NEWS · OCTOBER 23, 2026

# Nicht nur Delphi: Wie drei Quartale 2026 meinen Job neu geschrieben haben

Ein Interessent verließ meine Website mit dem Eindruck "der macht nur Delphi" -- hier ist, wie drei Quartale 2026 tatsächlich aussahen, mit TypeScript, Swift, Kotlin, C# und Delphi in Produktion, und warum der Job heute Architektur und Verifikation heißt, nicht Fließendsprechen in einer Syntax.

By Dr. Holger Flick




Vor ein paar Wochen suchte ein Unternehmen jemanden, der eine Anwendung für sie baut. Jemand verwies sie auf meine Website. Sie lasen sie und gingen mit einem Eindruck wieder: "Ach, der macht nur Delphi. Wir benutzen kein Delphi." Ich habe das aus zweiter Hand gehört, und es war genau das Feedback, das ich nicht hören wollte -- nicht, weil es unfair war, sondern weil meine eigene Startseite es sich verdient hatte. Das Wort "Delphi" stand im Hero, in den Kennzahlen, im Footer und im ersten Service auf der Liste.

Die Sache ist aber die: In den neun Monaten seit Januar habe ich Produktionscode in TypeScript, Swift, Kotlin, C#, SQL und, ja, Delphi ausgeliefert. Nichts davon ist weit hergeholt, und nichts davon ist für mich in dem Sinne neu, auf den es ankommt. Neu ist, wie die Arbeit erledigt wird. 2025 hätte ich mich als Delphi-Experten beschrieben, der auch für das Web baut. Im Oktober 2026 ist diese Beschreibung schlicht falsch. Ich bin der Architekt. Ich kenne die Frameworks und die Architekturen, ich entscheide, was das System sein soll, und ich leite die Werkzeuge an, die heute den größten Teil des Codes schreiben -- in welcher Sprache auch immer für die Aufgabe richtig ist. Dann verifiziere ich jede Zeile, denn 35 Jahre in diesem Geschäft haben mich gelehrt, wie korrekt aussieht.

Dieser Beitrag ist die lange Fassung dessen, was die Startseite hätte sagen sollen. Sie bekommen eine öffentliche Auswahl dessen, was ich dieses Jahr gebaut habe und in welcher Sprache, das Argument, warum

"welche Sprache" die uninteressanteste Frage in meiner Arbeit geworden ist, die Belege für und gegen dieses Argument, und -- weil ich keine Brücken abbreche -- was all das bedeutet, wenn Sie ein Delphi-Entwickler sind, der auf denselben Kalender schaut.

## Was die Website sagte, und was sie hätte sagen sollen

Die alte Startseite hat nicht gelogen; sie war nur ein Jahr veraltet. Die Formulierung "30+ Jahre Delphi" war wahr, der Titel Embarcadero MVP war wahr, und die Delphi-zu-Web-Migrationen, die ich als Fallstudien aufgeführt habe, waren echt und laufen noch. Aber ein Besucher ohne jede Delphi-Vergangenheit liest eine solche Seite und schließt vernünftigerweise, dass der Delphi-Teil der Punkt ist und der Web-Teil ein Nebeneffekt. Für den größten Teil meiner Laufbahn wäre diese Lesart ungefähr richtig gewesen. 2026 ist sie es nicht.

Also habe ich die Seite umgeschrieben. Die Stacks, die ich dieses Jahr ausgeliefert habe, stehen jetzt dort, wo die Kennzahlen früher mit einer Sprache geprahlt haben. Die Services beginnen mit individuellen Systemen und Apps für jede Plattform, und die Delphi-Arbeit -- Migrationen, Beratung, Code-Rettung -- ist weiterhin da, als ein Service unter sieben. Warum sie bleibt, dazu komme ich noch. Zuerst ein paar Belege.

