NEWS · OCTOBER 23, 2026

Não Só Delphi: Como Três Trimestres de 2026 Reescreveram Meu Trabalho

Um cliente em potencial saiu do meu site com a impressão de que "esse cara só faz Delphi" -- então aqui está como três trimestres de 2026 realmente foram, com TypeScript, Swift, Kotlin, C# e Delphi em produção, e por que o trabalho agora se chama arquitetura e verificação, e não fluência em uma única sintaxe.

By Dr. Holger Flick

Há algumas semanas, uma empresa procurava alguém para construir um aplicativo. Alguém indicou o meu site. Eles leram e foram embora com uma única impressão: "Ah, esse cara só faz Delphi. Nós não usamos Delphi." Ouvi isso de segunda mão, e foi exatamente o retorno que eu não queria ouvir -- não porque fosse injusto, mas porque a minha própria página inicial o tinha merecido. A palavra "Delphi" aparecia no destaque, nas estatísticas, no rodapé e no primeiro serviço da lista.

Mas a questão é esta. Nos nove meses desde janeiro, entreguei código de produção em TypeScript, Swift, Kotlin, C#, SQL e, sim, Delphi. Nada disso é exagero, e nada disso é novo para mim no sentido que importa. O que é novo é a forma como o trabalho é feito. Em 2025, eu teria me descrito como um especialista em Delphi que também constrói para a web. Em outubro de 2026, essa descrição está simplesmente errada. Eu sou o arquiteto. Conheço os frameworks e as arquiteturas, decido o que o sistema deve ser e dirijo as ferramentas que hoje escrevem a maior parte do código -- em qualquer linguagem que seja a certa para a tarefa. Depois verifico cada linha, porque 35 anos neste ramo me ensinaram como o correto se parece.

Este artigo é a versão longa do que a página inicial deveria ter dito. Você vai receber uma amostra pública do que construí este ano e em qual linguagem, o argumento de por que "qual linguagem" se tornou a pergunta menos interessante do meu trabalho, as evidências a favor e contra esse argumento e -- porque não estou

queimando pontes -- o que tudo isso significa se você é um desenvolvedor Delphi olhando para o mesmo calendário.

O que o site dizia, e o que deveria ter dito

A antiga página inicial não mentia; estava apenas um ano desatualizada. A frase "30+ anos de Delphi" era verdadeira, o título de Embarcadero MVP era verdadeiro, e as migrações de Delphi para a web que listei como estudos de caso eram reais e continuam rodando. Mas um visitante sem nenhuma história com Delphi lê uma página assim e conclui, com razão, que a parte do Delphi é o ponto central e a parte da web é um efeito colateral. Durante a maior parte da minha carreira, essa leitura teria sido mais ou menos correta. Em 2026, não é.

Então reescrevi a página. As stacks que entreguei este ano agora aparecem onde as estatísticas antes se gabavam de uma única linguagem. Os serviços começam com sistemas sob medida e aplicativos para todas as plataformas, e o trabalho com Delphi -- migrações, consultoria, resgate de código -- continua lá, como um serviço entre sete. Volto a explicar por que ele fica. Primeiro, algumas provas.

Uma amostra do que foi entregue em 2026, por linguagem

Em vez de alegar versatilidade, deixe-me mostrar uma amostra. Quero deixar claro que é uma amostra, e não o livro-razão: a maior parte do meu trabalho é para clientes e nunca vira artigo ou estudo de caso, e há mais aplicativos iOS e Android por trás dela do que os citados abaixo. O que a tabela faz é escolher, para cada linguagem, uma ou duas coisas que posso apontar publicamente. Tudo nela está publicado neste site ou descrito em um estudo de caso anonimizado na página de trabalhos, com uma exceção honesta que vou sinalizar.

