schulden an allen möglichen Stellen.“ Und nach einer kurzen Gedankenpause fügte er an: „Aber nur diejenigen, die damit selbstkritisch und transparent umgehen, bekommen das Problem langfristig in den Griff.“ Besser hätte ich es nicht sagen können!

Fazit

Wir haben in dieser und in den vergangenen drei Episoden [1] bis [3] ausführlich das Thema der Schulden in der Softwareentwicklung diskutiert. Ich habe Ihnen gezeigt, welche Arten es gibt, welche Konsequenzen sie haben, wie sie sich auf Ihr Projekt auswirken, wie sie abgebaut werden können, und Ihnen damit den Schlüssel für die nachhaltige Lösung der Schuldenproblematik in die Hand gegeben. Sie haben gesehen, dass das Thema durchaus eine hohe Komplexität aufweist, sowohl auf der Ebene der Schulden selbst als auch auf der Lösungsebene, und dass zur Erkennung, Abarbeitung und anschließenden Vermeidung solcher Schulden eine Menge Erkenntnisse bei verschiedenen Personen reifen und Entscheidungen getroffen werden müssen – aber das alles sind Tatsachen, denen sich jedes Softwareentwicklungsteam früher oder später stellen muss. Die Schulden gehören zur Softwareentwicklung leider dazu, den Schlüssel zum Erfolg habe ich Ihnen hoffentlich mit dieser kleinen Artikelserie gegeben. Ich wünsche Ihnen viel Erfolg, Mut und Ausdauer dabei! ■

[1] David Tielke, *Der Schuldensumpf*, dotnetpro 1/2025,

Seite 48 f., www.dotnetpro.de/A2501DDD

[2] David Tielke, *Nicht nur in der Technik ...*, dotnetpro

2/2025, Seite 54 f., www.dotnetpro.de/A2502DDD

[3] David Tielke, *Die Konsequenz von Schulden*, dotnetpro

3/2025, Seite 42 f., www.dotnetpro.de/A2503DDD



David Tielke

ist freiberuflicher Berater und von Microsoft zertifizierter Trainer für die Anwendungsentwicklung auf der .NET-Plattform. Darüber hinaus hat er sich auf die Bereiche Softwarearchitektur, Softwarequalität und ALM spezialisiert. mail@david-tielke.de

dnpCode

A2504-05DDD

ENTWURF UND ENTWICKLUNG WARTBARER SOFTWARESYSTEME

SOLIDe Architektur

Wer einige wenige Prinzipien versteht und sich an sie hält, produziert besseren Code.

Bevor wir über Prinzipien sprechen, sollten wir uns klar darüber sein, was gute Softwarearchitektur auszeichnet und warum wir überhaupt Architektur brauchen. Guter Code und das Ergebnis von guter Architektur bedeuten für mich:

- Man findet Code da, wo man ihn vermutet.
- Man kann Code testen.
- Man kann leicht Funktionalität ändern, ohne dass „alles zusammenbricht“.
- Code ist lesbar und deshalb optisch leicht verständlich.
- Code ist nicht komplex, sondern einfach und deshalb fehlerfreier.

Wie erreiche ich gute Architektur?

Gute Architektur erreicht man, indem Prinzipien angewendet werden, die durchdacht und erprobt sind. Je mehr dieser Prinzipien man kennt, desto flexibler ist man in der eigenen Vorgehensweise. Denn nicht immer hat man einen Nagel vor

sich, wenn man einen Hammer in der Hand hält. Sollten Sie von Softwarearchitektur noch gar nichts gehört haben, dann reicht es, mit einer einzigen Regel zu beginnen: Stellen Sie sich bei der Entwicklung immer die Frage: Ist der Code, den ich hier einfüge, eine Aufgabe, die diese Klasse oder Komponente zwingend erfüllen muss, oder gehört das Problem eigentlich an einer anderen Stelle gelöst? Schon sind wir beim Grundprinzip guter Architektur: Eine Komponente macht nur das, wofür sie da ist.

Schauen Sie sich [Listing 1](#) an. Es dient als Ausgangspunkt für alle unsere Überlegungen. Ich gehe davon aus, dass Sie Code dieser Art schon oft gesehen haben. Was fällt beim allerersten Blick auf? Welche Gedanken gehen einem dabei durch den Kopf?

Positiv ist, dass hier eine Stored Procedure verwendet wird. Das verhindert SQL Injection – ein Pluspunkt. Wir haben auch ein gewisses Logging. Auch das ist schon mal gut.

● Listing 1: Die Ausgangslage – der Mahnlauf mit Steuerberechnung

```
public static void Mahnlauf()
{
    SqlConnection conn = new SqlConnection(
        "Server=myServerAddress;Database=myDataBase;
        User Id=myUsername;Password=myPassword;");
    conn.Open();
    SqlCommand cmd = new SqlCommand("SELECT * FROM
        OffenePosten WHERE Status = 'offen'", conn);
    SqlDataReader reader = cmd.ExecuteReader();

    while (reader.Read())
    {
        decimal betrag =
            Convert.ToDecimal(reader["Betrag"]);
        string land = reader["Land"].ToString();
        decimal steuersatz = 0.19M;
        if (land == "DE")
            steuersatz = 0.19M;
        else if (land == "AT")
            steuersatz = 0M;

        decimal steuer = betrag * steuersatz;

        Console.WriteLine($"Erzeuge Mahnung für: " +
            $"{reader["Kunde"].ToString()}, " +
            "zu zahlen: {betrag + steuer} " *
            "(inkl. {steuersatz * 100}% Steuer)");

        SqlCommand insertMahnungCmd = new SqlCommand(
            "INSERT INTO Mahnungen (Kunde, Betrag, Steuer)
            VALUES (@Kunde, @Betrag, @Steuer)", conn);
        insertMahnungCmd.Parameters.AddWithValue(
            "@Kunde", reader["Kunde"]);
        insertMahnungCmd.Parameters.AddWithValue(
            "@Betrag", betrag);
        insertMahnungCmd.Parameters.AddWithValue(
            "@Steuer", steuer);
        insertMahnungCmd.ExecuteNonQuery();

        SqlCommand updateCmd = new SqlCommand(
            "UPDATE OffenePosten SET Status = 'gemahnt',
            Steuer = @Steuer WHERE Id = @Id", conn);
        updateCmd.Parameters.AddWithValue(
            "@Id", reader["Id"]);
        updateCmd.Parameters.AddWithValue(
            "@Steuer", steuer);
        updateCmd.ExecuteNonQuery();
    }
    conn.Close();
}
```

Als Nächstes fallen sicher Code-Styling-Richtlinien ins Auge. Vielleicht ist in denen, die in Ihrer Firma gelten, definiert, dass man Variablen immer mit *var* statt mit dem Datentyp deklariert, um Code lesbarer zu machen. Der Vorteil ist, dass alle Variablennamen untereinanderstehen. Aber das sind kosmetische Korrekturen.

Wenn es nur zwei Länder gibt, dann könnte man den Code hier vielleicht etwas vereinfachen:

```
var steuersatz = (land=="DE") ? 0.19M : 0M;
```

Weniger Zeilen Code sind immer besser, denn man kann mehr auf einmal lesen und den Code besser korrigieren.

Und sonst? Na ja, muss ein *Select* * unbedingt sein? Was ist, wenn die Tabelle 100 Spalten hat? Dann zieht man unnötig Daten über den Äther.

Das sind alles berechtigte Einwände, sie haben aber noch nichts mit Architektur zu tun. Wir hatten von guter Architektur gesprochen, wenn eine Methode nur das macht, was sie zwingend machen muss, und keine anderen Aufgaben erledigt. Was meinen Sie? Ist das im Beispiel der Fall?

Oder lassen Sie uns anders an die Sache herangehen: Was ist, wenn der Chef fragt: Hast du den Code auch getestet? Wenn der Kunde sagt: Wir haben jetzt auch Kunden aus der Schweiz. Wenn der Gesetzgeber beschließt, die Steuer von 19 auf 20 Prozent zu erhöhen. Dann beginnen die Probleme. Und dann merkt man, dass es schlechte Architektur war. Man müsste in allen Code-Teilen des Programms suchen, wo „19“ steht. Oder alle Codestellen ausfindig machen, in denen ein Land verwendet wird.

Und wie ist das mit einer Testmethode, wenn in der Datenbank keine offenen Posten da sind. Wie kann man denn sehen, ob die Mahnungen alle korrekt erzeugt wurden, ohne manuell in der Datenbank nachzuschauen? Weil das eben so aufwendig ist, wird lieber nicht getestet.

Und wie sieht das mit dem Logging aus, wenn die Anwendung beim Kunden läuft? Wie komme ich an den Output der Console? Es ist zwar nur ganz wenig Code, und man sollte doch nicht für alles so ein Fass aufmachen, aber es fühlt sich auch etwas komisch an. Wie ginge es besser?

Genau hier hilft SOLID [1]. Der Name SOLID ist ein Akronym für fünf Prinzipien, mit denen wir den Code nach und nach optimieren. Los geht es gleich mit dem, was wir als das Grundprinzip guter Architektur kennengelernt haben.

Single-Responsibility-Prinzip (SRP)

Eine Klasse sollte nur eine Aufgabe oder Verantwortlichkeit haben.

Das klingt einfacher, als es ist. Denn wenn einmal Code geschrieben wurde, ist es schwierig, Elemente auseinanderzuziehen. Welche Stellen in Listing 1 verletzen dieses Prinzip? Wir haben Datenbankzugriff, Steuerberechnung, Mahnungserstellung und so weiter, und das alles in einer Methode!

Offensichtlich eine Verletzung dieses Prinzips ist es immer, wenn das Laden oder Speichern von Daten dort vorgenommen wird, wo auch berechnet wird. Ein Beleg sollte keine *Save*-Methode haben, dazu gibt es ein *Beleg-Repository*, dem

Listing 2: Das Laden und Speichern wird abstrahiert

```
public interface IOffenePostenRepository
{
    IEnumerable<dynamic> GetOffenePosten();
    void InsertMahnung(string kunde, decimal betrag,
        decimal steuer);
    void UpdateOffenenPostenStatusUndSteuer(
        int id, decimal steuer);
}
```

der Beleg übergeben wird. Leicht erkennbar ist die Verletzung des Prinzips auch, wenn Hilfsaufgaben in die Methode „mal eben schnell“ reingenommen werden. Es sind ja nur fünf Zeilen Code, die das Monatsende eines Datums berechnen, da kann ich das doch auch in der Belegklasse reinnehmen ...

Was an dieser Stelle hilft, ist die Frage: Kann ich diesen Code auch woanders brauchen? Oder wie teste ich den Code, wenn ich die Werte statt von der Datenbank von einem Mock-up geliefert bekommen möchte? Das sind Indizien, dass der Code nicht in diese Methode gehört. Sollte eine Klasse mehr als hundert Zeilen Code haben, ist das auch ein sicherer Hinweis, dass hier etwas nicht stimmen könnte. Es hilft, wenn man gleich nach der Erstellung der Klasse einen Kommentar einfügt, wozu diese Klasse gebraucht wird, und dann jeden Code an diesem Kommentar misst.

Ist die Verletzung des Prinzips erkannt, geht es darum, wie man die Funktionen aus der Klasse herausbekommt. Das einfachste ist, wenn man aus den einzelnen Funktionen *Extensions* schreibt. Dann kann man

```
beleg.Valutadatum = beleg.Belegdatum.EndeDesMonats()
```

schreiben und die Funktion *EndeDesMonats()* bei jedem Datum verwenden.

Eine Klasse kann auch um Funktionalität angereichert werden, indem ihr *Services* zur Verfügung gestellt werden, die die Aufgaben erledigen. Beispielsweise so:

```
void CalcValutatdatum(IDatumsService service){
    beleg.ValutaDatum =
        service.GetEndeDesMonats(beleg.Belegdatum)
}
```

Zurück zu unserem Beispiel. Zuerst muss der Datenzugriff raus aus dem Code. Dazu eignen sich *Repositories*. Legen wir uns in Listing 2 ein Interface *IOffenePostenRepository* an.

Das *Repository* selbst in Listing 3 bezieht sich immer auf einen Datenspeicher, sodass es vollkommen in Ordnung ist, wenn in der Implementierung ein SQL-Zugriff stattfindet. Gegebenenfalls könnte man, wenn man mit einem ORM-Mapper arbeitet, diesen noch zwischen die Datenbankschicht und das *Repository* legen. Bei der Verwendung von *Entity Framework* lässt sich dann der *DataStore* immer ►

● **Listing 3: In dieser Implementierung werden die Werte aus der Datenbank geholt**

```

public class OffenePostenRepository :
    IOffenePostenRepository
{
    private readonly string _connectionString;

    public OffenePostenRepository(
        string connectionString)
    {
        _connectionString = connectionString;
    }

    public IEnumerable<dynamic> GetOffenePosten()
    {
        var offenePosten = new List<dynamic>();
        using (var conn = new SqlConnection(
            _connectionString))
        {
            conn.Open();
            using (var cmd = new SqlCommand("SELECT * FROM
                OffenePosten WHERE Status = 'offen'", conn))
            {
                using (var reader = cmd.ExecuteReader())
                {
                    while (reader.Read())
                    {
                        offenePosten.Add(new
                        {
                            Id = reader["Id"],
                            Kunde = reader["Kunde"],
                            Betrag = reader["Betrag"],
                            Land = reader["Land"]
                        });
                    }
                }
            }
            return offenePosten;
        }
    }

    public void InsertMahnung(
        string kunde, decimal betrag, decimal steuer)
    {
        using (var conn = new SqlConnection(
            _connectionString))
        {
            conn.Open();
            using (var cmd = new SqlCommand("INSERT INTO
                Mahnungen (Kunde, Betrag, Steuer) VALUES (
                    @Kunde, @Betrag, @Steuer)", conn))
            {
                cmd.Parameters.AddWithValue("@Kunde", kunde);
                cmd.Parameters.AddWithValue(
                    "@Betrag", betrag);
                cmd.Parameters.AddWithValue(
                    "@Steuer", steuer);
                cmd.ExecuteNonQuery();
            }
        }
    }

    public void UpdateOffenenPostenStatusUndSteuer(
        int id, decimal steuer)
    {
        using (var conn = new SqlConnection(
            _connectionString))
        {
            conn.Open();
            using (var cmd = new SqlCommand(
                "UPDATE OffenePosten SET Status = 'gemahnt',
                Steuer = @Steuer WHERE Id = @Id", conn))
            {
                cmd.Parameters.AddWithValue("@Id", id);
                cmd.Parameters.AddWithValue("@Steuer", steuer);
                cmd.ExecuteNonQuery();
            }
        }
    }
}

```

schön in das Repository injecten. Es ist also nicht wichtig, dass genau dieses Repository auf die Datenbank geht. Denn wir könnten es durch ein anderes austauschen. Dadurch entsteht die Flexibilität.

So lässt sich ein Repository schreiben, das immer die gleichen (Dummy-)Daten zurückliefert und so stabile Tests ermöglicht, ohne dass eine Datenbank dazu nötig ist. ChatGPT ist hier wunderbar, wenn es darum geht, Beispieldaten einzufügen. In [Listing 4](#) sehen Sie, dass ChatGPT auch kreativ bei der Namensfindung helfen kann. Denn nicht jeder Code muss von Ihnen kommen. Es reicht, wenn Sie die Architektur vorgeben und nach Ihren Vorgaben Code erzeugen lassen.

Exkurs: API-Controller

Als ein weiteres konkretes Beispiel für Single Responsibility soll uns ein API-Controller dienen, der Kommentare entgegennimmt. Was ist seine Aufgabe?

Streng genommen ist die einzige Aufgabe, Aufrufe über HTTPS entgegenzunehmen und das richtige Ergebnis zurückzugeben. Ein Aspekt ist die Sicherheit, also wer darf was. Man könnte vielleicht noch sagen, dass er die internen Objekte in DTOs konvertiert, oder besser, diese Aufgabe an die DTOs weiterreicht. Ein weiterer Aspekt ist das Logging.

Was gehört jetzt also wie in den Controller? Die Sicherheit kann man mit einem Attribut abdecken:

```
[Authorize(Role="Editor")]
```

Das Logging kann eine Middleware übernehmen, die vor und nach dem Aufruf eines jeden Controllers die Parameter und gegebenenfalls Fehler mitloggt. Beide Aspekte (Sicherheit und Logging) betreffen ja nicht nur den Kommentar-Controller, sondern jeden Controller.

Die Aufgabe, die richtigen Kommentare aus der Datenbank zu holen, kann ein Service übernehmen. Denn vielleicht brauche ich an anderer Stelle, wenn ich einen Artikel zurückgebe, auch wieder alle Kommentare dazu. Zudem kann er so separat vom Controller getestet werden.

Die Umwandlung der Entities in DTOs kann das DTO selbst übernehmen, indem es das Entity im Konstruktor auf-

nimmt. Oder man schreibt eine Extension *ToDto()*. Wichtig ist, dass nicht ein spezifischer Storage injected wird, sondern ein Interface *IDataStore*. So kann zum Testen ein Store verwendet werden, der Demodaten zurückgibt anstatt Werte aus der Datenbank.

Jede Klasse konzentriert sich auf die Aufgaben, die spezifisch für sie sind. Das gibt lesbaren und testbaren Code und weniger Fehler. Das [Listing 5](#) zeigt den Controller.

Open/Closed-Prinzip (OCP)

Software-Entitäten sollen offen für Erweiterungen, aber geschlossen für Modifikationen sein.

Das Problem ist, wie wir damit umgehen, wenn sich der Steuersatz ändert oder wenn ein weiteres Land dazukommt. ►

● Listing 4: Ein Repository, das Testdaten zurückliefert und somit getestet werden kann

```
public class MockOffenePostenRepository :
    IOffenePostenRepository
{
    private List<dynamic> offenePosten =
        new List<dynamic>();
    private List<dynamic> mahnungen =
        new List<dynamic>();

    public MockOffenePostenRepository()
    {
        // Erzeugen von 5 deutschen Kunden mit typischen
        // Namen
        offenePosten.AddRange(new List<dynamic>
        {
            new { Id = 1, Kunde = "Max Müller",
                Betrag = 500m, Land = "DE" },
            new { Id = 2, Kunde = "Sophia Schmidt",
                Betrag = 450m, Land = "DE" },
            new { Id = 3, Kunde = "Leon Bauer",
                Betrag = 600m, Land = "DE" },
            new { Id = 4, Kunde = "Emma Wagner",
                Betrag = 550m, Land = "DE" },
            new { Id = 5, Kunde = "Felix Meyer",
                Betrag = 650m, Land = "DE" }
        });

        // Erzeugen von 3 österreichischen Kunden mit
        // typischen Namen
        offenePosten.AddRange(new List<dynamic>
        {
            new { Id = 6, Kunde = "Lukas Huber",
                Betrag = 700m, Land = "AT" },
            new { Id = 7, Kunde = "Anna Gruber",
                Betrag = 750m, Land = "AT" },
            new { Id = 8, Kunde = "Tobias Lehner",
                Betrag = 800m, Land = "AT" }
        });
    }

    // Ein Schweizer Kunde mit einem typischen Namen
    offenePosten.Add(new { Id = 9, Kunde =
        "Nicolas Frei", Betrag = 900m, Land = "CH" });
}

public IEnumerable<dynamic> GetOffenePosten()
{
    return offenePosten;
}

public void InsertMahnung(
    string kunde, decimal betrag, decimal steuer)
{
    mahnungen.Add(new { Kunde = kunde,
        Betrag = betrag, Steuer = steuer });
    Console.WriteLine($"Mahnung eingefügt für: " +
        $"{kunde}, Betrag: {betrag}, Steuer: {steuer}");
}

public void UpdateOffenenPostenStatusUndSteuer(
    int id, decimal steuer)
{
    foreach (var posten in offenePosten)
    {
        if (posten.Id == id)
        {
            posten.Steuer = steuer; // Da 'dynamic' ver-
            // wendet wird, ist diese Zuweisung möglich,
            // aber in realen Szenarien sollte ein stärker
            // typisiertes Modell verwendet werden.
            Console.WriteLine($"Offenen Posten " +
                "aktualisiert: {id}, neue Steuer: {steuer}");
            break;
        }
    }
}
```

● Listing 5: Single Responsibility bei einem Controller

```
[Authorize(Role="Editor")]
[Route("[controller]")]
[ApiController]
public class CommentsController :
    DPSController<CommentsController>
{
    private readonly ICommentService _CommentService;
    public CommentsController(
        ILogger<CommentsController> logger,
        ICommentService commentService,
        IDataStore storage,
        IContext context) : base(logger, storage, context)
    {
        CommentService = commentService;
    }

    [HttpGet("GetAll")]
    public async Task<ActionResult<IEnumerable<
        CommentDto>>> GetAll([FromQuery]
        RequestGetSingle request)
    {
        var comments = await
            _CommentService.GetAllCommentsAsync(
                DataStore, request);
        return Ok(comments.Where(_ => _.Parent == null).
            Select(_ => new CommentDto(_)).ToList());
    }

    [HttpPost("Update")]
    public async Task<ActionResult<CommentDto>>
        Put(CommentUpdateDto dto)
    {
        var result = await _CommentService.AddComment(
            DataStore, dto);
        if (result == null) return NotFound();
        return result.ToDto();
    }

    [HttpPost("Insert")]
    public async Task<ActionResult<CommentDto>>
        InsertComment(CommentUpdateDto dto)
    {
        return await CommentService.AddComment(
            DataStore, dto).ToDto();
    }
}
```

● Listing 6: Die Steuerberechnung je nach „Strategie“

```
public interface ISteuerStrategie
{
    decimal BerechneSteuer(decimal betrag);
}

public class InlandSteuerStrategie : ISteuerStrategie
{
    public decimal BerechneSteuer(decimal betrag)
    {
        return betrag * 0.19M; // 19% Steuersatz
    }
}

public class AuslandSteuerStrategie : ISteuerStrategie
{
    public decimal BerechneSteuer(decimal betrag)
    {
        return betrag * 0M; // 0% Steuersatz
    }
}

// So würde es mit Strategy Pattern aussehen. Das
// wäre aber sehr „statisch“:

public class SteuerBerechner
{
    private readonly ISteuerStrategie _steuerStrategie;

    public SteuerBerechner(
        ISteuerStrategie steuerStrategie)
    {
        _steuerStrategie = steuerStrategie;
    }

    public decimal BerechneSteuer(decimal betrag)
    {
        return _steuerStrategie.BerechneSteuer(betrag);
    }
}

public static void MahnungInland()
{
    // Auswahl der Steuerstrategie basierend auf dem
    // Land oder anderen Kriterien
    var steuerStrategie = new InlandSteuerStrategie();
    // Kann dynamisch basierend auf Bedingungen gesetzt
    // werden
    var berechner = new SteuerBerechner(steuerStrategie);
}
```

Das Open/Closed-Prinzip gibt uns vor, dass wir am Code für das Mahnwesen nichts ändern sollten. Denn die Berechnung der Steuer muss von außen kommen, und der Code sollte erweiterbar sein, um bei Bedarf andere Berechnungslogiken zu ermöglichen.

Gehen wir etwa davon aus, dass wir zwei Fälle unterscheiden möchten: Inlandsbesteuerung und Auslandsbesteuerung. Dann können wir dazu zwei Strategien, wie man die Steuer berechnet, in jeweils einer eigenen Steuerstrategie umsetzen. Das Interface ist simpel:

```
public interface ISteuerStrategie
{
    decimal BerechneSteuer(decimal betrag);
}
```

Um jetzt die jeweils richtige Strategie anzuwenden, gibt es zwei Patterns: Das Factory Pattern und das Strategy Pattern. Beim Strategy Pattern wird von außen die Strategie geändert. In [Listing 6](#) sehen Sie, dass im Konstruktor die jeweilige Strategie übergeben wird und man dann mit dem Steuerrechner arbeiten kann.

In unserem Beispiel ist aber das Factory Pattern besser geeignet, denn je nach Land soll die richtige Strategie ermittelt, also zurückgegeben werden. Die statische Factory hat also nur eine Aufgabe: Ermitteln (und notfalls Instanzieren, deshalb der Name Factory) der richtigen Strategie.

Liskov-Substitution-Prinzip (LSP)

Objekte in einem Programm sollten durch Instanzen von Untertypen ohne Beeinträchtigung der Korrektheit des Programms ersetzt werden können.

Das klingt zunächst etwas abstrakt und schwer verständlich. Das LSP besagt, dass Objekte einer Basisklasse durch Objekte von Subklassen ersetzt werden können, ohne die Korrektheit des Programms zu beeinträchtigen. Es legt den Schwerpunkt auf die korrekte Vererbung, sodass die abgeleiteten Klassen die Basisfunktionalitäten nicht verändern oder deren Erwartungen verletzen.

In unserem Code findet das LSP keine Anwendung. Um es transparent zu machen, könnte man sich vorstellen, dass man eine Basisklasse hat, die Mahntexte erzeugt ([Listing 7](#)). Je nach Mahnstufe leitet eine Klasse von dieser Klasse ab und überschreibt die Standardeinstellung. Sie verschärft quasi den Ton. Man erwartet aber immer, dass die Methode *GeneriereMahnText* einen Text zurückliefert. Bei Mahnstufe 3 ist dieses Prinzip verletzt, da ein leerer String zurückgegeben wird. Die Erwartung an die Klasse wird also nicht erfüllt.

Hier vielleicht noch ein allgemeiner Satz zu guter Softwarearchitektur. Denn Sie werden sagen: „Das ist doch keine große Sache, wenn hier String-Empty zurückgegeben wird. Das ‚siehe‘ ich doch, wenn ich in die Klasse reinschaue. Das weiß ich doch, da ich sie selbst geschrieben habe. Ich bin nur eben noch nicht so weit, das wird später dann gemacht.“ Das Problem dabei ist: Wenn Sie eine Klasse verwenden, die Sie ►

```
var betrag = 100; // Beispielbetrag
var steuer = berechner.BerechneSteuer(betrag);
Console.WriteLine($"Steuer: {steuer}");
}

// Oder unter Verwendung einer Factory

public static class SteuerStrategieFabrik
{
    public static ISteuerStrategie
        GetSteuerStrategie(string land)
    {
        switch (land)
        {
            case "DE":
                return new InlandSteuerStrategie();
            case "AT":
                return new AuslandSteuerStrategie();
            case "CH":
                return new AuslandSteuerStrategie();
            default:
                throw new NotImplementedException($"Keine " +
                    "Steuerstrategie implementiert für: {land}");
        }
    }
}
```

```
}

public static void MahnungMitLandFactory()
{
    try
    {
        var land = "DE"; // Kann dynamisch gesetzt werden,
        // z.B. durch Benutzereingabe
        var steuerStrategie =
            SteuerStrategieFabrik.GetSteuerStrategie(land);

        var betrag = 100.0; // Beispielbetrag
        var steuer =
            steuerStrategie.BerechneSteuer(betrag);

        Console.WriteLine($"Für das Land {land}: Zu " +
            "zahlende Steuer auf {betrag} ist {steuer}.");
    }
    catch (NotImplementedException ex)
    {
        Console.WriteLine(ex.message);
    }
}
```


vorher selbst geschrieben haben, verfügen Sie über implizites Wissen, wie mit dieser Klasse umzugehen ist. Und genau das sollte nicht sein. Die Klasse ist dann nicht mehr selbsterklärend. Sie kann gegebenenfalls nicht für einen anderen Kontext verwendet werden, weil sie davon ausgeht, dass *LadeSprache()* immer vor *GeneriereTexte()* aufgerufen wird. Schlechtes Design also, da implizite Annahmen gemacht werden. Mein Merkspruch dazu ist: Gute Entwickler sind immer ein bisschen schizophr. Sie schreiben eine Klasse als Persönlichkeit 1 und verwenden sie dann als Persönlichkeit 2. Gute Architektur macht aus, dass das möglich ist: Klassen kann man verwenden, ohne zu wissen, wie sie konkret implementiert sind.

Interface-Segregation-Prinzip (ISP)

Kein Client sollte gezwungen sein, von ihm nicht genutzte Interfaces zu implementieren.

Wer sich noch einmal das Interface *IOffenePostenRepository* ansieht, stellt schnell fest, dass hier zwei Aspekte – einmal der Mahnlauf und das andere Mal die Ermittlung von offenen Posten – in einem Interface stecken und jede Klasse, die das implementiert, nicht nur die offenen Posten zurückgeben, sondern auch noch einen Mahnlauf implementieren muss. Man zwingt sie dazu. Und wenn sie das nicht tut, sondern ein leeres Array zurückliefert, dann widerspricht sie dem LSP, weil sie ein erwartetes Verhalten nicht umsetzt. Wie also damit umgehen?

Da Interfaces voneinander erben können, ergibt es Sinn, wenn pro Aspekt ein eigenes Interface geschrieben wird und die Interfaces dann später bei Bedarf zusammengefasst werden. Konkretes Beispiel: Eine Raumreservierung kann aktiv oder inaktiv sein. Und da sie eine zeitliche Spanne abdeckt, hat sie einen Start und ein Ende. Anstelle eines Interfaces, das Start, Ende und *IsActive* beinhaltet, kann man pro Aspekt ein Interface schreiben und beide dann kombinieren.

```
public interface IIsActive
{
    bool IsActive {get;}
}

public interface IDuration
{
    DateTime Start {get;}
    DateTime End {get;}
}

public interface IActiveDuration: IIsActive, IDuration
```

Wenn jetzt ein Raum kein Start und Ende hat, aber aktiv und inaktiv sein kann, dann muss er nur *IIsActive* implementieren und nicht das Interface *IActiveDuration*. Auf der anderen Seite können alle, die *IsActive* implementieren und kein Start und Ende haben, diese Erweiterung nutzen:

```
public static IQueryable<T> WhereIsActive<T>(this
    IQueryable<T> query) where T : IIsActive
{
    // Wende den Filter an
    return query.Where(item => item.IsActive);
}
```

Somit ließe sich ein Basis-Repository schreiben, das aufgrund einer allgemeinen Einstellung nur aktive oder alle Elemente zurückliefert. Oder eine Extension, die prüft, ob ein Termin innerhalb einer *IDuration* liegt oder eben nicht.

```
Reservierung.IsGeblockt(DateTime.Now)
```

Und nun wären wir wieder an einem interessanten Punkt: *DateTime.Now* ist der Tod jeder guten Architektur. Denn was

● Listing 7: Verschiedene Überschreibungen der Basisklasse

```
public abstract class MahnTextGenerator
{
    public virtual string GeneriereMahnText(
        decimal betrag)
    {
        // Grundannahme: Eine einfache Mahnung wird
        // generiert.
        return $"Sie haben eine offene Forderung in " +
            "Höhe von {betrag:C}.";
    }
}

public class Mahnstufe2TextGenerator :
    MahnTextGenerator
{
    public override string GeneriereMahnText(
        decimal betrag)
    {
        return base.GeneriereMahnText(betrag) +
            Environment.NewLine + "Aber bitte zackig!";
    }
}

public class Mahnstufe3TextGenerator :
    MahnTextGenerator
{
    public override string GeneriereMahnText(
        decimal betrag)
    {
        return string.Empty;
    }
}
```

● Listing 8: Ein Service-Container nimmt konkrete Instanzen auf und löst deren Abhängigkeiten

```

public class SimpleServiceContainer
{
    private readonly Dictionary<Type, Type>
        _typeMappings = new();

    public void Register<TService, TImplementation>()
    {
        _typeMappings[typeof(TService)] =
            typeof(TImplementation);
    }

    public object GetService(Type serviceType)
    {
        if (_typeMappings.TryGetValue(
            serviceType, out Type implementationType))
        {
            ConstructorInfo constructor =
                implementationType.GetConstructors().First();
            ParameterInfo[] constructorParameters =
                constructor.GetParameters();
            if (constructorParameters.Length == 0)
            {
                return
                    Activator.CreateInstance(implementationType);
            }
            else
            {
                object[] parameters =
                    new object[constructorParameters.Length];
                for (int i = 0; i <
                    constructorParameters.Length; i++)
                {
                    parameters[i] = GetService(
                        constructorParameters[i].ParameterType);
                }

                return constructor.Invoke(parameters);
            }
        }
        else
        {
            throw new InvalidOperationException(
                "No registered service for type " +
                serviceType.Name);
        }
    }

    public T GetService<T>()
    {
        return (T)GetService(typeof(T));
    }
}

```

genau *Now* ist, kann ich nicht von außen in die Klasse geben. Und somit kann sie nicht getestet werden. Sie liefert immer einen anderen Output. *DateTime.Now* verletzt das Prinzip, dass eine Methode, wenn ich sie mit den gleichen Parametern aufrufe, immer das gleiche Ergebnis liefern muss – auf gut Deutsch: Idempotenz (Idem = dasselbe). Zudem ist *Now* abhängig von der Zeitzone, in der ich mich bewege.

Besser wäre ein *DateTimeService*, den man injectet und der eine Methode *GetCurrentTime()* hat. Damit lässt sich wunderbar testen. Und gerade bei ERP-Anwendungen oder in der Finanzbuchhaltung ist der Tag, an dem ich buche, nicht immer der Tag, für den ich buche. Anders gesagt: Die Aufgabe, die aktuelle Zeit zu ermitteln, darf nicht von der Methode ausgeführt werden, die diese Aufgabe nicht hat. Klare Verletzung des Single-Responsibility-Prinzips, nicht wahr?

Dependency-Inversion-Prinzip (DIP)

Abhängigkeiten sollten auf Abstraktionen
und nicht auf Konkretionen beruhen.

Das bedeutet, dass Module höherer Ebene nicht von Modulen niedrigerer Ebene abhängen sollten, sondern beide von Abstraktionen. In unserem Beispiel sollte weder der Mahnlauf vom Repository noch das Repository vom Mahnlauf abhängig sein. Sondern beide sollten nur auf Abstraktionen des jeweiligen Dienstes zurückgreifen.

Dependency Inversion ist gar nicht immer leicht umzusetzen. Ein Trick ist, mit einem Service-Container zu arbeiten und

sich die jeweilige Implementierung aus dem Container zu holen. Vorher muss, je nach (Test-)Szenario oder (Live-)Umgebung, die jeweilige Instanz in den Container gelegt werden.

Wir haben ja gesehen, dass es ein Repository geben kann, das gegen die Datenbank geht, und eines, das Beispieldaten zurückliefert, um es zu testen. Je nachdem, was vorher in den Container gelegt wird, arbeitet dann das Mahnwesen (live) gegen die Datenbank oder (im Test) gegen einen Dummy-Service. Da bei einem Service-Container viel „Magic behind the scene“ von den Klassen von .NET erledigt wird, ist in [Listing 8](#) der Code eines Beispiel-Containers abgebildet, wie er in vereinfachter Form aussehen könnte. Er soll dem besseren Verständnis dienen.

Was passiert hier im Detail: Man kann eine Implementierung, also beispielsweise das *OffenePostenRepository*, als Implementierung des Interfaces *IOffenePostenRepository* registrieren. Es wird dabei nichts anderes gemacht, als dass ein Dictionary aufgebaut wird, in dem für jedes Interface eine Implementierung liegt. Wenn jetzt aber das Repository beispielsweise eine *IConfiguration* braucht, aus der es sich die Verbindung zur Datenbank holt, dann reicht es, wenn man im Konstruktor das Interface angibt. Also:

```

public OffenePostenRepository(IConfiguration config)
{
    _connectionString = config.ConnectionString;
}

```

● Listing 9: Der Mahnlauf und die Verwendung mit einem Service-Container

```

public interface IMahnlauf
{
    void Execute();
}

public class Mahnlauf : IMahnlauf
{
    private readonly ILogger _Logger;
    private readonly IOffenePostenRepository _Repository;
    private readonly ISteuerStrategieFabrik _Factory;

    public Mahnlauf(ILogger logger,
        IOffenePostenRepository repository,
        ISteuerStrategieFabrik factory)
    {
        _Logger = logger;
        _Repository = repository;
        _Factory = factory;
    }

    public void Execute()
    {
        var offenePosten = _Repository.GetOffenePosten();
        foreach (var posten in offenePosten)
        {
            var steuer = _Factory.GetSteuerStrategie(
                posten.Land).BerechneSteuer(posten.Betrag);
            _Logger.Log(LogLevel.Debug, $"Erzeuge Mahnung " +
                "für: {posten.Kunde}, zu zahlen: * +
                \"{posten.Betrag + steuer}\"");
            _Repository.InsertMahnung(posten.Kunde,
                posten.Betrag, steuer);
            _Repository.UpdateOffenenPostenStatusUndSteuer(
                posten.Id, steuer);
        }
    }
}

```

In dem Moment, in dem man sich vom Service-Container die konkrete Implementierung für *IOffenePostenRepository* holt, schaut dieser nach, welche Parameter im Konstruktor liegen, holt sie aus seinem eigenen Dictionary und kümmert sich so um die Auflösung der Abhängigkeiten.