## Eine Auswahl dessen, was 2026 ausgeliefert wurde, nach Sprache

Statt Vielseitigkeit zu behaupten, zeige ich Ihnen eine Auswahl. Ich möchte klarstellen, dass es eine Auswahl ist und nicht das Hauptbuch: Der größte Teil meiner Arbeit ist Kundenarbeit und bekommt nie einen Blogbeitrag oder eine Fallstudie, und dahinter stehen mehr iOS- und Android-Apps als die unten genannten. Die Tabelle greift für jede Sprache ein oder zwei Dinge heraus, auf die ich öffentlich zeigen kann. Alles darin ist entweder auf dieser Website veröffentlicht oder in einer anonymisierten Fallstudie auf der Work-Seite beschrieben, mit einer ehrlichen Ausnahme, die ich kennzeichne.

| Sprache | Ein öffentliches Beispiel | Wo nachzulesen |
|---|---|---|
| TypeScript, Next.js, React, PostgreSQL | Diese Website, ein Lagerverwaltungssystem, ein B2B-Shop, eine Offline-First-Lieferapp, eine Delphi-7-Anwendung neu gebaut für den Browser | Fallstudien, die Delphi-zu-Next.js-Serie |
| Swift, SwiftUI | Hearth: Apps für iPhone, iPad, Apple TV und Mac auf einem Backend, 313 automatisierte Tests, 48 Releases in zwölf Tagen -- das eine iOS-Projekt, das ich vollständig zeigen kann | Bauen statt kaufen, Kein Vibe Coding |
| Kotlin | Android-Apps, gebaut auf bestehenden Webanwendungen, für Kunden | Noch kein Beitrag -- das ist die Ausnahme; ich schreibe einen |
| C# und .NET | Ein C#-Berichtsdienst gelesen, verstanden und in die Next.js-App eingefaltet, in die er gehörte; unterwegs zwei Rezensionen zu .NET-10-Büchern | Von einem C#-Berichtsdienst zu einer Next.js-Codebasis, C# 14 und .NET 10 |
| Delphi | FireDAC und PostgreSQL von null, WebStencils mit SQLite, Delphi 13.2 unter Linux, ein natives Tool an einem Nachmittag per Spezifikation gebaut | FireDAC-CLI, WebStencils und SQLite, Spec Coding mit Delphi |

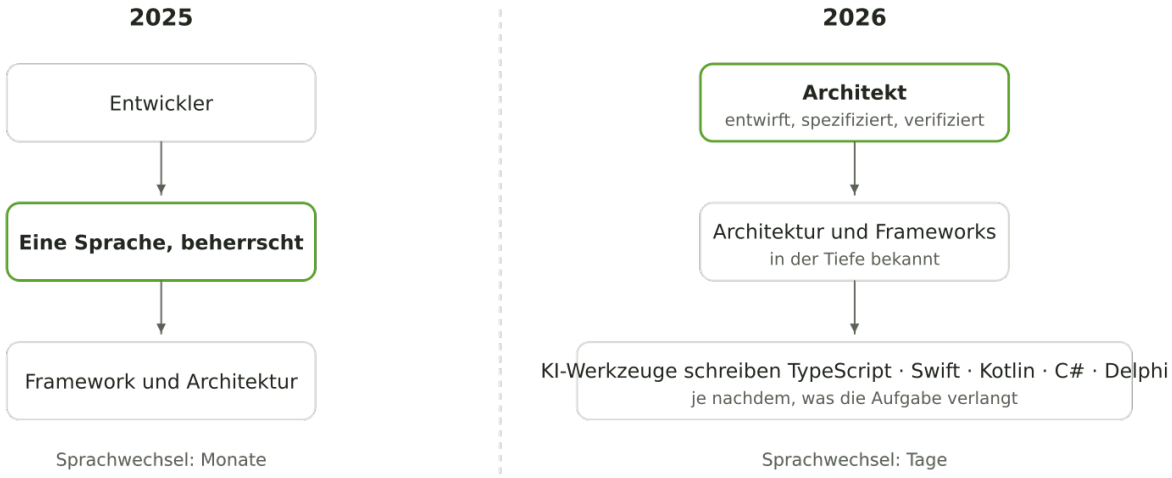
Lesen Sie die Tabelle noch einmal und achten Sie darauf, was fehlt: eine Zeile mit "sechs Monate Swift gelernt". Es gibt sie nicht. Die Swift-Apps existieren, weil ich wusste, wie ein iPad-Client für ein Next.js-Backend mit Passkey-Anmeldung aussehen sollte, das aufgeschrieben habe und die Werkzeuge so lange angeleitet habe, bis das Ergebnis passte -- und es dann auf dem echten Gerät getestet habe, wie der Beitrag Kein Vibe Coding ziemlich detailliert dokumentiert. Die Kotlin-Apps entstanden auf dieselbe Weise, auf einer Web-App, die bereits

