NEWS · OCTOBER 23, 2026

Pas seulement Delphi : comment trois trimestres de 2026 ont réécrit mon métier

Un prospect a quitté mon site avec l'impression « ce type ne fait que du Delphi » -- voici donc à quoi ont vraiment ressemblé trois trimestres de 2026, avec TypeScript, Swift, Kotlin, C# et Delphi en production, et pourquoi le métier s'appelle désormais architecture et vérification, et non maîtrise d'une seule syntaxe.

By Dr. Holger Flick



Il y a quelques semaines, une entreprise cherchait quelqu'un pour lui construire une application. On lui a indiqué mon site. Elle l'a lu, et elle en est repartie avec une seule impression : « Ah, ce type ne fait que du Delphi. Nous n'utilisons pas Delphi. » Je l'ai appris de seconde main, et c'était précisément le retour que je ne voulais pas entendre -- non parce qu'il était injuste, mais parce que ma propre page d'accueil l'avait mérité. Le mot « Delphi » figurait dans le bandeau, dans les chiffres, dans le pied de page et dans le premier service de la liste.

Voici pourtant le fond de l'affaire. Depuis janvier, en neuf mois, j'ai livré du code de production en TypeScript, Swift, Kotlin, C#, SQL et, oui, Delphi. Rien de tout cela n'est tiré par les cheveux, et rien n'est nouveau pour moi au sens qui compte. Ce qui est nouveau, c'est la façon dont le travail se fait. En 2025, je me serais décrit comme un expert Delphi qui construit aussi pour le web. En octobre 2026, cette description est tout simplement fausse. Je suis l'architecte. Je connais les frameworks et les architectures, je décide de ce que le système doit être, et je dirige les outils qui écrivent désormais l'essentiel du code -- dans le langage qui convient à la tâche, quel qu'il soit. Puis je vérifie chaque ligne, parce que 35 ans dans ce métier m'ont appris à quoi ressemble du code correct.

Cet article est la version longue de ce que la page d'accueil aurait dû dire. Vous y trouverez un échantillon public de ce que j'ai construit cette année et dans quel langage, l'argument expliquant pourquoi « quel langage » est

devenu la question la moins intéressante de mon travail, les preuves pour et contre cet argument, et -- parce que je ne brûle aucun pont -- ce que tout cela signifie si vous êtes un développeur Delphi qui regarde le même calendrier.

Ce que le site disait, et ce qu'il aurait dû dire

L'ancienne page d'accueil ne mentait pas ; elle avait simplement un an de retard. La formule « 30+ ans de Delphi » était vraie, le titre d'Embarcadero MVP était vrai, et les migrations Delphi vers le web que je présentais comme études de cas étaient réelles et tournent encore. Mais un visiteur sans le moindre passé Delphi lit une page de ce genre et conclut raisonnablement que la partie Delphi est l'essentiel et la partie web un effet secondaire. Pendant la plus grande partie de ma carrière, cette lecture aurait été à peu près juste. En 2026, elle ne l'est plus.

J'ai donc réécrit la page. Les stacks que j'ai livrées cette année figurent désormais là où les chiffres se vantaient d'un seul langage. Les services commencent par les systèmes sur mesure et les applications pour toutes les plateformes, et le travail Delphi -- migrations, conseil, sauvetage de code -- est toujours là, comme un service parmi sept. Je reviendrai sur la raison pour laquelle il reste. D'abord, quelques preuves.

Un échantillon de ce qui a été livré en 2026, par langage

Plutôt que de revendiquer de la polyvalence, laissez-moi vous montrer un échantillon. Je tiens à être clair : c'est un échantillon et non le grand livre. La plus grande partie de mon travail est destinée à des clients et ne donne jamais lieu à un article ou à une étude de cas, et il y a derrière plus d'applications iOS et Android que celles nommées ci-dessous. Ce que fait le tableau, c'est choisir, pour chaque langage, une ou deux choses que je peux montrer publiquement. Tout ce qui y figure est soit publié sur ce site, soit décrit dans une étude de cas anonymisée sur la page des réalisations, avec une exception honnête que je signale.