Baut man einen Container für einen Datenbankzugriff, sollte man vorher eine Implementierung des *IConfiguration*-Interfaces in den Container legen. Ansonsten ist das nicht nötig. Bei der Programmierung selbst macht man sich keine Gedanken, woher die konkrete Instanz kommt. Wichtig ist nur, dass man sie eben benötigt. Ein solcher Ansatz bietet viel Freiraum, um Code, der nicht in die Methode gehört, in Services auszulagern.

In Listing 9 sehen Sie den kompletten Code, wie er mit einem Service-Container verwendet wird. Das Schreiben von Tests wird so deutlich einfacher. Ebenso die Steuererhöhung. Oder die Aufnahme weiterer Länder. Mit einem solchen Container können wir komplexe Verschachtelungen leicht lösen.

Hier der Aufruf des Programms ohne Dummy-Daten:

```

public static void Main2()
{
    var serviceContainer = new SimpleServiceContainer();
    serviceContainer.Register<IOffenePostenRepository,
        OffenePostenRepository>();
    serviceContainer.Register<ISteuerStrategieFabrik,
        SteuerStrategieFabrik>();
    serviceContainer.Register<IMahnlauf, Mahnlauf>();
    var mahnlauf =
        serviceContainer.GetService<IMahnlauf>();
    mahnlauf.Execute();
}

```

Ein Test lässt sich jetzt leicht ausführen, indem man das *MockOffenePostenRepository* verwendet.

Die Verwendung eines Service-Containers erscheint in unserem Beispiel vielleicht etwas übertrieben. Aber wenn Sie eine größere Anwendung schreiben, dann verwenden Sie natürlich für alle Ihre Services den gleichen Container, und dann ist er schon vorhanden. Er wird mit .NET sowieso mitgeliefert. Sie werden sich wundern, um wie viel besser wartbar der Code wird, wenn Sie einmal mit einem Service-Container gearbeitet haben.

Fazit

Sie haben mit SOLID die wichtigsten Prinzipien kennengelernt, um Software mit guter Architektur zu bauen. Gute Architektur ist, wenn eine Wand nur eine Wand ist, ein Boden nur ein Boden und ein Dach nur ein Dach. Sobald man im Boden Fenster findet, stimmt etwas nicht. Und mit der Zeit bekommt man einen Blick dafür. ■

[1] SOLID und die Prinzipien objektorientierten Designs,
www.dotnetpro.de/SL2504-05SOLID1



Dipl.-Theol. Bernhard Pichler

war 20 Jahre 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

A2504-05SOLID

MANAGEMENT UND FÜHRUNG

Der Wert von Zielen

Bei der Definition von Zielen scheint der Graben zwischen theoretischen Veröffentlichungen und gelebter Praxis besonders groß zu sein. Warum ist das so, und was können wir sinnvoll dagegen tun?

Wenn wir im Spätherbst im Wald spazieren gehen, kann es sein, dass wir vor lauter Blättern den Weg nicht mehr erkennen können. Um dann auf dem Weg zu bleiben oder ihn schnell wiederzufinden, wenn wir von ihm abgekommen sind, ist es hilfreich, das Ziel oder zumindest ein Zwischenziel auf unserem Weg auszumachen. Können wir zum Beispiel einen bestimmten Baum anpeilen, werden wir den Weg sicherlich finden. Ohne entsprechende Landmarken werden wir erst nach großen Umwegen am Ziel ankommen.

Der Sinn von Zielen

Dieses simple Beispiel zeigt den Sinn auf, der darin liegt, Ziele zu setzen. Wir möchten irgendwo ankommen, doch der Weg ist unklar. Über ein konkretes Ziel kann nun ein gangbarer Weg gefunden werden. Vielleicht können sogar mehrere Wege gefunden werden, die gegeneinander abgewogen werden können. So oder so: Wir vermeiden Umwege und haben eine Basis für unsere Planungen.

Gerade wenn es darum geht, das Handeln einer Gruppe auszurichten, sind Ziele das Mittel der Wahl. Jede einzelne Person kann sich daran orientieren. Treten Störungen oder Hindernisse auf, können Lösungen dafür entwickelt werden, um dennoch ans Ziel zu gelangen. Diese Trennung zwischen Lösung(en) und Ziel ist essenziell, wenn wir mit Zielen arbeiten wollen. Ein einfaches Beispiel soll dies verdeutlichen.

Angenommen, mein Lebenstraum in jungen Jahren wäre es gewesen, an der Olympiade teilzunehmen, so ist der offensichtliche Weg zum Ziel, in einer Sportart herausragende Leistungen zu erbringen. Als Sprinter müsste ich so um die 10 Sekunden über 100 Meter laufen, um sicher dabei zu sein. Gelingt mir das jedoch trotz intensiven Trainings nicht, bleibt mir der Weg ins Olympische Dorf dennoch nicht verschlossen. Als Sportler kann ich versuchen, mich für die 4x100-Meter-Staffel zu qualifizieren. Ich müsste dann nur der Viertbeste sein. Wenn ich es als Athlet dennoch nicht schaffe, so kann ich zum Beispiel als Trainer, Funktionär oder Kampfrichter mein Ziel immer noch erreichen. Es gibt verschiedene Wege zum Ziel.

Vorgaben sind keine Ziele

An dem Beispiel erkennen wir noch einen anderen, wichtigen Unterschied. Eine Qualifikationsnorm, wie sie für Olym-



pische Spiele existiert, definiert Vorgaben, die zu erfüllen sind, um sicher teilnehmen zu dürfen. Eine Vorgabe stellt eine Rahmenbedingung dar, die es zu erfüllen gilt, um das Ziel zu erreichen.

Eine Vorgabe entspricht also eher einer Rahmenbedingung für den Weg zum Ziel. Leider sind manche Vorgaben so herausfordernd, dass das Ziel in den Hintergrund tritt. Um die Vorgabe zu erfüllen, wird der Trainingsplan darauf ausgerichtet, die Höchstleistung auf einem Qualifizierungswettkampf zu erreichen und damit zeitlich gesehen deutlich vor den Olympischen Spielen. Dort fällt dann die Leistung wieder schwächer aus. Bekannt ist dieses Phänomen als „Fluch des ‚Sports Illustrated‘“. Häufig kamen vor den Olympi- ►

Bild: Diklo Klaisataporn / Shutterstock

schen Spielen die Besten aus der Qualifikation auf das Titelbild des Sportmagazins, um dann bei Olympia doch nicht Gold zu holen. Die Kombination aus Trainingsplan und dem statistischen Effekt der Schwankungen um einen Mittelwert erklären diesen vermeintlichen Fluch.

Solche Effekte zwischen Zielen und Vorgaben lassen sich auch auf Softwareentwicklungsprojekte übertragen. Vorgaben ergeben sich häufig aus „politischen“ Interessen. Es muss mit einem Team aus einem anderen Standort zusammengearbeitet werden, da dort gerade die Auslastung zu gering ist, oder eine Lieferung muss unbedingt noch im Juni geliefert werden, obwohl die Software noch nicht ausreichend stabil ist, weil der nicht unerhebliche Bonus eines Bereichsleiters davon abhängt. Ich denke, viele haben so etwas Ähnliches bereits erlebt und unter den Folgen der Vorgaben gelitten. Ich glaube auch, dass viele technische Schulden genau darin ihren Ursprung haben.

Was macht Ziele oft so problematisch?

Damit wir sinnvoll Ziele definieren können, bedarf es eines stabilen Umfelds. Ziele sollen ja SMART sein. Diese Beschreibung geht auf Peter F. Drucker aus den 1970er-Jahren zurück und wurde mit der Zeit immer wieder leicht angepasst [1]:

- Spezifisch, also möglichst konkret formuliert
- Messbar, um in kurzen Intervallen regelmäßig zu prüfen, wie nah wir an das Ziel bereits herangekommen sind (oder auch nicht)
- Achievable (erreichbar), also von allen Beteiligten als ansprechend und für erstrebenswert erachtet
- Realistisch, relevant und angemessen, also mit ausreichender Anstrengung erreichbar
- Terminiert, also als Meilenstein mit einem möglichst konkreten Zieldatum formuliert

Damit wir das umsetzen können, brauchen wir eine ausreichende Stabilität unserer Ziele und der Rahmenbedingungen. Andernfalls würden wir die Ziele den veränderten Gegebenheiten permanent anpassen, was ihren Wert verblasen ließe. Die unterschiedlichen Ziele müssen des Weiteren auf jeder Hierarchieebene aufeinander abgestimmt sein. Diese Zusammenhänge werden im Business Motivation Model (BMM) übersichtlich dargestellt (Bild 1) [2]:

- Die Vision beschreibt das ultimative, womöglich unerreichbare Ziel, das sich ein Unternehmen setzt.
- Eine Zielstellung (englisch: Goal) führt die Vision weiter aus. Zielstellungen werden in Ziele umformuliert, die erreicht werden müssen, um die Vision umzusetzen.
- Ein Planziel (englisch: Objective) ist ein messbares, erreichbares und zeitlich begrenztes Ziel.
- Die Mission beschreibt das andauernde Verhalten eines Unternehmens, mit dem die Vision erreicht werden soll.
- Die Strategie unterstützt eine Zielstellung.
- Eine Taktik unterstützt ein Planziel.

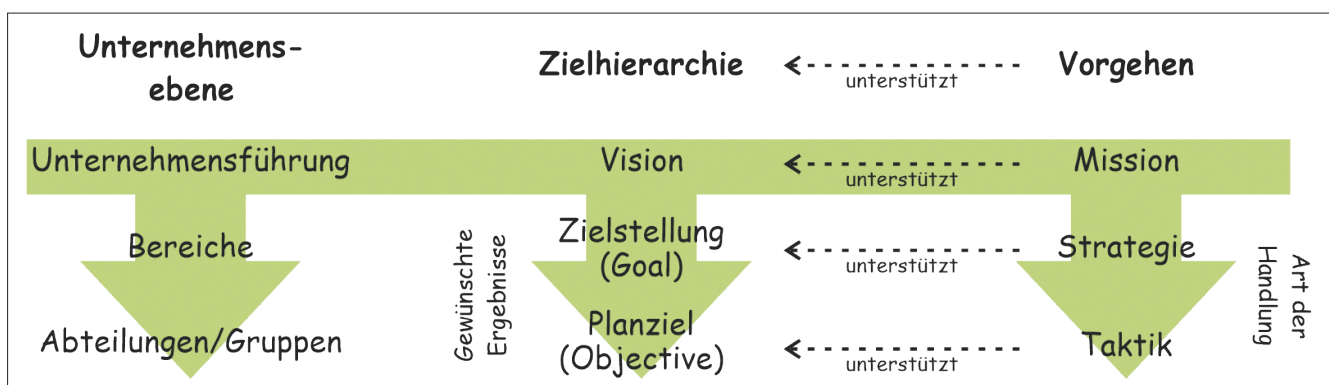
Ein solches Zielgeflecht ist ein komplexes Netz aus unterschiedlich abstrakten Elementen, aus dem sich auf den einzelnen Organisationsebenen konkrete Handlungen ableiten lassen. Das ist alles andere als trivial. Eine strategische Stabilität ist für den Erfolg, also die taktische und strategische Zielerreichung, unerlässlich. Ansonsten entstehen auf allen Ebenen Blindleistung und Ergebnisse, die kaum Wert liefern.

Ziel ist nicht gleich Maßnahme

Gerade in der IT werden auf die Frage nach den Zielen oft Maßnahmen genannt. Diese Verwechslung schränkt das Arbeiten mit Zielen stark ein, da sie die Lösung vorwegnimmt und die Frage nach dem Warum nicht beantwortet.

„Unser Ziel ist die Ablösung der aktuellen XY-Datenbank durch die ABC-Datenbank, die unser neuer Firmenstandard ist.“ Solche oder ähnliche Aussagen haben wir sicherlich bereits gehört oder vielleicht selbst verwendet. Doch ist das so richtig? Handelt es sich nicht eher um eine Maßnahme? Maßnahmen sind Aktivitäten, die wir durchführen, um ein Ziel zu erreichen. Den Unterschied können wir erkennen, wenn wir uns nach dem Warum fragen.

Warum wechseln wir auf die Standard-Datenbank? Warum haben wir überhaupt eine Standard-Datenbank? Was wollen wir damit auf welcher Zielebene erreichen? Durch die Vereinheitlichung möchten wir vielleicht Wartungs- und Betriebskosten reduzieren. Das wäre ein typisches Planziel. Dieses und andere ähnlich gelagerte Ziele gehören vielleicht zur Strategie, die Wirtschaftlichkeit einer Inhouse-IT zu verbessern, um im Vergleich zu Fremdanbietern konkurrenzfähig zu bleiben. Die Zielstellung wäre dann die Konkurrenzfähigkeit



Im Business Motivation Model werden Ziele entlang der Organisationsebenen strukturiert (Bild 1)

gegenüber externen Dienstleistern. Diese Zielstellung unterstützt eine Vision ähnlich wie „Wir sind der dauerhafte, anerkannte und renommierte IT-Partner unseres Mutterkonzerns!“

Nordstern oder OKR?

Werden wir etwas konkreter und schauen uns zwei weit verbreitete Konzepte zur Zielgestaltung genauer an. Zum einen können wir eine Zielhierarchie, wie bereits angedeutet, hierarchisch herunterbrechen. Zum anderen können wir mit einem generellen Ziel als Leitlinie (True North beziehungsweise Nordstern [3]) arbeiten (Bild 2).

OKR (Objectives and Key Results)

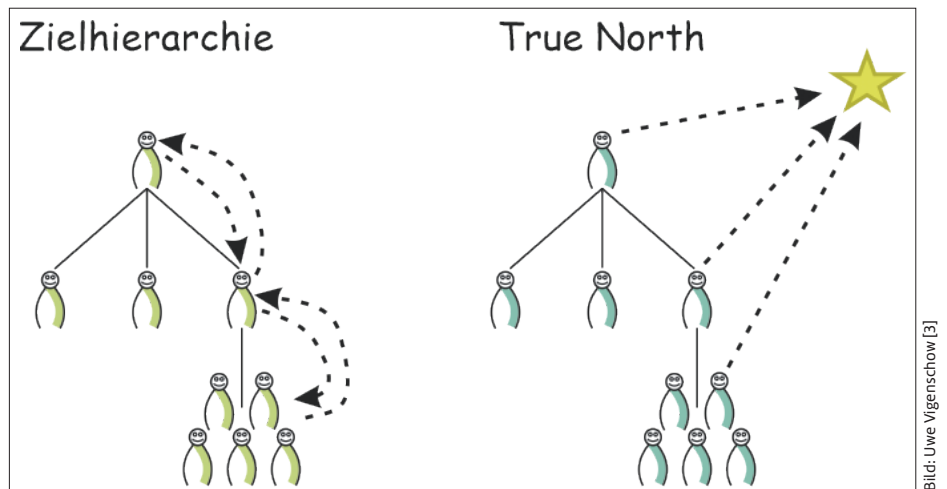
wurde bereits in den 1970er-Jahren bei Intel entwickelt und danach vor allen Dingen durch Google immer weiter aktualisiert. Die unternehmensweiten Ziele werden dabei in ein Zielvereinbarungssystem hierarchisch heruntergebrochen:

- **Ziel/Motivation:** Was will ich erreichen und warum?
- **Kenngroße/Überprüfungskriterium:** Was ist dann neu oder anders? Und wie kann ich das möglichst objektiv überprüfen?

Der Fokus wird dabei, anders als in anderen Zielvereinbarungssystemen, die eher top-down individuelle Jahresziele festlegen beziehungsweise vorgeben und an Boni gekoppelt sind, auf die Zusammenarbeit, gegenseitige Unterstützung und gemeinsam definierte Ziele gelegt. Diese Ziele sind transparent für alle Mitarbeitenden, gegenseitig top-down und bottom-up und vor allen Dingen horizontal, also zwischen den Abteilungen oder Bereichen ausgehandelt. Auch liegt der zeitliche Fokus deutlich kürzer als in anderen Zielvereinbarungssystemen bei typischerweise drei Monaten beziehungsweise vier Iterationen respektive Sprints. Auf keinen Fall sind OKR an ein Belohnungssystem gekoppelt [4].

OKRs ersetzen keine Vision und auch keine strategischen Ziele einer Organisation. Dafür bedarf es anderer Konzepte. Die Methode ist situativ und pragmatisch. Sie fokussiert nicht auf reinen Output wie zum Beispiel Lines of Code, sondern legt den Fokus auf den Outcome, also den aktiven Wertbeitrag, den eine Organisation für ihre Kunden leistet. OKR lassen sich damit auch gut auf Sprintziele abbilden.

OKR liegen also eher auf der taktischen Ebene. Sie können einen



Hierarchisch heruntergebrochene Ziele (links) und eine generelle Leitlinie (True North) als Ziel (rechts) (Bild 2)

Bild: Uwe Vigenschow [3]

Veränderungsprozess unterstützen, aber aufgrund ihrer vergleichsweise kurzen Zeithorizonte keinen Veränderungsprozess ersetzen. Dafür brauchen wir ein zweites Modell.

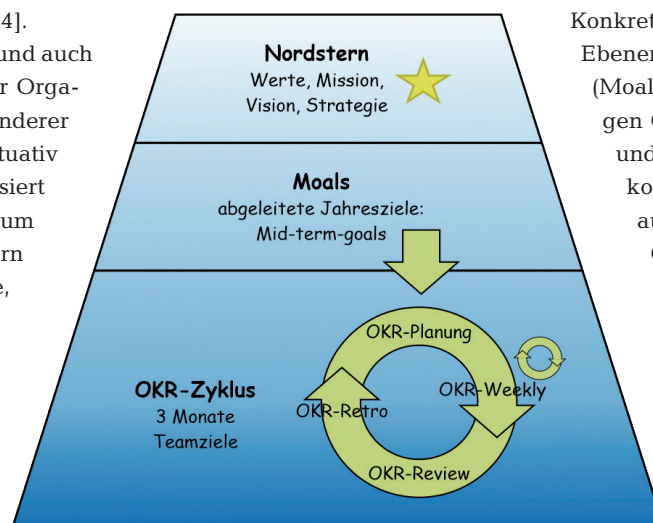
Beim Nordstern beziehungsweise True North geht es darum, einem kontinuierlichen Veränderungsprozess eine Richtung zu geben. Dieses übergreifende Ziel entspricht häufig eher einer Vision und ist als Ziel in den meisten Fällen nur unvollständig zu erreichen. Der Nordstern gibt die gemeinsame Richtung vor. Alle laufenden und geplanten Maßnahmen auf allen Hierarchieebenen müssen in einer Form auf dieses gemeinsame Ziel hinwirken. Konsequenzen aus diesen Maßnahmen wie zum Beispiel ein erhöhter Aufwand werden im Rahmen eines Lernprozesses in Kauf genommen. Durch flankierende Maßnahmen werden die negativen Effekte dann wieder minimiert, also zum Beispiel die Effizienz neuer Prozesse erhöht [1].

Konzepte im Zusammenspiel

Beide Konzepte ergänzen sich gemeinsam, sodass übergreifend und langfristig eine ideale Vision als Nordstern die Richtung für alle vorgibt.

Konkret werden die Ziele dann auf den Ebenen gemäß OKR in Jahresziele (Moals) und dann in den dreimonatigen OKR-Zyklus heruntergebrochen und auf diesem Weg immer weiter konkretisiert (Bild 3). Die Antwort auf die Frage „Nordstern oder OKR?“ lautet daher: Beides!

OKR hat einen Zyklus ähnlich dem aus Scrum bekannten Vorgehen aus Planung, auf einzelne Wochen iterativ heruntergebrochene Steuerungszyklen, Review und Retrospektive. Daher lassen sich beide Prozesse gut inte- ►



Nordstern und OKR im Zusammenspiel (Bild 3)

gieren und aneinander anpassen. Drei bis vier Sprints passen in einen OKR-Zyklus.

Wie kann so etwas konkret aussehen? Nehmen wir als Beispiel für eine Vision als Nordstern die Aussage: „Wir liefern nur noch fehlerfreie Software zu unseren Kunden!“ Es ist offensichtlich, dass das aller Wahrscheinlichkeit nach einem nicht erreichbaren Ideal entspricht. Dennoch kann es wertvoll sein, von einem Ideal auszugehen. Was bedeutet „zu unseren Kunden“? Wir können den Nordstern folgendermaßen umformulieren: „Im Anwendungsbereich der Kunden treten keine Fehler auf!“ Wir können es sogar noch weiter fokussieren: „Die Ziele unserer Kunden mit unserer Software werden stabil, dauerhaft und nachhaltig erreicht.“ Marginale Fehler, die sich vielleicht noch in unserer gelieferten Software befinden, spielen dann keine Rolle mehr.

Daraus lassen sich eine Reihe konkreter Ziele als Moals (Mid-term goals) ableiten, die die Anforderungsanalyse, das Requirements Engineering, den Entwicklungsprozess, die Teststrategie und die Lieferprozesse sowie die Ausbildung unserer Mitarbeitenden betreffen. Wir können beispielsweise über Personas oder definierte Business-Prozesse den Intended Use für unsere Software definieren und daran Entwicklungs- und Testprozesse ausrichten. Dabei kann auch durch gezieltes Weglassen die Komplexität unserer Lösung reduziert werden, indem zum Beispiel bestimmte Szenarien ausgeschlossen werden. Auf diese Weise wird der Fokus unserer Entwicklung immer klarer und kann in OKR-Teamziele heruntergebrochen werden.

Zwischenfazit

Wir kennen nun die Bedeutung von Zielen und wissen, wie das resultierende Zielgeflecht aussieht und mit welcher Kombination aus visionärem Ziel und konkreter Zielvereinbarung die Mitarbeitenden einer Organisation lang-, mittel- und kurzfristig auf ein gemeinsames Ziel ausgerichtet werden können. Das ist ein anspruchsvolles Unterfangen, doch es erscheint es mit etwas Energie gut umsetzbar. Und vielen Organisationen gelingt das auch, wenn vielleicht nicht perfekt, dann zumindest in brauchbarer Weise. Doch bei Weitem nicht alle bekommen das gut hin. Was geht hier schief?

Voraussetzungen für den Erfolg

Damit ein solches verschachteltes und aufeinander aufbauendes Zielgeflecht wirksam werden kann, bedarf es einer ausreichenden Stabilität auf strategischer Ebene. Solange es auf dieser Ebene Unklarheit oder wechselnde, konträre Aussagen gibt, kann ein Zielgeflecht nur mühsam und mit viel Aufwand aktuell und passend gehalten werden. Da ein solcher Aufwand unter diesen Rahmenbedingungen meist ver-

mieden wird, zerfällt das Zielgeflecht in einzelne, nicht mehr gut zusammenpassende Teile.

Außerdem bedarf es einer hohen Selbstdisziplin aller Beteiligten, um sich regelmäßig aus dem Tagesgeschäft herauszuziehen und die Ziele gemeinsam abzustimmen und deren Erreichungsgrad zu bewerten. Ohne einen hohen Grad an methodischem Arbeiten wird das Arbeiten mit Zielen schnell zerplatzen wie eine Seifenblase.

Der dritte Erfolgsfaktor ist die Unabhängigkeit des Zielvereinbarungssystems von dem Belohnungssystem einer Organisation. Sobald die Zielerreichung mit individuellen Zielen verknüpft wird, treten die gemeinsamen Ziele in den Hintergrund. Ein weiteres Problem ist die Laufzeit der Belohnungssysteme. Meist gehen diese über ein Jahr. Damit liegen wir jedoch bei einer viermal längeren Laufzeit als für OKR. Das passt nicht zusammen. Im schlimmsten Fall veraltet das Ziel aus dem Belohnungssystem und passt nicht mehr zu den angepassten Zielen im OKR. Solche Zielkonflikte sind hausgemacht und zu vermeiden.

Das in diesem Artikel dargestellte Zielgeflecht kann und darf nicht mit einem Belohnungssystem gekoppelt werden!

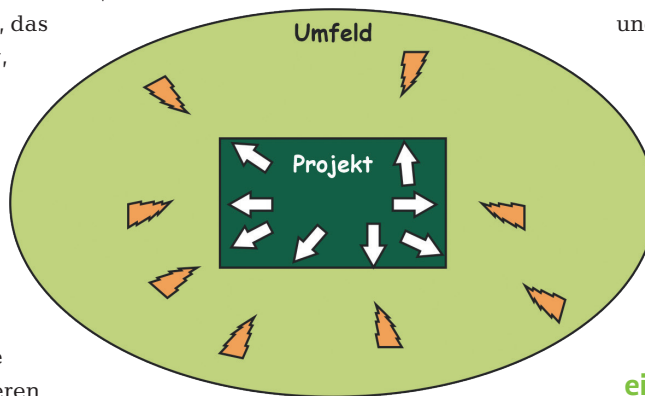
Reifegrad einer Organisation

Für mich folgt daraus, dass die Fähigkeit einer Organisation, ein erfolgreiches Zielgeflecht ähnlich dem in diesem Artikel skizzierten Modell umzusetzen, ein Indikator für den Reifegrad der Organisation im Sinne der Qualität der Ausbildung ihrer Mitarbeitenden und der gelebten Prozesse beziehungsweise Abläufe ist. Unternehmen, die erfolgreich mit Zielgeflechten arbeiten, können ihre Strategie wirksam umsetzen und werden dadurch am Markt noch erfolgreicher beziehungsweise können auch in unruhigen Zeiten ihren Markterfolg sichern.

Den Organisationen, die zum Beispiel OKR einsetzen, geht es also nicht deswegen gut, weil sie eine bestimmte Methode einsetzen, sondern, weil sie reif genug sind, diese erfolgreich einsetzen zu können. Wenn wir die Kombination aus Nordstern und OKR (oder vergleichbare Konzepte) in unserer Organisation umsetzen können, haben wir einen gewissen Reifegrad erreicht. Wenn wir das bereits längere Zeit tun, umso besser. Wenn wir jedoch immer wieder Schwierigkeiten mit der Umsetzung von Konzepten wie Nordstern, OKR, Balanced Scorecards et cetera. haben, dann müssen wir zuerst die Basis dafür legen. Klarheit in unserer Strategie und die Qualifizierung unserer Mitarbeitenden sind der Einstieg für den Weg hin zu einem methodischen Arbeiten.

Voraussetzungen schaffen

Das ist leichter gesagt als getan. Wie können wir uns ganz konkret im beruflichen Alltag in eine Richtung entwickeln,



Ashbys Gesetz, vereinfacht dargestellt (Bild 4)

in der wir lernen, methodischer zu arbeiten und uns schrittweise an Zielen und Zielgeflechten zu orientieren? Dazu betrachten wir die beiden grundlegenden Störfaktoren, die auf ein Team einwirken können. Zum einen kann unser Umfeld einfach zu dynamisch sein, als dass ein Ziel länger als wenige Wochen Bestand hat. Zum anderen formulieren wir vielleicht die falschen Ziele.

Arbeiten wir in einem Umfeld, in dem gleichsam im Wochentakt Anforderungsänderungen auf uns einprasseln, ist es mühsam und fast wertlos, mit konkreten Zielen im Bereich von ein bis drei Monaten zu arbeiten. In einem solch hochkomplexen, ja beinahe bereits chaotischem System können wir nur überleben, indem wir unsere eigene Struktur auf die gleiche Ausprägung der Komplexität bringen wie die Störungen aus unserem Umfeld: Wir wenden Ashbys Gesetz an (Bild 4). Über die hohe Flexibilität sowie die Wirk-, Handlungs- und Kommunikationsmöglichkeiten eines selbst organisierten Projektteams (weiße Pfeile) können wir bestmöglich auf die Anforderungen eines komplexen Umfelds (orange Blitze) reagieren [5]

Um das zu erreichen, brauchen wir Teammitglieder mit einer hohen Flexibilität, die sich schnell um eine auftretende Störung kümmern können. Das hat einerseits mit dem Mindset der Personen im Team zu tun und andererseits mit ihrer Ausbildung und Qualifikation in technischer und kommunikativer Hinsicht. Unsere Prozesse werden eher adaptiv sein und das Lernen im Team unterstützen. Scrum mit seinem integrierten Inspect-and-adapt-Zyklus kann hier die passende Wahl sein. Die Teams arbeiten dazu weitgehend selbstorganisiert, um sofort handlungsfähig zu sein und nicht auf Managemententscheidungen warten zu müssen.

Offene Fragen anstelle von Zielen

Wenn es uns auch mit selbst organisierten Teams (noch) nicht gelingt, ein funktionierendes Zielgeflecht aufzubauen und zu konkretisieren, kann es an falschen beziehungsweise wenig hilfreichen Zielen liegen oder an falschen Annahmen, auf deren Grundlage wir Ziele formulieren.

Wenn es uns noch nicht gelingt, wirkungsvolle Ziele in klare Worte zu fassen, kann es als Vorstufe helfen, offene Fragen zu formulieren. Offene Fragen sind Fragen, die sich nicht sinnvoll mit Ja oder Nein beantworten lassen. Wir kennen sie auch als sogenannte W-Fragen, denn sie beginnen mit Wörtern wie Warum, Weshalb, Wo, Wie, Wann und so weiter [6].

Offene Fragen öffnen den Raum und helfen uns, nächste Lernschritte zu definieren. Wir lenken damit unseren Fokus auf verschiedene Weise:

- **Detailebene**, zum Beispiel „Wie geht das genau?“, um konkreter zu werden, oder „Wie passt das ins Gesamtbild?“, um abstrakter zu werden.
- **Umfang**, zum Beispiel „Was gehört noch dazu?“, um in die inhaltliche Breite auf einer Detailebene zu gehen.
- **Zeit**, zum Beispiel „Was haben wir früher gemacht, um ein ähnliches Problem zu lösen?“ oder „Was genau soll in Zukunft anders sein?“
- **Vom Problem hin zur Lösung**, zum Beispiel: „Woran merkst du, dass das Problem gelöst ist?“

Die daraus resultierenden, konkreter werdenden Fragen aus dem Team können wir dann sammeln und in eine Reihenfolge bringen, nach der wir ihre Beantwortung angehen wollen. Wir lernen von Iteration zu Iteration mehr über die vorliegenden Probleme, ihre Lösungen und die Hindernisse, die uns im Weg stehen. So erreichen wir Schritt für Schritt ein Level, auf dem wir dann probieren können, sinnvolle Ziele zu formulieren und über Ziele zu lenken.

Fazit

Der Weg ist in der Softwareentwicklung nicht das Ziel, sondern dessen Wirkung. Auch wenn wir nicht wissen können, was in fünf Jahren sein wird, brauchen wir dennoch eine Orientierung für das Handeln einer Organisation. Die Vision, der wir dabei folgen, hat daher notwendigerweise eher abstrakten und idealen Charakter. Das konkrete Herunterbrechen in einzelne Teile bis hin zu einem zeitlichen Horizont von maximal drei Monaten führt zu konkreten, messbaren Zielen, die uns Schritt für Schritt näher an die Umsetzung unserer Vision führen.

Durch diese Kombination, wie sie Nordstern und OKR bieten, bleiben wir flexibel genug, um mehrmals im Jahr auf Veränderungen reagieren zu können, und schaffen gleichzeitig einen Rahmen für die Softwareentwicklung, der für zwei bis vier Sprints beziehungsweise Iterationen stabil ist. Gleichzeitig bildet der Grad der Umsetzung eines solchen Zielgeflechts über alle Abstraktions- und Organisationsebenen einen Indikator für den Reifegrad einer Organisation. Unternehmen sind erfolgreich, weil sie in der Lage sind, anspruchsvolle Zielgeflechts erfolgreich umsetzen und nutzen zu können. ■

- [1] Sandra Reupke-Sieroux, Stefan Roock, Henning Wolf, *Agile Leadership*, dpunkt.verlag, 2020, ISBN 978-3-86490-696-1
- [2] Uwe Vigerschow, Björn Schneider, Ines Meyrose, *Soft Skills für IT-Führungskräfte und Projektleiter*, dpunkt.verlag, 3. Auflage, 2016, ISBN 978-3-86490-395-3
- [3] Uwe Vigerschow, *Lernende Organisationen*, dpunkt.verlag, 2021, ISBN 978-3-86490-798-2
- [4] Julia Dellnitz et al., *Fokus – Das Handbuch für Product Owner*, Rheinwerk, 2023, ISBN 978-3-8362-9269-6
- [5] Uwe Vigerschow, *APM – Agiles Projektmanagement*, dpunkt.verlag, 2015, ISBN 978-3-86490-211-6
- [6] Uwe Vigerschow, Björn Schneider, Ines Meyrose, *Soft Skills für Softwareentwickler*, dpunkt.verlag, 4. Auflage, 2019, ISBN 978-3-86490-697-8



Uwe Vigerschow

ist Senior Consultant bei der SWAGLab GmbH und freiberuflicher Fachautor. Er bringt dabei seine Erfahrung aus über 30 Jahren als Softwareentwickler, Berater und Führungskraft für den Erfolg seiner Kunden und Leser:innen ein.

uwe.vigerschow@swaglab.rocks

dnpCode

A2504-05Ziele

SPIELERISCH PROGRAMMIEREN LERNEN

Die Lego-ToyPad-Challenge

Interaktive Spielsteine, Discofieber und Zahlenraten in einer neuen Dimension: Hier beginnt ein aufregender Einstieg in die Ausbildung zum Anwendungsentwickler.

Der Schritt, eine neue Ausbildung zu starten, ist oft mit viel Aufregung verbunden: neue Kollegen, spannende Technologien und die ersten Herausforderungen. Auch für die beiden Verfasser dieses Artikels war das nicht anders. Während der eine bereits eine vorherige Ausbildung als Systemintegrator abgeschlossen hatte und dadurch erste Programmiererfahrungen mitbrachte, betrat der andere als Quereinsteiger völliges Neuland. Doch wie schafft man es, beide Auszubildende auf einer Ebene abzuholen – ohne den einen zu unterfordern und den anderen zu überfordern? Die Antwort war einfach und effektiv: Spielerisch lernen!

Die erste Aufgabe bestand darin, ein Zahlenraten-Spiel als Konsolenanwendung in C# zu programmieren. Was auf den ersten Blick simpel erschien, erwies sich als vielseitig und herausfordernd. Für den einen Azubi eine triviale Übung, für den anderen ein echter Meilenstein. Doch statt sich davon entmutigen zu lassen, arbeiteten beide zusammen und begannen, Ideen zu sammeln: Was könnte man aus einem so einfachen Spiel wie Zahlenraten herausholen?

● Das Lego ToyPad

Das ToyPad, bekannt aus dem Spiel Lego Dimensions, ist ein innovatives Peripheriegerät, das über mehrere NFC-Leseflächen mit LEDs an der linken, rechten und zentralen Position verfügt. Darüber können reale Lego-Modelle ins digitale Spiel übertragen werden – und wie sich herausgestellt hat: Da geht noch einiges mehr!

Von der Implementierung verschiedener Schwierigkeitsgrade bis hin zur Entwicklung von Algorithmen, die die optimale Strategie für das Spiel ermitteln, wandelte sich die Aufgabe zu einem lehrreichen und kreativen Projekt. Am Ende präsentierten beide stolz ihre Ergebnisse – ein Moment, der nicht nur die ersten Schritte in der Softwareentwicklung zeigte, sondern auch den Grundstein für eine größere Herausforderung legte: die Lego-ToyPad-Challenge.

Diese Challenge, ins Leben gerufen von Tom Wendel [1], bestand darin, herauszufinden, welche kreativen Möglichkeiten das ToyPad von Lego bieten könnte.

Aufgabe war, diese Technologie mit dem Zahlenraten-Spiel zu verbinden. Die Idee: Kann man ein Zahlenraten-Spiel mit

dem ToyPad spielen? Kleiner Spoiler: Ja, und es ist richtig cool! Im folgenden Text wird erläutert, wie die Umsetzung erfolgte, welche Herausforderungen dabei auftraten und welche spannenden Möglichkeiten sich daraus ergeben haben.

