NEWS · OCTOBER 23, 2026

Niet alleen Delphi: hoe drie kwartalen van 2026 mijn baan herschreven

Een potentiële klant verliet mijn site met de indruk "die man doet alleen Delphi" -- dus hier is hoe drie kwartalen van 2026 er werkelijk uitzagen, met TypeScript, Swift, Kotlin, C# en Delphi in productie, en waarom het werk nu architectuur en verificatie heet, niet vloeiendheid in één syntaxis.

By Dr. Holger Flick



Een paar weken geleden zocht een bedrijf iemand om een applicatie voor hen te bouwen. Iemand wees ze naar mijn site. Ze lazen die, en ze liepen weg met één indruk: "O, die man doet alleen Delphi. Wij gebruiken geen Delphi." Ik hoorde dat uit de tweede hand, en het was precies de feedback die ik niet wilde horen -- niet omdat het oneerlijk was, maar omdat mijn eigen voorpagina het had verdiend. Het woord "Delphi" stond in de hero, in de cijfers, in de footer en in de eerste dienst op de lijst.

Maar dit is het punt. In de negen maanden sinds januari heb ik productiecode opgeleverd in TypeScript, Swift, Kotlin, C#, SQL en, ja, Delphi. Niets daarvan is vergezocht, en niets daarvan is nieuw voor mij in de zin die ertoe doet. Wat nieuw is, is hoe het werk gedaan wordt. In 2025 zou ik mezelf hebben beschreven als een Delphi-expert die ook voor het web bouwt. In oktober 2026 is die beschrijving simpelweg onjuist. Ik ben de architect. Ik ken de frameworks en de architecturen, ik bepaal wat het systeem moet zijn, en ik stuur de tools aan die nu het grootste deel van de code schrijven -- in welke taal ook geschikt is voor de klus. Daarna verifieer ik elke regel, want 35 jaar in dit vak hebben mij geleerd hoe correct eruitziet.

Dit artikel is de lange versie van wat de voorpagina had moeten zeggen. U krijgt een publieke greep uit wat ik dit jaar heb gebouwd en in welke taal, het argument waarom "welke taal" de minst interessante vraag in mijn

werk is geworden, het bewijs voor en tegen dat argument, en -- omdat ik geen bruggen verbrand -- wat dit alles betekent als u een Delphi-ontwikkelaar bent die naar dezelfde kalender kijkt.

Wat de site zei, en wat hij had moeten zeggen

De oude voorpagina loog niet; hij was alleen een jaar verouderd. De zin "30+ jaar Delphi" was waar, de titel Embarcadero MVP was waar, en de Delphi-naar-web-migraties die ik als casestudy's noemde waren echt en draaien nog steeds. Maar een bezoeker zonder enige Delphi-geschiedenis leest zo'n pagina en concludeert redelijkerwijs dat het Delphi-deel de kern is en het webdeel een bijverschijnsel. Voor het grootste deel van mijn loopbaan zou die lezing ongeveer juist zijn geweest. In 2026 is dat niet zo.

Dus heb ik de pagina herschreven. De stacks die ik dit jaar heb opgeleverd staan nu waar de cijfers vroeger met één taal pronkten. De diensten beginnen met maatwerksystemen en apps voor elk platform, en het Delphi-werk -- migraties, advies, code-redding -- is er nog steeds, als één dienst van zeven. Waarom het blijft, daar kom ik op terug. Eerst wat bewijzen.

Een greep uit wat in 2026 is opgeleverd, per taal

In plaats van veelzijdigheid te claimen, laat ik u een greep zien. Ik wil duidelijk zijn dat dit een greep is en niet het grootboek: het grootste deel van mijn werk is voor klanten en krijgt nooit een blogartikel of casestudy, en er zitten meer iOS- en Android-apps achter dan de hieronder genoemde. Wat de tabel doet, is per taal één of twee dingen kiezen die ik publiek kan aanwijzen. Alles erin is ofwel gepubliceerd op deze site, ofwel beschreven in een geanonimiseerde casestudy op de werkpagina, met één eerlijke uitzondering die ik markeer.