Linguagem	Um exemplo público	Onde ver
TypeScript, Next.js, React, PostgreSQL	Este site, um sistema de gestão de armazém, uma loja B2B, um aplicativo de entregas offline-first, um aplicativo Delphi 7 reconstruído para o navegador	Estudos de caso, a série Delphi para Next.js
Swift, SwiftUI	Hearth: aplicativos para iPhone, iPad, Apple TV e Mac em um único backend, 313 testes automatizados, 48 releases em doze dias -- o único projeto iOS que posso mostrar por inteiro	Construa, não compre, Não é vibe coding
Kotlin	Aplicativos Android construídos sobre aplicativos web existentes, para clientes	Ainda sem artigo -- esta é a exceção; vou escrever um
C# e .NET	Um serviço de relatórios em C# lido, entendido e incorporado ao aplicativo Next.js ao qual pertencia; duas resenhas de livros sobre .NET 10 pelo caminho	De um serviço de relatórios em C# para uma única base de código Next.js, C# 14 e .NET 10
Delphi	FireDAC e PostgreSQL do zero, WebStencils com SQLite, Delphi 13.2 no Linux, uma ferramenta nativa construída em uma tarde a partir de uma especificação	CLI com FireDAC, WebStencils e SQLite, Spec coding com Delphi

Leia a tabela mais uma vez e repare no que está faltando: uma linha dizendo "passei seis meses aprendendo Swift". Não existe. Os aplicativos Swift existem porque eu sabia como um cliente iPad de um backend Next.js com login por passkey deveria ser, escrevi isso no papel e dirigi as ferramentas até o resultado bater -- e então

testei no aparelho de verdade, como o artigo Não é vibe coding documenta com algum detalhe. Os aplicativos

Kotlin surgiram da mesma forma, sobre um aplicativo web que já definia cada regra. Na minha experiência, ter um aplicativo web funcionando como base é o que transforma um aplicativo nativo em questão de dias em vez de meses; o código específico da plataforma é fino por projeto, e toda a lógica que pode dar errado vive no servidor, onde já foi testada.

O trabalho mudou; a competência, não

Aqui está a parte que considero realmente nova, e quero descrevê-la com cuidado em vez de inflá-la. Durante a maior parte da minha carreira, a unidade de competência era a linguagem. Você era desenvolvedor Delphi, ou desenvolvedor C#, ou desenvolvedor Java, e trocar significava meses sendo uma versão pior de si mesmo antes de ficar bom de novo. Os frameworks e a arquitetura importavam, claro, mas você os acessava por meio de uma sintaxe que precisava dominar por completo.

Não é mais assim que os meus dias funcionam. O diagrama abaixo mostra o que quero dizer, e é o retrato mais honesto que consigo desenhar de 2025 contra 2026.

2025: um especialista em uma linguagem chega à arquitetura por meio de uma única sintaxe. 2026: o arquiteto projeta o sistema e dirige as ferramentas para a linguagem que se encaixa.

A coluna da esquerda é como trabalhei por cerca de trinta anos, e não a desmereço; ela produziu muito software bom. A coluna da direita é como trabalhei este ano. Repare em qual caixa tem a borda grossa. Em 2025, a linguagem era a camada insubstituível. Em 2026, a camada insubstituível é a pessoa no topo que sabe o que o sistema deve ser, conhece os frameworks e a arquitetura bem o suficiente para especificá-lo com precisão e sabe o suficiente sobre todas essas linguagens para ler o que volta e rejeitar o que está errado.

Essa última oração é todo o trabalho agora. As ferramentas conseguem produzir Kotlin idiomático mais rápido do que eu consigo digitar. O que elas não conseguem é decidir que o isolamento entre famílias pertence a triggers de banco de dados e não ao código do aplicativo, ou notar que uma tela de login no iPad parece a tela de emparelhamento de uma televisão, ou reconhecer que um relatório que "parece certo" está calculando um imposto do jeito errado. Essas decisões vieram de mim, e o artigo Não é vibe coding as lista uma a uma.