Kommunikation zwischen Rechner und ToyPads in C#

Beim Start der Challenge ging es primär darum, zu überlegen, wie ein greifbares Peripheriegerät mit dem zuvor entwickelten Zahlenraten-Spiel verbunden werden könnte. Es musste also dafür gesorgt werden, dass über PCs mit den Lego ToyPads kommuniziert werden kann. Glücklicherweise bietet das NuGet-Paket LegoDimensions [2] genau die Werkzeuge, die benötigt werden. Es fungiert als Schnittstelle zwischen PC und ToyPad und erlaubt das Auslesen und Modifizieren von NFC-Tags – genau, NFC-Tags. Es kam die Idee auf, solche Tags ins Spiel einzubringen, um damit die ToyPads gezielt anzusprechen und das Spielerlebnis interaktiver zu gestalten. Dies geschieht durch sogenannte Tag-Events, die das NuGet-Paket zur Verfügung stellt und damit die Identifizierung von NFC-Tags deutlich vereinfacht.

```
legoToyPad.LegoTagEvent += LegoToyPad_LegoTagEvent;
```

```
private static void LegoToyPad_LegoTagEvent(
    object? sender, LegoDimensions.LegoTagEventArgs e)
{
    // NFC-Tags auslesen
}
```

NFC-Tags

Nachdem die grundlegende Kommunikation zwischen dem PC und dem Lego ToyPad mithilfe des LegoDimensions-Pakets etabliert war, ging es darum, eigene NFC-Tags nahtlos in das Projekt zu integrieren. Ziel war es, mehreren Tags eindeutige Zahlen zuzuordnen, die im Spiel verwendet werden können – von 0 bis 9. Die Lösung: ein Dictionary für Zuordnungen.

Um die Verbindung zwischen den Tags und den entsprechenden Zahlen herzustellen, wurde eine Dictionary-Datenstruktur implementiert. In diesem Dictionary werden die UUID (Unique Identifier) jedes NFC-Tags als Schlüssel und die dazugehörige Zahl als Wert gespeichert. Dadurch können die Tags sowohl effizient als auch zuverlässig identifiziert werden, und man kann ihre zugewiesenen Zahlen im Spiel verwenden.

```
public static readonly Dictionary<string, int>
    TagWerte = new()
{
    { "04-FD-BA-68-B8-2A-81", 0 },
    { "04-8E-DD-5F-B8-2A-81", 1 },
    { "04-71-D3-5F-B8-2A-81", 2 },
    { "04-1E-C9-5F-B8-2A-81", 3 },
    { "04-8B-83-5F-B8-2A-81", 4 },
    { "04-B3-33-5F-B8-2A-81", 5 },
    { "04-82-3A-5F-B8-2A-81", 6 },
    { "04-FE-A9-5F-B8-2A-81", 7 },
    { "04-62-34-5F-B8-2A-81", 8 },
    { "04-C5-27-5F-B8-2A-81", 9 },

    // Weitere Tags...
};
```

Sobald ein NFC-Tag auf das ToyPad gelegt wird, löst das LegoDimensions-Paket einen Tag-Event aus, der die UID des Tags übermittelt. Diese UID wird dann mit dem Dictionary abgeglichen, um die zugehörige Zahl zu ermitteln.

Spielsteine

Um das NFC-Tag interessanter zu gestalten, sollte es nicht nur als flache, kleine Platte verwendet werden. Deshalb fiel die Wahl darauf, mithilfe eines 3D-Druckers Hüllen herzustellen, die das NFC-Tag zu einem „Spielstein“ machen.

Mit der kostenlosen Software Tinkercad [3] wurden diese „Spielsteine“ entworfen, in denen die NFC-Tags schließlich eingebettet waren. Nach einigen Testdrucken und kleineren Anpassungen konnten mehrere Sets produziert werden, damit jeder aktive Spieler eines zur Hand hat (Bild 1).

Erste Tests, neue Modi

Mit den fertigen Spielsteinen in der Hand war der nächste Schritt klar: Sie in die Praxis zu bringen und die ersten Tests durchzuführen. Die vielversprechenden Ergebnisse führten schnell zu der Frage: Wie lässt sich das Spiel noch spannender gestalten? Vielleicht durch neue Spielmodi! Mit den bereitgestellten Werkzeugen und der Integration der NFC-Tags konnte die Entwicklung direkt starten. Der erste Modus war der Singleplayer-Modus, bei dem der Spieler gegen den Computer antritt, um eine Geheimzahl zu erraten: Der Computer denkt sich zu Beginn des Spiels eine zufällige Zahl zwischen 0 und 100 aus; die Aufgabe des Spielers ist es, diese Zahl zu erraten, indem er eine Zahl auf das Lego ToyPad legt. Das ToyPad dient dabei nicht nur als Eingabegerät, sondern dank der LEDs auch als visuelles Feedback-System, das den Spieler durch das Spiel führt.

Visuelles Feedback mit dem ToyPad

Die Beleuchtungsfunktionen des ToyPads sind ein zentrales Element, um dem Spieler intuitives Feedback zu geben. Hierbei sind folgende Farben wichtig:

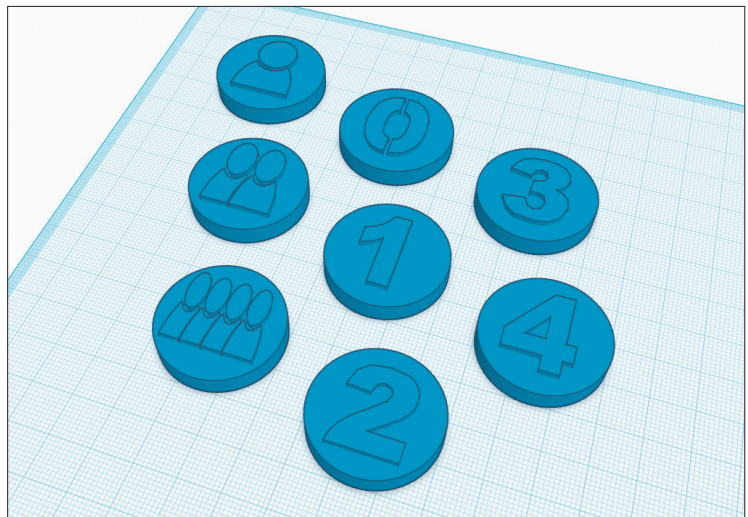
- **Grün:** Der Spieler hat die Geheimzahl erraten.
- **Rot links:** Die gelegte Zahl ist größer als die Geheimzahl.
- **Rot rechts:** Die gelegte Zahl ist kleiner als die Geheimzahl.

Dank der flexiblen Steuerungsmöglichkeit durch das LegoDimensions-Paket ist es relativ einfach, die entsprechenden Beleuchtungseffekte zu programmieren:

```
legoToyPad.SetColor(
    LegoDimensions.Portal.Pad.All, Color.Green);
```

Technische Umsetzung

Der Singleplayer-Modus basiert auf einer Kombination aus Zufallszahlengenerierungen, Event-Handling und visuellem Feedback. Zu Beginn wird eine zufällige Zahl zwischen 1 und 100 generiert, die der Spieler erraten muss. Die Eingaben werden über NFC-Tags getätigt, indem zwei dieser Tags auf die Flächen des ToyPads gelegt werden. Da bereits die UID und die zugehörige Zahl der jeweiligen NFC-Tags gespeichert wurden, kann das System aus den gelegten Tags eine ein- oder sogar zweistellige Zahl erkennen, welche dann als Rateversuch gilt. Das System registriert sowohl das Platzieren



3D-Druckvorlagen der Spielchips (Bild 1)

als auch das Entfernen von Tags, wodurch die Spieler ihre Eingabe dynamisch ändern können.

Während der Singleplayer-Modus den Spieler gegen den Computer antreten lässt, bringt der Versus-Mode eine ganz neue Dynamik in Spiel: Zwei Spieler treten gegeneinander an, wobei jeder versucht, die Geheimzahl des Gegners schneller zu erraten. Damit die Spieler parallel gegeneinander antreten können, erhalten beide Spieler ein ToyPad und NFC-Tags mit den jeweiligen Nummern. Beide Spieler legen ihre Geheimzahl mithilfe des ToyPads fest, und das Spiel startet, sobald die Farben das entsprechende Signal geben. In diesem Modus geht es um gute Strategie und Schnelligkeit, was den Spaßfaktor erhöht – sogar unter Erwachsenen, aber dazu später mehr. ►

Da der Singleplayer-Modus-Code die Basis des Spiels schon beinhaltet, wurde dieser einfach an die Versus-Modus-Umgebung angepasst. Nun war es möglich, mit mehreren ToyPads gleichzeitig und unabhängig Zahlen zu legen.

Diese Entwicklung erlaubte es auch, das Spielgeschehen weiter auszubauen und einen Turnier-Modus zu entwickeln. In diesem Modus können mehrere Spieler in einem Turnierformat gegeneinander antreten. Pro Runde treten jeweils zwei Spieler gleichzeitig gegeneinander an, bevor die nächsten beiden Spieler folgen. Dieser Ablauf wiederholt sich, bis am Ende nur noch ein Spieler übrig bleibt, der als Sieger aus dem Turnier hervorgeht.

Die Zielgruppe

Da oft schon im Lauf der Ausbildung, spätestens aber danach mit tatsächlichen Kunden in Projekten gearbeitet wird, um unter anderem Individualsoftware nach Vorgabe zu entwickeln, kam die Frage auf, wer denn die Zielgruppe für dieses Projekt sein könnte. Nun, es ist ein Zahlenraten-Spiel, und dabei kommt Lego zum Einsatz!

Die Antwort lag nahe: Es richtet sich an Kinder. Die Schwierigkeit bestand nun darin, dieses Spiel so zu programmieren, dass es von Kindern ganz intuitiv mit so wenig Erklärung und Hilfestellung wie möglich gespielt werden kann.

Dabei war zu bedenken, auf welchem Entwicklungsstand diese Kinder sein könnten. Das Resultat: Die Altersklasse von etwa sechs bis zehn Jahren sollte es sein. Zu diesem Zeitpunkt wurde auch ein Name für das Projekt gewählt: „ZahlenZauber“.

Ein Name, der kindgerecht ist, die Verbindung zu Zahlen betont und gleichzeitig den spielerischen und magischen Charakter unterstreicht. (Eventuell war an dieser Stelle ein nicht näher benanntes LLM bei der Namensfindung unterstützend tätig ...)

Disco Mode

Das eigenständige Recherchieren, Ausprobieren und Finden von Lösungen für auftretende Probleme ist eine effektive Methode, um neues Wissen zu erwerben. Unter dem Motto „Learning by Doing“ wurde der „Disco-Modus“ entwickelt – eine kleine, spielerische Erweiterung des Codes. Die Idee dahinter ist simpel: Offenbar stehen Kinder total auf bunte, blinkende Dinge – besonders wenn sie selbst einen gewissen Einfluss darauf haben. Ziel ist es, die Farben der ToyPads zufällig und abwechslungsreich aufblinken zu lassen.

Eine entsprechende Klasse, die die Logik der flachen Disco-Kugel enthält, Farben initialisiert und zufällig alle angeschlossenen ToyPads ansteuert, war schnell entwickelt.

Der gesamte Modus läuft maximal 30 Sekunden – gesteuert durch eine Stopwatch. Danach wird der Disco-Modus beendet und in das Hauptmenü zurückgeführt.

Test und Feedback

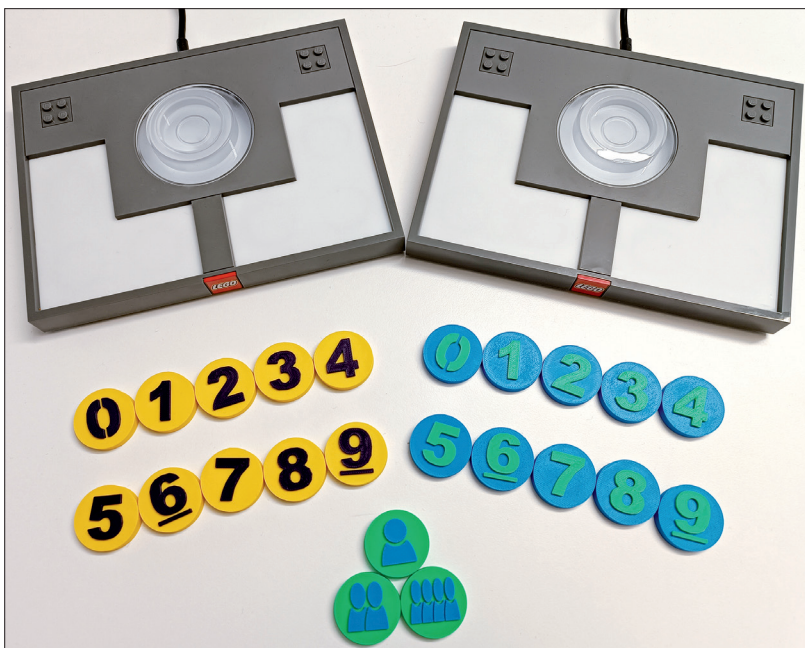
Um sicherzugehen, dass die Ideen und die technische Umsetzung des Spiels auch wirklich bugfrei und intuitiv funktionieren, wurde zunächst ein interner Spieltest durchgeführt. Dazu durften einige Mitarbeiter in mehreren Runden gegeneinander antreten. Während des Tests konnten wertvolle Erkenntnisse gesammelt werden, die bei der weiteren Entwicklung eine wichtige Rolle spielten. Schnell zeigte sich jedoch vor allem, dass dem Spiel noch etwas Entscheidendes fehlte: eine Unterstützung, die den Spielern bei der Navigation und Handhabung mit den ToyPads hilft (Bild 2).

Zu diesem Zeitpunkt beschränkte sich das Feedback der ToyPads auf visuelle Signale über LEDs, die unter den Flächen des ToyPads angebracht sind. Aus diesem Grund entstand die Idee, Soundeffekte hinzuzufügen, um das Spielgeschehen atmosphärischer und intuitiver zu gestalten. Diese Ergänzung sollte nicht nur die Interaktion mit dem ToyPad erleichtern, sondern auch das Spielerlebnis spannender und immersiver machen.

Soundeffekte

Zur Verwaltung und Nutzung von Soundeffekten wurde eine JSON-Konfigurationsdatei ergänzt. Diese listet die verfügbaren Sounds und ihre Dateipfade, sodass Anpassungen möglich sind, ohne den Code zu ändern. Eine zentrale *SoundEffect*-Klasse bietet Eigenschaften für die Soundtypen und eine *PlaySound*-Methode, die unter Windows das Abspielen von Sounds ermöglicht:

```
public static void PlaySound(
    string soundFile)
{
    if (OperatingSystem.IsWindows())
    {
        try { new SoundPlayer(
            soundFile).Play(); }
        catch (Exception ex) {
            Console.WriteLine(ex.Message); }
```



Hardware des ZahlenZaubers (Bild 2)


```

}
else { Console.WriteLine("Nur Windows unterstützt."); }
}

```

Fehler beim Abspielen werden behandelt, und bei nicht unterstützten Systemen erfolgt ein Hinweis. Die *SoundManager*-Klasse lädt und verwaltet die Soundeffekte, indem sie den Zugriff auf die JSON-Datei und die Wiedergabe der Sounds koordiniert. Die *PlaySound*-Methode in der *SoundManager*-Klasse ist genauso simpel, wie sie klingt:

```

public static void PlaySound(string soundPath)
{
    if (!string.IsNullOrEmpty(soundPath))
        new SoundPlayer(soundPath).Play();
}

```

Durch diese Struktur wird eine einfache und zentrale Handhabung der Soundeffekte gewährleistet. Der Aufruf kann zum Beispiel wie folgt aussehen:

```

SoundManager.PlaySound(
    Program.SoundEffects!.SelectedMenuSound);

```

Vorstellung in einem Hort

Das Projekt war nun in einem Zustand, in dem es präsentiert werden konnte, und da die Zielgruppe ebenfalls feststand, war es naheliegend, es dieser vorzustellen. Dank einiger Beziehungen konnte der Kontakt zu einer Kindertagesstätte mit angeschlossenem Hort hergestellt werden, um das Anliegen zu erklären. Anschließend wurde ein Termin ausgemacht und die Vorbereitungen begannen (inklusive dem einen oder anderen kleineren Bugfix). Als der „Feinschliff“ beendet war, stand auch schon der Tag des vereinbarten Termins an.

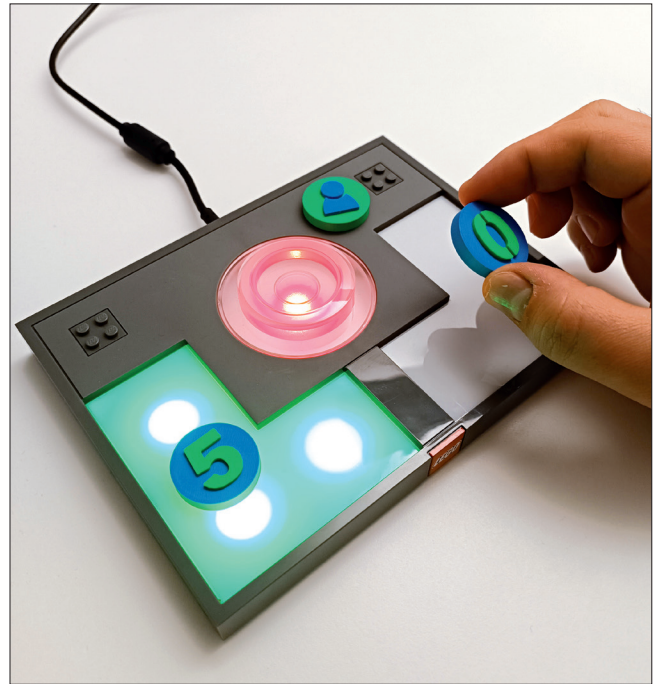
Auf dem Weg zur Kita kamen einige Gedanken auf – würden die Kinder das Spiel mögen? Würden sie verstehen, was sie tun müssen, um das Spiel spielen zu können? Hätten sie Lust, es auszuprobieren, oder fänden sie es von Beginn an langweilig? Dies war das erste gemeinsame Projekt, welches eigenständig entwickelt wurde und nun kurz vor der Präsentation stand; selbstverständlich ging dies mit etwas Nervosität einher. Diese legte sich allerdings schnell, denn es lief einfach großartig (Bild 3)!

Nach einer kleinen Vorstellungsrunde über das Spiel für die Kinder, die sich zahlreich im Gemeinschaftsraum versammelt hatten, war die Beteiligung sehr groß bei der Frage, wer dieses Spiel denn gerne ausprobieren möchte. So versammelten sich alle Freiwilligen (darunter auch einer der Betreuer) um einen Tisch und es erfolgte eine kleine Einweisung in die Handhabung der Pads, Regeln, Spielsteine et cetera.

Zu Beginn wurde der „Singleplayer“ präsentiert und von den Kindern der Reihe nach ausprobiert. Das grundlegende Prinzip des Spiels wurde von den meisten recht schnell verstanden: Richtige Zahl legen, Pads leuchten grün, gewonnen! Der Weg zum Ziel, also das Legen der Zahlen, wurde in vielen Fällen von Kind zu Kind unterschiedlich angegangen. Es war sehr interessant zu beobachten, wie jedes Kind nach

einer kurzen Zeit anfang, seine eigenen Strategien zu entwickeln. Auch in Bezug darauf, auf welchem Entwicklungsstand sie sich befanden.

So hat ein Kind, welches bereits in der vierten Klasse war, schnell herausgefunden, nach welchem Muster die Zahlen gelegt werden müssen, um schneller ans Ziel zu gelangen. Während ein anderes Kind, zweite Klasse, sich dazu entschied, beim Legen einfach nach und nach von 1 bis 99 alle Zahlen durchzuprobieren, nach der Logik: Irgendwann wird



Zahlen schnappen, platzieren, Licht an! (Bild 3)

es schon die richtige sein. Zugegeben, der Weg dauert länger, aber am Ende wurde dennoch die richtige Zahl gefunden und die Freude war dementsprechend groß, es selbst geschafft zu haben.

Die Modi „Player-vs-Player“ und der „Turnier-Modus“ waren vom Ziel her gleich: Lege die richtige Zahl. Der Twist war aber: Sei schneller als dein Gegenüber!

Die Kinder waren sehr motiviert, schneller zu sein, und haben sich eine ganze Weile mit dem Spiel beschäftigt. Ein Eingriff ins Spielgeschehen, um gegebenenfalls noch mal die Regeln zu erklären oder durch aktives Zutun den Kindern zu helfen, war kaum noch notwendig. Als abschließendes Feedback von den Kindern reichen wohl zwei Zitate aus: „Kann man das Spiel vielleicht kaufen?“ und „Das ist das beste Spiel der ganzen Welt!“ (Kein Spaß, das hat ein kleines Mädchen wirklich so gesagt.) Anhand dessen halten wir fest: Das Projekt war ein voller Erfolg!

Herausforderungen

Die Herausforderungen zeigten sich jeweils an anderer Stelle, was auch hauptsächlich daran lag, dass die beiden Auszubildenden mit unterschiedlich viel Erfahrung gestartet ►

waren. Für einen 32-jährigen Familienvater, der vorher über zehn Jahre in einem pflegerischen Berufsfeld tätig war, sind die Basics in der Anwendungsentwicklung teils schon eine ordentliche Hürde, die zu bewältigen ist – rein subjektiv betrachtet. Auf der anderen Seite steht man als 22 Jahre junger



ZahlenZauber bei der .NET Developer Conference (DDC) (Bild 4)

Mensch, der bereits eine Berufsausbildung als Systemintegrator hat, ganz anders davor und bringt einiges an Vorerfahrung mit. Dennoch war zum Beispiel die Gestaltung einer intuitiven „User Experience“ für das ToyPad eine große Herausforderung, da es in der Hinsicht bislang wenig Berührungspunkte zur eigentlichen Zielgruppe gab.

Trotz dieser Unterschiede ergänzte man sich jedoch. Der jüngere Auszubildende konnte beispielsweise seinem älteren Kollegen viel in Bezug auf die technischen Aspekte und die Basics der Softwareentwicklung erklären, was dessen Lernkurve deutlich beschleunigte. Im Gegenzug konnte der 32-jährige Auszubildende mit seiner Erfahrung als Vater wichtige Perspektiven zur Gestaltung der Benutzererfahrung einbringen. Besonders bei der Entwicklung einer kinderfreundlichen „User Experience“ für das ToyPad, einem Bereich, in dem es für beide bislang wenig Berührungspunkte gab, half ihm seine praktische Erfahrung mit der Zielgruppe sehr, um das Produkt ansprechend und intuitiv zu gestalten.

Fazit

Wieso war diese Challenge so wertvoll? Sie bot die Möglichkeit, viele wichtige Fähigkeiten zu entwickeln – von technischem Know-how bis hin zu kreativen Problemlösungsfähig-

keiten und Teamarbeit. Das Projekt zeigte, dass Azubis nicht nur passiv lernen, sondern aktiv eigene Ideen umsetzen können. Durch Kreativität und Eigeninitiative wurde etwas Wertvolles geschaffen. Ein solcher Einstieg in die Ausbildung hilft nicht nur, technische Fähigkeiten zu verbessern, sondern auch persönlich und beruflich zu wachsen. Und jemandem damit am Ende noch eine so große Freude bereiten zu können, ist ein magisches Gefühl.

.NET Developer Conference (DDC)

Ein besonderes Highlight des Projekts war, dass es von Tom Wendel, der diese Challenge ins Leben gerufen hat, auf der .NET Developer Conference (DDC) vorgestellt wurde. Für Azubis definitiv ein unglaubliches Gefühl, dass die eigene Arbeit es bis auf eine so renommierte Bühne geschafft hat – auch wenn man nicht selbst vor Ort war.

Die Vorstellung des Projekts auf der DDC hat gezeigt, wie viel durch Engagement und Kreativität erreicht werden kann. Es motiviert, weiter an spannenden Projekten zu arbeiten und das eigene Wissen in der IT auszubauen. Zu wissen, dass eine kleine Idee so positiv aufgenommen werden kann, war ein großartiger Abschluss für ein erstes großes Ausbildungsprojekt (Bild 4).

Abschließend gilt es noch einen herzlichen Dank dafür auszurichten, die Gelegenheit bekommen zu haben, das ZahlenZauber-Projekt in diesem Artikel vorstellen zu dürfen. Die Hoffnung ist groß, damit bei einigen Lesern das Interesse zu wecken, selbst ähnliche Projekte anzugehen oder diese als Inspiration zu nutzen, um Auszubildenden einen spannenden und motivierenden Einstieg in eine interessante Ausbildung zu ermöglichen. ■

[1] Tom Wendel auf YouTube, *What is the Toypad Challenge?*, www.youtube.com/watch?v=SL2504-05LegoToypad1

[2] *LegoDimensions-Paket bei NuGet*, <https://www.nuget.org/packages/LegoDimensions>

[3] *Tinkercad*, <https://www.tinkercad.com>



Robin-Niklas Fokuhl

absolviert eine Ausbildung zum Fachinformatiker Anwendungsentwicklung mit Schwerpunkt C# bei der Blue Sky Innovations GmbH. Als Quereinsteiger bringt er vielseitige berufliche Erfahrungen mit.

rfokuhl@blueskyinnovations.de



Fabian Mohdrich

ist gelernter Fachinformatiker Systemintegration und erweitert sein Portfolio durch eine Ausbildung zum Fachinformatiker Anwendungsentwicklung mit Fokus auf Microsoft .NET bei der Blue Sky Innovations GmbH.

fmohdrich@blueskyinnovations.de

dnpCode

A2504-05LegoToyPad

DEV CONTAINER MIT VISUAL STUDIO CODE

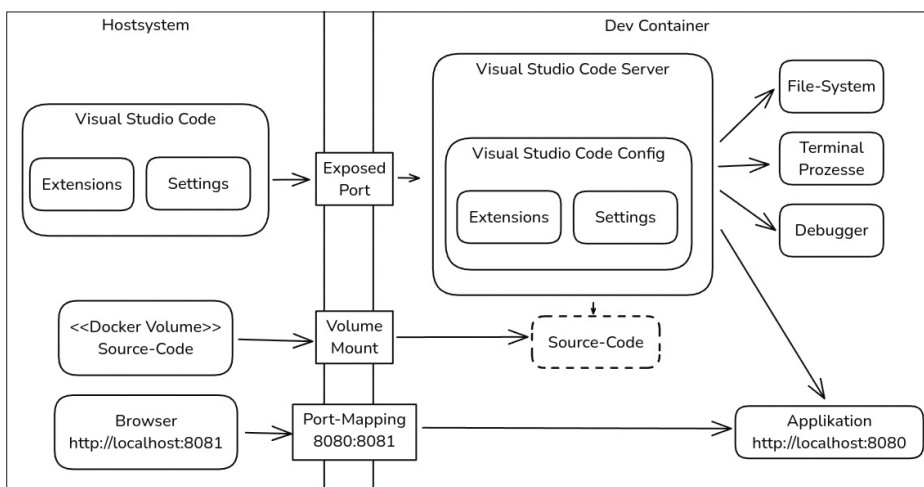
Entwickeln im Container

Mit einem Dev Container lässt sich die komplette Umgebung eines Softwareentwicklungsprojekts in der Versionskontrolle ablegen und in kürzester Zeit wiederherstellen.

Neulich ist es passiert: Mitten in einer stressigen Projektphase fiel mir der Laptop herunter – Totalschaden. Hinderlich ist in solchen Fällen neben dem finanziellen Verlust vor allem die wertvolle Zeit, die dann für Installation und Konfiguration des neuen Rechners benötigt wird. Gerade bei etwas

schrieben und in der Versionskontrolle abgelegt werden kann. Anhand dieser Beschreibung wird ein Docker-Container gestartet. Die Entwicklungsumgebung (beispielsweise Visual Studio Code) ist auf dem Host-System installiert und verbindet sich mit dem Container. Hierbei werden Einstellun-

gen, Plug-ins und Tools nach Vorgabe des Containers in der Entwicklungsumgebung installiert. Der Source-Code befindet sich in einem Docker-Volume. Auch die zu entwickelnde Applikation wird innerhalb des Containers ausgeführt und über Portmapping dem Host-System zur Verfügung gestellt. Es fühlt sich praktisch an, als entwickle man „innerhalb“ dieses Containers (vergleiche Bild 1). Dadurch wird die Entwicklungsumgebung reproduzierbar und unabhängig vom Host-System. Somit lässt sich ein neues System sehr schnell aufsetzen, denn es muss lediglich der Container gestartet werden.



Zusammenspiel zwischen Host-System und Dev Container (Bild 1)

komplexeren Setups, in denen mehrere Programmiersprachen, Third-Party-Services und spezielle Systemeinstellungen nötig sind, kann dieses initiale Setup schon einmal einen halben Tag in Anspruch nehmen. Idealerweise gibt es dafür eine Installationsanleitung, die man Schritt für Schritt abarbeitet. In der Praxis zeigt sich jedoch immer wieder, dass diese Installationsanleitungen oft unvollständig oder veraltet sind.

Hier bilden Dev Container eine interessante Alternative. Dabei handelt sich um einen offenen Standard, mit dem die gesamte Konfiguration einer Entwicklungsumgebung be-

Vorteile von Dev Containern

Verglichen mit dem „klassischen“ Entwickeln direkt auf dem Host-System bietet das Entwickeln in einem Dev Container zahlreiche Vorteile. Hat man sich einmal an das Arbeiten damit gewöhnt, möchte man dieses nützliche Tool nicht mehr im eigenen Werkzeugkasten missen. Die wichtigsten Argumente für Dev Container sind:

- **Konsistente Entwicklungsumgebung:** Dev Container ermöglichen es Entwicklern, eine genau definierte und reproduzierbare Entwicklungsumgebung zu schaffen, die von allen Teammitgliedern geteilt werden kann. Dadurch wird sichergestellt, dass alle Entwickler mit denselben Tools und Einstellungen arbeiten.
- **Einfache Abhängigkeitsverwaltung:** Dev Container isolieren Abhängigkeiten und Bibliotheken innerhalb des Containers. Dadurch kann man an verschiedenen Projekten mit unterschiedlichen Abhängigkeiten auf demselben Host-System arbeiten, ohne Konflikte zu erzeugen.
- **Plattformunabhängigkeit:** Entwickler können auf unterschiedlichen Betriebssystemen arbeiten (zum Beispiel Windows, macOS, Linux), während der Dev Container eine ►

Listing 1: Eine erste devcontainer.json

```
{
  "name": "C# (.NET)",
  "image": "mcr.microsoft.com/devcontainers/
    dotnet:1-9.0-bookworm"
}
```

einheitliche Umgebung bereitstellt. Dies reduziert Probleme, die durch unterschiedliche Betriebssysteme entstehen können. Ein prominentes Beispiel für solche Probleme sind die betriebssystemabhängigen Zeichen für das Zeilenende.

- **Vereinfachtes Onboarding:** Neue Entwickler können schneller produktiv werden, da sie lediglich den Dev Container starten müssen, um auf eine vollständig konfigurierte Entwicklungsumgebung zuzugreifen. Man spart sich die Zeit, die benötigt wird, wenn man komplexe Setups händisch durchführen muss.
- **Vereinfachte Kollaboration:** Bei der Arbeit in Projektteams bietet ein Dev Container die Möglichkeit, die gesamte Entwicklungsumgebung als Code zu definieren und Versionen zu kontrollieren. Das sorgt für eine bessere Zusammenarbeit und weniger Abstimmungsprobleme, weil Änderungen über die Versionskontrolle einfach an das ganze Team ausgerollt werden können.
- **Schnelles Umschalten zwischen Projekten:** Da jede Projektumgebung in einem eigenen Dev Container enthalten ist, können Entwickler schnell zwischen Projekten wechseln, ohne sich über Umgebungsvariablen oder Abhängigkeiten Gedanken machen zu müssen. Dies ist insbesondere im Dienstleister- und Agenturgeschäft ein großer Vorteil.
- **Experimentierfreudigkeit fördern:** Entwickler können gefahrlos neue Tools oder Technologien innerhalb eines Containers testen, ohne Risiko für das Host-System oder andere Projekte.

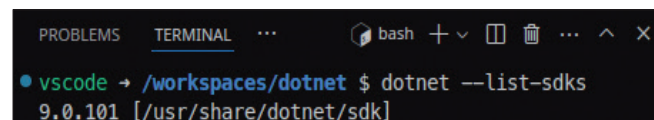
Allerdings setzt man trotz dieser Vorzüge oft nicht auf Dev Container, weil das initiale Setup als kompliziert oder zeitaufwendig eingeschätzt wird. Dieser Artikel zeigt, dass dem nicht so ist und die Einstiegshürde niedriger ist, als man denkt. Im Folgenden wird gezeigt, wie man einen einfachen Dev Container für die .NET-Entwicklung mithilfe von Visual Studio Code einrichtet und Schritt für Schritt bis hin zu einer Multi-Container-Umgebung erweitert.

Einen ersten Dev Container einrichten

Um mit Dev Containern zu arbeiten, benötigt man Docker sowie in Visual Studio Code das Plug-in *Dev Containers* von Microsoft. Anschließend führt man über die Command-Palette (erreichbar mit [Strg]+[Umschalt]+[P]) den Befehl *Dev Containers: New Dev Container* aus und wählt *Add configuration to workspace*. Danach kann man sich ein vordefiniertes Con-

tainer-Template aussuchen – wir wählen *C# (.NET)*. Schließlich vergibt man noch einen Namen und selektiert *Create Dev Container*. Nun lädt Visual Studio Code das Container-Image herunter und startet den Container. Nach einer kurzen Wartezeit öffnet sich ein neues Fenster innerhalb des Containers, und man findet eine Datei namens *devcontainer.json* im Unterverzeichnis *.devcontainer/* vor. Diese Datei ist sozusagen ein „Kochrezept“, das beschreibt, wie man unsere Entwicklungsumgebung aufsetzt. Wie man in [Listing 1](#) sieht, ist sie recht einfach aufgebaut. Sie enthält lediglich einen Namen und den Verweis auf ein Docker-Image, das beim Starten geladen wird (die echte Datei enthält noch einige Kommentare, sie wurden aus Gründen der Lesbarkeit hier weggelassen).

Öffnet man nun ein Terminal innerhalb von Visual Studio Code und lässt sich die installierten .NET-SDK-Versionen anzeigen, so sieht man, dass .NET 9.0 verfügbar ist ([Bild 2](#)), und



```

PROBLEMS TERMINAL ... bash + v [ ] [ ] ... ^ X
vscode → /workspaces/dotnet $ dotnet --list-sdks
9.0.101 [/usr/share/dotnet/sdk]
  
```

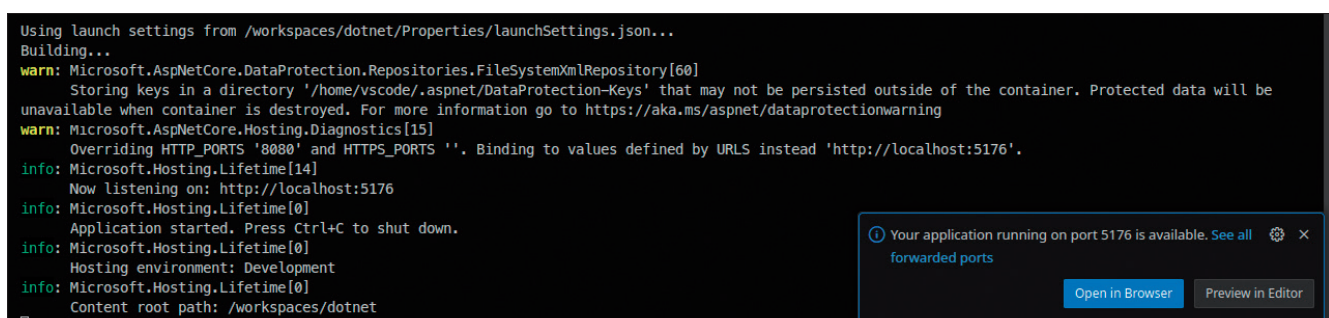
.NET-SDK-Versionen im Dev Container ([Bild 2](#))

zwar unabhängig davon, was an Frameworks auf dem Host-System installiert ist.

Mit dem Basis-Image kann man bereits eine vollständige .NET-Anwendung entwickeln. Gibt man nun das Kommando *dotnet new blazor && dotnet run* ein, so wird eine neue Blazor-Anwendung erstellt und gestartet. Visual Studio Code erkennt dies, richtet automatisch eine Portfreigabe für den Container ein und bietet uns an, die Anwendung im Browser zu öffnen ([Bild 3](#)).