Taal	Een publiek voorbeeld	Waar te vinden
TypeScript, Next.js, React, PostgreSQL	Deze website, een magazijnbeheersysteem, een B2B-webshop, een offline-first bezorgapp, een Delphi 7-applicatie herbouwd voor de browser	Casestudy's, de Delphi-naar-Next.js-serie
Swift, SwiftUI	Hearth: apps voor iPhone, iPad, Apple TV en Mac op één backend, 313 geautomatiseerde tests, 48 releases in twaalf dagen -- het ene iOS-project dat ik volledig kan laten zien	Bouwen, niet kopen, Geen vibe coding
Kotlin	Android-apps gebouwd op bestaande webapplicaties, voor klanten	Nog geen artikel -- dit is de uitzondering; ik schrijf er een
C# en .NET	Een C#-rapportservice gelezen, begrepen en opgevouwen in de Next.js-app waar hij thuishoorde; onderweg twee recensies van .NET 10-boeken	Van een C#-rapportservice naar één Next.js-codebase, C# 14 en .NET 10
Delphi	FireDAC en PostgreSQL vanaf nul, WebStencils met SQLite, Delphi 13.2 op Linux, een native tool op één middag gebouwd vanuit een specificatie	FireDAC-CLI, WebStencils en SQLite- , Spec coding met Delphi

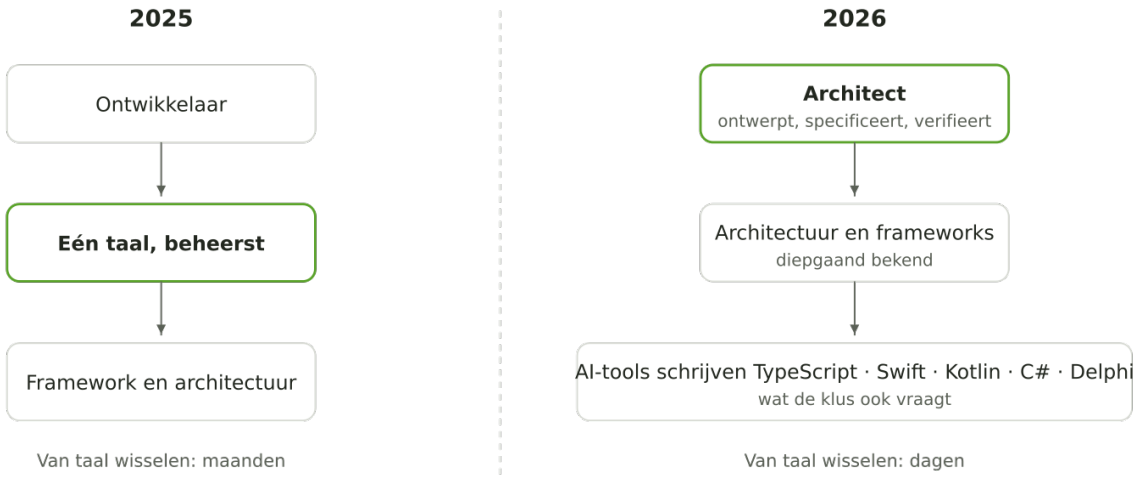
Lees die tabel nog eens en let op wat ontbreekt: een regel die zegt "zes maanden Swift geleerd". Die is er niet. De Swift-apps bestaan omdat ik wist hoe een iPad-client voor een Next.js-backend met passkey-aanmelding eruit moest zien, dat opschreef en de tools aanstuurde tot het resultaat klopte -- en het daarna op het echte apparaat testte, zoals het artikel Geen vibe coding tamelijk gedetailleerd documenteert. De Kotlin-apps ontstonden op

dezelfde manier, bovenop een webapp die al elke regel had vastgelegd. In mijn ervaring is een werkende webapplicatie als basis precies wat een native app tot een kwestie van dagen in plaats van maanden maakt; de platformcode is bewust dun, en alle logica die mis kan gaan leeft op de server, waar ze al getest was.

Het werk veranderde; de expertise niet

Hier komt het deel dat ik werkelijk nieuw vind, en ik wil het zorgvuldig beschrijven in plaats van ophemelen. Voor het grootste deel van mijn loopbaan was de eenheid van expertise de taal. Je was Delphi-ontwikkelaar of C#-ontwikkelaar of Java-ontwikkelaar, en overstappen betekende maanden lang een slechtere versie van jezelf zijn voordat je weer goed werd. De frameworks en de architectuur deden er natuurlijk toe, maar je bereikte ze via een syntaxis die je volledig moest beheersen.