jede Regel definiert hatte. Nach meiner Erfahrung ist eine funktionierende Webanwendung als Basis genau das, was eine native App zu einer Sache von Tagen statt Monaten macht; der Plattformcode ist absichtlich dünn, und alle Logik, die schiefgehen kann, lebt auf dem Server, wo sie bereits getestet war.

## Der Job hat sich geändert; die Expertise nicht

Hier kommt der Teil, den ich für wirklich neu halte, und ich möchte ihn sorgfältig beschreiben statt ihn hochzujubeln. Für den größten Teil meiner Laufbahn war die Einheit der Expertise die Sprache. Man war Delphi-Entwickler oder C#-Entwickler oder Java-Entwickler, und ein Wechsel bedeutete Monate, in denen man eine schlechtere Version seiner selbst war, bevor man wieder gut wurde. Die Frameworks und die Architektur zählten natürlich, aber man erreichte sie durch eine Syntax, die man vollständig beherrschen musste.

So laufen meine Tage nicht mehr ab. Das Diagramm unten zeigt, was ich meine, und es ist das ehrlichste Bild, das ich von 2025 gegenüber 2026 zeichnen kann.




2025: Ein Sprachspezialist erreicht die Architektur durch eine Syntax. 2026: Der Architekt entwirft das System und leitet die Werkzeuge in die Sprache, die passt.

Die linke Spalte ist, wie ich rund dreißig Jahre gearbeitet habe, und ich rede sie nicht schlecht; sie hat eine Menge guter Software hervorgebracht. Die rechte Spalte ist, wie ich dieses Jahr gearbeitet habe. Achten Sie darauf, welches Kästchen den dicken Rahmen hat. 2025 war die Sprache die unersetzliche Schicht. 2026 ist die unersetzliche Schicht die Person ganz oben, die weiß, was das System sein soll, die Frameworks und die Architektur gut genug kennt, um es präzise zu spezifizieren, und genug über all diese Sprachen weiß, um zu lesen, was zurückkommt, und abzulehnen, was falsch ist.

Dieser letzte Halbsatz ist heute der ganze Job. Die Werkzeuge können idiomatisches Kotlin schneller erzeugen, als ich es tippen kann. Was sie nicht können, ist zu entscheiden, dass die Trennung von Familiendaten in Datenbank-Trigger gehört und nicht in Anwendungscode, oder zu bemerken, dass ein Anmeldebildschirm auf dem iPad aussieht wie der Kopplungsbildschirm eines Fernsehers, oder zu erkennen, dass ein Bericht, der "richtig aussieht", eine Steuer falsch berechnet. Diese Entscheidungen kamen von mir, und der Beitrag Kein

Vibe Coding zählt sie einzeln auf. Ich muss Kotlin also nicht so lernen, wie ich 1995 Object Pascal gelernt habe,

mit einem dicken Buch und einem Jahr voller Fehler. Ich muss die Android-Architektur kennen, ich muss wissen, wie ein korrektes Kotlin-Programm aussieht, wenn ich eines lese, und ich brauche die Disziplin, jedes einzelne davon zu lesen. Die ersten beiden habe ich aus Jahrzehnten dieser Arbeit über Plattformen hinweg, und das dritte ist einfach Charakter.