Den Container mit Features erweitern

Jetzt könnten wir beispielsweise bereits eine vollständige Blazor-Anwendung entwickeln. Möchte man aber das Frontend als JavaScript-basierte Single Page Application entwickeln, benötigt man Node.js, das auf dem .NET-Standard-Image nicht installiert ist. Man sollte jedoch dem ersten Impuls widerstehen, Node.js einfach über die Kommandozeile zu installieren. Würde man dies tun, wäre Node.js nur in der aktuellen Containerinstanz verfügbar. Sobald man den Con-



```

Using launch settings from /workspaces/dotnet/Properties/launchSettings.json...
Building...
warn: Microsoft.AspNetCore.DataProtection.Repositories.FileSystemXmlRepository[60]
      Storing keys in a directory '/home/vscode/.aspnet/DataProtection-Keys' that may not be persisted outside of the container. Protected data will be
      unavailable when container is destroyed. For more information go to https://aka.ms/aspnet/dataprotectionwarning
warn: Microsoft.AspNetCore.Hosting.Diagnostics[15]
      Overriding HTTP_PORTS '8080' and HTTPS_PORTS ''. Binding to values defined by URLS instead 'http://localhost:5176'.
info: Microsoft.Hosting.Lifetime[14]
      Now listening on: http://localhost:5176
info: Microsoft.Hosting.Lifetime[0]
      Application started. Press Ctrl+C to shut down.
info: Microsoft.Hosting.Lifetime[0]
      Hosting environment: Development
info: Microsoft.Hosting.Lifetime[0]
      Content root path: /workspaces/dotnet
  
```

Meldung beim Starten des ASP.NET-Development-Servers ([Bild 3](#))

● Listing 2: Ein Feature in der devcontainer.json stellt Node.js in Version 20 bereit

```
{
  "name": "C# (.NET)",
  "image": "mcr.microsoft.com/devcontainers/
    dotnet:1-9.0-bookworm",
  "features": {
    "ghcr.io/devcontainers/features/node:1": {
      "version": "20"
    }
  }
}
```

tainer neu baut (oder ein Kollege das Repository auscheckt und dadurch einen neuen Container baut), wäre Node.js wieder verschwunden. Wir müssen also unser Kochrezept, die Datei *devcontainer.json*, erweitern.

Hierfür gibt es die Möglichkeit, sogenannte Features zu definieren. Ein Feature ist eine wiederverwendbare Beschreibung, die bestimmt, wie eine bestimmte Funktion (in diesem Fall die Installation von Node.js) in einem Dev Container bereitgestellt wird. Neben offiziellen Features von Microsoft gibt es auch eine Vielzahl von Features, die von der Community bereitgestellt werden. Auf der offiziellen Dev-Container-Webseite findet man eine Liste aller verfügbaren Features [1].

Möchte man ein Feature zu einem bestehenden Container hinzufügen, führt man in Visual Studio Code das Kommando *Dev Containers: Configure Container Features* aus. Anschließend kann man beliebig viele Features aussuchen, die dem Container hinzugefügt werden sollen. Für unser Beispiel wählt man *Node.js (via nvm)*, *yarn and pnpm* aus und bestätigt anschließend mit OK. Die Datei *devcontainer.json* wird daraufhin um ein Objekt namens *features* erweitert (siehe [Listing 2](#)). Jede Property dieses Objekts repräsentiert den Verweis auf ein Feature inklusive seiner Major-Version des Features (Version 1 bei *node*). Der Wert der Property ist wiederum ein Objekt, das die Konfiguration des Features enthält. Bei *node* kann man hier zum Beispiel die gewünschte Version

Mit Customizations Visual Studio Code anpassen

Der Dev-Container-Standard bietet auch die Möglichkeit, toolspezifische Konfigurationen (sogenannte Customizations) zu definieren. Diese Konfigurationen werden dann beim Verbinden des entsprechenden Tools mit dem Container ausgeführt.

Bei Visual Studio Code gibt es die Möglichkeit, Extensions zu definieren, die beim Verbinden zum Container mitinstalliert werden. Diese Extensions sind somit nur im Container verfügbar und nicht auf dem Host-System. Des Weiteren lassen sich Einstellungen für Visual Studio Code vornehmen, die ebenfalls nur innerhalb des Containers verfügbar sind. Mit diesen beiden Möglichkeiten kann man die komplette IDE individuell für das Projekt anpassen und diese Konfiguration leicht mit dem eigenen Team teilen.

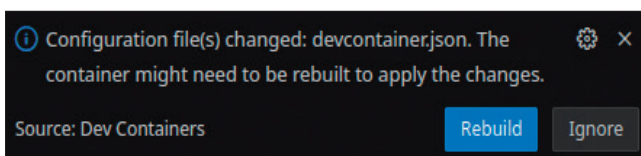
[Listing 3](#) zeigt, wie wir unserem Dev Container die Extension *Code Spell Checker* sowie ein deutsches Wörterbuch hinzufügen und ihn so konfigurieren, dass sowohl das deutsche als auch das englische Wörterbuch benutzt wird. Sobald der Container gestartet wird, steht somit die Rechtschreibprüfung in beiden Sprachen zur Verfügung. Praktisch: Um eine Extension hinzuzufügen, kann man einfach im Kontextmenü des Extension-Marktplatzes auf *add to devcontainer.json* klicken ([Bild 5](#)).

Das Container-Image erweitern

Bisher benutzten wir ein bestehendes Container-Image, das von Microsoft bereitgestellt wurde, und erweiterten es mit verfügbaren Features und Customizations. Manchmal ist es allerdings nötig, tiefer in den Bauprozess einzugreifen, um beispielsweise spezielle Abhängigkeiten zu installieren und zu konfigurieren. Hierfür bietet der Dev-Container-Standard die Möglichkeit, ein eigenes Dockerfile zu hinterlegen, das beim Bauen des Containers ausgeführt wird.

In unserem Beispiel möchten wir dem Image das .NET Core CLI Tool für Entity Framework Core hinzufügen, sodass im Anschluss das Kommando *dotnet ef* zur Verfügung steht. Hierfür legt man im Verzeichnis *.devcontainer* ein neues Dockerfile mit dem Inhalt aus [Listing 4](#) an. Zu beachten ist, dass die erste Zeile (beginnend mit *FROM*) das Basis-Image angibt, das bisher direkt über die *devcontainer.json* referenziert wurde. Anschließend ersetzt man die Property *image* in der *devcontainer.json* durch die Property *dockerFile* und gibt den Pfad (relativ zur *devcontainer.json*) zum Dockerfile an:

```
"dockerFile": "Dockerfile" ▶
```



Visual Studio Code erkennt eine Änderung der *devcontainer.json* und bietet uns einen Rebuild des Containers an ([Bild 4](#)).

von Node.js angeben (siehe [Listing 2](#)). Die Liste verfügbarer Konfigurationsoptionen unterscheidet sich selbstverständlich pro Feature und ist der Dokumentation des jeweiligen Features zu entnehmen.

Jetzt ist Node.js immer noch nicht verfügbar, da der Container zunächst neu gebaut werden muss. Glücklicherweise erkennt Visual Studio Code dies und weist uns bei Änderungen in der *devcontainer.json* darauf hin. Nach einem Klick auf *Rebuild* wird der Container neu gebaut, und Node.js ist verfügbar ([Bild 4](#)).

● Listing 3: Customization in der devcontainer.json am Beispiel der Extension Code Spell Checker

```
"customizations": {
  "vscode": {
    "extensions": [
      "streetsidesoftware.code-spell-checker",
      "streetsidesoftware.code-spell-checker-german"
    ],
    "settings": {
      "cSpell.language": "en,de-de"
    }
  }
},
```

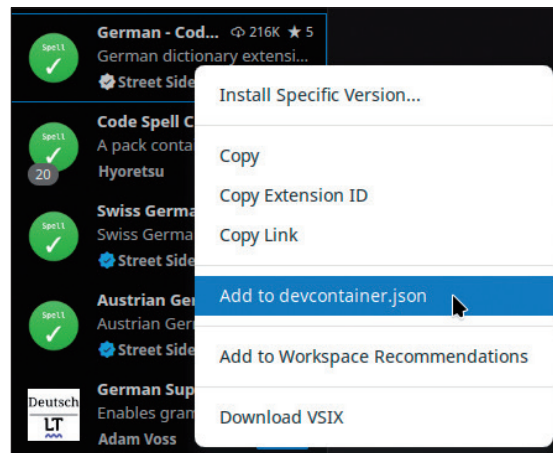
Baut man anschließend den Container neu, wird ein Image erstellt, das auf dem Basis-Image basiert, aber zusätzlich das Kommando *dotnet ef* installiert hat.

Einen zweiten Container hinzufügen

Als Nächstes soll im Dev Container die Möglichkeit geschaffen werden, auf einen SQL Server zuzugreifen, um Datenbanken zu erstellen und während der Entwicklung dagegen zu testen.

Microsoft selbst bietet Docker-Images des SQL Server an [2]. Wir machen jetzt also den Schritt

von einem Single-Container- zu einem Multi-Container-Setup, da beim Starten nun ein Dev Container sowie ein Container für den SQL Server gestartet werden müssen. Hier empfiehlt es sich, Docker Compose zu verwenden, um auf einfache Weise beliebig viele Container zu orchestrieren. Man legt



Extensions lassen sich per Rechtsklick hinzufügen (Bild 5)

eine Datei namens *docker-compose.yml* im Verzeichnis *.devcontainer/* an und definiert darin sowohl den SQL Server als auch den Dev Container (siehe Listing 5).

In der Datei definiert man zuerst einen Service für den Dev Container, der unser vorher erstelltes Dockerfile referenziert. Als Volume gibt man *../../workspaces:cached* an. Dadurch wird das Arbeitsverzeichnis des Hosts unter */workspaces* im Container gemountet sowie Caching aktiviert. Die Zeile *command: sleep infinity* bewirkt, dass der Container nicht sofort nach dem Start beendet wird; *network_mode: service:sqlserver* sorgt dafür, dass der Dev Container im gleichen Netzwerk wie der SQL Server ist und man somit vom Dev Container auf den SQL Server zugreifen kann.

Als Nächstes definiert man den Service für den SQL Server, wie in der Dokumentation von Microsoft [3] beschrieben. Be-

work_mode: service:sqlserver

● Listing 4: Ein Dockerfile, das das Standard-.NET-Image um das dotnet-ef-Kommando erweitert

```
FROM mcr.microsoft.com/devcontainers/
  dotnet:1-9.0-bookworm
RUN dotnet tool install -g dotnet-ef
```

● Listing 5: docker-compose.yml mit einer Definition für den Dev Container sowie einem Container für SQL Server

```
services:
  devcontainer:
    build:
      dockerfile: Dockerfile
    volumes:
      - ../../workspaces:cached
    command: sleep infinity
    # Prevents the container from exiting immediately
    network_mode: service:sqlserver
  sqlserver:
    image: mcr.microsoft.com/mssql/server:2022-latest
    environment:
      - SA_PASSWORD=Pass@word
      - ACCEPT_EULA=Y
    ports:
      - "1433:1433"
```

● Listing 6: devcontainer.json mit Verweis auf Docker Compose

```
{
  "name": "C# (.NET)",
  "dockerComposeFile": "docker-compose.yml",
  "service": "devcontainer",
  "workspaceFolder":
    "/workspaces/${localWorkspaceFolderBasename}",
  "features": {
    "ghcr.io/devcontainers/features/node:1": {
      "version": "20"
    }
  }
}
```

nötigt man weitere Services, fügt man diese entsprechend in der Datei unter *services* hinzu.

Jetzt muss die neue Docker-Compose-Datei noch in der *devcontainer.json* referenziert werden. Hierfür ersetzt man die Property *dockerFile* durch die Property *dockerComposeFile* und gibt den Pfad zur Docker-Compose-Datei an (Listing 6). Des Weiteren müssen der Service sowie ein Mountpunkt für das Arbeitsverzeichnis innerhalb des Containers angegeben werden. Beim nächsten Start des Containers wird nun auch der SQL Server gestartet.

Dateiaustausch zwischen Host und Container

Manchmal ist es nötig, Dateien zwischen Host und Container auszutauschen. Um eine Datei vom Host in den Container zu kopieren, kann man sie einfach per Drag and Drop in den Dateixplorer von Visual Studio Code ziehen. In der Gegenrichtung funktioniert es leider nicht so intuitiv. Um eine Datei vom Container auf den Host zu bringen, gibt es aber einen einfachen Trick: Man installiert das NPM-Paket *http-server*. Dies ist ein kleiner, leichtgewichtiger Webserver, der sich von der Kommandozeile aus mit dem Befehl *http-server* starten lässt. Das Verzeichnis, in dem das Kommando aufgerufen wurde, ist anschließend unter *http://localhost:8080* erreichbar, und man kann bequem Dateien aus dem Container herunterladen. Dies ist beispielsweise sehr nützlich, um Build-Artefakte aus dem Container auf den Host zu bekommen. *http-server* gibt es ebenfalls als Feature für Dev Container. Um das Paket einem Container hinzuzufügen, fügt man folgende Zeile unter *features* in die *devcontainer.json* ein:

```
"ghcr.io/devcontainers-extra/features/http-server:1": {}
```

Fazit

Ein Dev Container ist ein praktisches Werkzeug, um die Entwicklungsumgebung zu standardisieren. Er ermöglicht es, die gesamte Konfiguration der Entwicklungsumgebung in der Versionskontrolle abzulegen und somit für alle Teammitglieder zugänglich zu machen. Man kann sehr einfach mit vorgefertigten Container-Images starten und diese je nach Bedarf um Features erweitern. Benötigt man mehr Flexibilität, so kann man den Container auch mit einem eigenen Dockerfile definieren. Mit Docker Compose lassen sich leicht Multi-Container-Szenarien umsetzen. Die in diesem Artikel beschriebene Konfiguration gibt es übrigens bereits als fertiges Community-Template [4]. Die Funktionen des Templates

wurden in diesem Artikel Schritt für Schritt aufgebaut, um Kernfeatures von Dev Containern zu erklären.

Zugegeben: Dev Container haben eine gewisse Lernkurve, insbesondere für Teams, die noch nicht mit Docker vertraut sind. Allerdings ist es für sich genommen bereits eine große Zeitersparnis, wenn nicht mehr jedes Teammitglied sämtliche Tools und Einstellungen manuell konfigurieren muss. Insbesondere bei Projekten mit vielen externen Abhängigkeiten ist dies von unschätzbarem Wert.

Der Dev-Container-Standard [5] wurde als offener Standard von Microsoft für Visual Studio Code entwickelt. Um diesen Standard bildet sich bereits ein wachsendes Ökosystem [6] an Tools und Services, die Dev Container unterstützen – hier sind insbesondere die JetBrains-IDEs (zum Beispiel Rider oder Webstorm) sowie GitHub Codespaces zu nennen.

Wer tiefer in die Thematik eintauchen möchte, dem sei insbesondere die ausführliche Dokumentation von Visual Studio Code ans Herz gelegt [7]. ■

- [1] Verfügbare Dev Container Features,
<https://containers.dev/features>
- [2] Docker Images des SQL Servers,
www.dotnetpro.de/SL2504-05DevContainer1
- [3] .NET Architektur: Verwenden eines als Container
ausgeführten Datenbankservers,
www.dotnetpro.de/SL2504-05DevContainer2
- [4] Dev-Container-Template mit .NET, Node.js und MS SQL,
www.dotnetpro.de/SL2504-05DevContainer3
- [5] Development Container Specification,
<https://containers.dev/implementors/spec/>
- [6] Supporting tools and services,
<https://containers.dev/supporting>
- [7] Developing inside a Container (Visual Studio Code),
www.dotnetpro.de/SL2504-05DevContainer4



Alexander Rapp

ist Entwickler und Head of Software Engineering bei XITASO. Er unterstützt Teams dabei, sich in der Softwareentwicklung weiter zu professionalisieren. Seine Schwerpunkte sind Softwarequalität sowie Persönlichkeitsentwicklung für Softwareentwickler und -entwicklerinnen.

dnpCode

A2504-05DevContainer

POSTGRESQL

Performance Tuning

Datenzugriffsmethoden, Ausführungspläne und Statistiken.

Der PostgreSQL-Artikel [1] in der zurückliegenden dotnetpro hat einen breiten Überblick über PostgreSQL gegeben und zudem die wesentlichen Unterschiede zu Microsofts SQL Server benannt. Außerdem wurde darin gezeigt, wie Sie eine PostgreSQL-Datenbank anlegen, und es wurden erste Details zum Umgang mit Tabellen vorgestellt.

Dieser Artikel zeigt, wie Sie die Performance von PostgreSQL steigern, indem Sie Abfragen mithilfe von Indizes beschleunigen. Bevor es jedoch um die Details geht, möchte ich noch einige sehr wichtige Unterschiede zu Microsofts SQL Server im Bereich der Indizierung erklären.

Indizes in PostgreSQL

Weil eine PostgreSQL-Datenbank nur Heap-Tabellen unterstützt, ist es im Gegensatz zum SQL Server mit PostgreSQL nicht möglich, einen Clustered Index für eine Tabelle zu definieren und damit eine physische Reihenfolge der Tabellendaten zu erzeugen.

Eine Heap-Tabelle speichert die Datensätze sequenziell auf 8-KB-Seiten (Pages), ohne eine Schlüsselreihenfolge zu berücksichtigen. Obwohl PostgreSQL den Befehl *CLUSTER* unterstützt [2], bleibt die Sortierung aufgrund von späterem Einfügen, Ändern oder Löschen von Daten nicht bestehen.

Für Performance-Optimierungen bietet PostgreSQL jedoch an, auf einer Heap Table eine Reihe zusätzlicher Indizes zu definieren – wie zum Beispiel B-Tree, Hash, GIN, GiST, SP-GiST, BRIN und RUM.

Ebenfalls wichtig ist, dass PostgreSQL ein Multiversion Concurrency Control System (MVCC) einsetzt, um mehrere gleichzeitige Transaktionen voneinander zu isolieren. MVCC ist vergleichbar mit Optimistic Concurrency, das mit dem SQL Server 2005 eingeführt wurde.

Das Prinzip von MVCC ist, dass ein einmal eingefügter Datensatz nicht mehr verändert werden kann. Bei Änderungen an einem bestehenden Datensatz wird eine neue Version erzeugt und in die Heap-Tabelle aufgenommen. Dazu hat jeder Datensatz einen zeitlichen Gültigkeitsbereich, den die internen Spalten *xmin* und *xmax* abbilden. In diesen Spalten wer-

● **Tabelle 1: Erweiterungen des EXPLAIN-Befehls**

	Beschreibung
<i>ANALYZE</i>	Führt die Abfrage aus und liefert den Actual Execution Plan zurück.
<i>COSTS</i>	Zeigt die Kosten pro Operator an (standardmäßig aktiviert).
<i>TIMING</i>	Zeigt die Ausführungszeit pro Operator an (standardmäßig aktiviert).
<i>BUFFERS</i>	Zeigt weitere Informationen über die Verwendung des Buffer Managers an.
<i>WAL</i>	Liefert Infos zu den Log Records, die in das Write Ahead Log (WAL) geschrieben wurden.
<i>MEMORY</i>	Informiert über den verwendeten Hauptspeicher während der Query-Optimization-Phase.
<i>FORMAT</i>	Spezifiziert das Output-Format: Text (Default), XML, JSON, YAML.
<i>SUMMARY</i>	Erfragt Informationen über die Ausführungszeit der Abfrage (standardmäßig aktiviert).

den die Transaktions-IDs gespeichert, für deren Zeitraum die Datensatzversion gültig ist. Weitere Details zu MVCC erfahren Sie in einem später folgenden PostgreSQL-Artikel. Darin lernen Sie auch, wie nicht mehr benötigte Datensatzversionen mit sogenannten *VACUUM*-Operationen automatisch oder manuell bereinigt werden.

Ausführungspläne

Leser, die schon einmal Performance-Optimierungen beim SQL Server durchgeführt haben, kennen dessen Ausführungspläne. Ein Ausführungsplan ist das Programm, das ausgeführt wird, wenn die relationale Datenbank eine Abfrage verarbeitet.

Ein solcher Ausführungsplan ist erforderlich, weil die SQL-Abfragesprache eine deklarative Programmiersprache ist. Sie als Entwickler definieren das Ergebnis, das die Abfrage

zurückliefern soll, bestimmen aber zu keinem Zeitpunkt, wie dieses Ergebnis von der Datenbank-Engine ermittelt werden soll. Dieses „Wie“ steuert der Ausführungsplan, der vom Query Planner (PostgreSQL) beziehungsweise dem Query Optimizer

1	-- The "EXPLAIN" mode returns the Estimated Execution Plan, without running the query
2	EXPLAIN
3	SELECT COUNT(*) FROM Sales.SalesOrderHeader;
4	
Results Messages	
	QUERY PLAN
1	Aggregate (cost=902.93..902.94 rows=1 width=8)
2	--> Index Only Scan using "PK_SalesOrderHeader_SalesOrderID" on salesorderheader (cost=0.29..824.26 rows=31465 width=0)

EXPLAIN liefert den Estimated Execution Plan, ohne die Abfrage auszuführen (Bild 1)

5	-- Returns the Actual Execution Plan
6	EXPLAIN (ANALYZE)
7	SELECT COUNT(*) FROM Sales.SalesOrderHeader;
Results Messages	
QUERY PLAN	
1	Aggregate (cost=902.93..902.94 rows=1 width=8) (actual time=9.721..9.722 rows=1 loops=1)
2	-> Index Only Scan using "PK_SalesOrderHeader_SalesOrderID" on salesorderheader (cost=0.29..824.26 rows=31465 width=0) (actual time=3.557..8.116 rows=31465 loops=1)
3	Heap Fetches: 0
4	Planning Time: 0.112 ms
5	Execution Time: 9.768 ms

Der Actual Execution Plan zu einer Abfrage (Bild 2)

(SQL Server) erzeugt und anschließend ausgeführt wird. Ähnlich wie beim SQL Server gibt es auch in PostgreSQL den sogenannten Estimated Execution Plan sowie den Actual Execution Plan. Der Estimated Execution Plan ist der Ausführungsplan, der vom Query Planner erstellt wurde. Im Gegensatz dazu ist der Actual Execution Plan der tatsächlich ausgeführte Ausführungsplan, der zusätzlich mit weiteren leistungsrelevanten Informationen versehen ist.

Um diese beiden Ausführungspläne einsehen zu können, bietet PostgreSQL die Befehle *EXPLAIN* und *EXPLAIN ANALYZE* an. *EXPLAIN* liefert den Estimated Execution Plan für ein SQL-Statement zurück, ohne die Abfrage auszuführen. Hier ein einfaches Beispiel:

```
EXPLAIN SELECT COUNT(*) FROM Sales.SalesOrderHeader;
```

Es genügt also, den Befehl *EXPLAIN* vor das eigentliche SQL-Statement zu schreiben. Das Ergebnis des Befehls sehen Sie in **Bild 1**: Er liefert den Estimated Execution Plan als Text zurück. Ein grafisches Format (wie im SQL Server) wird ohne Zusatz-Tools nicht unterstützt.

In **Tabelle 1** finden Sie weitere Optionen, die Sie dem Befehl *EXPLAIN* mitgeben können, um zusätzliche Informationen abzufragen. Die Option *ANALYZE* führt die Abfrage aus und liefert den Actual Execution Plan:

```
EXPLAIN(ANALYZE) SELECT COUNT(*)
FROM Sales.SalesOrderHeader;
```

In **Bild 2** sehen Sie das Ergebnis dieser Abfrage. Wie Sie erkennen können, liefert der Actual Execution Plan Werte, die

Listing 1: EXPLAIN mit weiteren Optionen

```
EXPLAIN
(
  VERBOSE,
  ANALYZE,
  SETTINGS,
  COSTS,
  TIMING,
  BUFFERS,
  WAL,
  MEMORY,
  SUMMARY
)
SELECT ctid, AccountNumber
FROM Sales.SalesOrderHeader
WHERE AccountNumber = '10-4020-000646';
```

im Estimated Execution Plan fehlen, zum Beispiel die aktuellen Kosten (*Actual Costs*) sowie Informationen hinsichtlich der Ausführungszeiten. In **Listing 1** finden Sie eine Abfrage mit mehreren aktivierten Optionen, und **Bild 3** zeigt das Ergebnis dieser Abfrage.

Wie Sie **Tabelle 1** entnehmen können, kann PostgreSQL den Ausführungsplan in unterschiedlichen Formaten zurückgeben, etwa als Text, XML, JSON oder YAML.

Besonders interessant ist das Format JSON, da sich diese Daten mit anderen Tools visualisieren lassen. Ein Beispiel ►

Ergebnis der
EXPLAIN-
Abfrage aus
Listing 1
(Bild 3)

Results Messages	
QUERY PLAN	
1	Seq Scan on sales.salesorderheader (cost=0.00..1115.31 rows=2 width=21) (actual time=0.025..6.491 rows=5 loops=1)
2	Output: ctid, accountnumber
3	Filter: ((salesorderheader.accountnumber)::text = '10-4020-000646')::text)
4	Rows Removed by Filter: 31460
5	Buffers: shared hit=722
6	Planning:
7	Buffers: shared hit=11
8	Memory: used=10kB allocated=16kB
9	Planning Time: 0.090 ms
10	Execution Time: 6.507 ms

dafür ist die Website <https://explain.dalibo.com>, welche die JSON-Daten visuell aufbereiten kann. **Bild 4** zeigt eine komplexere Abfrage, die grafisch aufbereitet wurde.

Diese Art der Visualisierung erinnert sehr stark an die grafischen Ausführungspläne des SQL Server. Der Hauptunterschied: Im SQL Server lesen Sie einen Ausführungsplan von rechts nach links, den Ausführungsplan in **Bild 4** dagegen von unten nach oben.

Heap-Tabellen und Indizierung

Wie zu Beginn dieses Artikels erwähnt, stellt PostgreSQL ausschließlich Heap-Tabellen zur Verfügung, wobei die Daten ohne spezifische physische Sortierreihenfolge abgelegt werden. Jeder Datensatz wird in PostgreSQL eindeutig über eine sogenannte Tuple ID (TID) identifiziert, die folgendes Format hat:

```
(block_number, tuple_index)
```

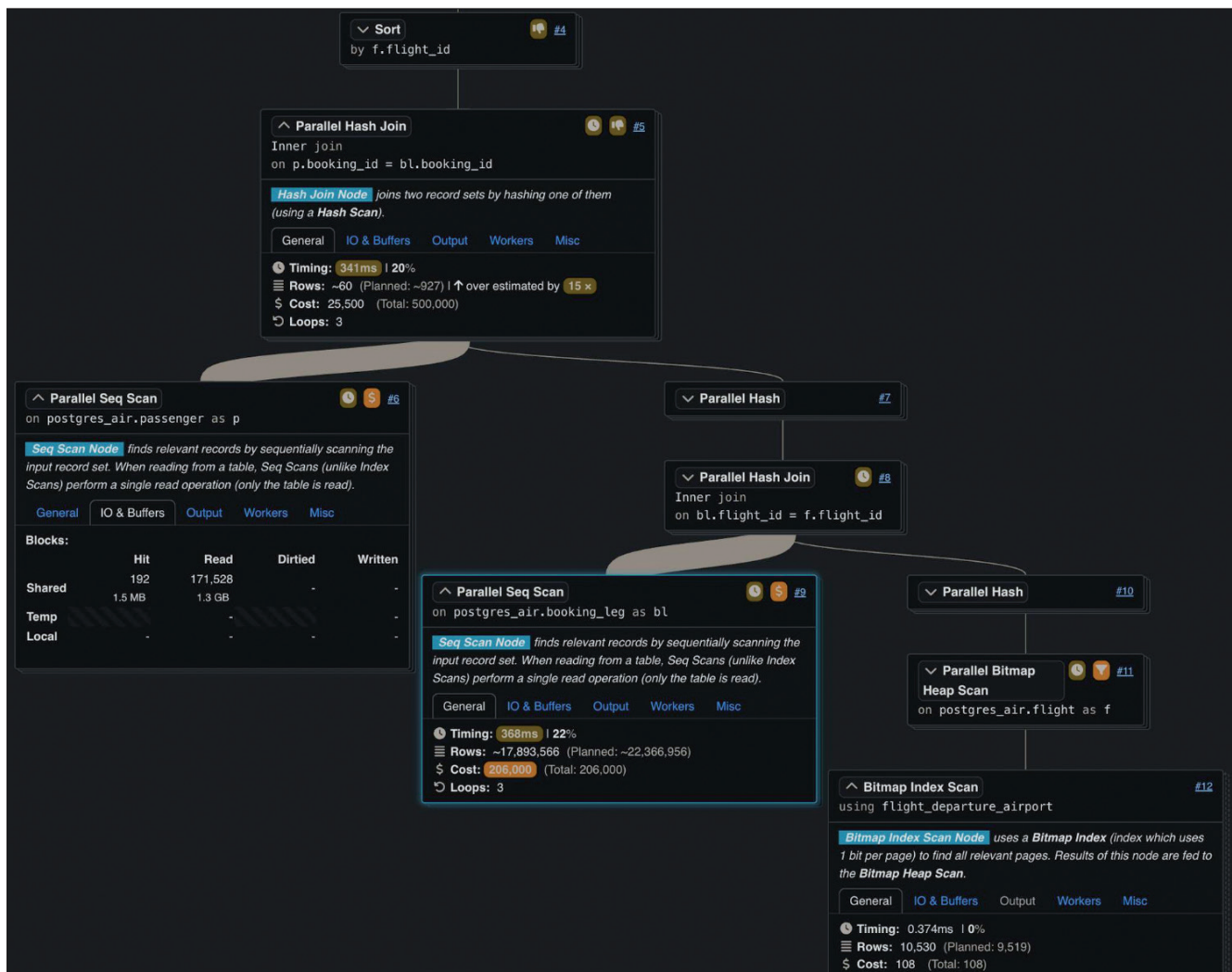
Die *block_number* benennt die 8-KB-Page, auf welcher der Datensatz physisch gespeichert ist. Da jede Heap-Tabelle und jeder Index in PostgreSQL in einer eigenen Datei gespeichert werden, bezieht sich die erste Seite immer auf die ers-

ten 8 KByte der Heap-Tabelle beziehungsweise des betreffenden Index.

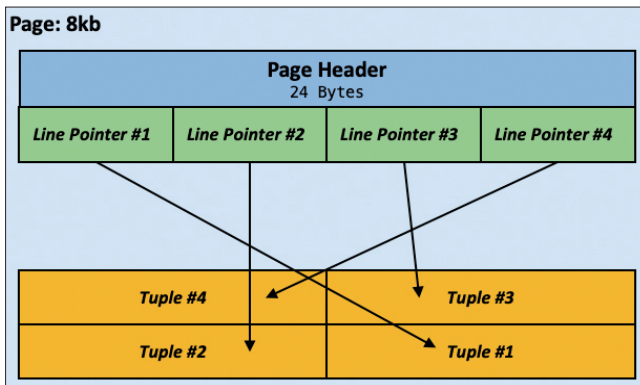
Der *tuple_index* referenziert den Eintrag im sogenannten Line Pointer Array. Dieses ist vergleichbar mit dem Row Offset Array einer SQL Server Page, wobei das Row Offset Array am Ende einer Seite gespeichert wird und das Line Pointer Array direkt auf den Page Header (24 Bytes) folgt. Der Eintrag im Line Pointer Array verweist auf die physische Position auf der Seite, an welcher der Datensatz zu finden ist.

Ein wesentlicher Unterschied zwischen Pages in PostgreSQL und SQL Server liegt darin, dass Datensätze in PostgreSQL immer vom Ende einer Page zum Anfang hin geschrieben werden. Im Gegensatz dazu folgen Datensätze im SQL Server direkt auf den Page Header (96 Bytes) und werden bis zum Ende der Page geschrieben. **Bild 5** zeigt den Aufbau einer PostgreSQL Page von 8 KByte.

Jeder Index für eine Tabelle in PostgreSQL verweist über TIDs auf die in der Heap-Tabelle gespeicherten Daten. Die TIDs können Sie sich wie einen Row Identifier (RID) im SQL Server vorstellen, mit denen RID-Lookups auf Heap-Tabellen durchgeführt werden, wenn nicht sämtliche angeforderten Spalten im referenzierten Non-Clustered-Index vorhanden sind.



Komplexe PostgreSQL-Abfrage, die grafisch aufbereitet wurde (**Bild 4**)



Schematischer Aufbau einer 8-KB-Page von PostgreSQL (Bild 5)

Und das gleiche Konzept trifft auch auf PostgreSQL zu: Die Indexing-Engine muss für jede TID, die von einem Index zurückgeliefert wird, einen Lookup gegen die Heap-Tabelle durchführen, um den gesuchten Datensatz zu lesen. Dies geschieht aus zwei unterschiedlichen Gründen:

- Es wurden in der SQL-Abfrage Spalten referenziert, die nicht Bestandteil des B-Tree-Indexes sind.
- Es muss eine Validierung in der Heap-Tabelle durchgeführt werden, welche Datensatzversion für die aktuelle Transaktion gültig ist.

Sehen wir uns hier den zweiten Punkt im Detail an. Wie bereits erwähnt, implementiert PostgreSQL ein MVCC-System, mit dessen Hilfe mehrere Datensatzversionen in der Heap-Tabelle gespeichert werden. Über die beiden internen Spalten *xmin* und *xmax* wird definiert, von wann bis wann eine Datensatzversion für Transaktionen sichtbar ist.

Führen Sie nun beispielsweise ein *UPDATE* auf einem Datensatz durch, wird eine neue Datensatzversion erzeugt, und jeder Index benötigt für diese neue Datensatzversion einen entsprechenden Eintrag – auch wenn eine Spalte aktualisiert wurde, die nicht indiziert wurde!

Der Index selbst speichert keine Versionsinformationen, diese befinden sich nur in der Heap-Tabelle in den Spalten *xmin* und *xmax*. Daher können beim Zugriff auf einen Index mehrere TIDs zurückgeliefert werden, die auf unterschiedliche Datensatzversionen verweisen.

Durch den zusätzlichen Zugriff auf die Heap-Tabelle werden letztendlich die Datensatzversionen verworfen, die innerhalb der aktuellen Transaktion nicht sichtbar sind (abhängig vom gewählten Transaction Isolation Level).

Daraus folgt, dass in PostgreSQL Lookup-Operationen von Indizes auf Heap-Tabellen immer stattfinden müssen. Sie werden im Ausführungsplan jedoch nicht explizit ausgewiesen, da sie Bestandteil der Architektur sind. Eine gewisse Erleichterung schafft hierbei die Visibility Map (VM), auf die ich in einem späteren Artikel im Zusammenhang mit den Details von MVCC noch näher eingehen werde.

Datenzugriffsmethoden

Die grundlegendste Methode für den Datenzugriff in PostgreSQL ist der Sequential Scan Operator, der dem Table Scan

Operator im SQL Server entspricht. Von ihm wird die gesamte Heap-Tabelle Seite für Seite gelesen und dabei werden eventuelle Filterprädikate ausgewertet, um zu bestimmen, ob ein Datensatz zum Ergebnis gehört. In Listing 2 wird dazu eine neue Tabelle mit 100 000 Datensätzen erstellt. Danach werden einige Abfragen ausgeführt, die zu einem Sequential Scan Operator im Ausführungsplan führen. Bild 6 zeigt die drei Estimated-Execution-Pläne, die jeweils einen Sequential Scan auf *TestTable1* durchführen. Der Sequential Scan Operator weist ein ähnliches Verhalten wie der Table Scan Operator im SQL Server auf: Er skaliert nicht proportional zur Datenmenge. Mit einer zunehmenden Anzahl von Seiten in einer Heap-Tabelle verlängert sich die Ausführungsdauer, weil jede Page gelesen und verarbeitet werden muss.

Abhilfe schafft hier nur die Definition eines Indexes. Über einen Index und dessen B-Tree-Struktur können Datensätze effizienter gefunden und Abfragen beschleunigt werden. Im folgenden Code-Schnipsel wird ein neuer Index angelegt und in unterschiedlichen Ausführungsplänen verwendet:

```
-- Create an index on the first column
CREATE INDEX idx_TestTable1_Col1
```

Listing 2: Sequential Scan

```
-- Create a new table
CREATE TABLE TestTable1
(
    Col1 INT,
    Col2 TEXT,
    Col3 BOOLEAN
);

-- Insert some test data
INSERT INTO TestTable1(Col1, Col2, Col3)
SELECT
    s.id,
    chr((32+random()*94)::integer),
    random() < 0.01
FROM generate_series(1,100000) AS s(id)
ORDER BY random();

-- Sequential Scan
EXPLAIN
SELECT * FROM TestTable1;

-- Sequential Scan + Aggregate
EXPLAIN
SELECT COUNT(*) FROM TestTable1;

-- Filter + Sequential Scan
EXPLAIN
SELECT * FROM TestTable1
WHERE Col1 < 59000;
```

```
ON TestTable1(Col1);
ANALYZE TestTable1;
```

```
-- This query just returns 1
-- record, and therefore the Query
-- Planner chooses an "Index Scan".
```

```
EXPLAIN
SELECT * FROM TestTable1
WHERE Col1 = 1;
```

Nach Anlage des neuen Indexes wird über den Befehl *ANALYZE* sichergestellt, dass die Statistiken zur betreffenden Tabelle aktuell sind, damit der Query Planner den neuen Index auch berücksichtigt. In **Bild 7** sehen Sie den erzeugten Ausführungsplan.