Langage	Un exemple public	Où regarder
TypeScript, Next.js, React, PostgreSQL	Ce site web, un système de gestion d'entrepôt, une boutique B2B, une application de livraison offline-first, une application Delphi 7 reconstruite pour le navigateur	Études de cas, la série Delphi vers Next.js
Swift, SwiftUI	Hearth : des applications pour iPhone, iPad, Apple TV et Mac sur un seul backend, 313 tests automatisés, 48 versions en douze jours -- le seul projet iOS que je peux montrer en entier	Construire plutôt qu'acheter, Pas du vibe coding
Kotlin	Des applications Android construites au-dessus d'applications web existantes, pour des clients	Pas encore d'article -- c'est l'exception ; j'en écrirai un
C# et .NET	Un service de rapports en C# lu, compris et replié dans l'application Next.js à laquelle il appartenait ; deux critiques de livres sur .NET 10 en chemin	D'un service de rapports C# à une seule base de code Next.js, C# 14 et .NET 10
Delphi	FireDAC et PostgreSQL à partir de zéro, WebStencils avec SQLite, Delphi 13.2 sous Linux, un outil natif construit en un après-midi à partir d'une spécification	CLI FireDAC, WebStencils et SQLite- , Spec coding avec Delphi

Relisez ce tableau et remarquez ce qui manque : une ligne qui dirait « six mois passés à apprendre Swift ». Il n'y en a pas. Les applications Swift existent parce que je savais à quoi devait ressembler un client iPad d'un backend Next.js avec connexion par passkey, que je l'ai écrit noir sur blanc, et que j'ai dirigé les outils jusqu'à

ce que le résultat corresponde -- puis je l'ai testé sur l'appareil réel, comme l'article Pas du vibe coding le documente en détail. Les applications Kotlin sont nées de la même façon, au-dessus d'une application web qui définissait déjà chaque règle. D'expérience, c'est le fait de disposer d'une application web fonctionnelle comme base qui transforme une application native en une affaire de jours plutôt que de mois ; le code spécifique à la plateforme est mince par conception, et toute la logique susceptible de se tromper vit sur le serveur, où elle était déjà testée.

Le métier a changé ; l'expertise, non

Voici la partie que je crois réellement nouvelle, et je tiens à la décrire avec soin plutôt qu'à l'enjoliver. Pendant la plus grande partie de ma carrière, l'unité d'expertise était le langage. On était développeur Delphi, développeur C# ou développeur Java, et changer signifiait des mois à être une version dégradée de soi-même avant de redevenir bon. Les frameworks et l'architecture comptaient, bien sûr, mais on y accédait par une syntaxe qu'il fallait posséder entièrement.

Ce n'est plus ainsi que se déroulent mes journées. Le diagramme ci-dessous montre ce que je veux dire, et c'est l'image la plus honnête que je puisse dessiner de 2025 face à 2026.



2025 : un spécialiste d'un langage atteint l'architecture à travers une seule syntaxe. 2026 : l'architecte conçoit le système et dirige les outils vers le langage qui convient.

La colonne de gauche, c'est ainsi que j'ai travaillé pendant une trentaine d'années, et je ne la dénigre pas ; elle a produit beaucoup de bons logiciels. La colonne de droite, c'est ainsi que j'ai travaillé cette année. Remarquez quelle case porte le trait épais. En 2025, le langage était la couche irremplaçable. En 2026, la couche irremplaçable est la personne tout en haut qui sait ce que le système doit être, connaît les frameworks et l'architecture assez bien pour le spécifier précisément, et en sait assez sur tous ces langages pour lire ce qui revient et rejeter ce qui est faux.

Cette dernière proposition, c'est tout le métier désormais. Les outils peuvent produire du Kotlin idiomatique plus vite que je ne peux le taper. Ce qu'ils ne peuvent pas faire, c'est décider que l'isolation entre familles relève des déclencheurs de base de données et non du code applicatif, ou remarquer qu'un écran de connexion sur iPad ressemble à l'écran d'appairage d'un téléviseur, ou reconnaître qu'un rapport qui « a l'air juste » calcule une taxe de travers. Ces décisions venaient de moi, et l'article Pas du vibe coding les énumère une à une. Je n'ai donc pas besoin d'apprendre Kotlin comme j'ai appris Object Pascal en 1995, avec un gros livre et une année d'erreurs. Je dois connaître l'architecture Android, savoir à quoi ressemble un programme Kotlin correct quand j'en lis un, et avoir la discipline de les lire tous. Les deux premiers me viennent de décennies à faire ce travail sur toutes les plateformes, et le troisième n'est qu'une question de caractère.

Ce n'est pas du vibe coding, et voici où cette affirmation s'arrête

Je tiens à dire explicitement que le paragraphe ci-dessus est mon opinion, et que je sais qu'il sonne comme chaque article « 10x grâce à l'IA » que vous avez fait défiler cette année. Laissez-moi donc l'étayer, puis le contester sérieusement.

L'étayage est simple : le processus est documenté et les chiffres sont publics. Hearth est passé par 52 pull requests, chacune décrite, relue et testée, avec 313 tests automatisés, 34 contrôles d'isolation en base de données et un portail de déploiement que nous avons délibérément cassé une fois pour prouver qu'il refuse de livrer. Simon Willison a tracé la ligne que je trouve la plus utile : le vibe coding, c'est « construire un logiciel avec un LLM sans relire le code qu'il écrit », et si vous le relisez, le testez à fond et pouvez expliquer comment il fonctionne, c'est simplement du développement logiciel. Sa règle d'or : ne jamais valider du code dont il ne pourrait pas expliquer exactement à quelqu'un d'autre ce qu'il fait. Je m'impose la même règle, et c'est la règle que 35 ans rendent possible : on ne peut pas relire du Kotlin qu'on ne sait pas lire, et on ne lit pas bien du Kotlin si l'on n'a jamais réfléchi sérieusement à la façon dont une application Android doit être structurée.