Portanto, não preciso aprender Kotlin do jeito que aprendi Object Pascal em 1995, com um livro grosso e um ano de erros. Preciso conhecer a arquitetura Android, preciso saber como um programa Kotlin correto se parece quando leio um, e preciso da disciplina de ler cada um deles. Os dois primeiros eu tenho de décadas fazendo isso em várias plataformas, e o terceiro é só caráter.

Isto não é vibe coding, e aqui é onde essa afirmação para

Quero deixar explícito que o parágrafo acima é a minha opinião, e que tenho consciência de que ele soa como todo artigo "10x com IA" que você rolou a tela para passar este ano. Então deixe-me sustentá-lo e depois desafiá-lo de forma adequada.

A sustentação é simples: o processo está documentado e os números são públicos. O Hearth passou por 52 pull requests, cada um descrito, revisado e testado, com 313 testes automatizados, 34 verificações de isolamento no banco de dados e um portão de deploy que quebramos de propósito uma vez para provar que ele se recusa a entregar. Simon Willison traçou a linha que considero mais útil: vibe coding é "construir software com um LLM sem revisar o código que ele escreve", e se você o revisa, testa a fundo e consegue explicar como funciona, isso é simplesmente desenvolvimento de software. A regra de ouro dele é não fazer commit de código que não conseguiria explicar exatamente a outra pessoa o que faz. Eu me mantenho na mesma regra, e é a regra que 35 anos tornam possível: você não consegue revisar Kotlin que não consegue ler, e não consegue ler Kotlin bem se nunca pensou a sério sobre como um aplicativo Android deve ser estruturado.

Agora o desafio, e eu o levo a sério.

> **Onde isso não se sustenta**
>
> A METR conduziu no início de 2025 um estudo randomizado com 16 desenvolvedores open source experientes em 246 issues reais de seus próprios repositórios. Com ferramentas de IA permitidas, eles levaram 19 por cento *mais* tempo -- e ainda assim acreditavam depois ter sido 20 por cento mais rápidos. Os próprios autores listam as ressalvas (amostra pequena, desenvolvedores novos na ferramenta específica, resultados já superados pelo acompanhamento deles em 2026), mas a lacuna de percepção é o achado que deveria preocupar todo mundo, inclusive eu. A sensação de velocidade não é prova de velocidade. A Pesquisa de Desenvolvedores do Stack Overflow de 2025 encontrou 84 por cento dos desenvolvedores usando ou planejando usar ferramentas de IA, enquanto 46 por cento dizem não confiar no resultado, contra 31 por cento um ano antes. Mais uso, menos confiança. E o relatório DORA 2025 constata que a adoção de IA agora se correlaciona com maior vazão *e* com maior instabilidade -- mais falhas em mudanças, mais retrabalho. A frase deles é que a IA "amplifica o que já está lá".

Então aqui está a versão delimitada da minha afirmação, a única que vou defender. Dirigir ferramentas de IA entre linguagens funciona quando a pessoa que as dirige já tem o julgamento arquitetural e a capacidade de leitura para pegar o que está errado, e quando o processo força essa leitura a acontecer: pull requests, testes, revisão, um portão de deploy. Sob essas condições, entreguei este ano mais software, e melhor, do que em qualquer ano anterior. Sem essas condições, a pesquisa diz que você provavelmente vai entregar software que parece mais rápido e é menos estável, e não vai perceber. O "amplifica o que já está lá" do DORA está exatamente certo: no meu caso, as ferramentas amplificaram 35 anos de experiência. Elas amplificariam a ausência disso com a mesma fidelidade.