Der Index Scan Operator ist vergleichbar mit dem Non-Clustered Index Seek Operator im SQL Server. In PostgreSQL gibt es keinen separaten Seek Operator, da diese Funktionalität im Index Scan Operator implementiert ist. Dieser kann im besten Fall genau einen Datensatz zurückliefern, oder im schlechtesten Fall auch sämtliche Datensätze einer Tabelle.

Enthält eine Abfrage mehrere Bedingungen oder Filter, setzt PostgreSQL häufig auf den Bitmap Index Scan Operator – eine Technik, die effiziente Index-Zugriffe mit effizienten Datenzugriffen kombiniert.

Dabei erstellt PostgreSQL zunächst eine Bitmap-Maske – eine komprimierte Darstellung aller Datensätze, die die Abfragebedingungen erfüllen. Diese Bitmap-Maske wird mithilfe der relevanten Indizes aufgebaut. Statt jeden Datensatz einzeln abzurufen, kann PostgreSQL durch die Bitmap-Maske gezielt große Datenblöcke auf einmal laden und verarbeiten. Diese Vorgehensweise reduziert Random-I/O-Zugriffe und verbessert die Performance von Abfragen erheblich.

Besonders vorteilhaft ist dieses Verfahren bei großen Tabellen mit mehreren Filterbedingungen – zum Beispiel, wenn nach dem Alter und gleichzeitig nach dem Standort eines

Results Messages	
QUERY PLAN	
1	Seq Scan on testtable1 (cost=0.00..1443.00 rows=100000 width=7)
QUERY PLAN	
1	Aggregate (cost=1693.00..1693.01 rows=1 width=8)
2	→ Seq Scan on testtable1 (cost=0.00..1443.00 rows=100000 width=0)
QUERY PLAN	
1	Seq Scan on testtable1 (cost=0.00..1693.00 rows=58944 width=7)
2	Filter: (col1 < 59000)

Drei Estimated-Execution-Pläne, die jeweils einen Sequential Scan auf TestTable1 durchführen (**Bild 6**)

Results Messages	
QUERY PLAN	
1	Index Scan using idx_testtable1_col1 on testtable1 (cost=0.29..8.31 rows=1 width=7)
2	Index Cond: (col1 = 1)

Ausführungsplan zum Index-Scan (**Bild 7**)

Kunden gefiltert wird. In solchen Fällen kann der Bitmap Index Scan mehrere Indexzugriffe bündeln und effizienter gestalten.

Der SQL Server bietet kein direktes Pendant zum Bitmap Index Scan an, nutzt aber eine ähnliche Technik namens Bitmap Filtering innerhalb von parallelen Ausführungsplänen. Obwohl dies in bestimmten Szenarien ähnliche Vorteile bringt, ist es nicht so flexibel oder universell einsetzbar wie der Bitmap Index Scan in PostgreSQL.

Insbesondere wenn eine Abfrage mehrere Index Scans erfordern würde, sorgt der Bitmap Index Scan für eine optimierte und ressourcenschonende Ausführung. **Listing 3** zeigt eine Abfrage, die zur Verwendung eines Bitmap Index Scan Operators führt. Die Abfrage liefert auf Basis des Filters 199 Datensätze zurück, und daher entscheidet sich der Query Planner für einen Bitmap Index Scan Operator in Kombination mit einem Bitmap Heap Scan Operator. Diese beiden Operatoren verwendet PostgreSQL immer gemeinsam; Sie werden sie niemals alleine in einem Ausführungsplan vorfinden.

Wie Sie in **Bild 8** sehen, baut der Bitmap Index Scan Operator zunächst die Bitmap-Maske im Hauptspeicher auf. Dafür werden die relevanten Pages unter Zuhilfenahme des zuvor angelegten Index ermittelt.

Anschließend werden diese Pages über den Bitmap Heap Scan Operator einmalig gelesen und die relevanten Datensätze zurückgeliefert. Dank des Bitmap Heap Scan Operators muss eine Page nur einmal gelesen und verarbeitet werden, auch wenn mehrere Datensätze dieser Page verarbeitet werden müssen.

Listing 3: Bitmap Index Scan 1/2

```
-- This query returns 199 records, and therefore
-- "Bitmap IndexScan" the Query Planner chooses
-- an operation in combination with a
-- "Bitmap Heap Scan".
```

```
EXPLAIN
SELECT * FROM TestTable1
WHERE Col1 < 200;
```

Auch wenn mehrere Filterbedingungen gemeinsam evaluiert werden müssen, können Bitmap Index Scan und Bitmap Heap Scan verwendet werden. Die Abfrage in [Listing 4](#) formuliert zwei Filterbedingungen, wobei für jede Bedingung ein Index vorhanden ist. Im Rahmen des Ausführungsplans wird auf jeden Index über einen Bitmap Index Scan Operator zugegriffen, wobei jeder Operator eine eigene Bitmap-Maske erzeugt.

Die beiden Bitmap-Masken werden anschließend über einen *BitmapAnd*-Operator kombiniert und schlussendlich alle relevanten Pages genau ein Mal vom Bitmap Heap Scan Operator verarbeitet.

Bei diesem Vorgehen handelt es sich um eine klassische Index-Intersection, die im SQL Server durch physische Join-Operatoren (*Nested Loop*, *Merge Join*, *Hash Join*) realisiert wird. [Bild 9](#) zeigt den generierten Ausführungsplan.

PostgreSQL bietet mit dem Index Only Scan noch eine weitere Datenzugriffsmethode an, die vergleichbar mit einem Covering Non-Clustered Index im SQL Server ist. Ein Index Only Scan wird vom Query Planner immer dann durchgeführt, wenn sämtliche von einer Abfrage referenzierten Spalten Bestandteil eines Indexes sind. Wie bereits erwähnt, muss PostgreSQL trotzdem auf die Heap-Tabelle zugreifen, um festzustellen, ob es sich um eine Datensatzversion handelt, die für die aktuelle Transaktion sichtbar ist, da ein Index diesbezüglich keine Informationen enthält.

Um dieses Verhalten zu verbessern, implementiert PostgreSQL eine sogenannte Visibility Map (VM) für jede Heap-Tabelle. Auch diese Funktionalität wird in einem späteren Artikel noch näher erläutert. [Listing 5](#) zeigt eine Abfrage, die zu einem Index-Only-Scan führt. Wenn Sie sich den dazu generierten Ausführungsplan in [Bild 10](#) ansehen, sehen Sie beim Index Only Scan Operator auch die Eigenschaft *Heap Fetches*. Diese informiert darüber, wie oft ein Lookup in der Heap-Tabelle durchgeführt wurde. Im Beispiel wurden keine Lookups durchgeführt (*Heap Fetches: 0*), weil diese durch die Verwendung der Visibility Map vermieden werden konnten.

Sobald Sie jedoch Datensätze ändern, wird auch die Visibility Map geändert, wodurch wieder *Heap Fetches* auftreten, siehe [Bild 11](#).

Statistiken

Ähnlich wie beim SQL Server nutzt der Query Planner von PostgreSQL Statistiken, um bereits während der Kompilierung einer Abfrage abschätzen zu können, wie viele Datensätze eine Abfrage finden wird.

PostgreSQL legt hierzu Statistiken auf Tabellen-Ebene und auch auf Spalten-Ebene an. Generelle Statistiken über die Tabelle finden Sie in der System-View *pg_class*, die bereits im zurückliegenden PostgreSQL-Artikel [1] verwendet wurde. Im Rahmen von Statistiken bietet diese System-View folgende Informationen:

● Listing 4: Bitmap Index Scan 2/2

```
-- Create an additional index on the second
-- column
CREATE INDEX idx_TestTable1_Col2 ON
    TestTable1(Col2);
ANALYZE TestTable1;

-- This query returns 3 records, and each search
-- predicate is indexed by a separate index.
EXPLAIN
SELECT * FROM TestTable1
WHERE Col1 < 200
AND Col2 = 'k';
```

● Listing 5: Index Only Scan

```
-- This query references only columns which are
-- part of the previous created index. Therefore,
-- the Query Planner uses an "Index Only Scan"
-- operation in the query plan, because the used
-- index is for this query a "Covering Index".
EXPLAIN (ANALYZE)
SELECT Col1 FROM TestTable1
WHERE Col1 < 6000;
```

- Anzahl der Datensätze in einer Tabelle (Spalte *reltuples*),
- Anzahl der Pages einer Tabelle (Spalte *relpages*),
- Anzahl der Pages, die über die Visibility Map sichtbar sind (Spalte *relallvisible*). ▶

Results	Messages
QUERY PLAN	
1	Bitmap Heap Scan on testtable1 (cost=17.62..25.25 rows=2 width=7)
2	Recheck Cond: ((col1 < 200) AND (col2 = 'k'::text))
3	-> BitmapAnd (cost=17.62..17.62 rows=2 width=0)
4	-> Bitmap Index Scan on idx_testtable1_col1 (cost=0.00..5.73 rows=191 width=0)
5	Index Cond: (col1 < 200)
6	-> Bitmap Index Scan on idx_testtable1_col2 (cost=0.00..11.64 rows=980 width=0)
7	Index Cond: (col2 = 'k'::text)

Die Abfrage aus [Listing 3](#) generiert diesen Ausführungsplan ([Bild 8](#))

Results	Messages
QUERY PLAN	
1	Bitmap Heap Scan on testtable1 (cost=5.77..357.08 rows=191 width=7)
2	Recheck Cond: (col1 < 200)
3	-> Bitmap Index Scan on idx_testtable1_col1 (cost=0.00..5.73 rows=191 width=0)
4	Index Cond: (col1 < 200)

Der zu [Listing 4](#) generierte Ausführungsplan ([Bild 9](#))

Results	Messages
	QUERY PLAN
1	Index Only Scan using idx_testtable1_col1 on testtable1 (cost=0.29..171.17 rows=5879 width=4) (actual time=0.036..0.548 rows=5999 loops=1)
2	Index Cond: (col1 < 6000)
3	Heap Fetches: 0
4	Planning Time: 0.130 ms
5	Execution Time: 0.728 ms

Heap Fetches: 0 – es wurden keine Lookups in der Heap-Tabelle durchgeführt (Bild 10)

Results	Messages
	QUERY PLAN
1	Index Only Scan using idx_testtable1_col1 on testtable1 (cost=0.29..174.50 rows=6069 width=4) (actual time=0.064..0.753 rows=5999 loops=1)
2	Index Cond: (col1 < 6000)
3	Heap Fetches: 225
4	Planning Time: 0.483 ms
5	Execution Time: 0.932 ms

Nach dem Ändern von Daten treten wieder Heap Fetches auf, hier 225 (Bild 11)

Die folgende Abfrage zeigt, wie Sie mithilfe dieser System-View weitere Informationen über eine Tabelle ermitteln:

```
SELECT
  oid,
  relname,
  reltuples,
  relpages,
  relallvisible
```

```
FROM pg_class
WHERE relname = 'testtable1';
```

Hier werden statistische Informationen für die Tabelle *TestTable1* ermittelt. Führen Sie ein einfaches *SELECT*-Statement ohne weitere Filterung gegen die Tabelle aus, wird die Spalte *reltuples* für das Abschätzen der Datenmenge herangezogen, vergleiche Bild 12. Wichtig ist allerdings zu wissen, dass die Informationen der System-View *pg_class* bei *INSERT*-

● Listing 6: Cardinality Estimation & ANALYZE

```
-- Create a copy of the table definition
CREATE TABLE TestTable1_copy (LIKE TestTable1);

-- The column "reltuples" is -1, because ANALYZE
-- wasn't yet done on the table
SELECT
  oid,
  relname,
  reltuples,
  relpages,
  relallvisible
FROM pg_class
WHERE relname = 'testtable1_copy';

-- The Cardinality Estimation is wrong, because
-- the Query Planner doesn't know it yet
-- better...
EXPLAIN
SELECT * FROM TestTable1_copy;

-- Let's run the ANALYZE command on the table
ANALYZE TestTable1_copy;

-- Now we have a Cardinality Estimation of
-- 1 row
EXPLAIN
SELECT * FROM TestTable1_copy;

-- Copy 100000 records into the new table
INSERT INTO TestTable1_copy
SELECT * FROM TestTable1;

-- The Cardinality Estimation is again wrong
EXPLAIN
SELECT * FROM TestTable1_copy;

-- Let's run the ANALYZE command on the table
ANALYZE TestTable1_copy;

-- Now we have a Cardinality Estimation of
-- 100000 rows
EXPLAIN
SELECT * FROM TestTable1_copy;
```

Results Messages					
	oid	relname	reltuples	relpages	relallvisible
1	25184	testtable1	100000	443	436

QUERY PLAN	
1	Seq Scan on testtable1 (cost=0.00..1443.00 rows=100000 width=7)

Der Spalte **reltuples** der Statistikdatei `pg_class` entnimmt der Ausführungsplan die Anzahl der Zeilen (Bild 12)

oder *DELETE*-Operationen nicht aktualisiert werden. Die Daten werden nur dann neu berechnet, wenn einer der folgenden Befehle ausgeführt wird:

- *ANALYZE* (Neuberechnen der Statistiken),
- *VACUUM* (Löschen nicht benötigter Datensatzversionen),
- *CLUSTER* (physisches Sortieren der Tabellendaten),
- *CREATE INDEX* (Anlage eines neuen Index),
- *REINDEX* (Rebuild eines bestehenden Index).

In Listing 6 finden Sie ein ausführliches Beispiel zu dieser Thematik. Wie Sie darin erkennen können, liefert die Schätzung der Datenmenge (*Cardinality Estimation*) nur dann richtige Abschätzungen, wenn die Statistiken der Tabelle von einem der oben genannten Befehle neu berechnet wurden. Weiterführende Informationen zum Befehl *ANALYZE* finden Sie im Web unter [3].

Neben den Statistiken auf Tabellen-Ebene bietet PostgreSQL auch Statistiken für einzelne Spalten einer Tabelle an, unabhängig davon, ob für eine Spalte ein Index definiert ist oder nicht.

Diese Statistiken können Sie über die System-View `pg_stats` einsehen. Deren wichtigste Spalten sind:

- *n_distinct*: Anzahl der eindeutigen Spaltenwerte,
- *most_common_vals*: am häufigsten vorkommende Werte,
- *histogram_bounds*: Histogramm der Werteverteilung.

Listing 7: GROUP BY & n_distinct

```
SELECT
    c.reltuples,
    s.n_distinct,
    round(c.reltuples / s.n_distinct) AS rows
FROM pg_class c
JOIN pg_stats s ON c.relname = s.tablename
WHERE
    s.tablename = 'flight'
    AND s.attname = 'departure_airport';

-- "n_distinct" will be used for the Cardinality
-- Estimation of the GROUP BY clause.
EXPLAIN
SELECT
    departure_airport,
    COUNT(*)
FROM flight
GROUP BY departure_airport;
```

Sehen wir uns diese Informationen ein wenig näher an. Für die folgenden Beispiele wird die Datenbank *postgres_air* verwendet, die unter [4] zum Download bereit steht. Die Anzahl der eindeutigen Spaltenwerte wird zum Beispiel für die Abschätzung von *GROUP-BY*-Klauseln verwendet, vergleiche Listing 7. In Bild 13 sehen Sie die Abschätzungen innerhalb des dafür generierten Ausführungsplans.

Ist die Anzahl der eindeutigen Werte falsch, können Sie diese in PostgreSQL sogar auf Spaltenebene setzen. Listing 8 zeigt, wie das geht. Wichtig ist dabei, dass Sie nach dem Befehl *ALTER TABLE ALTER COLUMN* die Tabelle mit *ANALYZE* prüfen lassen, damit die neu gesetzten Werte auch in den Statistiken berücksichtigt werden. Ist die Datenverteilung in einer Spalte nicht gleichmäßig – normalerweise ist das der ►

Results Messages			
	reltuples	n_distinct	rows
1	683178	647	1056

QUERY PLAN	
1	Finalize GroupAggregate (cost=1000.45..11450.74 rows=647 width=12)
2	Group Key: departure_airport
3	-> Gather Merge (cost=1000.45..11437.80 rows=1294 width=12)
4	Workers Planned: 2
5	-> Partial GroupAggregate (cost=0.42..10288.42 rows=647 width=12)
6	Group Key: departure_airport
7	-> Parallel Index Only Scan using flight_departure_airport on flight (cost=0.42..8858.66 rows=284658 width=4)

Ausführungsplan mit Abschätzungen für die Abfrage aus Listing 7 (Bild 13)

Listing 8: Ändern von n_distinct

```
-- Alter the "n_distinct" value
ALTER TABLE flight
ALTER COLUMN departure_airport
SET (n_distinct = 647);

-- Recalculate the statistics, so that we see the
-- new "n_distinct" value.
ANALYZE flight;

-- n_distinct contains now the custom value
SELECT
    attname,
    n_distinct
FROM pg_stats
WHERE tablename = 'flight'
AND attname = 'departure_airport';
```

Fall –, speichert PostgreSQL die häufigsten Spaltenwerte mit ihrer Häufigkeit in den sogenannten Most Common Values (MCV).

Auf Basis dieser Informationen kann der Query Planner die Cardinality Estimation für Filter-Prädikate durchführen. Die entsprechenden Werte finden Sie in der Spalte *most_common_vals* und in *most_common_freqs* ihre Häufigkeit. Bis zu 100 Werte speichert PostgreSQL hier (Standardeinstellung).

Listing 9 zeigt ein einfaches Beispiel dafür, wie diese Informationen für die Cardinality Estimation herangezogen werden. Darin wird über die Funktion *array_position* der Index eines Eintrags aus dem Array *most_common_freqs* ermittelt und der zurückgelieferte Wert mit der Anzahl der Datensätze (*reltuples*) multipliziert. Das Ergebnis ist die Abschätzung der zurückgelieferten Datensätze beim Operator Bitmap Index Scan im Ausführungsplan, vergleiche Bild 14.

Da in der Standardeinstellung maximal 100 Werte als Most Common Values aufgenommen werden, erfolgt die Darstellung der Verteilung aller weiteren Werte in einem Histogramm. Die verbleibenden Werte werden dabei gleichmäßig auf 100 Buckets verteilt.

Die Spalte *histogram_bounds* enthält die Grenzwerte der einzelnen Buckets. Zwischen den einzelnen Buckets befinden sich durchschnittlich immer gleich viele Datensätze. So erhält der Query Planner eine ungefähre Vorstellung über die Datenverteilung der restlichen Spaltenwerte.

Fazit

Dieser Artikel hat sich mit Performance-Tuning-Techniken in PostgreSQL beschäftigt. Nach einem Überblick der Ausführungspläne haben Sie die Datenzugriffsmethoden kennengelernt und es wurden einige größere Unterschiede zum SQL Server herausgearbeitet.

Abschließend wurden die Statistiken vorgestellt, mit deren Hilfe der Query Planner Ausführungspläne erzeugen kann. Auch hier gibt es sehr große Unterschiede zum SQL Server, da alle Spalten einer Tabelle entsprechende Statistiken besitzen, unabhängig davon, ob sie indiziert sind oder nicht. ■

Listing 9: Most Common Values

```
-- Most Common Values - Unnested
SELECT
    s.attname,
    c.reltuples,
    s.most_common_vals,
    s.most_common_freqs,
    unnest(s.most_common_vals::text::text[]) AS
        most_common_value,
    unnest(s.most_common_freqs::text::text[]) AS
        most_common_value_frequency
FROM pg_stats s
JOIN pg_class c ON s.tablename = c.relname
WHERE tablename = 'flight'
AND attname = 'departure_airport';

-- Most Common Values
SELECT
    s.attname,
    c.reltuples,
    s.most_common_vals,
    s.most_common_freqs,
    ROUND(c.reltuples * s.most_common_freqs[
        array_position((s.most_common_vals::text::text[]),
            'VIE')]) AS estimation_vie,
    ROUND(c.reltuples * s.most_common_freqs[
        array_position((s.most_common_vals::text::text[]),
            'FRA')]) AS estimation_fra
FROM pg_stats s
JOIN pg_class c ON s.tablename = c.relname
WHERE tablename = 'flight'
AND attname = 'departure_airport';

-- Estimation on the Bitmap Index Scan: 2163
EXPLAIN
SELECT * FROM flight
WHERE departure_airport = 'VIE';

-- Estimation on the Bitmap Index Scan: 9838
EXPLAIN
SELECT * FROM flight
WHERE departure_airport = 'FRA';
```

Results		Messages					
	attname	reltuples	most_common_vals	most_common_freqs	estimation_vie	estimation_fra	
1	departure_airport	683178	{ATL,ORD,DEN,LAX,CDG,LAS,JFK,PHX,FRA,IAD,PHL,SFO,SEA...	{0.022666667,0.0191,0.0181,0.017833333,0.0159,0.0156...	2163	9838	

QUERY PLAN	
1	Bitmap Heap Scan on flight (cost=25.19..5018.17 rows=2163 width=71)
2	Recheck Cond: (departure_airport = 'VIE'::bpchar)
3	→ Bitmap Index Scan on flight_departure_airport (cost=0.00..24.65 rows=2163 width=0)
4	Index Cond: (departure_airport = 'VIE'::bpchar)

QUERY PLAN	
1	Bitmap Heap Scan on flight (cost=112.67..9276.16 rows=9838 width=71)
2	Recheck Cond: (departure_airport = 'FRA'::bpchar)
3	→ Bitmap Index Scan on flight_departure_airport (cost=0.00..110.21 rows=9838 width=0)
4	Index Cond: (departure_airport = 'FRA'::bpchar)

Abschätzung der bei einem Bitmap Index Scan zurückzugebenden Datensätze (Bild 14)

- [1] Klaus Aschenbrenner, *Open-Source-Datenbank, PostgreSQL*, dotnetpro 3/2025, Seite 26 ff., www.dotnetpro.de/A2503PostgreSQL
- [2] *Cluster-Befehl*, www.dotnetpro.de/SL2504-05PostgreSQL1
- [3] *Optimizing Query Performance with Statistics Updates in PostgreSQL and Microsoft SQL Server*, www.dotnetpro.de/SL2504-05PostgreSQL2
- [4] *Datenbank postgres_air*, www.dotnetpro.de/SL2504-05PostgreSQL3



Klaus Aschenbrenner

berät Unternehmen in Europa beim Einsatz des Microsoft SQL Server und beschäftigt sich mit Windows-Programmierung und .NET. Zweimal zeichnete in Microsoft für sein Engagement als Microsoft MVP aus. Er ist zu erreichen über www.SQLpassion.at.

dnpCode A2504-05PostgreSQL

DEV ACADEMY

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



1 TAG

INHOUSE ODER REMOTE

Technische Schulden - Chancen, Probleme, Lösungsansätze

TRAINING!

- Was sind technische Schulden und welche Wirkung haben sie?
- Ursachen und Konsequenzen von technischen Schulden
- Verhinderung und Abbau von Technischen Schulden
- Kennenlernen von Methoden, Modellen und Werkzeugen, z.B. Story Splitting, Delegation Poker, Projekt versus Produkt
- Technische Schulden in der DoD und dem Refinement



Stefan Mintert
Trainer

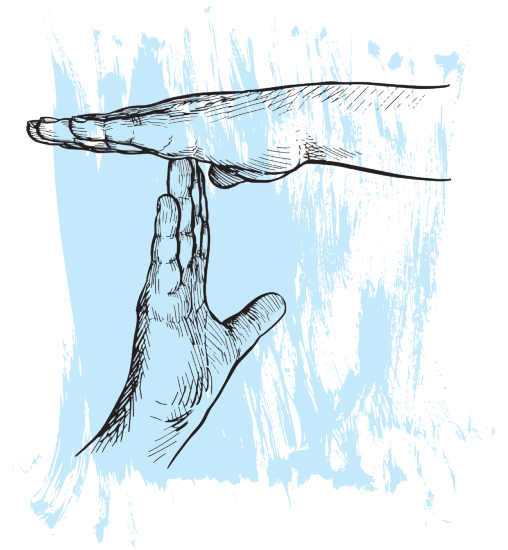
TIME-OUT

Alle für alle oder alle für mich?

Eine Atmosphäre des Vertrauens schafft die Grundlage für den gemeinsamen Erfolg.



Time-out Start



Bei mir liegen zwei Bücher auf dem Nachttisch: „Erfolgreich denken: Lernen Sie wie Millionäre zu denken und Grenzen zu überwinden“ von Bodo Schäfer und „Die Kunst des Schenkens: Von der Lust, Freude zu bereiten“ von Fabian Vogt. Welches soll ich zuerst lesen? Welches würden Sie wählen? Ich habe mir das erste geschnappt, denn schenken kann ich ja erst, wenn ich etwas habe. Außerdem ist Erfolg einfach sexy, das zieht.

Natürlich liegen die Bücher nicht wirklich auf meinem Nachttisch. Sie sind Metaphern für zwei unterschiedliche Haltungen: Bei der ersten geht es um das „Ich“ und darum, selbst besser und stärker zu werden, bei der zweiten um das „Wir“ und darum, andere glücklich (und besser) zu machen. Es wird Sie nicht wundern, dass die Literatur zum ersten Thema deutlich umfangreicher ist. Die Suche nach dem Stichwort „Teilen“ warf als Erstes Kinderbücher über Sankt Martin aus. Und dann lange nichts. Das „Wir“ und das „Für andere“ stehen nicht hoch im Kurs.

Dabei ist das Teilen Wesenskern unserer abendländischen christlichen Tradition: „Ich für euch“ sind die Worte, die Jesus beim Abendmahl gesprochen hat. Selbsthingabe statt Selbstbereicherung ist die Botschaft des christlichen Glaubens. Zumindest in der Theorie.

Im Wechselspiel der Hormone

Wenn Sie eher an Wissenschaft und Biochemie glauben, versuche ich es anders. Die Natur steuert über Hormone menschliches Verhalten; mit Dopamin fördert sie Leistung. Es wird ausgeschüttet, wenn wir erfolgreich sind, etwas geleistet haben, Ziele erreicht sind. Dopamin macht glücklich, denn es motiviert, zu neuen Ufern aufzubrechen. Der Nachteil: Man wird leicht süchtig. Denn glücklich fühlen wir uns nur, wenn der Dopamin-Spiegel steigt. Haben wir Erfolg, wollen wir noch mehr; sind wir mächtig, wollen wir mehr Macht. Das geht so weiter, bis wir den Gipfel erreicht haben. Dann suchen wir den nächsten Kick, und der Kreislauf beginnt von Neuem.

Es gibt ein weiteres Hormon, das uns ebenfalls glücklich macht, aber anders wirkt: das Oxytocin. Es wird immer dann ausgeschüttet, wenn Menschen zusammenarbeiten, sich helfen und gegenseitig unterstützen. Im Team lassen sich bessere Ergebnisse erzielen, das Team ist immer stärker als der Einzelne. Und trotzdem muss sich der Einzelne anstrengen. Das Wechselspiel der beiden Hormone ist es, was das Überleben unserer Art garantiert hat.

Oxytocin wirkt langsamer und beständiger als Dopamin. Bei jeder Umarmung, bei jeder Berührung wird ein bisschen mehr ausgeschüttet – die Natur will nicht nur schnellen Erfolg, sondern auch langfristige Bindung. Wobei „Die Natur will das“ eine Metapher ist. Die Natur ist keine eigene Größe mit einem Willen. Man sollte also davon sprechen, dass dieses Muster von individuellem Erfolg und Teamarbeit bis heute unser Überleben am effizientesten gesichert hat.

Oxytocin wird nicht nur ausgeschüttet, wenn wir in das Gesicht des Menschen schauen, dem wir gerade das Leben gerettet haben; es wird auch dann ausgeschüttet, wenn wir zuschauen, wie ein Mensch dem anderen hilft. Unentgeltlich. Man könnte auch sagen, es ist das Schnulzen-Hormon, von dem Hollywood lebt.

Es gibt ein drittes Hormon, das Cortisol, das sogenannte Stresshormon. Immer dann, wenn wir uns schützen müssen, weil wir uns bedroht fühlen, wird es ausgeschüttet. Es bereitet uns darauf vor, dass jetzt eine Leistung zu erbringen ist, es fordert uns sozusagen heraus. An sich ist das nichts Schlechtes, doch ein dauerhaft hoher Cortisol-Spiegel macht krank und senkt die Leistungsfähigkeit. Etwas Stress hin und wieder ist also kein Problem. Aber dauerhafter Stress ist nicht förderlich für den Erfolg des Einzelnen und der Gruppe.

So weit die Biochemie. Doch lassen Sie uns über die moderne Arbeitswelt reden. Denn wie wir miteinander umgehen, wie wir unsere Mitarbeiter führen oder von ihnen geführt werden, lässt sich anhand dieser drei Hormone gut veranschaulichen.

- Den Erfolgskick haben wir, wenn wir ein Ziel erreichen (Dopamin). Er ist zwar motivierend, aber kurz. Er lebt von Zahlen, Siegen und Macht. Nichts Schlechtes, aber auch nichts, worauf man sein Leben aufbauen sollte.
- Stress spüren wir, wenn wir bedroht sind (Cortisol). Etwa wenn wir unter Leistungsdruck stehen, in einem Klima des Misstrauens arbeiten und es mehr darum geht, uns selbst zu schützen, als darum, etwas Produktives für die Firma zu erreichen. Darf sein, aber nicht auf Dauer.
- Wahres Glück (Oxytocin) spüren wir, wenn wir mit unseren Kollegen im Austausch sind, wenn wir ihnen helfen, wir uns bei ihnen sicher fühlen. Dazu müssen wir sie kennen, Vertrauen aufbauen und einander verstehen.

Der Ring der Sicherheit

Ich muss noch einmal die Disziplin wechseln, bevor wir im Büro weitermachen. Die meiste Zeit unserer Menschheitsgeschichte haben wir in Gruppen von 150 Personen (Sippe) zusammengelebt und uns gemeinsam gegen äußerst widrige Umstände behauptet. Wir haben Lasten geteilt und Aufgaben verteilt, um gemeinsam zu überleben. Ein Einzelgänger wäre gnadenlos untergegangen. Jeder in der Gruppe konnte sich in seiner Sippe sicher fühlen und sich auf sie verlassen.

Ein ähnliches Konstrukt finden wir beim Militär. Dort heißt es nicht Sippe, sondern Kompanie: eine Einheit aus ebenfalls 100 bis 150 Personen, die sich gegen extreme Herausforderungen wehrt. Das funktioniert nur, wenn sich jeder auf den anderen verlassen kann, man sich kennt und vertraut und bereit ist, für den anderen alles zu geben. Innerhalb dieser Gruppe gibt es einen Ring der Sicherheit, einen Bereich, in dem man seine Waffen ablegen kann. Der Ring wird nach außen hart verteidigt: Kein Feind kommt rein. Und ein Freund auch nur nach langer Aufnahmeprüfung, hartem Training, dem Test des richtigen Mindsets und oft einem Initiationsritus.

Kommt Ihnen das jetzt zu archaisch vor? Zu brutal? Zu weit weg von der modernen Welt? Lassen Sie mich ein paar Analogien ziehen, und überlegen Sie selbst, wo und wie Sie sich positionieren wollen.

In einem Unternehmen oder einer Gesellschaft, die so groß sind, dass die „obere Etage“ nicht weiß, wie es „denen da unten“ geht, ist viel Potenzial für Cortisol. Hier muss ich mich selbst schützen, muss sehen, wo ich bleibe. Denn wenn es mit der Wirtschaft bergab geht: Wer kann mir garantieren, dass ich nicht unter die Räder komme? Schnell werden einfach mal 1000 Stellen gestrichen (eine Zahl, keine Personen, mit denen man mitfühlen kann).

In einer Welt, in der der Aktiengewinn das alleinige Ziel des Unternehmens ist, gibt es keinen Ring der Sicherheit. Da ist der Cortisol-Spiegel dauerhaft hoch, was Kreativität, Mut und Innovation verhindert.

Wenn wir Cortisol niedrig und gleichzeitig Oxytocin auf dauerhaft hohem Niveau halten wollen und es hin und wieder Dopamin-Schübe geben soll, ist es wichtig, dass die Menschen in der Firma sich kennen und vertrauen. Es sollte nicht so leicht sein, in die Firma zu hineinzukommen, aber wenn man sich bewiesen hat und den gleichen Spirit teilt, sollte man die Waffen ablegen können. Denn ab jetzt greift der Ring der Sicherheit.

Was könnte das sein? Es könnte die Ansage sein, dass niemand in der Firma entlassen wird, ganz egal wie stark die wirtschaftliche Schieflage ist. Denn diese steht man gemeinsam durch. Es ist besser, wenn jeder kurzfristig weniger bekommt, als wenn wenige nichts mehr bekommen. Die Menschen sind immer wichtiger als der Gewinn. Denn der Gewinn kommt, wenn Menschen konstruktiv zusammenarbeiten, eine gemeinsame Vision haben und mit Einsatz dabei sind. Es wird viel Energie frei, wenn man nicht das Gefühl hat, sich rechtfertigen zu müssen, wenn man zehn Minuten zu spät kommt, oder wenn man sich nicht mehr ärgert, weil man das Gefühl hat, der andere leiste zu wenig. Diese Energie stecken alle in das Unternehmensziel.

Es gibt ein schönes Beispiel, wie eine gute Fehlerkultur Innovation hervorbringt: Der Chemiker bei 3M, der an Klebstoff arbeiten sollte, hatte einen Misserfolg und nur einen schwach klebenden Stoff hergestellt. Statt sich Sorgen um seine Anstellung zu machen, erzählte er davon. Ein anderer Kollege hatte das Problem zu lösen, Zettel so an den Bildschirm zu kleben, dass sie keine Spuren hinterlassen. Seit diesem Zeitpunkt gibt es Post-It. Innovation kann in diesem geschützten Raum entstehen, wenn man offen miteinander umgeht und seine Fehler teilt.

Mit dieser Anregung will ich es hier bewenden lassen. Vielleicht hat es bei Ihnen ja „Klick“ gemacht. Aber bevor ich es vergesse: Das Buch, das tatsächlich auf meinem Nachttisch lag, heißt „Leaders eat last“ von Simon Sinek. Von ihm habe ich all diese Anregungen, und er wiederum hat vieles vom „Circle of Safety“ des U.S. Marine Corps abgeschaut. Und die wiederum haben ihre Ideen von ...

Aber lassen wir das, denn uns geht es ja nicht darum, wer den Erfolg verbucht, sondern darum, dass wir alle miteinander in Beziehung stehen und so gemeinsam Großartiges vollbringen können. Old Europe eben.

In diesem Sinne: Arbeiten Sie zusammen, schaffen Sie eine Vertrauensbasis, erreichen Sie das Unmögliche – dann kommt der Erfolg. Happy Coding! ■



Time-out End



Dipl.-Theol. Bernhard Pichler

war 20 Jahre 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

A2504-05Timeout

TECHNISCHE SCHULDEN IN DER SOFTWAREENTWICKLUNG

Eine Frage der Führung

Technische Schulden haben in Entstehung und Abbau vor allem mit einem Mangel an Führung zu tun. Womit noch nicht gesagt ist, wer verantwortlich ist und handeln sollte.

Technische Schulden sind allgegenwärtig. Das liegt auch daran, dass der Begriff mit mehreren Bedeutungen überladen ist. Dazu möchte ich hier nicht in die Tiefe gehen, weil das schon eine Reihe anderer dotnetpro-Artikel in den vergangenen Jahren erledigt hat. Doch für den weiteren Verlauf ist eine kleine Klassifikation hilfreich.

Der Begriff „technische Schulden“ geht auf Ward Cunningham zurück, der ihn Anfang der 1990er-Jahre geprägt hat. Bemerkenswert daran ist, dass Cunningham damals an einer Finanzsoftware arbeitete. Das Wort „Schulden“ wählte er, weil er der Meinung war, dass sein Auftraggeber damit etwas anfangen könne. Hätte er eine Software für den Gesundheitsbereich geschrieben, würden wir heute vielleicht über „technische Mangelernährung“ nachdenken oder, wäre sein damaliger Auftraggeber ein Autokonzern gewesen, vielleicht über „technischen Rost“. Doch es kam anders. Heute sprechen wir fortwährend von Schulden, obwohl die meisten von uns nicht mit einem FinTech-Unternehmen arbeiten und der Begriff oft gar keine gute Metapher darstellt.