Maintenant, la contestation, et je la prends au sérieux.

> **Là où cela ne tient pas**
>
> METR a mené début 2025 un essai randomisé avec 16 développeurs open source expérimentés sur 246 tickets réels dans leurs propres dépôts. Avec les outils IA autorisés, ils ont mis 19 pour cent *plus* de temps -- et croyaient encore ensuite avoir été 20 pour cent plus rapides. Les auteurs énumèrent eux-mêmes les réserves (petit échantillon, développeurs novices avec l'outil précis, résultats déjà dépassés par leur propre suivi de 2026), mais l'écart de perception est le résultat qui devrait inquiéter tout le monde, moi compris. La sensation de vitesse n'est pas une preuve de vitesse. L'enquête Stack Overflow 2025 auprès des développeurs a trouvé 84 pour cent de développeurs utilisant ou prévoyant d'utiliser des outils IA, tandis que 46 pour cent disent ne pas faire confiance au résultat, contre 31 pour cent un an plus tôt. Plus d'usage, moins de confiance. Et le rapport DORA 2025 constate que l'adoption de l'IA est désormais corrélée à un débit plus élevé *et* à une instabilité plus élevée -- plus d'échecs de changement, plus de reprises. Leur formule : l'IA « amplifie ce qui est déjà là ».

Voici donc la version délimitée de mon affirmation, la seule que je défende. Diriger des outils IA à travers plusieurs langages fonctionne quand la personne qui les dirige possède déjà le jugement architectural et la capacité de lecture pour attraper ce qui est faux, et quand le processus force cette lecture : pull requests, tests, relecture, portail de déploiement. Dans ces conditions, j'ai livré cette année plus de logiciels, et de meilleurs, qu'aucune année auparavant. Sans ces conditions, la recherche dit que vous livrerez probablement des logiciels qui semblent plus rapides et sont moins stables, sans vous en rendre compte. Le « amplifie ce qui est déjà là » de DORA est exactement juste : dans mon cas, les outils ont amplifié 35 ans d'expérience. Ils amplifieraient leur absence avec la même fidélité.

> **L'autre face de l'argument architectural**
>
> Une objection légitime : si les applications natives sont « minces par conception » et que la logique vit sur le serveur, n'ai-je pas simplement choisi le cas facile ? En partie, oui. Une application Kotlin dont le rôle est d'afficher des données et de parler à une API bien définie est un problème bien plus petit qu'une application Kotlin avec un état local complexe, des traitements en arrière-plan et la résolution de conflits hors ligne. Je soutiendrais quand même que pousser la logique vers un seul endroit testé est la bonne architecture pour la plupart des applications métier, et que c'est précisément le genre de décision qu'un architecte prend avant qu'un langage soit choisi. Mais je ne prétendrais pas que toute application native peut être produite à si bon compte, et un projet qui exige réellement une expertise profonde de la plateforme a toujours besoin de quelqu'un qui la possède.

Ce que cela signifie pour la communauté Delphi

Voici maintenant la partie promise. Je ne laisse pas Delphi derrière moi, et je veux dire clairement pourquoi, car les chiffres honnêtes ne sont pas tendres cette année et je ne ferai pas semblant du contraire.

L'indice TIOBE d'octobre 2026 place Delphi/Object Pascal au 19e rang avec 0,83 pour cent, contre le 9e rang un an plus tôt. C'est une chute abrupte dans un classement qui, reconnaissons-le, mesure l'intérêt des recherches et non les logiciels en production, et je mettrais quiconque en garde contre la lecture d'un seul indice comme d'un verdict. Dans le même temps, Embarcadero a livré RAD Studio 13 Florence en septembre 2025 et 13.1 en mars 2026, j'ai écrit en septembre sur Delphi 13.2 sous Linux, et je reçois encore chaque semaine du courrier de personnes qui maintiennent des systèmes Delphi faisant tourner de vraies entreprises. Ces systèmes ne vont pas disparaître de sitôt, et le besoin de gens capables de les lire non plus.

Mon opinion -- et c'est une opinion, pesez-la donc comme telle -- est que Delphi comme *compétence de carrière principale* est sur le déclin, au sens où moins de nouveaux systèmes seront démarrés avec, tandis que Delphi comme *parc de logiciels en service* aura besoin de soins experts pendant longtemps. Si vous avez passé vingt ans dans Delphi, ce sont là deux bonnes nouvelles, et voici pourquoi.