## Das ist kein Vibe Coding, und hier endet diese Behauptung

Ich möchte ausdrücklich sagen, dass der Absatz oben meine Meinung ist, und dass mir bewusst ist, dass er klingt wie jeder "10x mit KI"-Beitrag, an dem Sie dieses Jahr vorbeigescrollt sind. Also lassen Sie mich ihn stützen und dann ordentlich herausfordern.

Die Stütze ist einfach: Der Prozess ist dokumentiert und die Zahlen sind öffentlich. Hearth durchlief 52 Pull Requests, jeder beschrieben, geprüft und getestet, mit 313 automatisierten Tests, 34 Datenbank-Isolationsprüfungen und einem Deploy-Gate, das wir absichtlich einmal gebrochen haben, um zu beweisen, dass es die Auslieferung verweigert. Simon Willison hat die Linie gezogen, die ich am nützlichsten finde: Vibe Coding ist "Software mit einem LLM bauen, ohne den Code zu prüfen, den es schreibt", und wenn man ihn prüft, gründlich testet und erklären kann, wie er funktioniert, dann ist es einfach Softwareentwicklung. Seine goldene Regel lautet, dass er keinen Code committet, von dem er nicht jemand anderem genau erklären könnte, was er tut. Ich halte mich an dieselbe Regel, und es ist die Regel, die 35 Jahre erst möglich machen: Man kann kein Kotlin prüfen, das man nicht lesen kann, und man kann Kotlin nicht gut lesen, wenn man nie gründlich darüber nachgedacht hat, wie eine Android-App strukturiert sein sollte.

Nun die Herausforderung, und ich nehme sie ernst.

> **Wo das nicht gilt**
>
> METR führte Anfang 2025 eine randomisierte Studie mit 16 erfahrenen Open-Source-Entwicklern an 246 echten Issues in ihren eigenen Repositories durch. Mit erlaubten KI-Werkzeugen brauchten sie 19 Prozent *länger* -- und glaubten danach trotzdem, sie seien 20 Prozent schneller gewesen. Die Autoren nennen die Einschränkungen selbst (kleine Stichprobe, Entwickler neu im konkreten Werkzeug, Ergebnisse bereits durch ihre eigene Folgestudie 2026 überholt), aber die Wahrnehmungslücke ist der Befund, der alle beunruhigen sollte, mich eingeschlossen. Das Gefühl von Geschwindigkeit ist kein Beweis für Geschwindigkeit. Die Stack Overflow Developer Survey 2025 fand 84 Prozent der Entwickler, die KI-Werkzeuge nutzen oder nutzen wollen, während 46 Prozent sagen, sie vertrauen der Ausgabe nicht, gegenüber 31 Prozent ein Jahr zuvor. Mehr Nutzung, weniger Vertrauen. Und der DORA-Report 2025 stellt fest, dass KI-Einsatz heute mit höherem Durchsatz *und* mit höherer Instabilität korreliert -- mehr fehlgeschlagene Änderungen, mehr Nacharbeit. Ihre Formulierung: KI "verstärkt, was bereits da ist".