Nehmen wir einmal an, dass wir agil an einem neuen Produkt arbeiten. Es soll ein einfach zu transportierendes Surfbrett sein. Mit unserem Team entwickeln wir einen besonderen Schaumstoff. Er ist stabil genug, um darin ein Scharnier verankern zu können, sodass wir unser späteres Surfbrett zusammenklappen können. Der Schaumstoff ist brilliant. Noch nie in der Geschichte der Menschheit hat jemand einen derart brillanten Schaumstoff entwickelt. Und umweltfreundlich ist er obendrein, denn er ist kompostierbar und wird aus Atommüll produziert; und die radioaktive Strahlung geht bei der Herstellung auf magische Art und Weise verloren. Besser geht es nicht.

Im ersten Review stellen wir dann fest, dass die ursprüngliche Anforderung „Es soll platzsparend sein“ ungenau war: Wir haben ein Inkrement entwickelt, das durch den Klappmechanismus mit einer geringen Grundfläche auskommt; die Stakeholder wünschen sich aber ein geringes Volumen. Also müssen wir den Schaumstoff aufblasbar machen. Das ist eine große Herausforderung, weil wir ihn dafür nicht entworfen haben. Dieses Delta meint Cunningham, wenn er von „technischen Schulden“ spricht. Er meint ausdrücklich nicht, dass schlechter Code geschrieben wird.

Im obigen Beispiel: Der Schaumstoff ist brilliant geworden. Makellos. Sollten wir in dieser Situation wirklich von technischen Schulden im heutigen Sinn sprechen? Ich finde, dass der Begriff hier nicht passt. Mir gefällt in diesem Fall der Ausdruck „agile Schulden“ etwas besser. Aber am liebsten be-

zeichne ich alle Probleme, über die wir hier sprechen, als „Kursabweichung“.

Das Team aus meinem Beispiel hat einen Kurs auf ein herausragendes Produkt eingeschlagen. Es hat handwerklich alles richtig gemacht. Doch während der Entwicklung zeigt sich, dass man durch eine ungenaue Anforderung oder eine fehlerhafte Kommunikation vom Kurs abgekommen ist, und zwar ohne es zu merken. Jetzt ist eine Kurskorrektur notwendig. Das klingt für mich viel netter als Schulden, die man zurückzahlen muss.

Der einzige Fall, in dem das Wort „Schulden“ aus meiner Sicht angemessen ist: Man möchte sich Zeit leihen und nimmt dafür übergangsweise eine geringere Qualität in Kauf. Will man die geringe Qualität wieder loswerden, muss man die geliehene Zeit zurückzahlen. Mit Zinsen, versteht sich.

Es gibt noch ein drittes Phänomen, das gerne unter technischen Schulden subsumiert wird: miserabel geschriebener Code. Doch auch für diesen gibt es gute Gründe. Zum Beispiel programmieren Anfänger in der Regel schlechter als erfahrene Entwickler, die das schon seit Jahrzehnten machen. Dieser „schlechte“ Code kann aber auch durchaus etwas Gutes haben: Er konnte mit dem verfügbaren Personal geschrieben werden. Wäre das nichts Gutes, warum sollte man es dann geschehen lassen?

Ein Aspekt dieser Angelegenheit wird gerne übersehen: Die Frage, was „schlechter Code“ ist, ist nicht pauschal zu beantworten. Denn irgendjemand hat ihn so geschrieben, wie er nun mal ist. Wenn der Urheber weiß, dass er nicht gut programmiert hat, wird er hoffentlich um Hilfe bitten; andernfalls hat das Team ein ganz anderes Problem.

Also kann man davon ausgehen, dass der Urheber mit seinem Programm einigermaßen zufrieden ist. Und das hat wichtige Vorteile: Der Urheber versteht das Programm und kann es weiterentwickeln. Das ist keine Selbstverständlichkeit. Ich habe bereits Teams erlebt, in denen ein großes Know-how-Gefälle festzustellen war.

Ein Team hatte einen externen „Experten“ ins Team geholt. Hätte man andere Experten gefragt, hätten sie zweifellos gesagt, dass Quellcode und Architektur, die der Externe geliefert hatte, nach „objektiven“ Maßstäben außerordentlich gut waren. Das Problem war nur: Niemand aus dem Team sah sich in der Lage, an dem Programmcode Änderungen vorzunehmen. In diesem Sinne war der Code schlecht wartbar. Die objektiven Maßstäbe gibt es schlicht nicht. Quellcode ist Kommunikation. Wenn ich ihn nicht verstehe, kann ich damit nichts anfangen.

Wer ist schuld an Schulden ...

Wenden wir uns der Frage zu, wer für technische Schulden verantwortlich ist. Genauer: Wer ist verantwortlich, dass sie überhaupt entstehen?

Den Anfang machen die „agilen Schulden“, also jene Aufwände, die dadurch notwendig werden, dass sich das Verständnis des Entwicklungsziels im Lauf der Zeit verändert (Bild 1). Auslöser sind die Wünsche, Ideen, Anforderungen und User Stories, die das Entwicklerteam bekommt. Sie kommen vom Produktmanagement, Product Owner oder irgendeiner anderen, ranghöheren Person.

Ganz offensichtlich sind die Softwareentwickler nicht dafür verantwortlich, dass diese Schulden angehäuft werden.

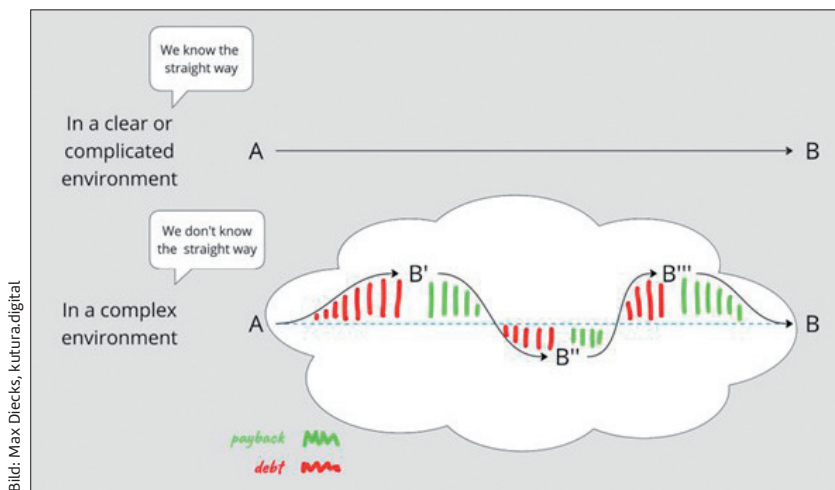


Bild: Max Diecks, kutura.digital

Agiles Arbeiten kommt in einer komplexen Umgebung zum Einsatz, wenn wir den geradlinigen Weg vom Start A zum Ziel B nicht kennen. Ein erstes Zwischenziel B' ist dann mehr oder weniger weit von der Ideallinie entfernt. Diese Abweichung brachte Ward Cunningham dazu, den Begriff „technische Schulden“ zu prägen. Zahlt man sie nicht sofort zurück, entfernt man sich mit dem nächsten Ziel B'' noch weiter von der Ideallinie, was auf lange Sicht zum Bankrott führt. Nur wenn der Code „clean“ ist, kann man die Schulden zügig begleichen. Zahlt man sie zurück, können die nächsten Ziele B'', B''' und so weiter näher an der Ideallinie liegen (Bild 1)

Sie entwickeln, was bestellt wird. Und das in so hoher Qualität, wie es das eigene Können und die von außen auferlegten Vorgaben zulassen. Wenn man es so betrachtet, liegt die Verantwortung außerhalb des Dev-Teams.

So ähnlich ist es auch bei den technischen Schulden, die man freiwillig und bewusst eingeht, um zum Beispiel einen vom Management vorgegebenen Termin einzuhalten. Die Argumentation ist hier die gleiche, auch wenn der Quellcode oder die Architektur dabei vielleicht Qualitätseinbußen erleiden. Wer schnell etwas liefern soll, führt womöglich das sinnvolle Refactoring einer Klasse nicht durch und ergänzt einfach ein paar Methoden, die die aktuelle Aufgabe irgendwie erledigen.

Als Letztes bleiben noch Code-Mängel, die zum Beispiel entstehen, weil es sich um ein unerfahrenes Team mit Junior-Entwicklern handelt. Jeder gibt sein Bestes, und das ist eben noch lange nicht „perfekt“. Auch hier kann man die Ver-

antwortung bei den Vorgesetzten suchen. Sie haben das Team zusammengestellt und verwalten die Fortbildungsbudgets. Was soll das Team oder gar ein einzelner Entwickler da machen?

... und wer begleitet sie?

In einer späteren Phase stellt sich die Frage, wer für den Abbau der entstandenen Schulden und Mängel verantwortlich ist. Zu diesem Zeitpunkt kann man in analoger Weise die Verantwortung ganz klar bei den ranghöheren Personen ansiedeln. Der PO priorisiert im Backlog die Feature-Tickets höher als die Schulden-Tickets? Der Vorgesetzte gibt neue Termine vor, die höchste Priorität haben? Und so weiter, und so weiter.

Kurz gesagt: Es liegt an der Führung. Nichts Neues eigentlich. Das steht ja auch schon so im Titel des Artikels. Und nun? Wie geht es weiter? Soll das heißen, Augen zu und mit Höchstgeschwindigkeit in den technischen Bankrott? Hoffentlich nicht. Irgendetwas muss sich ändern. Und das bedeutet immer: Irgendjemand muss seine Handlungsweise verändern. Da stellt sich die Frage, wer das sein kann. Wer kann das Team aus dem Schlamassel führen? Die naive Antwort lautet: die schon genannte ranghöhere Person. Und genau dort befindet sich der Knackpunkt. Das Wort „ranghöher“ ist das Problem.

Vielleicht ist Ihnen aufgefallen, dass ich „Führung“ und „Rang“ gleichbedeutend verwendet habe. Das entspricht dem weitverbreiteten Bild, dass Vorgesetzte auch führen (sollten). Viel weiter komme ich jedoch, wenn ich Führung so fasse, wie es Svenja Hofert und Claudia Thonet formulieren [1]:

„Führung ist völlig unabhängig von einer Funktion und Position. Führung ist das Bestimmen der Richtung von Bewegung und erfolgreiches Intervenieren in kritischen Situationen.“

Nach meinem Geschmack sollte der Rang, also die Position im Organigramm, beim Abbau von technischen Schulden keine Rolle spielen. Ohne in die Tiefe zu gehen, kann man hier einfach mal von flachen Hierarchien oder lateraler Führung sprechen. Letzteres ist interessant, weil das Wort „Führung“ darin vorkommt. Ein Wort, das wir in keinem (deutschen) Positionstitel finden. Wir haben Abteilungsleiter, Teamleiter, Bereichsleiter, aber keine Abteilungsanführer, Teamanführer, Bereichsanführer. Die Silbe „an“ habe ich eingebaut, um Konflikte mit der deutschen Geschichte zu vermeiden, denn darum geht es hier ganz sicher nicht.

Abgesehen davon hat das Wort „Anführer“ einen ziemlich guten Klang. Bei einem Anführer denken wir an klassische Banküberfallfilme (Heist-Movies [2]) wie etwa „Oceans Eleven“. Von der moralischen und juristischen Komponente der illegalen Handlung einmal abgesehen haben sie einen Riesenvorteil: den charismatischen Anführer. Ihm folgt nicht nur der Rest der Bande, sondern auch der Zuschauer bereitwillig. ►

Erst der Akt des Folgens macht den Anführer zu einem solchen. Wenn keiner mitmacht, ist er nur irgendein Kerl. Die Vorgesetzten, die mir beruflich begegnen, sind mehrheitlich nicht solche Typen. Sie haben nicht die Ausstrahlung von George Clooney. Man folgt ihnen nicht, sondern tut, was sie sagen, weil sie den Paycheck unterschreiben. Das zeigt sich in Studien und Umfragen immer wieder. Wer an dieser Stelle sagt: „Mein Chef ist anders, ich würde für ihn durchs Feuer gehen“, kann sich glücklich schätzen. Die Regel ist es nicht.

Nur am Rande sei festgestellt, dass sich an unserer emotionalen Bindung rein gar nichts ändert, wenn wir aus dem „Teamleiter“ einen „Team Lead“ machen. Da steckt dann zwar das richtige Wort drin, aber die meisten Team Leads haben mit einem Anführer so viel gemeinsam wie ein Lichtjahr mit einer Zeiteinheit.

So weit, so gut. Nun habe ich den Mangel an Führung in dieser Angelegenheit beklagt, doch sachliche Gründe dafür, dass die Vorgesetzten besser nicht für technische Schulden verantwortlich gemacht werden sollten, habe ich noch nicht angeführt. Dabei ist die Sache ganz einfach: Sie wissen schlicht und ergreifend nichts von den technischen Schulden. Oder sie verstehen es nicht. Oder beides.

Über viele Jahre habe ich Entwicklern, mit denen ich gearbeitet habe oder die an einem unserer Workshops zu technischen Schulden teilgenommen haben, immer wieder die gleichen Fragen gestellt.

Die erste lautete: „Wie gut können die Personen außerhalb des Dev-Teams den Umfang/das Ausmaß der technischen Schulden einschätzen?“ Die Antworten waren stets eindeutig (Bild 2): Nach Meinung der Entwickler wissen andere Personen ziemlich wenig über die technischen Schulden in der eigenen Software.

Die Selbsteinschätzung (Bild 3) sieht ganz anders aus. Auf die Frage, wer über den Umgang mit technischen Schulden entscheide, die Entwickler oder die anderen, sind die Antworten ebenfalls deutlich auf einer Seite: die anderen (Bild 4).

Die spannende Frage: Warum lassen die Entwickler das zu? Vielleicht ist das so eine Mentalität der Art „Der Kunde ist



Die meisten Entwickler, mit denen wir arbeiten, glauben, dass Personen außerhalb des Dev-Teams eine sehr schlechte Vorstellung von technischen Schulden in der eigenen Software haben (Bild 2)

König“ oder „Wer die Kapelle bezahlt, bestimmt, was gespielt wird“. Klingt vernünftig und kulturell gelernt, ist aber mindestens in gewissen Situationen nicht zielführend. Man stelle sich den Flugpassagier vor, der dem Piloten in wilden Turbulenzen Anweisungen gibt, wie er die Maschine zu fliegen hat. Kommt natürlich nicht vor, schließlich heißt der Chef an Bord nicht umsonst Flug „kapitän“. Hier kommen Kompetenz und Rang zusammen. Diesen Satz kann man nicht oft genug schreiben und lesen: *Hier kommen Kompetenz und Rang zusammen*.

Wenn es um technische Schulden in der Softwareentwicklung geht, ist das aber nicht der Fall. Entscheidet der Rang und nicht die Kompetenz über den Auf- und Abbau von technischen Schulden, setzt die Abwärtsspirale ein: *Bei technischen Schulden sind Kompetenz und Rang getrennt*.

Führungswechsel

Um das zu verhindern, gibt es einen Ausweg: Die Entwickler (= Kompetenz) müssen beim Thema technische Schulden die Führung übernehmen. Zumal dieses der Mehrzahl der Entwickler auch keineswegs egal ist (Bild 5). Die konkrete Ausgestaltung dieser Idee hängt zweifellos vom Unternehmenskontext und der vorherrschenden Kultur ab. Im weiteren Verlauf des Artikels gehe ich zuerst von einer Unternehmenskultur aus, in der man miteinander sprechen kann.

Eine grundlegende Unterscheidung besteht darin, ob die Entwickler offen oder verdeckt die Führung übernehmen. Da die verdeckte Vorgehensweise einige Risiken mit sich bringt, beginne ich mit der offenen Führung.

Mit offener Führung meine ich, dass alle Beteiligten sich darüber einig sind, dass technische Schulden und weitere Code-Mängel in der Verantwortung der Entwickler liegen. Für diese Aussage bekommt man sehr leicht vom Management grünes Licht. Wer will sich schon mit Technik abgeben? Aber dazu gehört untrennbar auch, dass die Entwickler autonom entscheiden können, wann und wie sie technische Schulden abbauen. Ein wesentliches Merkmal dieser Aussage ist, dass alle Beteiligten den Entwicklern zuge-



Bei der Selbsteinschätzung wird das Bild erwartungsgemäß positiver. Entwickler denken, dass sie technische Schulden in der eigenen Software relativ gut beurteilen können. Bemerkenswert ist, dass bei dieser Frage die Spreizung deutlich größer ist. Die Selbsteinschätzung gerät aber nicht in den roten Bereich (Bild 3)

stehen, zu neuen Feature-Wünschen oder Terminvorgaben Nein sagen zu können.

In der Praxis beobachte ich eher den Fall, dass beispielsweise Terminvorgaben des Managements als nicht verhandelbar wahrgenommen werden. In der Folge verzichten die Entwickler auf dringend notwendige Modernisierungen der Software, und wenn das über lange Zeit geschieht, rutscht man in den technischen Bankrott. Dann ist das Wehklagen groß und man macht sich auf die Suche nach den Schuldigen. Aus der Luft gegriffene Terminvorgaben werden dabei in der Regel nicht als Ursache erkannt.

Wie kann man dazu kommen, dass die Entwickler die Führung übernehmen? Der erste und wichtigste Punkt besteht darin, dass man zuerst klärt, wie der Handlungsspielraum im Moment überhaupt aussieht. Ich habe mit einer ganzen Reihe von Teams gearbeitet, die mir mit voller Überzeugung berichteten, was sie alles nicht entscheiden dürfen. Als ich die jeweiligen Vorgesetzten oder die Geschäftsführung darauf ansprach, gab es erstaunte Gesichter: In den meisten Fällen sahen die Vorgesetzten die Entscheidungsbefugnis sehr wohl beim Team.

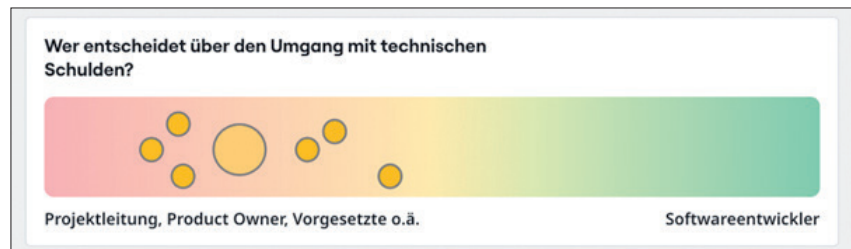
Kurz gesagt: Die Verantwortlichkeiten und der Handlungsspielraum, den beide Seiten sehen, sind oft unterschiedlich. Deshalb lautet mein erster Tipp: Klärt als Team mit allen relevanten Personen, wer welche Verantwortung im Bereich technischer Schulden hat. Wer diese Personen sind, hängt vom jeweiligen Kontext ab. Manchmal ist es der Product Owner, manchmal ein Vorgesetzter, und manchmal wird ein Product Owner als Vorgesetzter wahrgenommen und bekommt kein Veto.

Es empfiehlt sich auch, etwaige Terminvorgaben vorsichtig zu hinterfragen. Mancher Manager meint, man müsse stramme Termine vorgeben, damit die Mitarbeiter richtig spüren. Wenn dieser Glaubenssatz vorherrscht, die Zeitvorgaben ohne Widerspruch bleiben und das Projekt später im technischen Bankrott endet, haben alle verloren. Also sollte man zumindest eine Klärung versuchen.

Für das Gespräch kann es hilfreich sein, den Begriff der technischen Schulden zu überdenken und eventuell durch einen anderen zu ersetzen. Dafür gibt es mehrere Gründe: Erstens erfasst der Begriff nicht alle Aspekte von Problemen adäquat. Zweitens sind beide Worte irreführend, Schulden und Technik. Schulden macht man in der Regel bewusst, aber viele Code-Probleme entstehen im Lauf der Zeit „von selbst“. Und die Probleme nur auf „Technik“ zu reduzieren senkt das Interesse von Vorgesetzten oder Kunden rapide. Außerdem untergräbt es die Glaubwürdigkeit der Entwickler, wenn sie dauernd berichten, dass mit der „Technik“ etwas nicht

stimme. Es gibt durchaus Vorschläge für ein anderes Vokabular, etwa Softwareverschleiß oder Softwareerosion [3]. Ich verwende gerne das Segeln als Metapher, wie oben bereits angedeutet. In der klassischen Navigation (das heißt ohne GPS-gesteuerte Autopiloten und Ähnliches) sind Kurskorrekturen [4] etwas völlig Normales.

Wikipedia schreibt dazu: „Der direkte Weg von einem Ausgangspunkt (origin) zum Ziel (destination) führt über den Kurs. Das ist die Verbindungslinie beider Punkte [...]. Bei



Diejenigen Personen, die die technischen Schulden am wenigsten einschätzen können, sind nach Meinung der Entwickler diejenigen, die mehrheitlich die Entscheidungen darüber treffen (Bild 4)

einer Störung, beispielsweise eine Strömung oder Querwind, wird das Ziel verfehlt. Der zurückgelegte Weg, der Weg über Grund oder Track, weicht vom Kurs ab. Der Schiffs- oder Flugzeugführer steuert gegen.“

Diese Störungen haben wir in der Softwareentwicklung fortwährend, zum Beispiel:

- Neue Anforderungen lassen die getroffenen Architekturentscheidungen in einem neuen Licht erscheinen.
- Plötzlich bekannt gewordene Sicherheitslücken in einer verwendeten Bibliothek verlangen nach sofortigem Eingreifen.
- Der neue Mitarbeiter kennt unsere Coding Practices noch nicht, aber genau jetzt haben wir keine Zeit, seinen (funktionierenden) Code zu überarbeiten.
- Der nächste Termin drängt und „Programm läuft“ ist wichtiger als „Code ist clean“.

Wichtig bei diesen Beispielen ist: Niemand handelt hier falsch. Es gibt keinen Schuldigen. Es gibt nur Ereignisse, die nach einer Kurskorrektur verlangen. Und wie beim Segeln ►



Technische Schulden lassen die Mehrheit der Entwickler nicht kalt. Vorgesetzte sollten auch diesen Aspekt hinterfragen, um rechtzeitig reagieren zu können, bevor die Unzufriedenheit zu groß wird (Bild 5)

stellt sich die Frage, wann ich den Kurs korrigiere. Je länger ich warte, desto größer wird die Abweichung zum Ziel.

Aus meiner Sicht ist es unausweichlich, dass die Stimme der Entwicklerteams absolut gleichwertig ist, wenn es darum geht, den neuen Kurs zu wählen. Dazu gehört ein Veto-Recht, um zu große Kursabweichungen zu verhindern. Dies einzufordern ist Aufgabe der Entwickler.

Tun sie das nicht, bleibt noch die verdeckte Führung. Damit ist gemeint, dass die Teams die Aufwände für die Kurskorrektur in anderen Arbeitspaketen „verstecken“. Wer mit Story Estimation arbeitet, kann zum Beispiel nach dem Schätzen den ermittelten Wert immer ein wenig erhöhen, um in dem Overhead das Refactoring oder Ähnliches unterzubringen. Alternativ kann man die Velocity senken und in der damit frei gewordenen Zeit die notwendigen Arbeiten unterbringen. Eine Alternative ist die bekannte 80:20-Regel: Vier Tage pro Woche arbeitet das Team an neuen Features, an einem Tag zahlt man Schulden zurück.

Mir gefallen diese verdeckten Ansätze nicht sehr gut, weil es an der – aus meiner Sicht notwendigen – Kommunikation aller Beteiligten fehlt. Ein Product Owner, mit dem ich gearbeitet habe, hat sich regelmäßig darüber beklagt, dass die Entwickler „den Maschinenraum polieren“. Aus seiner Sicht fand zu viel Arbeit am Code statt. Die Gefahr ist nicht von der Hand zu weisen und besonders groß, wenn die Arbeiten verdeckt stattfinden.

Kommunikation

Die Ursache dieser unterschiedlichen Wahrnehmungen besteht darin, dass beide Seiten – Entwickler und Business-Leute/Stakeholder/Kunden – die jeweils andere Seite nicht gut genug verstehen. Die Entwickler beschwerten sich über unrealistische Termine, und die Stakeholder beklagen die faulen Entwickler. Das eine ist so falsch wie das andere. Meistens zumindest. Der Ausweg ist leider schwierig, weil hier sehr unterschiedliche Menschen zusammenkommen.

Der Ausweg heißt: Kommunikation auf Augenhöhe. Wenn ich als Entwickler Teil des Business bin, verstehe ich, welchen Nutzen die Termine haben. Welches Ergebnis erzielt meine Firma denn, wenn wir einen Termin einhalten? Welchen Nutzen und welchen Schaden richte ich eventuell an, wenn ich den Code „perfekt“ haben möchte? Solange ich als Entwickler an den Business-Entscheidungen nicht beteiligt bin, stehen alle Entscheidungen über das Aufnehmen oder die Rückzahlung von technischen Schulden auf tönernen Füßen. Eine Kurskorrektur kann ich nur seriös durchführen, wenn ich beides kenne: Position und Ziel. Wer nur eines kennt, kommt vielleicht nicht lebend an.

Wenn Entwickler über Softwarequalität sprechen, ist stets die Rede von Fehlerfreiheit, Wartbarkeit, Testabdeckung. Doch zur Qualität meines Produkts gehört auch, dass es Umsatz und Gewinn erzeugt. Das ist eine schöne, einfache Welt-sicht. Wir entwickeln ein Programm, das wir verkaufen. Verdienen wir mit einem Feature mehr Geld als vorher, dann ist es ein gutes Feature.

Leider ist diese einfache Weltsicht in Konzernen aufgrund der Größe oft nicht anwendbar. Bei interner Software, die ein

Rädchen im großen Getriebe ist, ist die Kopplung „Feature => Umsatz/Gewinn“ nicht leicht herzustellen. Trotzdem stehe ich zu der Grundaussage. Personen auf unterschiedlichen Hierarchieebenen betrachten sehr unterschiedliche Qualitätsmerkmale. Wer wesentliche Merkmale außer Acht lässt, wird schlechtere Entscheidungen für den Kurswechsel treffen. Da ich sowohl als Manager als auch als Entwickler nur einen Ausschnitt der „gesamten“ Qualität sehen kann, braucht es einen Austausch über die verschiedenen Perspektiven.

Die Organisation einer Firma in Abteilungen (lateral) und in Hierarchien (vertikal) behindert diesen Austausch. Es kommt zur Silobildung auf beiden Achsen. Um sie aufzubrechen, braucht es einen Dialog, den beide Seiten anstoßen können. Da die höheren Ränge aufgrund des fehlenden Kontakts zur Software die Probleme erst spüren, wenn es richtig weh tut, sind die Entwickler gefordert, frühzeitig in die Kommunikation zu gehen, Verantwortungsübernahme anzubieten, die nötigen Rechte (insbesondere das Veto-Recht) einzufordern und diese Rechte und Pflichten anzuwenden.

Wer sich ein wenig auf Management-Sprache einstellen will, kann sich mit dem Begriff „Andon“ vertraut machen. Dieses aus dem Japanischen stammende Wort wird meist mit einer Reißleine in Verbindung gebracht, die die Mitarbeiter in den Toyota-Werken bereits vor Jahrzehnten ziehen konnten und sollten, wenn sie Qualitätsprobleme erkannten. Es „bedeutet in diesem Kontext, dass jeder Mitarbeiter befugt ist, die Fertigungslinie bei Problemen unverzüglich anzuhalten, damit Qualitätsmängel sofort behoben werden können“ [5].

Denkt man an die Kosten, die ein Stillstand bedeutet, ist das erhebliche Macht in der Hand jeder einzelnen Person in der Fertigung. Denkt man an die Kosten, die Qualitätsmängel der Produkte auf der Straße bedeuten, ist es klar, weshalb die Manager diese Macht gerne abgeben. Auch hier ist die Führungsmacht vom Rang zur Kompetenz gewandert. Was hält uns davon ab, eine analoge Regelung für uns Entwickler einzufordern? ■

[1] Svenja Hofert, Claudia Thonet, *Der agile Kulturwandel*, www.dotnetpro.de/SL2504-05TechnischeSchulden1

[2] Heist-Movie, <https://de.wikipedia.org/wiki/Heist-Movie>

[3] *Softwareverschleiß versus technische Schulden*, www.dotnetpro.de/SL2504-05TechnischeSchulden2

[4] *Kurskorrekturen auf Wikipedia*, www.dotnetpro.de/SL2504-05TechnischeSchulden3

[5] *Andon (Prozessoptimierung)*, www.dotnetpro.de/SL2504-05TechnischeSchulden4



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

A2504-05TechnischeSchulden



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

WIE MOTIVATION ENTSTEHT

Als Coden noch Bock machte

Warum war Programmieren früher pure Euphorie – und ist heute oft nur noch Pflicht?
Eine Spurensuche nach dem verlorenen Flow.

Wenn ich zurückdenke an meine Zeit als Jugendlicher oder als Student, dann waren Computer, Informatik und Software damals die Erfüllung meines Lebens. Darüber mit Freunden zu sprechen konnte Abende füllen, und wenn es mal Partys gab, wurde angetrunken über die Tücken von JavaScript gerappt. Es war wunderbar!

Auch später noch: Nicht selten hörte ich mich sagen, dass ich den tollsten Job der Welt habe, dass ich hier in der luxuriösen Situation sei, mein liebstes Hobby zum Beruf gemacht zu haben – und dafür noch ordentlich Asche zu bekommen. Für mich war es unvorstellbar, dass andere oft nicht schon seit dem sechsten Lebensjahr genau gewusst hatten, was sie bis übers Rentenalter hinaus machen wollten.

Wenn ich mich so umhöre, geht es verdammt vielen in meiner Branche genau wie mir. Und das ist irre – gibt es doch zahlreiche Zeitgenossen, die sich tagtäglich durch ihren Job quälen und sich eigentlich nur noch auf den Feierabend und das Wochenende freuen. Unsereins freut sich, wenn endlich, endlich Montag ist und man wieder im Rechner versinken kann. Man denke nur an die Begeisterung, die jedes neue Framework, jede neue Technologie, jede neue Kleinigkeit in einem ausgelöst hat: Man hatte spontan tausend Ideen für Anwendungen, und schon ging es los mit dem nächsten Grüne-Wiese-Projekt. Diese Art der Motivation schien unerschütterlich. Es konnte eigentlich passieren, was wollte – solange ich und meinesgleichen einen warmen, dunklen Ort hatten, an dem ein Computer mit Internet stand, konnte eigentlich nichts schief laufen.

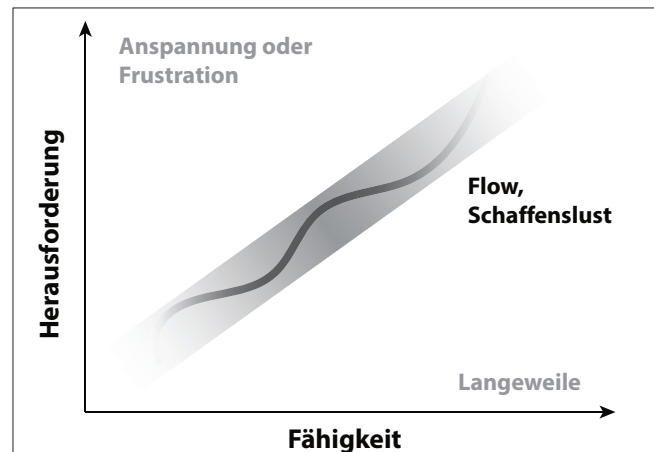
Und dennoch: Im Lauf des Lebens hat sich daran doch entscheidend etwas getan. Natürlich interessiert man sich weiterhin für die News der Branche. Aber die Dinge sind längst nicht mehr so mitreißend, wie sie einmal waren. Die Gründe, immer auf dem Laufenden zu sein, sind schlichtweg andere geworden: Man bleibt für die tägliche Arbeit up to date, will eigentlich nur abgleichen, ob der neue Quatsch auch nicht besser ist als das, was man bereits verwendet, oder ob man vielleicht Kleinigkeiten im Arbeitsablauf verbessern kann.

Das mag einerseits daran liegen, dass man einfach schon viel gesehen hat und die schiere Menge an neuen Frameworks nicht mehr überschaubar ist. Man wird außerdem älter und will auch nicht mehr auf jeden neuen Zug aufspringen. Gleichzeitig gibt es aber doch noch das Kribbeln, wenn man doch mal wieder die Chance hat, ein neues Projekt anzulegen, um etwas auszuprobieren. Und man muss zugeben: Das Jucken in den Fingern ist noch da, es schlummert zwischenzeitlich einfach nur.

Was passiert da im Kopf?

Bevor wir uns die Frage stellen, was uns da nicht nur den Tag, sondern gleich unseren ganzen Frohsinn im Leben versemelt, lohnt sich ein Blick auf das, was jenen Flow-Zustand eigentlich ausmacht.

Müsste ich den Flow-Zustand beschreiben, würde ich von ultimativem Fokus, Glückseligkeit und Zeitlosigkeit sprechen. Mihaly Csíkszentmihályi definiert es in seinem Buch „Finding Flow: The Psychology of Engagement with Everyday Life“ [1] etwas umfangreicher mit den Worten: „Being completely involved in an activity for its own sake. The ego falls away. Time flies. Every action, movement, and thought follows inevitably from the previous one.“ Volltreffer.



Der Flow-Tunnel: Er zeigt das Zusammenspiel zwischen Fähigkeit und Anforderung (Bild 1)

Jetzt ist der Flow-Modus des Programmierers natürlich der Flow-Mode auf Steroiden. Warum das bei unserer Zunft besonders intensiv ist, besprechen wir weiter unten. Für uns ist an dieser Stelle die Frage wichtig, was den Wald-und-Wiesen-Flow landläufig so ausmacht. Denn dauerhaft im Tunnel zu sein hat auch seine Tücken.

Im Grunde ist dieser Zustand allgemeingültig für alle Menschen, Tätigkeiten und Gebiete. Er ist dadurch definiert, dass die angewendete Tätigkeit genau dem Skill-Level des Anwenders entspricht – nicht überfordernd (auch bekannt als Stress) und nicht zu unterfordernd (formally known as Langeweile) (Bild 1). Am ehesten kann man sich das vielleicht anhand eines Computerspiels vor Augen führen. Sind die Auf-

gaben im Spiel zu einfach, fehlt die Herausforderung und das Spiel wird schnell langweilig. Sind die Aufgaben dagegen zu schwer, ist das Spiel frustrierend und macht ebenso wenig Spaß. Die Challenge des Game-Designers ist es also, genau das Fertigungslevel des Spielers zu erwischen, damit der Spieler Spaß hat.

Abgesehen davon, dass die Fähigkeiten einzelner Spieler sehr individuell sind, verändert sich das Skillset durchs Spielen. Und das ist erschreckend nah am echten Leben und unserem beruflichen Alltag. Dies ist eine Erkenntnis von bestechender Einfachheit: Sind die Aufgaben zu simpel, langweilen wir uns. Haben wir zu viele Aufgaben oder überfordern uns diese durch ihre Komplexität, sind wir gestresst.

On top kommen weitere Dinge, die den Programmierer im Tunnel halten und fast schon süchtig machen – im Übrigen auch Faktoren, die in Computerspielen goldene Regeln für ein erfolgreiches Game-Design sind. Beispielsweise „Instant Gratification“, also die sofortige Belohnung für eine Aktion. In Spielen sind das das Sternesystem, die eingesammelten Coins oder Erfahrungspunkte. Bei uns ist die Belohnung für eine gute Codepassage der Moment, wenn alles kompiliert, der Test grün ist und alles wie erwartet funktioniert. Das sorgt für einen ordentlichen Dopamin-Kick.

Doch nicht genug. Es greift außerdem ein weiteres Grundbedürfnis des Menschen: bestimmte Fertigkeiten zu meistern („Mastery“). Und dabei ist ein Fortschritt selten so schnell und deutlich spürbar, wie wenn man ein neues Framework über nur wenige Stunden hinweg zuerst noch nicht versteht, sich dann langsam einarbeitet, hier und da scheitert (was ja nur bestätigt, wie schwer es zu meistern ist) und es am Ende für sich gebändigt hat. Das Gegenstück beim Computerspiel könnte hier ein komplexes Kampfsystem sein, das sich erst durch längeres Spielen zur vollen Pracht entfaltet beziehungsweise bei dem man sich Stück für Stück durch gewonnene Kämpfe verbessert.

Zuletzt kommt ein Aspekt ins Spiel, der in der menschlichen Bedürfnispyramide ganz weit oben steht: Autonomie. Was so philosophisch klingt, drückt eigentlich nur aus, dass wir in unseren Entscheidungen unabhängig sein wollen. Dieses Bedürfnis ist das höchste Gut in der eigenen Selbstverwirklichung und kann im Flow – gerade bei Grüne-Wiese-Hobbyprojekten – uneingeschränkt ausgelebt werden. Alle Entscheidungen im Hier und Jetzt obliegen einzig und allein dem Programmierer. Wow, oder?