Tout ce qui faisait votre valeur la fait encore : vous comprenez votre domaine métier, vous savez comment une application multi-tiers est structurée, vous avez débogué des transactions FireDAC à trois heures du matin, et vous savez à quoi ressemble un programme correct quand vous en lisez un. Ce qui a changé, c'est que le langage n'est plus le fossé qui protège ce savoir ; c'en est désormais la partie la moins chère. Cela signifie que ce qui vous retenait dans Delphi -- le coût de redevenir débutant en TypeScript ou en C# -- s'est en grande partie évaporé. Vous n'avez pas à redevenir débutant. Vous devez devenir l'architecte que vous êtes déjà, à voix haute, et laisser les outils taper.



Les atouts d'un développeur Delphi, et ceux que les outils changent. Les couches précieuses restent les vôtres ; la couche syntaxique est celle qui est devenue bon marché.

Les trois cases pleines sont ce qu'un vétéran de Delphi apporte à n'importe quel projet, et c'est exactement ce que j'ai apporté au travail Swift et Kotlin cette année. La case en pointillés était autrefois la barrière. Ma série Delphi vers Next.js a été écrite pour cette transition, et si vous vous trouvez du côté Delphi, c'est toujours la première chose que je vous mettrais entre les mains. La stack de destination compte moins que vous ne le pensez ; l'habitude de tout lire avant de livrer compte plus que jamais.

> **Si vous préférez rester sur Delphi**
> C'est un choix légitime, et rien ici ne s'y oppose pour les systèmes que vous exploitez déjà. Le même outillage IA fonctionne dans un projet Delphi ; cette année, j'ai construit un outil Delphi natif en un après-midi

> à partir d'une spécification et intégré un modèle de langage local dans une application Delphi. L'important n'est pas la destination. L'important, c'est que le langage a cessé d'être ce qui vous limite, dans un sens comme dans l'autre.

Ce que cela signifie si vous recrutez

Si vous êtes l'entreprise qui est repartie il y a quelques semaines, cette section est pour vous, et je la ferai courte. Apportez-moi le problème, pas le langage. Décrivez-moi le flux de travail qui a besoin d'un logiciel, les systèmes avec lesquels il doit dialoguer, les personnes qui l'utiliseront et les appareils qu'elles tiennent en main. Je proposerai une architecture et la stack qui lui convient -- Next.js et PostgreSQL là où une application navigateur est le bon choix, C# et ASP.NET là où votre équipe vit déjà dans .NET, Swift ou Kotlin là où il faut du natif, Delphi là où un backend Delphi fonctionnel doit rester exactement où il est. Puis je le construirai avec les outils, lirai chaque ligne, le testerai et vous montrerai les pull requests.

Quel langage je « fais » est devenu la question la moins intéressante au sujet de mon travail. Ce que je fais, c'est décider de ce que le système doit être et m'assurer qu'il le soit. D'expérience, c'est aussi la partie pour laquelle les clients paient réellement.

À retenir

Trois trimestres de 2026 ont changé ce qu'est mon métier, et la page d'accueil le dit enfin. Voici la version courte.

- J'ai livré cette année du code de production en TypeScript, Swift, Kotlin, C#, SQL et Delphi. Le tableau ci-dessus en est un échantillon public, pas la liste complète, et pour l'un de ces langages je vous dois encore un article.
- Le métier, c'est désormais l'architecture, la spécification et la vérification. Les outils écrivent la syntaxe ; je décide de ce qu'est le système et je lis tout ce qui revient. Cela fonctionne grâce à 35 ans à savoir à quoi ressemble le correct -- et la recherche dit que cela ne fonctionnerait pas sans.
- Les preuves contre l'exagération sont réelles : des développeurs expérimentés mesurablement plus lents dans l'essai de METR, une confiance en baisse dans les résultats de l'IA, et une instabilité en hausse là où le processus est faible. Mon affirmation est délimitée à un processus avec tests, relectures et portail de déploiement, et je délimiterais la vôtre de la même façon.
- Delphi reste. Son classement a lourdement chuté cette année et Embarcadero continue de livrer ; les deux sont vrais. Pour les développeurs Delphi, les couches précieuses -- domaine, architecture, jugement -- sont intactes, et la barrière qui vous retenait dans un seul langage s'est en grande partie dissoute.

> Le langage n'a jamais été l'expertise. Il était la porte d'entrée. En 2026, la porte d'entrée est une spécification et un lecteur attentif, et je suis devenu le second.

Si vous avez un système à construire et aucun attachement particulier à la façon dont il sera écrit, parlons-en.

Et si vous êtes un développeur Delphi qui regarde le même calendrier que moi, commencez par la série, et écrivez-moi quand vous bloquez. Ce pont reste debout.

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 —