Hier also die eingegrenzte Fassung meiner Behauptung, die einzige Fassung, die ich verteidige. KI-Werkzeuge über Sprachen hinweg anzuleiten funktioniert, wenn die Person, die sie anleitet, bereits das architektonische Urteilsvermögen und die Lesefähigkeit hat, um zu erkennen, was falsch ist, und wenn der Prozess dieses Lesen erzwingt: Pull Requests, Tests, Review, ein Deploy-Gate. Unter diesen Bedingungen habe ich dieses Jahr mehr und bessere Software ausgeliefert als in jedem Jahr davor. Ohne diese Bedingungen sagt die Forschung, dass Sie wahrscheinlich Software ausliefern, die sich schneller anfühlt und weniger stabil ist, und es nicht bemerken. DORAs "verstärkt, was bereits da ist" trifft es genau: Die Werkzeuge haben in meinem Fall 35 Jahre Erfahrung verstärkt. Deren Abwesenheit würden sie genauso treu verstärken.

> **Die andere Seite des Architektur-Arguments**
>
> Ein fairer Einwand: Wenn die nativen Apps "absichtlich dünn" sind und die Logik auf dem Server lebt, habe ich mir dann nicht einfach den leichten Fall ausgesucht? Teilweise, ja. Eine Kotlin-App, deren Aufgabe es ist, Daten darzustellen und mit einer sauber definierten API zu sprechen, ist ein weit kleineres Problem als eine Kotlin-App mit komplexem lokalem Zustand, Hintergrundverarbeitung und Offline-Konfliktauflösung. Ich würde trotzdem argumentieren, dass Logik an eine getestete Stelle zu schieben für die meisten Geschäftsanwendungen die richtige Architektur ist, und dass genau das die Art Entscheidung ist, die ein Architekt trifft, bevor irgendeine Sprache gewählt wird. Aber ich würde nicht behaupten, dass jede native App so günstig entstehen kann, und ein Projekt, das wirklich tiefe Plattformexpertise braucht, braucht weiterhin jemanden, der sie hat.

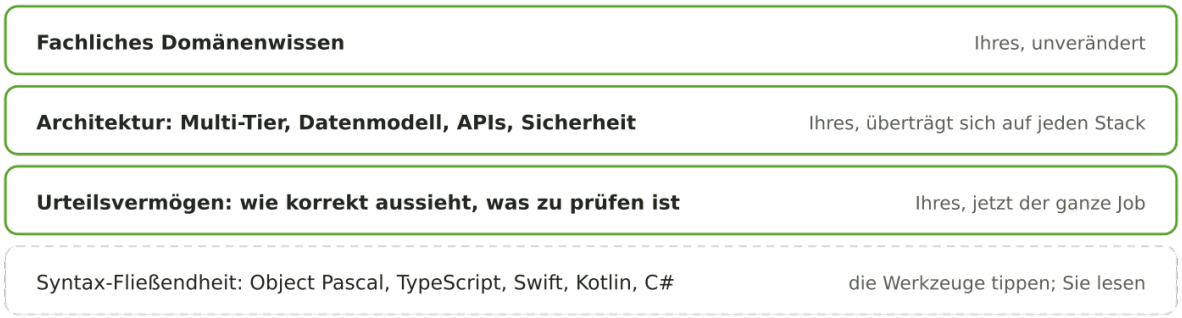
## Was das für die Delphi-Community bedeutet

Nun der versprochene Teil. Ich lasse Delphi nicht hinter mir, und ich möchte klar sagen, warum, denn die ehrlichen Zahlen sind dieses Jahr nicht freundlich, und ich werde nicht so tun, als wären sie es.

Der TIOBE-Index für Oktober 2026 führt Delphi/Object Pascal auf Platz 19 mit 0,83 Prozent, nach Platz 9 ein Jahr zuvor. Das ist ein steiler Abstieg in einer Rangliste, die zugegebenermaßen Suchinteresse misst und nicht Software in Produktion, und ich würde jeden davor warnen, einen einzelnen Index als Urteil zu lesen. Gleichzeitig hat Embarcadero im September 2025 RAD Studio 13 Florence und im März 2026 13.1 ausgeliefert, ich habe im September über Delphi 13.2 unter Linux geschrieben, und ich bekomme immer noch jede Woche Post von Menschen, die Delphi-Systeme pflegen, auf denen echte Unternehmen laufen. Diese Systeme verschwinden nicht so bald, und der Bedarf an Menschen, die sie lesen können, auch nicht.

