mdt.

Ex Datis, Claritas

¿Cómo será el desarrollo de software en el futuro?

El desarrollo de software será igual y totalmente distinto: hice sin escribir código un plugin de 4.800 horas, pero el cambio no llega igual a todo el software.

Manuel Delgado Tenorio · 13 de octubre de 2026

Publicado originalmente en www.manueldelgado.com/blog/como-sera-desarrollo-software-futuro



El otro día, me preguntaron cómo creo que será el desarrollo de software dentro de unos años, cuando las herramientas de IA hayan avanzado todavía más respecto al estado actual. Mi respuesta, aunque improvisada, resultó resumir perfectamente mi opinión al respecto: será completamente igual y totalmente distinto.

EN LA RADIO (14 DE OCTUBRE DE 2026)
He estado en el programa "No son horas", de Onda Cero, charlando con Gemma Ruiz sobre este tema. Si no pudiste escucharlo en directo, ya puedes oír la grabación del programa en Spotify.

A estas alturas de la película, ya deberíamos estar acostumbrados a que **dos cosas que parecen opuestas puedan ser ciertas a la vez.** En el caso que nos ocupa, la razón principal de esa aparente paradoja es que "el desarrollo de software" no es, ni de lejos, un único ente monolítico que solo se mueve en una dirección.

Hay un buen puñado de variables con las que podemos trocear y parcelar esta disciplina: software de misión crítica vs. software cotidiano; software de negocio vs. software de sistemas vs. software de consumo; software embebido vs. software de propósito general[1]; software de un solo uso (como un script ad hoc para cargar unos datos) o software mantenible[2]; software desarrollado por un amateur vs. software desarrollado por profesionales... y así hasta el infinito[3].

La cosa es que todas esas parcelas forman parte del desarrollo de software, pero **los cambios no afectan de la misma forma, ni a la misma velocidad, a unas y a otras.** Así que si debatimos sobre cómo está cambiando el desarrollo de software o cómo va a cambiar en los próximos años, más nos vale acotar bien el ámbito del que estamos hablando, porque lo que se aplique a, por ejemplo, el software de control de los motores de un avión comercial ni de lejos se aplica a una chorrada programada en una tarde para mi Arduino o a una macro de Excel para dar formato a una tabla.

Toda esta historia y este debate surgió a raíz de un *side project* que he publicado hace unos pocos días. Se llama "Matters that Matter" y, en su forma actual, es un plugin para Obsidian para la gestión de tareas y pequeños proyectos personales. Lo he hecho yo solo, en unos pocos días, y ya está disponible (junto con su companion theme, Calm Matters) en el directorio de la comunidad de Obsidian. Es la primera vez que genero un producto de software sin abrir ni una sola vez un IDE, todo desde Claude Code. Literalmente, y también por primera vez, **no he escrito ni una sola línea de las 27.000 (más 12.000 de tests) que lo componen** aunque, eso sí, he escrito páginas de texto para aburrir y he leído más código que en todo el último año junto.

La cosa es que, cuando anuncié la disponibilidad del plugin, me surgió la curiosidad de cuánto habría tardado (y cuánto habría costado) desarrollar este plugin (incluyendo la plantilla, su CLI y su servidor MCP, la documentación, la web, etc.).

[1] Me cuesta horrores encontrar una forma de referirme al software "normal", el que no está embebido en un dispositivo, que no sea decir "software no embebido". Espero que se me entienda.

[2] Me pasa lo mismo aquí. Para los muy cafeteros, esta es la distinción que hace Fred Brooks, en "The Mythical Man-Month", que distingue entre un "programa" (algo que funciona para su autor en su contexto) y un "producto de programación" (generalizado, probado y documentado para que otros lo usen y mantengan).

[3] Quizá no tanto como el infinito, pero sí más de lo que tiene sentido seguir enumerando aquí.