Zo verlopen mijn dagen niet meer. Het diagram hieronder laat zien wat ik bedoel, en het is het eerlijkste beeld dat ik van 2025 tegenover 2026 kan tekenen.



2025: een taalspecialist bereikt de architectuur via één syntaxis. 2026: de architect ontwerpt het systeem en stuurt de tools naar de taal die past.

De linkerkolom is hoe ik ruwweg dertig jaar heb gewerkt, en ik praat het niet af; het heeft veel goede software opgeleverd. De rechterkolom is hoe ik dit jaar heb gewerkt. Let op welk vakje de dikke rand heeft. In 2025 was de taal de onvervangbare laag. In 2026 is de onvervangbare laag de persoon bovenaan die weet wat het systeem moet zijn, de frameworks en de architectuur goed genoeg kent om het precies te specificeren, en genoeg van al die talen weet om te lezen wat terugkomt en te verwerpen wat fout is.

Die laatste bijzin is nu het hele werk. De tools kunnen idiomatisch Kotlin sneller produceren dan ik het kan typen. Wat ze niet kunnen, is besluiten dat de scheiding tussen gezinnen in databasetriggers thuishoort en niet in applicatiecode, of opmerken dat een aanmeldscherm op de iPad eruitziet als het koppelscherm van een televisie, of herkennen dat een rapport dat "er goed uitziet" een belasting verkeerd berekent. Die beslissingen kwamen van mij, en het artikel Geen vibe coding somt ze één voor één op. Ik hoef Kotlin dus niet te leren zoals

ik in 1995 Object Pascal leerde, met een dik boek en een jaar aan fouten. Ik moet de Android-architectuur kennen, ik moet weten hoe een correct Kotlin-programma eruitziet als ik er een lees, en ik heb de discipline nodig om ze allemaal te lezen. De eerste twee heb ik uit decennia van dit werk over platformen heen, en het derde is gewoon karakter.

Dit is geen vibe coding, en hier stopt die bewering

Ik wil uitdrukkelijk zeggen dat de alinea hierboven mijn mening is, en dat ik me ervan bewust ben dat hij klinkt als elk "10x met AI"-artikel waar u dit jaar voorbij hebt gescrold. Laat me hem dus onderbouwen en daarna behoorlijk uitdagen.

De onderbouwing is eenvoudig: het proces is gedocumenteerd en de cijfers zijn openbaar. Hearth ging door 52 pull requests, elk beschreven, beoordeeld en getest, met 313 geautomatiseerde tests, 34 database-isolatiecontroles en een deploy-gate die we één keer bewust hebben gebroken om te bewijzen dat hij weigert op te leveren. Simon Willison trok de lijn die ik het nuttigst vind: vibe coding is "software bouwen met een LLM zonder de code te beoordelen die het schrijft", en als u die beoordeelt, grondig test en kunt uitleggen hoe hij werkt, dan is het gewoon softwareontwikkeling. Zijn gouden regel is dat hij geen code commit waarvan hij niet precies aan iemand anders kan uitleggen wat hij doet. Ik houd mezelf aan dezelfde regel, en het is de regel die 35 jaar mogelijk maken: u kunt geen Kotlin beoordelen dat u niet kunt lezen, en u kunt Kotlin niet goed lezen als u nooit grondig hebt nagedacht over hoe een Android-app gestructureerd hoort te zijn.

Nu de uitdaging, en ik neem die serieus.

> **Waar dit niet geldt**
>
> METR voerde begin 2025 een gerandomiseerd onderzoek uit met 16 ervaren open-source-ontwikkelaars op 246 echte issues in hun eigen repositories. Met AI-tools toegestaan deden ze er 19 procent *langer* over -- en geloofden achteraf nog steeds dat ze 20 procent sneller waren geweest. De auteurs noemen de kanttekeningen zelf (kleine steekproef, ontwikkelaars nieuw met de specifieke tool, resultaten al achterhaald door hun eigen vervolgonderzoek uit 2026), maar de perceptiekloof is de bevinding die iedereen zorgen zou moeten baren, mij inbegrepen. Het gevoel van snelheid is geen bewijs van snelheid. De Stack Overflow Developer Survey 2025 vond dat 84 procent van de ontwikkelaars AI-tools gebruikt of wil gebruiken, terwijl 46 procent zegt de uitvoer niet te vertrouwen, tegenover 31 procent een jaar eerder. Meer gebruik, minder vertrouwen. En het DORA-rapport 2025 stelt vast dat AI-adoptie nu correleert met hogere doorvoer *en* met hogere instabiliteit -- meer mislukte wijzigingen, meer herstelwerk. Hun formulering: AI "versterkt wat er al is".