Meine Meinung -- und es ist eine Meinung, wägen Sie sie also als solche -- ist, dass Delphi als *primäre Karrierefähigkeit* auf dem Weg nach draußen ist, in dem Sinne, dass weniger neue Systeme darin begonnen werden, während Delphi als *Bestand laufender Software* noch lange fachkundige Pflege braucht. Wenn Sie zwanzig Jahre in Delphi verbracht haben, sind das beides gute Nachrichten, und zwar deshalb.

Alles, was Sie wertvoll gemacht hat, ist weiterhin wertvoll: Sie verstehen Ihre Fachdomäne, Sie wissen, wie eine Multi-Tier-Anwendung aufgebaut ist, Sie haben FireDAC-Transaktionen um drei Uhr morgens debuggt, und Sie wissen, wie ein korrektes Programm aussieht, wenn Sie eines lesen. Geändert hat sich, dass die Sprache nicht mehr der Burggraben um dieses Wissen ist; sie ist jetzt der billigste Teil. Das bedeutet, dass das, was Sie in Delphi gehalten hat -- die Kosten, in TypeScript oder C# wieder Anfänger zu werden -- größtenteils verdampft ist. Sie müssen kein Anfänger werden. Sie müssen der Architekt werden, der Sie bereits sind, laut und deutlich, und die Werkzeuge tippen lassen.




Die Vermögenswerte eines Delphi-Entwicklers, und welche davon die Werkzeuge verändern. Die wertvollen Schichten bleiben Ihre; die Syntaxschicht ist das, was billig geworden ist.

Die drei durchgezogenen Kästchen sind das, was ein Delphi-Veteran in jedes Projekt mitbringt, und sie sind genau das, was ich dieses Jahr in die Swift- und Kotlin-Arbeit mitgebracht habe. Das gestrichelte Kästchen war früher die Barriere. Meine Delphi-zu-Next.js-Serie wurde für diesen Übergang geschrieben, und wenn Sie auf der Delphi-Seite davon stehen, ist das weiterhin das Erste, was ich Ihnen in die Hand geben würde. Der Ziel-Stack ist weniger wichtig, als Sie denken; die Gewohnheit, alles zu lesen, bevor es ausgeliefert wird, ist wichtiger als je zuvor.

> **Wenn Sie lieber bei Delphi bleiben**
> Das ist eine legitime Entscheidung, und nichts hier spricht für die Systeme, die Sie bereits betreiben, dagegen. Dieselben KI-Werkzeuge funktionieren in einem Delphi-Projekt; ich habe dieses Jahr ein natives Delphi-Tool an einem Nachmittag per Spezifikation gebaut und ein lokales Sprachmodell in eine Delphi-App

> eingebettet. Der Punkt ist nicht das Ziel. Der Punkt ist, dass die Sprache aufgehört hat, das zu sein, was Sie begrenzt -- in beide Richtungen.

## Was das bedeutet, wenn Sie jemanden suchen

Wenn Sie das Unternehmen sind, das vor ein paar Wochen weitergezogen ist, ist dieser Abschnitt für Sie, und ich halte ihn kurz. Bringen Sie mir das Problem, nicht die Sprache. Beschreiben Sie mir den Arbeitsablauf, der Software braucht, die Systeme, mit denen sie sprechen muss, die Menschen, die sie benutzen werden, und die Geräte, die sie in der Hand halten. Ich schlage eine Architektur vor und den Stack, der dazu passt -- Next.js und PostgreSQL, wo eine Browser-Anwendung richtig ist, C# und ASP.NET, wo Ihr Team bereits in .NET zu Hause ist, Swift oder Kotlin, wo es nativ sein muss, Delphi, wo ein funktionierendes Delphi-Backend genau da bleiben sollte, wo es ist. Dann baue ich es mit den Werkzeugen, lese jede Zeile, teste es und zeige Ihnen die Pull Requests.