Hice un rápido cálculo de servilleta y me salieron unas cifras como para quedarse helado, propias de una startup dedicada a esto. Más datos, en la imagen:



No me voy a pelear con nadie por esas cifras: si en lugar de 4.800 horas de trabajo o 320.000 euros de coste alguien me dice que la realidad es solo un 20% o un 10% de eso, pues también me vale, porque **eso seguiría siendo, como mínimo, varias docenas de veces más que el esfuerzo que realmente me ha supuesto**. Hablamos de **ahorros de varios órdenes de magnitud,** en cualquier caso. Eso sí, no hagamos trampas al solitario: esas cifras son solo de creación y, como recuerda Scott Brinker, la regla general es que la construcción inicial supone el 30% del trabajo de ingeniería y el mantenimiento posterior, el 70% restante (un mantenimiento que la IA también acelera, pero que no consiste solo en escribir código).

Por supuesto, nada de esto es nuevo. Ya hablé de esta tendencia a principios de año. En estos diez meses que nos separan, lo que ha cambiado es que lo que entonces solo empezábamos a intuir empieza a ser realidad en algunos

ámbitos como este, el software de consumo y gratuito. Esto no hace sino reforzarme más en la idea de entonces: es posible que llegue un momento en que dejemos de instalar aplicaciones básicas en nuestros dispositivos; en su lugar, nos limitaremos a explicarle lo que necesitamos y el dispositivo las creará para nosotros, completamente personalizadas.[4] Es un escenario todavía futurista, pero hoy ya es posible trazar un camino para llegar desde aquí hasta allí.

La discusión sobre lo que estos avances significan para (esta parte de) la industria del software, cómo está en riesgo el modelo de pago de muchísimas aplicaciones o cómo, aún más que hasta ahora, es la distribución (y no la creación del software) la clave para construir un negocio exitoso es, sin duda, un debate apasionante[5]. Y, sin embargo, **pensar que, a su vez, eso afecta a todo el software por igual y a la misma velocidad es completamente ingenuo.**

Y no hace falta irse a los extremos para comprobarlo. Germán Del Zotto, un ingeniero con treinta años de oficio, lo contaba muy bien el pasado mes de marzo en La Bonilista: en un proyecto personal desplegó una aplicación en producción en 27 días, y solo 9 de los 545 cambios en el código fueron suyos (un 1,7%). Pero de lunes a viernes, en una empresa de más de mil empleados, las revisiones de seguridad, los procesos de aprobación y la coordinación de decenas de personas siguen ahí, igual que siempre, y la regla es clara: "si tú lo subes a un repositorio, es TU código, da igual si lo genera la IA o tu vecino". **La misma persona, con las mismas herramientas, vive dos realidades completamente distintas** según la parcela en la que esté trabajando.

Pero, atención, lo contrario también es cierto: **es exactamente igual de ingenuo pensar que hay parcelas del desarrollo de software que son completamente inmunes a lo que está pasando** y a lo que está por pasar. Llegará más tarde y/o llegará con matices, pero llegará.

Hay voces, por ejemplo, que se aferran a que **mucho del código producido por LLMs no sigue las buenas prácticas de desarrollo o que acarrea riesgos de seguridad**[6] y eso es, también sin duda, muy cierto. Pero igual de cierto es que el software que producen estos sistemas a día de hoy, octubre de 2026, **ya es mejor y más seguro que el de una buena parte**[7] **de los profesionales de la programación**[8]**.** Porque, claro, recordemos que no todo el mundo puede ser *committer* del kernel de Linux: también existen los que desarrollan, qué se yo, la web de Renfe.