Dus hier is de afgebakende versie van mijn bewering, de enige versie die ik verdedig. AI-tools over talen heen aansturen werkt wanneer de persoon die ze aanstuurt al het architectonische oordeel en het leesvermogen heeft om te vangen wat fout is, en wanneer het proces dat lezen afdwingt: pull requests, tests, review, een deploy-gate. Onder die voorwaarden heb ik dit jaar meer en betere software opgeleverd dan in elk jaar ervoor. Zonder die voorwaarden zegt het onderzoek dat u waarschijnlijk software oplevert die sneller voelt en minder stabiel is, zonder het te merken. DORA's "versterkt wat er al is" is precies juist: de tools versterkten in mijn geval 35 jaar ervaring. De afwezigheid daarvan zouden ze even trouw versterken.

> **De andere kant van het architectuurargument**
>
> Een eerlijk bezwaar: als de native apps "bewust dun" zijn en de logica op de server leeft, heb ik dan niet gewoon het makkelijke geval gekozen? Deels, ja. Een Kotlin-app die data moet weergeven en met een goed gedefinieerde API moet praten is een veel kleiner probleem dan een Kotlin-app met complexe lokale toestand, achtergrondverwerking en offline-conflictoplossing. Ik zou nog steeds betogen dat logica naar één geteste plek duwen de juiste architectuur is voor de meeste bedrijfsapplicaties, en dat dit precies het soort beslissing is dat een architect neemt voordat er een taal wordt gekozen. Maar ik zou niet beweren dat elke native app zo goedkoop te maken is, en een project dat werkelijk diepe platformexpertise nodig heeft, heeft nog steeds iemand nodig die die heeft.

Wat dit betekent voor de Delphi-gemeenschap

Nu het deel dat ik beloofde. Ik laat Delphi niet achter me, en ik wil duidelijk zeggen waarom, want de eerlijke cijfers zijn dit jaar niet vriendelijk en ik ga niet doen alsof dat anders is.

De TIOBE-index voor oktober 2026 zet Delphi/Object Pascal op plaats 19 met 0,83 procent, tegenover plaats 9 een jaar eerder. Dat is een steile val in een ranglijst die, toegegeven, zoekinteresse meet en geen software in productie, en ik zou iedereen waarschuwen om één index niet als vonnis te lezen. Tegelijkertijd leverde Embarcadero in september 2025 RAD Studio 13 Florence en in maart 2026 13.1, schreef ik in september over Delphi 13.2 op Linux, en krijg ik nog elke week post van mensen die Delphi-systemen onderhouden waarop echte bedrijven draaien. Die systemen verdwijnen niet snel, en de behoefte aan mensen die ze kunnen lezen ook niet.

Mijn mening -- en het is een mening, weeg haar dus als zodanig -- is dat Delphi als *primaire carrièrevaardigheid* op zijn retour is, in de zin dat er minder nieuwe systemen in worden gestart, terwijl Delphi als *bestand aan draaiende software* nog lang deskundige zorg nodig heeft. Als u twintig jaar in Delphi hebt doorgebracht, is dat allebei goed nieuws, en wel hierom.

Alles wat u waardevol maakte is nog steeds waardevol: u begrijpt uw vakdomein, u weet hoe een multi-tier-applicatie is opgebouwd, u hebt om drie uur 's nachts FireDAC-transacties gedebugd, en u weet hoe een correct programma eruitziet als u er een leest. Wat veranderd is, is dat de taal niet langer de slotgracht om die kennis is; ze is nu het goedkoopste deel. Dat betekent dat wat u in Delphi hield -- de kosten om in TypeScript of C# opnieuw beginner te worden -- grotendeels is verdampt. U hoeft geen beginner te worden. U moet de architect worden die u al bent, hardop, en de tools laten typen.