In Spielen findet dieses Bedürfnis übrigens an ganz vielen Stellen seinen Platz. Angefangen natürlich bei der freien Wahl dessen, was als Nächstes passiert – ein Schlüsselfaktor von interaktiven Erzählungen. Gerade in Open-World-Spielen steht dem Spieler oft eine ganze Bandbreite an Möglichkeiten offen: Sammle ich jetzt erst mal Grünzeug für den Eintopf, den ich dann machen werde, oder sollte ich doch erst den Drachen töten? Auch Skillbäume sind ein gutes Beispiel. Denn hier kann der Spieler entscheiden, worauf er sich im restlichen Spielerleben spezialisieren und wo er sich verbessern will.

Mit diesem Wissen können wir uns nun fragen, warum wir eigentlich nicht jede Minute unseres Arbeitslebens voll Dopamin sind.

Der Flow im Alltag

Nun ja. Der geneigte Leser und gegebenenfalls erfahrene Entwickler wird beim Lesen des vorherigen Abschnitts sicher schon ein paar Ideen gehabt haben, warum wir nicht mehr jeden Tag wie ein 15-jähriger Teenie im Rausch in den Feierabend gehen: Einige der Parameter sind eben bei unserer täglichen Arbeit nicht mehr gegeben.

So werden wir nicht tagtäglich mit vollkommen neuen Herausforderungen konfrontiert, um unser Bedürfnis nach Mastery an neuen Frameworks, kniffligen Algorithmen oder schier unlösbaren Problemen abzarbeiten. Vielmehr besteht ein Großteil des Programmiereralltags aus immer wiederkehrenden Aufgaben und Routinen. Je klarer und vorhersehbarer der Prozess und die Guidelines, umso besser fürs Team und das gemeinsame Verständnis des Codes. Schon klar.

Ähnliches gilt auch für die uneingeschränkte Autonomie bei künstlerischen Entscheidungen im Code-Projekt. Versucht man ernsthafte Software zu machen – am Ende noch im Team –, wird es mit den vollkommen freien Entscheidungen irgendwann schwierig. Angefangen bei den Code-Guidelines über die Richtlinien für externe Packages bis hin zur festgelegten Softwarearchitektur gibt es nun mal ein Rahmenwerk. Also auch hier: unvermeidbare Einschränkungen.

Und ganz ehrlich gesprochen kann es nicht das Ziel sein, sich permanent selbstvergessen in irgendwelche Probleme zu verbeißen. Denn auch der dauerhafte Betrieb in diesem Extremzustand bringt seine gesundheitlichen Tücken mit sich. Wie es also scheint, wird es im professionellen Umfeld nix mehr mit Flow. Nie mehr. Unmöglich. Und dennoch – manchmal passiert es trotzdem. Wenn es mal besonders knifflig wird, der Kollege bereits am Problem gescheitert ist oder schlichtweg was Neues um die Ecke kommt. Es gibt sie also noch, diese kleinen Momente.

Halten wir also fest, dass es nicht das Ziel sein kann, sich im Dauerrauschzustand zu befinden. Viel eher scheint es sinnvoll zu sein, sich ein Stück dieser Euphorie und Begeisterung dauerhaft zu bewahren, sodass man am Morgen mit einem Lächeln zur Arbeit geht. Denn im Grunde sind es dieselben Parameter – in abgeschwächter Form –, die uns mit Freude erfüllen.

Was bedeutet eigentlich Motivation?

Bisher haben wir nur davon gesprochen, wie extrem sich Eigenmotivation im perfekten Moment auswirken kann und wie sich das in einem idealen Alltag gestalten könnte. Ein paar der wichtigen Faktoren kamen auch schon zur Sprache. Doch die eigentliche Frage, die uns verstehen lässt, warum Menschen langfristig Lust auf ihre Arbeit haben, bleibt noch unbeantwortet: Was genau steckt denn hinter dem Phänomen Motivation?

Zusammengefasst beschreibt Motivation den Antrieb eines Menschen, Dinge zu tun – im Fall von Arbeitsmotivation idealerweise dauerhaft. Angelehnt an die „Selbstbestimmungstheorie“ [3] lässt sich die Motivation eines Menschen grundsätzlich in zwei Kategorien von Antriebsfaktoren einteilen (Bild 2). Zum einen sind da die intrinsischen Faktoren. Darunter fällt eine Reihe von menschlichen Grundbedürfnissen, ►

die es mithilfe der Tätigkeit zu befriedigen gilt. Zwei dieser mächtigen Triebfedern haben wir bereits bei der Flow-Analyse kennengelernt: Einerseits die Selbstverwirklichung in Form von Autonomie, die uns eine gewisse persönliche Entscheidungsfreiheit einräumt. Außerdem das Streben nach Kompetenz (Mastery), das uns Dinge wissen wollen lässt.

Die Dritte im Bunde entfaltet ihre Wirkung ganz besonders dann, wenn wir nicht mehr alleine an etwas arbeiten oder wenn wir Dinge nicht mehr nur für uns tun: die soziale Eingebundenheit – also die Bindung zu anderen Menschen. Viele Dinge unseres täglichen Lebens – und das betrifft nicht nur die Arbeitswelt – basieren auf diesem Grundbedürfnis. Es ist eines, das schon seit dem Kindesalter einen der wichtigsten Überlebensfaktoren darstellt und dementsprechend einen hohen Stellenwert im menschlichen Unterbewusstsein hat.

Je nach Interpretation der Theorie werden weitere intrinsische Faktoren wie Neugier, Spieltrieb und Sinnhaftigkeit der Tätigkeit genannt. Diese lassen sich im Grunde jedoch auf die oben genannten drei zurückführen: Neugier und Spieltrieb zählen auf das Kompetenzthema ein, die Sinnhaftigkeit eher auf die gesellschaftliche Bindung.

Der zweite dicke Teilbereich der Motivation sind die extrinsischen Motivationsfaktoren. Dies umfasst alles, was von außen kommt und im Einflussbereich unserer Tätigkeiten steht. Das sind beispielsweise Geld, der Firmenwagen, die Fleißsternchen in der Schule, Ranglisten und Punktesysteme im Allgemeinen. Diese Dinge funktionieren kurzfristig tatsächlich auch gut, um uns zur Leistung zu motivieren.

Das Fatale an dieser Art der Motivation ist jedoch, dass sie sich „abnutzt“, gleichzeitig die intrinsische Motivation zerstören kann und am Ende sogar ins Gegenteil kippt. Zwei Mal Mitarbeiter des Monats zu sein beflügelt, keine Frage. Es danach aber lange nicht mehr zu schaffen – denn es kann ja nur einen Mitarbeiter des Monats geben – frustriert zusätzlich. Besonders dann, wenn man keinen großen Einfluss auf die Bewertungskriterien hat.

Auch ein hohes Gehalt motiviert, schleift sich aber schnell als Selbstverständlichkeit ein und muss erhöht werden, um den immer schwächer werdenden Effekt aufrechtzuerhalten. Zwar klingt das alles sehr schlüssig und simpel, aber am En-

de ist es dann doch sehr abstrakt und in alltäglichen Aufgaben nur noch schwer zu mappen.

Versuchen wir es trotzdem mal.

Was die Motivation stört

Jetzt ist uns klar, wie Motivation funktioniert, und ich bin sicher, dass sich in den bisherigen Ausführungen irgendwie alle wiedergefunden haben. Mit diesem Wissen sollte es jetzt ein Leichtes sein, ein Arbeitsumfeld zu erschaffen, in dem alle in Freude und Wohlbefinden arbeiten können.

Hört man sich jedoch auf Konferenzen oder in einzelnen Firmen um, scheint es vielerorts dann doch nicht ganz so rosig auszusehen. Entwickler zeigen sich verärgert über das tägliche Zusammenspiel, klagen über zu hohe Arbeitslast, verfluchen Kollegen, Vorgesetzte, Nicht-Techies. Irgendwas scheint also schiefzulaufen bei der eigenen Motivation und Lebensfreude.

Das Gute ist ja, dass viele von uns Devs bereits einen schier unerschütterlichen, intrinsischen Drang zur Technik haben. Der ist auch nicht totzukriegen. Lasst uns daher auf die Suche nach den Dingen gehen, die uns den Tag vermiesen. Am einfachsten hangeln wir uns doch an den Parametern entlang, die uns überhaupt erst so begeistert sein lassen.

Die Wurzel des Übels scheint – wenn man den Betroffenen glauben darf – in den meisten Fällen das eigentliche Arbeitsumfeld zu sein. Kommunikation, Teammitglieder, Druck durch das anfallende Arbeitsvolumen, Manager und Chefs sowie die Art und Weise, Projekte zu organisieren, scheinen hier der Hemmschuh zu sein.

Doch am Ende sind das erst mal „nur“ die äußeren Umstände, die unsere innere Haltung zum Wanken bringen und uns daran hindern, unseren persönlichen Bedürfnissen zu folgen.

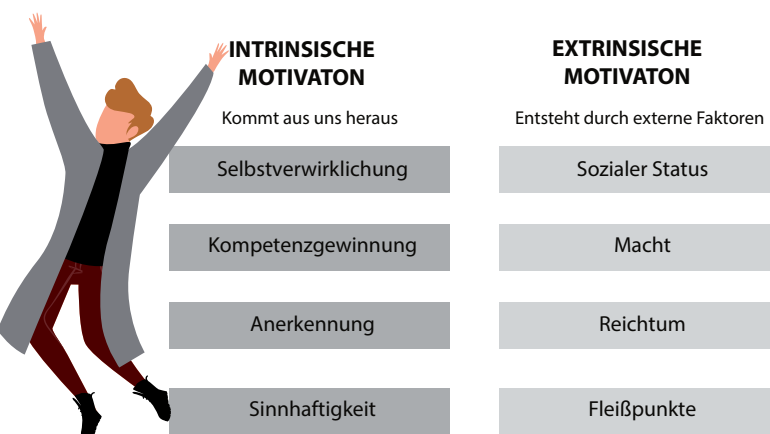
Ich möchte also auf diesem Weg mal die Welt hinter den scheinbar offensichtlichen Gründen für unser Unwohlsein beleuchten. Der perfekte Startpunkt hierfür ist eines der von uns bereits identifizierten Bedürfnisse – die Autonomie. Sie ist sicherlich eine der wichtigsten Bedingungen, die für Flow und Motivation erfüllt sein müssen.

Nimmt man jetzt Fachliteratur von Maslow [2] zur Hand, stößt man auf eine Auflistung der menschlichen Grundbedürfnisse, die aber nicht gleichberechtigt sind, sondern pyramidenförmig aufeinander aufbauen – um genau zu sein, werden fünf Ebenen beschrieben (Bild 3), die bereits weiter oben erwähnte Bedürfnispyramide.

Die Basis bildet die physiologische Grundversorgung in Form von Nahrung, Schlaf und Wärme.

Die Spitze krönt – wer ahnt es schon? – die Selbstverwirklichung. Das bedeutet im Umkehrschluss: Bevor wir wirklich sorgenfrei kreativ werden können, darf keines der anderen Bedürfnisse im Alarmzustand sein. Ohne Essen kein Flow.

Daher lohnt sich ein intensiver Blick auf alle Ebenen, um dann eine Relation zur „echten Welt“ herzustellen, um tatsächliche Probleme zu identifizieren und gegebenenfalls Gegenmaßnahmen einzuleiten. Da sich dieser Artikel an Arbeitnehmer wie an Privatpersonen und auch an Menschen mit personeller Verantwortung richtet, versuche ich, Einblicke und nützliche Tipps für alle Perspektiven zu sammeln. Denn am



Gegenüberstellung von intrinsischen und extrinsischen Motivationsfaktoren (Bild 2)

Ende tragen alle Beteiligten eines Systems zur Heilung bei – nicht nur Personen in offizieller Verantwortung.

Level 1: Die physiologische Basis

Hier geht es um nichts Geringeres als den notwendigen Input, um grundsätzlich überlebensfähig zu sein: Essen, Trinken, Schlaf, Wärme ... Im Endeffekt triviale Dinge, an die man – im wahrsten Sinne des Wortes – im Flow als Letztes denkt. Vergisst man doch oft, rechtzeitig zu essen.

Natürlich brauche ich hier niemandem zu sagen, wie wichtig eine ausgewogene Ernährung ist, was das im Einzelnen bedeutet und welchen Einfluss ein guter Rhythmus hat. Denn so richtig gut tut Fastfood nur, wenn es allerhöchste Eisenbahn ist und der Körper sich schon Sorgen um die Versorgung macht.

Dennoch sind es die Feinheiten, die sich bereits schnell im Wohlbefinden bemerkbar machen. Man erinnere sich nur an die zwei Grad niedrigere Raumtemperatur während der Energiepanik. Oder daran, wie wichtig es ist, gute Kulinarik in der Nähe des Büros vorzufinden – vielleicht sogar eine Kantine zur Verfügung zu haben. Sich nicht jeden Tag aufs Neue die zermürbende Frage zu stellen, was man heute essen soll, ist sehr befreiend.

Level 2: Allgemeine Sicherheit

Damit sind Rahmenparameter gemeint, die grundsätzliche Existenzängste verursachen können. Um einen sicheren Arbeitsplatz zu wissen, keinen Säbelzahn timer fürchten zu müssen oder sich auf ein auf absehbare Zeit zuverlässiges Einkommen verlassen zu können sind Stabilitätsfaktoren, die mehr Freiraum im Kopf schaffen. Wer sich ständig Gedanken darüber machen muss, womit er im kommenden Monat seine Brötchen bezahlt, wird sich mit kreativen Experimenten schwertun.

Was genau „sicheres Einkommen“ bedeutet, ist hier eine sehr subjektive Sache. Es gibt Berufsfelder, in denen nur kurz in die Zukunft geplant werden kann – man denke hier an den typischen Unternehmer. Wohingegen Berufe, bei denen eine lebenslange Anstellung wahrscheinlich ist, nicht zwangsläufig eine große Kreativität erwarten lassen. Doch das mag andere Gründe haben.

Aus Arbeitgebersicht bedeutet das natürlich, dem Mitarbeiter eine möglichst hohe Zuverlässigkeit und Planbarkeit zu vermitteln. Leichter gesagt als getan. Denn die Zuversicht des Mitarbeiters wird nicht dadurch gestärkt, dass man ständig verbalisiert, dass man nächsten Monat auch noch Gehalt bezahlt. Vielmehr ist diese Sicherheit tief verankert in der Unternehmenskultur und wird durch eine gute Fehlerkultur, Wertschätzung, transparente Kommunikation und Ähnliches untermauert.

Sind Sie nun auf Arbeitnehmerseite, sind Sie natürlich dem „ausgeliefert“, was Ihr Arbeitsumfeld so an Sicherheit ausstrahlt. Aus dieser Warte lohnt sich ein Blick auf die subjek-



Die Bedürfnispyramide, die einzelne Bedürfnisse als aufeinander aufbauende Struktur ordnet (Bild 3)

tive Wahrnehmung in Resonanz mit Ihrem Sicherheitsbedürfnis. Wie hoch hier Ihr persönlicher Sorgenpegel ist, hängt mit den Erfahrungen zusammen, die Sie in Ihrem Leben gemacht haben, oder welches Sicherheitsgefühl Ihnen Ihre unmittelbaren Bezugspersonen vermittelt haben. Es reicht schon, dass man möglicherweise als Kind unbewusst miterlebt hat, wie im Elternhaus um die eine oder andere Heizölrechnung gebangt wurde, um sensibler auf existenzverändernde Gefahren zu reagieren.

Der Trost ist, dass es im Arbeitsumfeld – gerade in der IT – ehrlicherweise keine wirklich existenzbedrohenden Gefahren gibt.

Es lohnt sich also ein objektiver

Faktencheck: In der Regel verfügt man über einige Monate Kündigungsfrist, fällt ins soziale Netz und sollte recht schnell eine neue Arbeitsstelle finden. Sollte Sie diese Sorge aber so sehr belasten, dass Ihre Arbeit darunter leidet, lohnt es sich, der Ursache auf den Grund zu gehen.

Level 3: Soziales Umfeld

Auf diesem Level befindet sich das Bedürfnis zur zwischenmenschlichen Bindung und der Teamzugehörigkeit. Hier geht es also um Teamspirit, gemeinsame Werte und Visionen.

Und hier wird es gerade bei Entwicklern wirklich interessant, denn besonders Menschen unseres Schlages fällt es oft schwer, sich tatsächlich auf andere einzulassen. Das mag zwar nicht auf alle Techniker zutreffen, aber hier ist die Dichte an Menschen recht hoch, bei denen das Vertrauen in andere Personen und das Bedürfnis nach echter Nähe nachhaltig beeinträchtigt sind.

Die Ursachen hierfür zu ergründen sprengt den Rahmen dieses Artikels und tut am Ende auch nichts zur Sache. Doch es ist wichtig zu wissen, dass diese Haltung in Extremfällen dazu führt, sein urmenschliches Bedürfnis nach Bindung zu anderen vollständig zu verleugnen. Daraus ergeben sich oft sonderbare und fast schon schizophrene Verhaltensmuster.

Wenn beispielsweise die Ablehnung der Zugehörigkeit zu einer Gruppierung auf den gleichzeitigen Bedarf an Anerkennung trifft. Wenn es sich plötzlich sonderbar anfühlt, mal das Team für einen erreichten Milestone ausgelassen zu feiern, aber für die Lösung eines kniffligen Problems keine angemessene Würdigung zu erfahren.

Worauf ich hinauswill: Manager und Teamleiter – gerade solche, die nicht selbst auch in einer Nerdhaut stecken – tun sich oft verdammt schwer, einzelne Mitglieder im Team aus der Reserve zu locken. Mitarbeiter zeigen sich verschlossen, unnahbar und verstecken sich lieber hinterm Bildschirm. Das liegt in der Natur der Entwickler. Die Tendenz zur ausschließlichen Remote-Arbeit, die hier natürlich auf großen Anklang stößt und gerne genommen wird, zahlt ihr Übriges auf die Entfremdung der Teammitglieder ein.

Das Resultat – nicht zuletzt durch übermenschliche Arbeitslasten – sind Teams, in denen kein wirkliches Miteinander ►

mehr herrscht, sondern bestenfalls ein Nebeneinander, schlimmstenfalls sogar ein Gegeneinander.

Wer hier gegensteuern will, muss mit Feingefühl ran. Ein Rafting-Event oder der Hochseilgarten könnten hier vielleicht das Ziel verfehlen. Viel wichtiger ist es, dem konflikt-sensiblen Entwickler, der sich leicht zurückzieht, ein Umfeld anzubieten, in dem er sich offen und angstfrei zeigen kann. Das funktioniert dort, wo eine wohlwollende und wertungs-freie Atmosphäre herrscht und jeder sein darf, wie er ist.

Das Schwierige daran: Alle müssen mitmachen. Das Gute daran: Jeder kann damit anfangen. Das ist nichts, was das Management bestimmt und was dann so gemacht wird. Es ist eine Frage der Haltung jedes Einzelnen und des Umgangs miteinander – eben wieder die Kultur des Teams oder des Unternehmens. Und wie in jedem systemischen Kontext führt jede Aktion zu einer Reaktion – jede Verhaltensänderung auch zu einer Verhaltensänderung des Gegenübers.

Fühlen Sie sich in Ihrem Kontext nicht wertgeschätzt, dann versuchen Sie doch das nächste Mal, wenn einer Ihrer Kollegen etwas Beeindruckendes erreicht hat, Ihre ehrliche Wertschätzung auszudrücken – ganz ohne Hintergedanken. Das mag nach einem billigen Vorschlag klingen, wirkt in der Praxis aber oft Wunder.

Level 4: Anerkennung

Mit einem stabilen sozialen Umfeld, das wohlwollend mit allen Teammitgliedern umgeht, ist das nächste Level – die Anerkennung beziehungsweise Wertschätzung für meine erbrachte Leistung im Team – im Grunde ein Kinderspiel.

Worum geht es im Konkreten? Mit dem Bedürfnis nach Anerkennung bestätigt mir das Umfeld, dass ich wertvoll für die Gruppe bin und somit meinen Platz gut ausfülle. Entgegen dem ersten Impuls geht es aber keineswegs darum, nur durch übermenschliche Leistung eine Daseinsberechtigung in der Gruppe zu „erarbeiten“. Dieses Verhalten geht nämlich oft mit einem geringen Selbstwert einher, der einen immer weiter in die Leistungs-spirale presst.

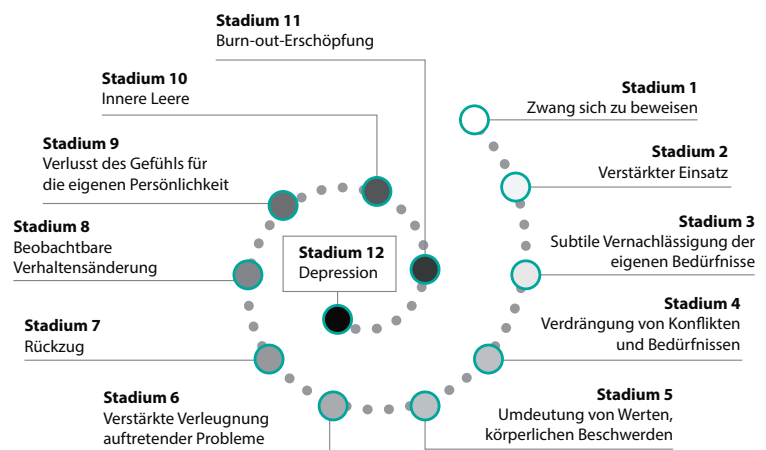
Vielmehr geht es darum, dass ich mein Möglichstes tue, um die Gruppe mit ihren Aufgaben vorwärtszubringen. Je nach Fähigkeit, Tätigkeitsfeld, Tagesform, gesundheitlichem Zustand kann das selbstverständlich variieren – das liegt in der Natur des Menschseins. Doch in allen Fällen werden die Mühen durch das Umfeld anerkannt und geschätzt.

Hier können wir den Flow-Tunnel wieder aufgreifen. Je nach Fähigkeit, Erfahrung und Arbeitsbelastung bewegt sich jeder Mensch in einem individuellen Leistungstunnel, in dem er eine gesunde Leistung bringen kann und dabei auch noch Spaß hat. Unterschreiten die angelegten Aufgaben die Ansprüche des Mitarbeiters, so langweilt er sich. Überschreiten sie aber dessen Leistungsfähigkeit – etwa durch permanente Ticketflut –, so gerät er in Stress. Das mag für eine Weile funktionieren, überlastet aber auf Dauer den Mitarbeiter.

Kommen hier jetzt zwei fatale Dinge zusammen, die in der freien Wildbahn recht oft vorzufinden sind, kommt es zur Ka-

tastrophe. Besteht der innere Glaube daran, Übermenschliches leisten zu müssen, um sich einen Platz in der Gruppe zu sichern, sorgen Mitarbeiter oft sogar selbst für den Stress und die Überarbeitung, indem sie sich aktiv Arbeit aufhalsen (um nützlich zu sein) und sich gleichzeitig in Perfektion verlieren (man wills ja gescheit machen). Niemals ist die Arbeit gut genug, nie ist ein Ende der Arbeit in Sicht, und im schlimmsten Fall führt das sogar zu mitleidigen Reaktionen des Umfelds – nicht zur gewünschten Anerkennung. Diese Menschen verfallen in den Glauben, nicht genug zu sein, verlieren den Sinn und stürzen in den berühmten Burn-out (Bild 4).

Was kann also getan werden, um jedem Menschen die Anerkennung zukommen zu lassen, die er benötigt? Im Grunde sind es die gleichen Mechanismen, die bereits zu einem gesunden Zugehörigkeitsgefühl führen. Eine starke Gruppe, die alle ihre Mitglieder schätzt und akzeptiert, wie sie sind,



Die Burn-out-Spirale zeigt den typischen Verlauf in die Depression (Bild 4)

bringt bereits alle Grundvoraussetzungen mit, um diese Anerkennung an alle ihre Mitglieder weiterzureichen.

Das Perfide ist auch hier wieder die Subjektivität der Sache. Wie misst man Anerkennung? Braucht jeder gleich viel davon? Ist der Empfänger empfänglich dafür? Wie gerade im Falle eines Burn-out-Kandidaten beschrieben, ist die Erwartungshaltung beziehungsweise das notwendige Maß jenseits von realistischen Parametern und wird niemals genug sein. Klagt also jemand darüber, nicht genug Wertschätzung erhalten zu haben, kann das unterschiedliche Ursachen haben.

Das Naheliegendste: Der Mitarbeiter wird nicht wertgeschätzt. Es kann aber ebenso bedeuten, dass die Wertschätzung anders ausgedrückt wird, als der Mitarbeiter sich das wünscht. Mein Lieblingsbeispiel ist, dass ein Unternehmen Erfolgsprämien auszahlt, der Mitarbeiter aber lieber ein ehrliches und herzliches Dankeschön gehabt hätte. Und dann gibt es eben noch die Möglichkeit, dass die Erwartungshaltung des Mitarbeiters nicht in einem gesunden Verhältnis steht und eher aus der Not heraus geboren ist.

Sollte das Thema Wertschätzung in der eigenen Motivationskette das schwache Glied sein, so lohnt sich ein ehrlicher Blick auf sich selbst. Welche Art der Wertschätzung würde

mir guttun? Ist diese Erwartung realistisch? Warum könnte sie nicht stattgefunden haben? Ein offenes Gespräch mit Kollegen und dem gewünschten Sender der Wertschätzung wirkt hier Wunder. Hält man sich doch vor Augen, dass jeder Mensch von der zwischenmenschlichen Ebene profitiert und normalerweise gerne darauf eingeht.

Final Level: Selbstverwirklichung

Kommen wir zur Königsdisziplin der Bedürfniskette: dem Wunsch nach Selbstverwirklichung und Autonomie. Was das in unserer Flow-Situation, dem Leben im Moment, bedeutet, hatten wir weiter oben ja bereits angerissen. Hier war es die Freiheit, eigene Entscheidungen zu treffen. Sei es die Wahl des Lösungswegs, die Wahl des zu verwendenden Frameworks oder die Frage, ob die geschweiften Klammern nun in der vorherigen Zeile sitzen oder eine eigene Zeile bekommen. Oder lasse ich sie komplett weg?

Im Kern bedeutet Selbstverwirklichung aber weitaus mehr, als in einem Moment eine (halbwegs) freie Entscheidung zu treffen. Es umfasst ebenso die Freiheit, im geeigneten Augenblick den für mich sinnvollsten Weg einzuschlagen. Also selbstwirksam zu sein. Das hat im echten Arbeitsleben unterschiedliche Ausprägungen. Natürlich braucht es Code-Guidelines und klare Prozesse, von denen Entwickler nicht nach Belieben abweichen können. Aber ist das schon eine Einschränkung der Selbstwirksamkeit?

Dieses Thema spielt sich hier auf einer anderen Ebene ab. Selbstwirksam bin ich auch, wenn ich im Konsens mit dem Team Kompromisslösungen erarbeite. Dann geht es zwar nicht vollständig nach meinem Willen, aber ich hatte die Chance, Einfluss zu nehmen. Und darum geht es im Grunde. Es steht also nicht im vermeintlich krassen Gegensatz zum Teamgedanken. In einem Unternehmen, in dem die Regeln eben nicht nach dem „Weil es halt so ist“-Muster gestrickt sind, sondern jederzeit durch gute Argumente und einen gemeinsamen Findungsprozess verändert werden können, bringt das schon einen großen Teil Selbstwirksamkeit eines jeden Mitarbeiters ins Rennen.

Ähnlich verhält es sich bei der persönlichen Entwicklung. Denn auch dort wirken sich Selbstbestimmung und Selbstverwirklichung aus. Indem ich mich nach meinen Vorstellungen persönlich weiterentwickeln kann – sei es durch Fortbildungen, Positionswechsel innerhalb des Unternehmens oder die Gestaltung meines Arbeitsmodus –, bin ich selbstwirksam. Auch hier kommt schon wieder das wohlwollende Team ins Spiel, das mich als Kollegen so akzeptiert, wie ich bin und wie mich weiterentwickeln will.

Bei diesem Aspekt kommt es in der Tat sehr auf die Möglichkeiten innerhalb des Unternehmens an. Aus Sicht des Arbeitgebers gibt es hier ganz viel Potenzial. Zumal ein Investment in den Mitarbeiter nicht nur dessen technische Expertise erhöht, sondern ihn rundum glücklicher macht.

Als Arbeitnehmer lässt sich nur so viel Autonomie einnehmen, wie es das Umfeld gestattet. Das Gute ist: Lebensentscheidungen, Fortbildung und Persönlichkeitsentwicklung finden auch außerhalb des Unternehmens statt. Fortbildungen und Weiterentwicklung lassen sich bequem auch extern

konsumieren und ebenso in externen Projekten einbringen, um die eigene Selbstverwirklichung dort anzuwenden.

Über eines muss man sich aber im Klaren sein: Wenn man sich außerhalb des Unternehmens immer neuen Herausforderungen stellt und sich weiterentwickelt, während man weiterhin in den starren, stagnierenden Vorstellungen des Unternehmens arbeitet, wird die entstehende Lücke immer größer und offensichtlicher. Dann resigniert man entweder (und verharret im Unternehmen), oder man zieht weiter.

Fazit

Was für ein Ritt durch die Gefühlswelt des Menschen! Ich hoffe, dieser Artikel hat gezeigt, dass die Zufriedenheit des Menschen, die Motivation zur Arbeit und das Finden innerer Erfüllung keineswegs fremdgesteuerte Maßeinheiten sind, auf die man als Einzelperson keinen Einfluss hat. Vielmehr ist es eine Frage der inneren Haltung und davon, wie man mit den äußeren Einflüssen umgeht.

Das Coole an diesem Thema ist, dass man im Prinzip keine wirklich konkreten Tipps niederschreiben muss. Mal abgesehen davon, dass die konkreten Maßnahmen ohnehin von Person zu Person vollkommen unterschiedlichen Nutzen haben, reicht hier schon eine ganz wichtige Sache: Seien Sie sich Ihres Bedürfnismangels bewusst! Denn nur Sie wissen, was Ihnen persönlich am besten hilft.

Wenn ich also drei Tipps aufschreiben müsste, wären es die folgenden:

- Hören Sie auf Ihre inneren Bedürfnisse – sie verraten Ihnen, was schief läuft.
- Öffnen Sie sich Ihren Mitmenschen und Kollegen mit Wünschen und Bedürfnissen.
- Indem Sie sich ins System einbringen, verändern Sie das System.

In diesem Sinne wünsche ich Ihnen eine motivierte und erfüllte Zeit. Und vielleicht – ganz vielleicht – ja sogar wieder ein bisschen dieses Kribbeln in den Fingern. ■

[1] Mihaly Csikszentmihályi, *Finding Flow: The Psychology of Engagement with Everyday Life*, Basic Books, 1997, ISBN 978-0-465-04513-6

[2] Abraham Maslow, *Motivation and Personality*, Harper, 1954

[3] Edward L. Deci, Richard M. Ryan, *Intrinsic Motivation and Self-Determination in Human Behavior*, Springer, 1985, ISBN 978-1-4899-2273-1



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

A2504-05Motivation

ARBEITSMARKT

TRENDS UND JOBS FÜR ENTWICKLER

Bertelsmann Stiftung

Den Job zu wechseln zahlt sich aus

Der Fachkräftemangel auf dem deutschen Arbeitsmarkt bietet Arbeitnehmern große Chancen. Tatsächlich werden jedes Jahr rund 30 Prozent der Beschäftigungsverhältnisse neu begonnen oder beendet.

Je größer der Sprung in einen neuen Job, desto stärker zahlt er sich aus: Einen Berufswechsel, verbunden mit einem höheren Anforderungsprofil, belohnt der Arbeitgeber im Schnitt mit einem Gehaltsplus von 6,9 Prozent.

Wer den Beruf wechselt, ohne dass sich das Anforderungsniveau ändert, kann immerhin ein Plus von fünf Prozent erwarten. Wer im gleichen Beruf bleibt, kommt noch auf 3,6 Prozent mehr. Das zeigt eine Studie der Bertelsmann Stiftung und des RWI – Leibniz-Institut für Wirtschaftsforschung. Ausgewertet wurden Daten der Jahre 2013 bis 2021 (Bild 1).

Zum Gehaltsplus kommt beim Jobwechsel ein Plus bei der Zufriedenheit. Wechsler sind auf einer Zufriedenheitsskala von 0 bis 10 Punkten an ihrer neuen Arbeitsstelle im Schnitt 1,1 Punkte zufriedener als vorher. Wer nicht wechselt, verliert dagegen im gleichen Zeitraum minimal um 0,2 Punkte. Zugleich besagt die Studie: Wer nicht wechselt, weil er im bestehenden Job rundum zufrieden ist, liegt auf der Zufriedenheitsskala bei 7,3 Punkten – und damit durchschnittlich 0,9 Punkte höher als die Wechsler.

Die Zufriedenheit beim Wechsel hängt nicht so sehr mit dem Plus auf dem Gehaltscheck zusammen. Steigt der Lohn von einem Jahr zum nächsten um zehn Prozentpunkte, steigt die Zufriedenheit im Schnitt nur um 0,05 Punkte. Das Wohlbefinden speist sich aus den neuen Aufgaben und dem Verhältnis zu neuen Führungskräften und Kollegen. „Ein höheres Einkommen ist ein wichtiges Argument“, sagt

Roman Wink, Arbeitsmarktexperte der Bertelsmann Stiftung. „Aber die wichtigere Botschaft ist: Arbeitgeber, die ihre Mitarbeiter halten wollen, müssen dafür sorgen, dass sie sich an ihrer Arbeitsstelle wohlfühlen.“

Apptio

Den KI-Nutzen bewerten

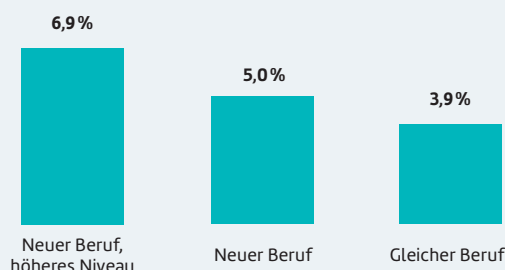
In einem bekannten Cartoon sagt ein Entwickler grinsend: „Gerade hat mir die KI in zwei Minuten 10 000 Zeilen Code geschrieben.“ Daraufhin trübt sich seine Miene ein, und er ergänzt: „Jetzt sitze ich wieder zwei, drei Tage dran, um den Code zu debuggen.“ Der Joke zeigt das Dilemma: Den Nutzen von KI kann man nicht anhand der anfänglichen Zeitersparnis bewerten, sondern man muss den Gesamtaufwand betrachten, und eine Führungskraft muss letztlich beziffern, wie hoch die Kosten für den Einsatz von KI sind und wie hoch der Return of Investment (ROI).

Laut dem neuen „Global Tech Investments Report“, den das IBM-Unternehmen Apptio veröffentlicht hat, erwarten im Jahr 2025 mehr als 90 Prozent der Unternehmen eine Erhöhung ihrer Technologiebudgets zur Unterstützung der bestehenden Infrastruktur, zur Aufrechterhaltung der betrieblichen Grundlagen und zur Förderung neuer Geschäfte durch Innovationen. Gleichzeitig sagen 55 Prozent der Führungskräfte, dass sie nicht über ausreichende Informationen verfügen, um Technologieausgaben effektiv zu bewerten.

Zudem besagt der Bericht, dass der Einsatz von KI rapide zunimmt, wobei Datenanalyse, Prozessautomatisierung und Cybersicherheit derzeit die wichtigsten Anwendungsfälle sind (Bild 2).

Um die wachsenden Investitionen in KI-Initiativen zu rechtfertigen, müssen die Führungskräfte den ROI bewerten, indem sie die Umsatzsteigerung, die betriebliche Effizienz und die

● Gehaltsplus beim Jobwechsel

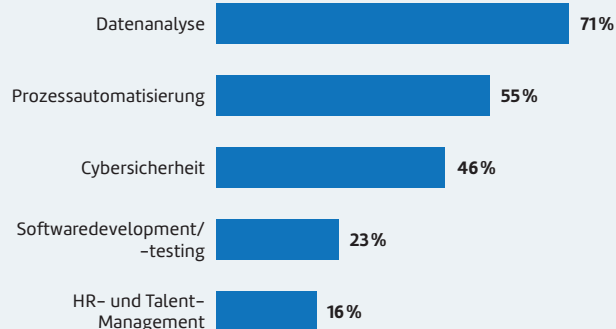


Ein Wechsel im gleichen Beruf verspricht 3,9 Prozent mehr Gehalt. In einem neuen Beruf mit höherem Niveau winkt ein Plus von 6,9 Prozent (Bild 1)

dotnetpro 4-5/2025

Quelle: Bertelsmann Stiftung / RWI – Leibniz-Institut

● KI-Einsatz findet statt für ...