> **O outro lado do argumento da arquitetura**
>
> Uma objeção justa: se os aplicativos nativos são "finos por projeto" e a lógica vive no servidor, eu não escolhi simplesmente o caso fácil? Em parte, sim. Um aplicativo Kotlin cujo trabalho é exibir dados e falar com uma API bem definida é um problema muito menor do que um aplicativo Kotlin com estado local complexo, processamento em segundo plano e resolução de conflitos offline. Ainda assim eu argumentaria que empurrar a lógica para um único lugar testado é a arquitetura certa para a maioria dos aplicativos de negócio, e que essa é precisamente a decisão que um arquiteto toma antes de qualquer linguagem ser escolhida. Mas eu não afirmaria que todo aplicativo nativo pode ser produzido a esse custo, e um projeto que realmente precisa de conhecimento profundo da plataforma continua precisando de alguém que o tenha.

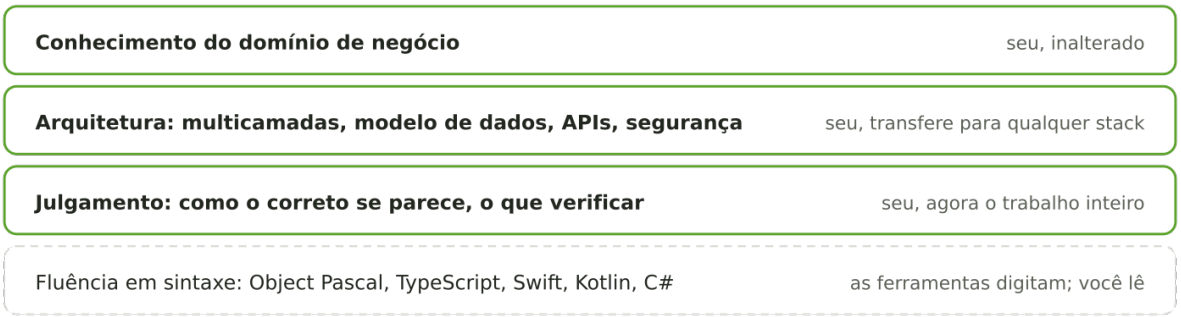
O que isso significa para a comunidade Delphi

Agora a parte que prometi. Não estou deixando o Delphi para trás, e quero dizer claramente por quê, porque os números honestos não são gentis este ano e não vou fingir o contrário.

O índice TIOBE de outubro de 2026 coloca Delphi/Object Pascal em 19º lugar com 0,83 por cento, contra o 9º lugar um ano antes. É uma queda acentuada em um ranking que, reconhecidamente, mede interesse de busca e não software em produção, e eu alertaria qualquer pessoa contra ler um único índice como veredito. Ao mesmo tempo, a Embarcadero lançou o RAD Studio 13 Florence em setembro de 2025 e o 13.1 em março de 2026, escrevi em setembro sobre o Delphi 13.2 no Linux, e ainda recebo e-mails toda semana de pessoas que mantêm sistemas Delphi que fazem empresas reais funcionarem. Esses sistemas não vão desaparecer tão cedo, nem a necessidade de pessoas capazes de lê-los.

Minha opinião -- e é uma opinião, então pese-a como tal -- é que o Delphi como *habilidade principal de carreira* está de saída, no sentido de que menos sistemas novos serão iniciados nele, enquanto o Delphi como *conjunto de software em funcionamento* vai precisar de cuidado especializado por muito tempo. Se você passou vinte anos no Delphi, ambas são boas notícias, e eis o porquê.

Tudo o que fazia de você alguém valioso continua valioso: você entende o seu domínio de negócio, sabe como um aplicativo multicamadas é estruturado, já depurou transações do FireDAC às três da manhã e sabe como um programa correto se parece quando lê um. O que mudou é que a linguagem não é mais o fosso em volta desse conhecimento; agora é a parte mais barata. Isso significa que o que mantinha você no Delphi -- o custo de voltar a ser iniciante em TypeScript ou C# -- praticamente evaporou. Você não precisa voltar a ser iniciante. Você precisa se tornar o arquiteto que já é, em voz alta, e deixar as ferramentas digitarem.