De activa van een Delphi-ontwikkelaar, en welke daarvan de tools veranderen. De waardevolle lagen blijven van u; de syntaxislaag is wat goedkoop is geworden.

De drie doorgetrokken vakjes zijn wat een Delphi-veteraan in elk project meebrengt, en het is precies wat ik dit jaar in het Swift- en Kotlin-werk meebracht. Het gestreepte vakje was vroeger de barrière. Mijn Delphi-naar-Next.js-serie is voor deze overgang geschreven, en als u aan de Delphi-kant ervan staat, is dat nog steeds het eerste wat ik u in handen zou geven. De doelstack doet er minder toe dan u denkt; de gewoonte om alles te lezen voordat het wordt opgeleverd doet er meer toe dan ooit.

> **Als u liever bij Delphi blijft**
> Dat is een legitieme keuze, en niets hier pleit ertegen voor de systemen die u al draait. Dezelfde AI-tooling werkt binnen een Delphi-project; ik heb dit jaar een native Delphi-tool op één middag vanuit een specificatie gebouwd en een lokaal taalmodel in een Delphi-app ingebed. Het punt is niet de bestemming. Het punt is dat de taal is opgehouden het ding te zijn dat u beperkt, in beide richtingen.

Wat dit betekent als u iemand zoekt

Als u het bedrijf bent dat een paar weken geleden wegliep, is deze sectie voor u, en ik houd hem kort. Breng mij het probleem, niet de taal. Vertel mij de workflow die software nodig heeft, de systemen waarmee ze moet praten, de mensen die haar gaan gebruiken en de apparaten die ze in handen hebben. Ik stel een architectuur voor en de stack die daarbij past -- Next.js en PostgreSQL waar een browserapplicatie juist is, C# en ASP.NET waar uw team al in .NET leeft, Swift of Kotlin waar het native moet zijn, Delphi waar een werkende Delphi-backend precies moet blijven waar hij is. Daarna bouw ik het met de tools, lees elke regel, test het en laat u de pull requests zien.

Welke taal ik "doe" is de minst interessante vraag over mijn werk geworden. Wat ik doe, is bepalen wat het systeem moet zijn en ervoor zorgen dat het dat is. In mijn ervaring is dat ook het deel waarvoor klanten werkelijk betalen.

De kern

Drie kwartalen van 2026 hebben veranderd wat mijn werk is, en de voorpagina zegt het eindelijk. De korte versie is deze.

- Ik heb dit jaar productiecode opgeleverd in TypeScript, Swift, Kotlin, C#, SQL en Delphi. De tabel hierboven is een publieke greep daaruit, niet de volledige lijst, en over één van die talen ben ik u nog een artikel schuldig.
- Het werk is nu architectuur, specificatie en verificatie. De tools schrijven de syntaxis; ik bepaal wat het systeem is en lees alles wat terugkomt. Dat werkt dankzij 35 jaar weten hoe correct eruitziet -- en het onderzoek zegt dat het zonder dat niet zou werken.
- Het bewijs tegen overdrijving is echt: ervaren ontwikkelaars meetbaar langzamer in METR's onderzoek, dalend vertrouwen in AI-uitvoer, en stijgende instabiliteit waar het proces zwak is. Mijn bewering is afgebakend tot een proces met tests, reviews en een deploy-gate, en ik zou de uwe net zo afbakenen.
- Delphi blijft. De ranking viel dit jaar hard en Embarcadero blijft leveren; beide zijn waar. Voor Delphi-ontwikkelaars zijn de waardevolle lagen -- domein, architectuur, oordeel -- intact, en de barrière die u in één taal hield is grotendeels opgelost.

> De taal was nooit de expertise. Ze was de weg naar binnen. In 2026 is de weg naar binnen een specificatie en een zorgvuldige lezer, en ik ben het tweede geworden.

Hebt u een systeem te bouwen en geen bijzondere gehechtheid aan hoe het geschreven wordt, laten we praten.

En bent u een Delphi-ontwikkelaar die naar dezelfde kalender kijkt als ik, begin dan met de serie, en schrijf me als u vastloopt. Die brug blijft staan.

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 —