[4] No es, ni mucho menos, la única visión posible de ese futuro. Satya Nadella, por ejemplo, habla de convertir Copilot en una "fábrica infinita de SaaS": software a medida, sí, pero construido dentro de la empresa y anclado a sus sistemas de registro (el CRM, el ERP...), no aplicaciones que el dispositivo se saca de la manga. Pero todas esas visiones, la suya, la mía y unas cuantas más, tienen algo en común: **asumen un cambio total de paradigma en cómo se crean, se distribuyen y se consumen este tipo de aplicaciones.**

[5] Un debate apasionante y, además, muy actual, porque esto ya está aquí: para que el mercado se ponga patas arriba no hace falta esperar a que incluso tu sobrino preadolescente sea capaz de desarrollar una aplicación para iPhone que pase los controles de la App Store; basta con que se multiplique por veinte o por cincuenta el número de aplicaciones disponibles. Y algo me dice que no vamos a tardar mucho en ver eso; de hecho, ya está empezando: según datos de Appfigures, los lanzamientos de nuevas aplicaciones para iOS crecieron en torno a un 80% interanual a principios de 2026, y la revisión de Apple ya lo está notando en sus tiempos de aprobación.

[6] Un buen ejemplo de esta postura es este artículo de Alex Ewerlöf, que recomiendo leer aunque no coincido con él, precisamente porque cae en los dos errores de los que hablo aquí: trata a toda la industria del software como un monolito uniforme y da por hecho que todos los programadores son excelsos artesanos del código (y, cuando acepta que no es así, lo hace cayendo en una descomunal falacia del escocés auténtico).

[7] Me cuesta aventurar una cifra, pero me pide el cuerpo decir que ya es mucho mejor que el percentil 40 de los desarrolladores profesionales.

[8] Entre los que, por si quedaba alguna duda, estoy lejos de considerarme (aunque, también, tengo más tiros pegados que algunos que seguro me mirarían por encima del hombro).

Así que no hay riesgo en afirmar que, por ejemplo, nadie va a desarrollar a mano plugins de organización personal para Obsidian nunca más y, a la vez, que el software de control de las centrales nucleares se va a seguir manteniendo a mano durante bastante tiempo. Pero sería demasiado atrevido decir que la IA nunca cambiará cómo se mantiene el software de las centrales nucleares, porque ya empieza a ser solo una discusión sobre cuándo ("cuándo", no "si") se darán las condiciones para que sea viable.

De hecho, ese camino ya ha empezado, aunque por la puerta de atrás: en el software más exigente, **la IA está entrando antes por la verificación que por la escritura**. Greg Kroah-Hartman, uno de los principales mantenedores del kernel de Linux, contaba esta primavera que los informes de errores generados con IA habían pasado, en cuestión de un mes, de ser ruido a ser mayoritariamente válidos. Y Mozilla corrigió en abril 423 vulnerabilidades de Firefox, tantas como en los dos años anteriores, cientos de ellas detectadas con ayuda de la IA. Sí, el territorio de esos mismos *committers* del kernel que mencionaba antes.

Así que, volviendo a la pregunta del principio: el desarrollo de software será completamente igual y totalmente distinto. **Lo difícil no es elegir cuál de las dos cosas es cierta, sino saber en qué parcela estamos en cada momento.**

Coda

Exactamente lo mismo que se aplica al desarrollo del software se puede decir de la investigación científica. Me remito a lo que ya conté la semana pasada en la radio y a este artículo que me ha gustado mucho (y cuya visión del futuro deseable comparto plenamente).



Sobre el autor

Manuel Delgado Tenorio

Manuel Delgado Tenorio es asesor independiente de estrategia de datos e inteligencia artificial, profesor e investigador. Cofundó y vendió una *startup* de inteligencia artificial, fue socio de la práctica de Data & Analytics para servicios financieros en EY y responsable de Data Analytics para el sur de Europa en AWS. Es profesor asociado en IE University, donde imparte la primera asignatura de grado sobre Customer Lifetime Value en España, investiga sobre el CLV en su doctorado en la Universidad Complutense de Madrid y es autor del libro "Customer Lifetime Value (CLV): No todos tus clientes valen lo mismo".

manueldelgado.com · LinkedIn · X

Recibe cada artículo en tu correo con la newsletter Ex Datis, Claritas: manueldelgado.com/newsletter/suscribete