Die wichtigsten KI-Anwendungsfälle sind derzeit Datenanalyse, Prozessautomatisierung und Cybersicherheit (Bild 2)

dotnetpro 4-5/2025

Quelle: Apptio Global Tech Investments Report, n>1000

Zahl des Monats

Cyberangriffe haben 2024 weltweit um **44 Prozent** zugenommen, und fast die Hälfte der kleinen und mittleren Unternehmen in Europa war in den vergangenen zwei Jahren von einem solchen Angriff betroffen. Gleichzeitig besteht in Europa im Bereich Cybersicherheit ein Fachkräftemangel von bis zu 500 000 Experten.

Quelle: Google

verbesserte Entscheidungsfindung quantifizieren. Gelingt ihnen dies, können sie künftigen Budgeterhöhungen zuversichtlich entgegensehen.

Bitkom

Digitale Souveränität

In einer Studie mit dem Titel „Digitale Souveränität“ hat der Branchenverband Bitkom untersucht, wie abhängig deutsche Unternehmen von digitalen Technologien und Services aus dem Ausland sind. Grundlage ist eine repräsentative Befragung von 603 Unternehmen aller Branchen in Deutschland ab 20 Beschäftigten. Mehr als 87 Prozent der Unternehmen beziehen digitale Technologien beziehungsweise Leistungen aus dem Ausland. Von diesen Unternehmen bezeichnen sich 56 Prozent als „stark“ und weitere 34 Prozent „eher stark“ abhängig von den Digitalimporten (Bild 3). Lediglich sechs Prozent der Befragten sagen, sie seien „eher nicht abhängig“, und wei-

tere drei Prozent sehen sich als „überhaupt nicht abhängig“ von Digitalimporten.

Obwohl die meisten Unternehmen (87 Prozent) bereits vielfältige Maßnahmen für den Fall treffen, dass bestimmte digitale Technologien oder Services eine Zeit lang nicht mehr geliefert oder importiert werden könnten, gehen nur drei Prozent der Unternehmen davon aus, dass sie zwei Jahre oder länger ohne Digitalimporte überleben könnten. Mehr als die Hälfte sagt, dass sie nicht länger als ein Jahr überbrücken könnten (Bild 4).

TU München

Mehr Fachkräfte für Cybersicherheit

Die Technische Universität München (TUM) bietet für über 200 Studierende mit unterschiedlichem Hintergrund ein neues Programm im Bereich der Cybersicherheit an. Die Studierenden haben dabei die Möglichkeit, die im Unterricht er-

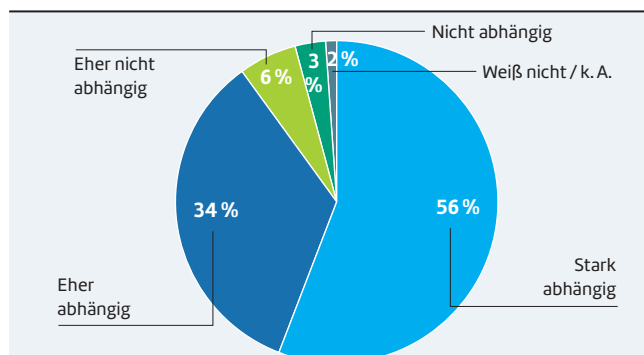
worbenen Fähigkeiten in realen Situationen anzuwenden. Das Ziel ist es, lokale Organisationen, die von Cyberangriffen besonders bedroht sind, bei der Cyberabwehr zu unterstützen.

Durch das neue, von Google geförderte Programm werden mehr als 250 lokale Unternehmen und gemeinnützige Organisationen unterstützt, die als besonders gefährdet für Cyberangriffe gelten. „Wir sind mit einem Mangel an Fachleuten für Cybersicherheit konfrontiert, und das zu einer Zeit, in der neue Technologien wie KI sowohl Chancen als auch Bedrohungen für das Feld darstellen“, sagt Dr. Wieland Holfelder, Vice President Engineering, Regional CTO für Google Cloud Security und Sovereignty & Site Lead Google München. „Dieses Modell ist eine Win-win-Situation. Studierende erlernen praktische Fähigkeiten im Bereich Cybersicherheit, und gefährdete Organisationen werden vor Cyberangriffen geschützt. Wir sind stolz darauf, die TUM und

andere Universitäten in der gesamten EMEA-Region dabei zu unterstützen, eine KI-versierte Belegschaft für Cybersicherheit aufzubauen.“

Der Lehrplan des Programms will theoretisches Wissen mit praktischen und realen Anwendungen verbinden. Die Studierenden sollen an unternehmensspezifischen Herausforderungen arbeiten. Dabei werden Theorie und Praxis durch das Lernen mit Serious Gaming und Hacking-Wettbewerbe kombiniert, und die Studierenden kommen in Kontakt mit Cybersicherheitskonzepten. Die Praxisprojekte befassen sich mit Technologien für elektronische Identifikation, Authentifizierung und Vertrauensdienste (eIDAS). Dazu zählen spezielle Anwendungen, die eine sichere Anmeldung ermöglichen, elektronische Signaturen sowie elektronische Siegel. Diese nutzen spezielle Sicherheits-Hardware und den deutschen Personalausweis als vertrauenswürdige Grundlage.

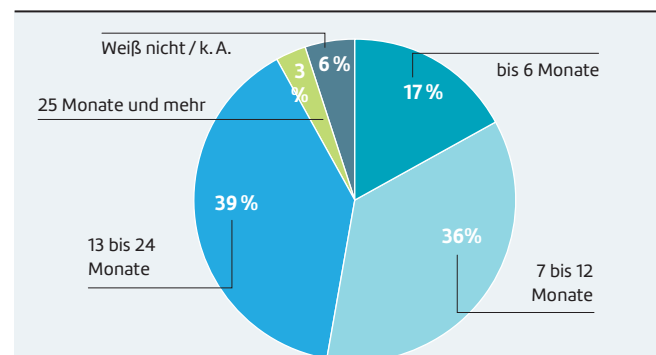
● Abhängigkeit von Digitalimporten



90 Prozent der deutschen Unternehmen fühlen sich „stark“ beziehungsweise „eher stark“ abhängig von Digitalimporten (Bild 3)

dotnetpro 4-5/2025 Quelle: Bitkom, n=526, repräsentativ, Rundungsdifferenzen

● Überleben ohne Digitalimporte



Rund 17 Prozent der Unternehmen sagen, dass sie ohne Digitalimporte nicht länger als sechs Monate weiterproduzieren könnten (Bild 4)

dotnetpro 4-5/2025 Quelle: Bitkom, n=526, repräsentativ, Rundungsdifferenzen



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 einbuchten unter jobs.developer-media.de



Diese und viele weitere aktuelle Job-Angebote finden Sie auf jobs.developer-media.de

robomotion GmbH	Senior Softwareentwickler (m/w/d) C# .NET	Leinfelden-Echterdingen
KOMSA AG	Vendor Manager (m/w/d) SaaS	Hartmannsdorf, Osnabrück, Kamen, Wolfsburg, bundesweit (Remote)
Landesbank Hessen-Thüringen	Position Application Manager SAP ERP (w/m/d)	Offenbach am Main, Erfurt
DFS Deutsche Flugsicherung GmbH	Software Engineer (w/m/d) für Flugsicherungssysteme	Langen (Hessen)
PMG AG	Fachinformatiker / Systemtechniker (Voice Solutions) (m/w/d)	Inning am Ammersee
ifp Executive Search Management Diagnostik	Bereichsleiter:in IT & Transformation	Aachen
LACOS GmbH	IT-Systemadministrator (m/w/d)	Zeulenroda-Triebes
Northrop Grumman LITEF GmbH	Entwicklungingenieur (m/w/d) für Software und System-Algorithmen	Freiburg
ESG Elektroniksystem- und Logistik GmbH	Softwareentwickler*in in der Drohnendetektion und -abwehr (w/m/d)	Fürstenfeldbruck
Hensoldt	Softwareentwickler*in C++ (w/m/d)	Wetzlar oder Oberkochen

Ihre Anzeige ist noch nicht dabei? Rufen Sie uns an:

Reinhold Fritsch, Tel.: +49 731 88005-8285, E-Mail an: jobs@developer-media.de

DIE DEVELOPER WEEK 2025 UND IHRE TRACK CHAIRS

Die ganze Welt der Softwareentwicklung

Das Programm der Developer Week formt ein riesiges Angebot an Wissen, das mit der Hilfe renommierter Expertinnen und Experten entsteht – den Track Chairs.

Was setzt man an den Anfang eines Artikels über das Programm der Developer Week [1], die vom 30. Juni bis zum 3. Juli 2025 im m:con Congress Center in Mannheim stattfindet? Preist man die Fortbildung zu den Soft Skills an oder lieber Sessions zu harten, technischen Themen? Hebt man Cloud-Informationen hervor, oder geht man etwa auf Nachhaltigkeit ein?

Sinn und Zweck der Developer Week ist es, den Teilnehmerinnen und Teilnehmern Wissen in ganz vielen Dimensionen zu vermitteln. Alles, damit Entwicklerinnen und Entwickler ihren Job besser ausführen können und wieder mit Motivation vollgetankt in den Alltag starten.

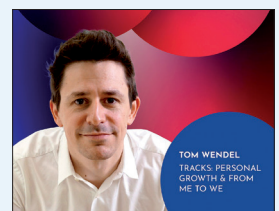
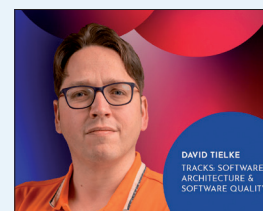
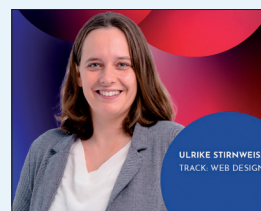
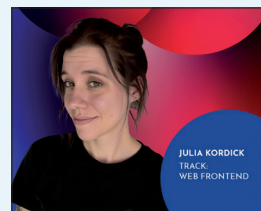
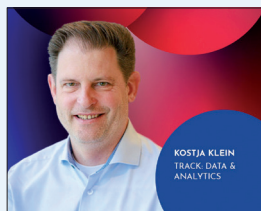
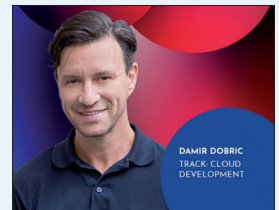
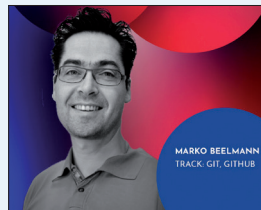
Und so lautet die überraschende Antwort auf diese Frage: den Menschen. Denn um ein solches Programm in der Breite und Tiefe zusammenzustellen, bedarf es Menschen, die diese Aufgabe auf sich nehmen.

Rund 160 Sessions sind zu besetzen und somit rund 160 Themen, Sprecherinnen und Sprecher auszuwählen. Vielfalt ist gefordert; Belanglosigkeit hingegen sollte im Programm der Developer Week keinen Platz haben.

Glücklicherweise gibt es diese Menschen, die sich bereit erklären, die Aufgabe des Track Chairs zu übernehmen. Sie sind nicht nur Expertin oder Experte in einem Bereich. Sie stellen sich der Mammutaufgabe, die besten Sessions auszu-

Das sind die Track Chairs der Developer Week

Unsere Expertinnen und Experten brennen für ihren Themenbereich – ob es sich um Architektur, Generative KI, Cloud, Soft Skills oder Green IT handelt. Auf du und du mit Trends und tiefem Fachwissen sorgen sie für das umfassende und vielfältige Programm der großen Softwareentwicklerkonferenz.



suchen. Aus weit über 1000 Vorschlägen müssen die Track Chairs die besonderen Sessions herausfiltern und aus diesen dann den Themenstrang zusammenstellen. Zusammen mit Ersatzsessions für den Fall, dass eine Sprecherin oder ein Sprecher ausfällt, bedeutet das, dass aus am Ende nur sieben Sessions pro Track übrig bleiben.

Ein großer Dank geht deshalb an unsere Expertinnen und Experten, die sich die Zeit nehmen, um für die Developer Week das Programm zu formen. Sie sollen hier den Platz erhalten, der ihnen gebührt.

26 Tracks Chairs, eine Welt

„Die ganze Welt der Softwareentwicklung“ lautet die Überschrift, aber das ist natürlich ein bisschen übertrieben: COBOL werden Sie genauso wenig im Programm finden wie Java-Programmierung. Aber wenn Sie sich in C#, JavaScript, TypeScript, KI, .NET, Web, Mobile, Datenbanken, Methoden und Vorgehensweisen, Architektur, Cloud oder Green IT zuhause fühlen, sind Sie auf der Developer Week genau richtig. Das Programm ist in 29 Themenstränge gegliedert. 26 Track Chairs sind dafür verantwortlich. Die Diskrepanz in den Zahlen ergibt sich, da manche Track Chairs mehrere Tracks verantworten. So kümmert sich Christian Giesswein um *.NET Doing* und *.NET Fundamentals*. Andere Tracks wiederum werden von zwei Track Chairs verantwortet. Den Track *Software Design Principles* stellen beispielsweise Carola Lilienthal und David Tielke gemeinsam zusammen.

Summa summarum kommen wir auf 26 Track Chairs, die hier alle im Bild zu Ehren kommen sollen.

Eine Session wählen, trotzdem nichts verpassen

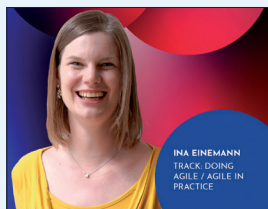
Wenn Teilnehmerinnen und Teilnehmer auf viele gleichzeitig stattfindende Vorträge stoßen, wird die Wahl zur Qual: Aus neun respektive zehn gleichzeitig stattfindenden Vorträgen einen zu erwählen und neun anderen einen Korb zu geben, ist mit Abwägen von Prioritäten verbunden.

Aber diese harte Auswahl bleibt ohne Folgen: Denn je nach Ticketart können die Teilnehmenden der Konferenz die Aufzeichnungen aller Sessions für drei beziehungsweise zwölf Monate ansehen. Sie verpassen also nichts, weil Sie sich vor Ort nicht zehnteilen können. Auch Slides und sonstige Materialien, die die Sprecherinnen und Sprecher zur Verfügung stellen, gibt das Team der Developer Week im Nachgang an die Teilnehmenden weiter. So erhalten Sie noch viel mehr, als das schlichte Ticket am Anfang suggeriert.

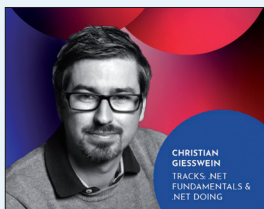
Ticket holen, dabei sein

Warum also noch lange überlegen? Bei dieser Masse an Themen, Expertinnen und Experten, Sessions, Workshops und Branchentalk ist das Dabeisein ein Muss. Teil der Community sein und sich frische Motivation für den Alltag holen – das bietet Ihnen die Developer Week. Ticket und hin! ■

[1] Developer Week, www.developer-week.de



INA EINEMANN
TRACK DOING
AGILE / AGILE IN
PRACTICE



CHRISTIAN
GIESSWEIN
TRACKS .NET
FUNDAMENTALS &
.NET DOING



FABIAN
GOSEBRINK
TRACK
WEB BACKEND



THOMAS CLAUDIUS
HUBER
TRACK
.NET FRONTEND



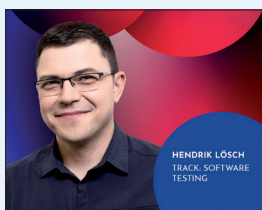
THORSTEN KANSY
TRACK: DATABASES



CHRISTIAN LIEBEL
TRACK: WEB HYBRID



CAROLA LILIENTHAL
TRACKS SOFTWARE
ARCHITECTURE &
SOFTWARE QUALITY



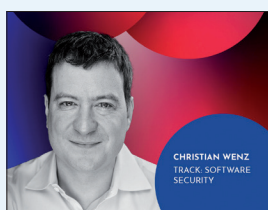
HENDRIK LÖSCH
TRACK: SOFTWARE
TESTING



STEFAN MINTERT
TRACKS PERSONAL
GROWTH & FROM
ME TO WE



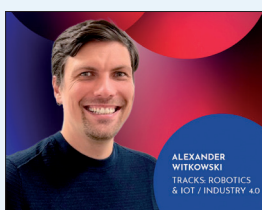
AYDIN MIR
MOHAMMADI
TRACK:
SUSTAINABILITY



CHRISTIAN WENZ
TRACK: SOFTWARE
SECURITY



CHRISTIAN WEYER
TRACK GENAI
TOOLS & SOLUTIONS



ALEXANDER
WITKOWSKI
TRACKS ROBOTICS
& IOT / INDUSTRY 4.0



LERNSTOFF

dnpCode A2504-05Lernstoff

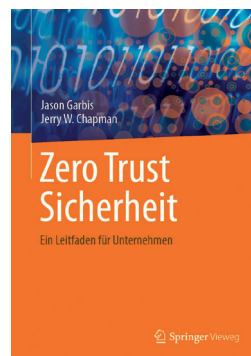
GEWÄHRLEISTUNG VON IT-SICHERHEIT

Vertrauen

Die immer größer werdenden Sicherheitsprobleme der IT-Welt haben ein von militärischen Erfahrungen beeinflusstes rigoroses Sicherheitskonzept in den Vordergrund gerückt, bei dem ein grundsätzliches Misstrauen vorherrscht.

Der Titel geht zunächst auf die Frage ein, warum Zero Trust erforderlich ist. Es folgt eine kompakte Chronik, wie es zur Entwicklung der aktuellen Zero-Trust-Umgebung gekommen ist und was Zero Trust im Kern ausmacht. Auf dieser Basis geht es insbesondere darum, wie Zero Trust in die vorhandene Betriebsarchitektur am besten integriert werden kann. Als Beleg für eine gelungene Vorgehensweise wird Google Zero Trust beschrieben. Die Bundesbehörden in den USA sind inzwischen auf Zero Trust verpflichtet.

Im zweiten Teil werden nacheinander die wesentlichen Komponenten vorgestellt. Allem voran geht es dabei um die Implementierung zur Kontrolle der Identität. Jeder, der auf eine Ressource zugreifen will, muss zunächst seine Berechtigung eindeutig nachweisen.



Jason Garbis, Jerry W. Chapman

Zero Trust Sicherheit. Ein Leitfaden für Unternehmen

Springer, 2024, 330 Seiten, Deutsch, ISBN 979-8-8688-0104-4, 29,99 Euro, E-Book 14,99 Euro

Zielgruppe: Verantwortliche für IT-Sicherheit und Führungskräfte

Voraussetzungen: Keine

Leseprobe: (Anmeldung erforderlich)

www.dotnetpro.de/SL2504-05Lernstoff1

Anschließend geht es um die Fragen der Netzwerkinfrastruktur, die Zugangskontrolle und die Systeme zur Erkennung und Abwehr von Eindringlingen. Dazu gehören Next-Generation Firewalls, spezielle Sicherheitsmaßnahmen in Cloud-Umgebungen und die Anforderungen an IOT-Geräte.

Da die Umstellung auf Zero Trust in der Regel ihre Zeit braucht und ohne Zustimmung und rege Mitarbeit nur schwer gelingen kann, werden zahlreiche praktische Ratschläge zu Anwendungsproblemen beigegeben. (vh) ■

EFFEKTIVE IT-PLANUNG

Strategie

Der in zweiter Auflage erschienene Leitfaden adressiert die zentralen Herausforderungen und Chancen der strategischen IT-Planung. Die Autorin verbindet theoretisches Wissen mit praktischen Ansätzen, um IT-Strategen, Unternehmensführer und digitale Change-Manager bei der Navigation durch die komplexen Anforderungen der digitalen Transformation zu unterstützen.

Das Werk bietet einen umfassenden Überblick über die strategische IT-Planung und behandelt ein breites Themenfeld, das von der Einbindung in die Unternehmensstrategie bis hin zu praktischen Umsetzungshilfen reicht. Die Inhalte sind praxisnah und unterfüttert mit Beispielen wie Bebauplänen oder CIO-Cockpits, die reale IT-Management-Szenarien abbilden. Dabei werden fundierte Erklärungen zu IT-Governance, Enterprise Architecture Management und agilem Demand Management gegeben. Besonders hilfreich sind die systematischen Leitfäden, die klare Anleitungen zur



Inge Hanschke

Einfache & effektive strategische IT-Planung

Hanser, 2. Auflage, 2024, 432 Seiten, Deutsch, ISBN 978-3-446-47676-9, 49,99 Euro

Zielgruppe: IT-Strategen, Führungskräfte

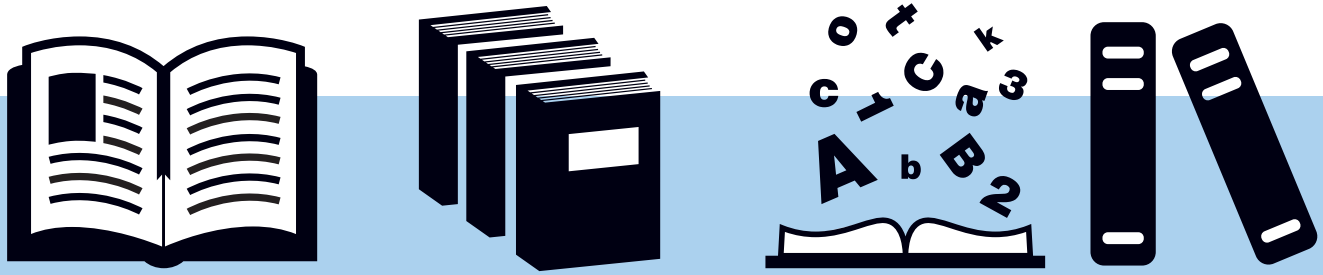
Voraussetzungen: Keine

Leseprobe:

www.dotnetpro.de/SL2504-05Lernstoff2

Entwicklung von IT-Strategien und Zielbildern bieten. Zudem wird auf aktuelle Trends wie Digitalisierung, die VUCA-Welt und die Auswirkungen von Pandemie und Energiekrise eingegangen.

Das Buch ist ein essenzielles Nachschlagewerk für jeden, der die strategische Rolle der IT in einem Unternehmen optimieren möchte. Trotz des hohen Detailgrads bleibt das Buch zugänglich und praxisnah, was es zu einer wertvollen Ressource in der IT-Strategie macht. (fd) ■



ERSTE SCHRITTE

SQL

Um den Einstieg in ein neues Gebiet zu erleichtern, ist die Konzentration auf das Wesentliche ebenso hilfreich wie eine überlegte Abfolge der Lernschritte. Autor Erik Schönwälder verteilt das gesamte zu erlernende Material auf zwei Wochen, und um den Erfolg zu unterstützen, wird der Zugang zu einer Beispieldatenbank als Spielwiese beigegeben.

Thema ist das Anlegen von und der Umgang mit relationalen Datenbanken, die Arbeit mit den dazugehörigen Datenbankmanagementsystemen und das Erlernen der Abfragesprache SQL. Alle Beispiele verwenden die Software von MySQL oder PostgreSQL als Muster, andere Varianten werden als nur leicht unterschiedliche Dialekte von SQL interpretiert. Der gemeinsame Kern der SQL-Syntax steht also immer im Mittelpunkt.

Nach der Einrichtung der Arbeitsumgebung werden die Kernbefehle der Abfragesprache direkt an einfachen Beispielen ausprobiert. In den nächsten Schritten werden die Möglichkeiten vorgestellt, die Datenmengen auf die Ziela-

Erik Schönwälder
SQL Schnelleinstieg
 Datenbanken abfragen und verwalten in 14 Tagen
 Einfach und ohne Vorkenntnisse
 Zahlreiche Praxisbeispiele
 mitp

Erik Schönwälder
 SQL Schnelleinstieg. Datenbanken abfragen und verwalten in 14 Tagen
 mitp, 2024, 272 Seiten, Deutsch, ISBN 978-3-7475-0868-8, 19,99 Euro
 Zielgruppe: Datenverwaltung
 Voraussetzungen: keine
 Leseprobe: www.dotnetpro.de/SL2504-05Lernstoff3

ten einzuschränken, sie zu gruppieren und zu aggregieren. Es folgen das Verknüpfen von Tabellen, die Manipulation von Daten und das Verknüpfen von Ergebnissen.

Die zweite Hälfte zeigt den Aufbau eigener Datenbanken, den Umgang mit den Datentypen sowie mit Primär- und Fremdschlüsseln und auch die Vorgehensweise beim Aktualisieren und Kontrollieren der Zugriffsrechte. Den Abschluss bilden Views, Indizes sowie Trigger, mit deren Hilfe erreicht werden kann, dass Änderungen an einer Stelle zu automatischen Reaktionen an anderer Stelle führen. (vh) ■

● Kurz vorgestellt

Hanspeter Mössenböck

Compilerbau

Behandelt die Grundlagen des Compilerbaus, von der Analyse bis zur Codeerzeugung. Außerdem: Übersetzungsprozesse mit attribuierten Grammatiken und Nutzung eines Compilergenerators. Beispiel ist ein MicroJava-Compiler, der ausführbaren Bytecode erzeugt.

dpunkt.verlag, 2024, 274 Seiten, Deutsch, ISBN 978-3-98889-008-5, 29,90 Euro

Ulrich Engelke

ChatGPT Power-Prompting

Das Buch liefert eine Einführung in das professionelle Prompt Engineering. Man erfährt, wie man ChatGPT Rollen zuweist, Kontext und Zielgruppe festlegt und differenzierte Antworten erzielt. Mit über 1000 Beispiel-Prompts.

mitp, 2024, 336 Seiten, Deutsch, ISBN 978-3-7475-0941-8, 24,99 Euro

Michael Kofler

Datenbanksysteme

Die Leser lernen grundlegende Begriffe und Konzepte kennen, die sie für gutes Datenbankdesign brauchen, und erfahren, wie Daten modelliert und strukturiert werden. Mit Schnellkurs in SQL und Praxiswissen zu Client-Entwicklung und DBMS-Administration.

Rheinwerk, 2. Auflage 2024, 782 Seiten, Deutsch, ISBN 978-3-367-10015-6, 49,90 Euro, E-Book 39,90 Euro

Ingrid Gerstbach

Customer Experience Design

Anhand von Fallstudien und praxiserprobten Methoden zeigt die Autorin, wie man Kundenerlebnisse verbessert und die Ergebnisse gezielter Designforschung in schlagkräftige Strategien umsetzt. Plus: Wie KI bei Personalisierung und Automatisierung im CX-Design helfen kann.

Hanser, 2025, 332 Seiten, Deutsch, ISBN 978-3-446-48086-5, 49,99 Euro

● Impressum

**Ebner Media Group GmbH & Co. KG****Büro München**

Brienner Str. 45 a-d, 80333 München

Sitz von Redaktion, Anzeigen, Vertrieb

Brienner Str. 45 a-d, 80333 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

Geschäftsführung

Annabel Ebner, Marco Parrillo

Chefredaktion

Fernando Schneider (ViSP)

Redaktion

Tilman Börner

Freie Mitarbeiter dieser Ausgabe**Redaktion**Ernst Altmannshofer, Sabine Block,
Bernhard Lauer**Autoren**Klaus Aschenbrenner, Elena Bochkor,
Dr. Fabian Deitelhoff, Mykola Dobrochynsky,
Robin-Niklas Fokuhl, Christian Giesswein, Peter Kröner,
Bernhard Lauer, Sebastian Meyer, Stefan Mintert,
Fabian Mohdrieh, Bernhard Pichler, Alexander Rampf,
Patrick Schnell, Dr. Holger Schwichtenberg,
Christian Sender, David Tielke, Uwe Vigenschow,
Helmut Vonhoegen, Tom Wendel, Christian Wenz,
Christian Weyer**Art Directorin**

Maria-Luise Steinkühler

Gestaltung**EMG Design Unit**Dagmar Breitenbach, Simone Köhnke,
Petra Reichensperner**Kundenservice / Einzelheftbestellung**EBNER MEDIA GROUP,
Brienner Str. 45 a-d, 80333 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 zweimonatlich.
Postvertriebskennzeichen B30751**Nachdruck**

© Nachdruck nur mit schriftlicher Genehmigung

Developer EventsCharlotte 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

AnzeigentarifeAnzeigenpreisliste gültig ab 1.1.2025
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 sechsmal im Jahr.
Das Einzelheft kostet im Inland 24,90 €.
Der Preis für das Premium-Plus-Abonnement
(6 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

Ebner Media Groupwww.ebnermedia.de 2, 29,
61-71, 77, 121, 129, 138-139, 147, 148

Autor:in werden!

Schreiben Sie
Artikel für die
dotnetpro


info@dotnetpro.de

Anbietersverzeichnis

Persönliche Beratung: Thomas Deck | +49 731 88005-8282 | thomas.deck@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 6-7/2025 erscheint am 19. Juni 2025

Foto: EMCenie



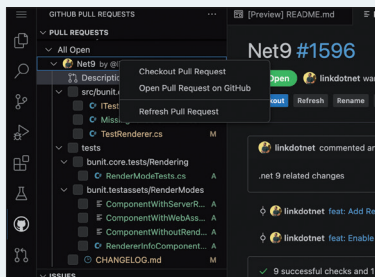
Low Code, No Code und die KI

Seit geraumer Zeit gewinnt das **Zusammenspiel von Low-Code-/No-Code-Methoden und -Tools mit Künstlicher Intelligenz** enorm an Bedeutung – nicht ohne Grund, denn der Einsatz von KI beschleunigt nicht nur die Entwicklung entsprechender Lösungen, sondern erleichtert die Erstellung komplexerer Anwendungen auch dadurch, dass der Zugang für Nutzer mit wenig beziehungsweise keinerlei Programmiererfahrung weiter erleichtert wird.

Grund genug also für die **dotnetpro 6-7/2025**, sich im Themenschwerpunkt eingehend mit Grundlagen und Möglichkeiten KI-basierter Entwicklung von No-Code-/Low-Code-Anwendungen zu befassen.

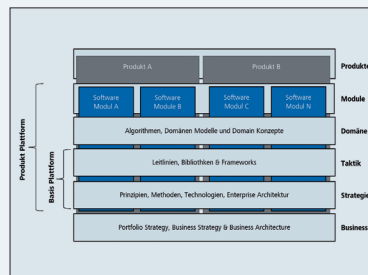
Collaborative Development

GitHub Codespaces senkt den Aufwand für das Aufsetzen von und die Arbeit an gemeinsamen Projekten signifikant.



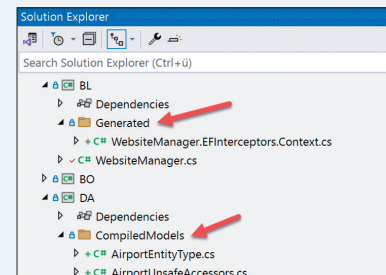
Modulithen in der Praxis

Bei der strukturierten Produktentwicklung kann das modulithische Architekturmuster entscheidend helfen.



Native AOT Compiling

Die Implementierung des Native-AOT-Compilers in EF Core 9.0 lässt noch Fragen offen und Luft nach oben.



End of file

Developer – an organism turning coffee into code.



Unsere digitalen Angebote



Wöchentlicher Entwickler-Newsletter
www.dotnetpro.de/newsletter



dotnetpro digital lesen: Android- und iPad-App
www.dotnetpro.de/dnpapps

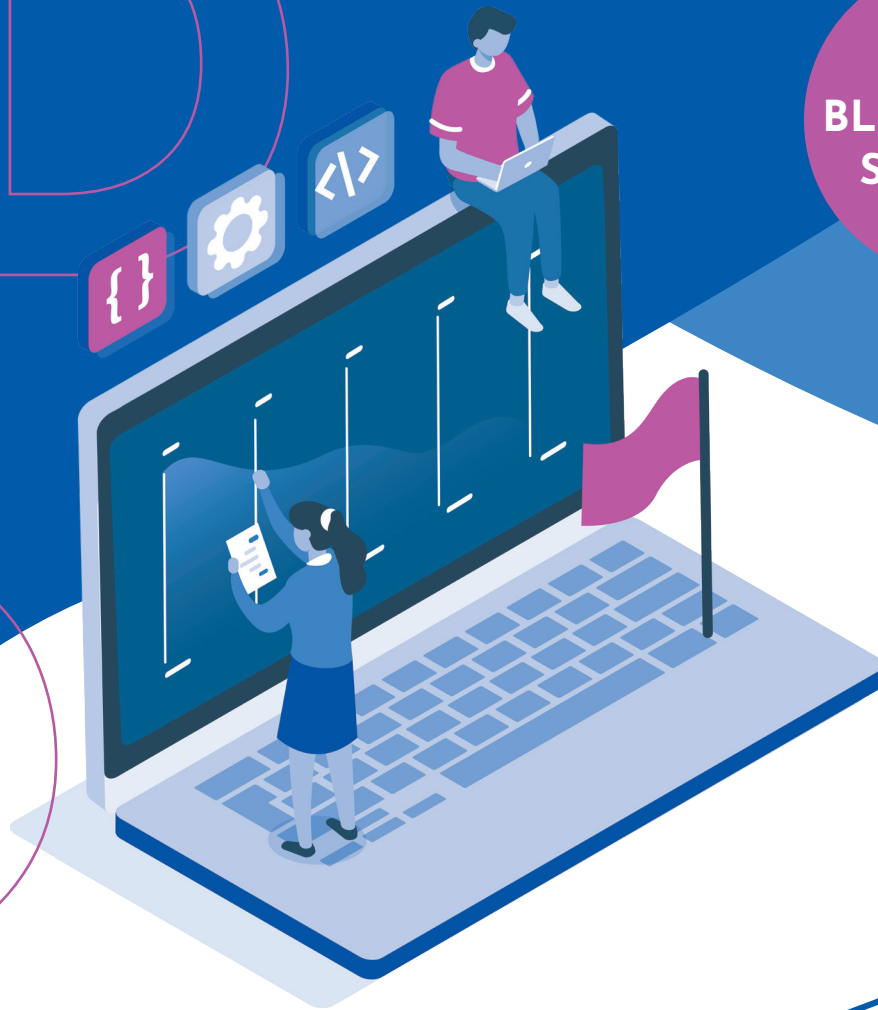


YouTube: Videos von den Konferenzen
www.youtube.com/@developerweek



Facebook: die News im Stream
<https://de-de.facebook.com/dotnetpro>

**JETZT
BLIND PRICE
SICHERN!**



.NET DEVELOPER CONFERENCE '25

DIE KONFERENZ FÜR .NET ENTWICKLER
24. - 27. NOVEMBER 2025 | PULLMAN COLOGNE

Die .NET Developer Conference ist eine der führenden Plattformen für Weiterbildung in .NET-Technologien. Mit einem umfangreichen Programm aus Vorträgen und Workshops bietet die Konferenz fundiertes Wissen und praxisnahe Einblicke in zentrale Themen des .NET-Bereichs im Zeitalter von KI. Ein Muss für alle, die ihr Know-how erweitern und aktuelle Trends verfolgen möchten.

dotnet-developer-conference.de | #DDC | Find us on [f](#) [i](#) [in](#)

Präsentiert von:



DEVELOPER WEEK '25

30. JUNI - 3. JULI 2025

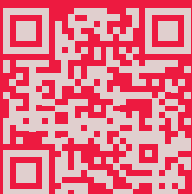
M:CON CONGRESS CENTER ROSENGARTEN
MANNHEIM

**JETZT
TICKET
SICHERN!**

DAS EVENT FÜR AI, CLOUD, WEB UND .NET

Software Design Principles | Software Architecture | Software Quality | Software Testing | Software Security | Doing Agile/Agile in Practice | DevOps | CI / CD
Git, GitHub | .NET Fundamentals | .NET Backend (Development)
.NET Frontend (Development) | .NET Doing | Web Frontend | Web Backend
Web Hybrid | Web Design | Web Languages | Cloud Development
Container | Data & Analytics | Databases | GenAI Fundamentals
GenAI Tools & Solutions | Personal Growth | From Me to We | Sustainability
Robotics | IoT & Industry 4.0

developer-week.de | #DWX | Find us on   



Präsentiert von: 