Os ativos de um desenvolvedor Delphi, e quais deles as ferramentas mudam. As camadas valiosas continuam suas; a camada de sintaxe é o que ficou barato.

As três caixas de linha contínua são o que um veterano de Delphi traz para qualquer projeto, e são exatamente o que trouxe para o trabalho com Swift e Kotlin este ano. A caixa tracejada era a barreira. Minha série Delphi para Next.js foi escrita para essa transição, e se você está do lado Delphi dela, continua sendo a primeira coisa que eu colocaria nas suas mãos. A stack de destino importa menos do que você pensa; o hábito de ler tudo antes de entregar importa mais do que nunca.

> **Se você prefere ficar no Delphi**
> É uma escolha legítima, e nada aqui argumenta contra ela para os sistemas que você já opera. As mesmas ferramentas de IA funcionam dentro de um projeto Delphi; este ano eu construí uma ferramenta Delphi nativa em uma tarde a partir de uma especificação e embuti um modelo de linguagem local em um aplicativo Delphi. A questão não é o destino. A questão é que a linguagem deixou de ser o que limita você, em qualquer direção.

O que isso significa se você está contratando

Se você é a empresa que foi embora há algumas semanas, esta seção é para você, e vou ser breve. Traga-me o problema, não a linguagem. Conte-me o fluxo de trabalho que precisa de software, os sistemas com os quais ele precisa conversar, as pessoas que vão usá-lo e os aparelhos que elas têm nas mãos. Vou propor uma arquitetura e a stack que se encaixa nela -- Next.js e PostgreSQL onde um aplicativo para navegador é o certo, C# e ASP.NET onde a sua equipe já vive em .NET, Swift ou Kotlin onde precisa ser nativo, Delphi onde um backend Delphi funcional deve ficar exatamente onde está. Depois vou construí-lo com as ferramentas, ler cada linha, testá-lo e mostrar os pull requests a você.

Qual linguagem eu "faço" se tornou a pergunta menos interessante sobre o meu trabalho. O que eu faço é decidir o que o sistema deve ser e garantir que ele seja isso. Na minha experiência, essa também é a parte pela qual os clientes realmente pagam.

Para levar

Três trimestres de 2026 mudaram o que é o meu trabalho, e a página inicial finalmente diz isso. A versão curta é esta.

- Entreguei este ano código de produção em TypeScript, Swift, Kotlin, C#, SQL e Delphi. A tabela acima é uma amostra pública disso, não a lista completa, e sobre uma dessas linguagens ainda devo um artigo a você.
- O trabalho agora é arquitetura, especificação e verificação. As ferramentas escrevem a sintaxe; eu decido o que o sistema é e leio tudo o que volta. Isso funciona por causa de 35 anos sabendo como o correto se parece -- e a pesquisa diz que não funcionaria sem isso.
- As evidências contra o exagero são reais: desenvolvedores experientes mensuravelmente mais lentos no estudo da METR, confiança em queda no resultado da IA e instabilidade em alta onde o processo é fraco. Minha afirmação está delimitada a um processo com testes, revisões e portão de deploy, e eu delimitaria a sua da mesma forma.
- O Delphi fica. Seu ranking caiu forte este ano e a Embarcadero continua lançando; ambos são verdade. Para desenvolvedores Delphi, as camadas valiosas -- domínio, arquitetura, julgamento -- estão intactas, e a barreira que mantinha você em uma única linguagem praticamente se dissolveu.

> A linguagem nunca foi a competência. Era a porta de entrada. Em 2026, a porta de entrada é uma especificação e um leitor cuidadoso, e eu me tornei o segundo.

Se você tem um sistema para construir e nenhum apego especial a como ele será escrito, vamos conversar. E se você é um desenvolvedor Delphi olhando para o mesmo calendário que eu, comece pela série e me escreva quando travar. Essa ponte continua de pé.

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 —