Welche Sprache ich "mache", ist die uninteressanteste Frage über meine Arbeit geworden. Was ich mache, ist zu entscheiden, was das System sein soll, und sicherzustellen, dass es das ist. Nach meiner Erfahrung ist das auch der Teil, für den Kunden tatsächlich bezahlen.

## Das Wesentliche

Drei Quartale 2026 haben verändert, was mein Job ist, und die Startseite sagt es endlich. Die Kurzfassung ist diese.

- Ich habe dieses Jahr Produktionscode in TypeScript, Swift, Kotlin, C#, SQL und Delphi ausgeliefert. Die Tabelle oben ist eine öffentliche Auswahl davon, nicht die ganze Liste, und zu einer dieser Sprachen schulde ich Ihnen noch einen Beitrag.
- Der Job heißt jetzt Architektur, Spezifikation und Verifikation. Die Werkzeuge schreiben die Syntax; ich entscheide, was das System ist, und lese alles, was zurückkommt. Das funktioniert wegen 35 Jahren Wissen, wie korrekt aussieht -- und die Forschung sagt, dass es ohne dieses Wissen nicht funktionieren würde.
- Die Belege gegen Übertreibung sind real: erfahrene Entwickler in METRs Studie messbar langsamer, sinkendes Vertrauen in KI-Ausgaben und steigende Instabilität, wo der Prozess schwach ist. Meine Behauptung ist auf einen Prozess mit Tests, Reviews und Deploy-Gate eingegrenzt, und ich würde Ihre genauso eingrenzen.
- Delphi bleibt. Sein Ranking ist dieses Jahr hart gefallen, und Embarcadero liefert weiter; beides ist wahr. Für Delphi-Entwickler sind die wertvollen Schichten -- Domäne, Architektur, Urteilsvermögen -- intakt, und die Barriere, die Sie in einer Sprache gehalten hat, hat sich größtenteils aufgelöst.

> Die Sprache war nie die Expertise. Sie war der Weg hinein. 2026 ist der Weg hinein eine Spezifikation und ein sorgfältiger Leser, und ich bin das Zweite geworden.

Wenn Sie ein System zu bauen haben und keine besondere Bindung daran, wie es geschrieben wird, lassen Sie uns reden. Und wenn Sie ein Delphi-Entwickler sind, der auf denselben Kalender schaut wie ich, beginnen Sie mit der Serie, und schreiben Sie mir, wenn Sie nicht weiterkommen. Diese Brücke bleibt stehen.

INDEPENDENT · AD-FREE · READER-SUPPORTED

# Free to read. Not free to make.

Every tutorial, article, and line of example code on Holger's Code is published without a paywall and without ads. This page is about what keeps it that way.

## Professional work, published free

The content here is held to the same standard as professional client work: every tutorial is built as a real project, tested against current versions, documented step by step, and revisited when the frameworks move on. A single in-depth tutorial routinely takes days — and that's before the bills behind it: servers, storage, build infrastructure, licenses, and the hardware everything is verified on.

If an article or tutorial here has saved you an afternoon of debugging, taught you a technique, or helped you ship, it created real value for you. Supporting it isn't charity — it's paying for professional work you found useful, at whatever price you choose. And if you can't, keep reading anyway. That's what the free part means.

## Ways to support

KO-FI

### Buy me a coffee

A one-time tip — no account, no subscription, any amount.

BOOKS & COURSES

### Buy the paid work

The paid tier of everything published free here — and you get something lasting in return.

SPREAD THE WORD

### Share an article

Send a tutorial to a colleague, cite an article, subscribe to the RSS feed.

Nothing here goes behind a paywall, whether you chip in or not. But if something on this site earned its keep today, this is where to say so.

**Support on Ko-fi —**