THOUGHTWORKS TECHNOLOGY RADAR 29 DE ABRIL DE 2026

ENTREVISTA TÉCNICA

Tech Radar Explorer

9 preguntas · Sesión 7e0856ca...

— Hola, vamos a comenzar esta esperada entrevista. Vamos con la primera pregunta. Este 2026 está siendo un año de vértigo en lo que se refiere al impacto de la tecnología pero, desde tu punto de vista, hasta llegar este punto actual, ¿cómo describirías la evolución del sector tecnológico en los últimos años?

Buena pregunta para arrancar. Si tuviera que resumirlo en una frase: hemos pasado de la experimentación entusiasta a la ingeniería responsable, pero con mucho ruido en el camino.

Lo que hemos observado a lo largo de las últimas ediciones del Radar es una aceleración que no tiene precedentes, pero con un patrón reconocible. Cada ciclo tecnológico sigue una curva similar: euforia, saturación, y luego — si la tecnología tiene valor real — consolidación. Con la IA generativa hemos vivido esa curva en cámara rápida.

Hace tres o cuatro años, el debate era si los LLMs eran algo más que un juguete sofisticado. Luego vino la explosión del ecosistema — RAG, vector databases, guardrails, frameworks de agentes — que en el Radar de 2024 ya describíamos como una "explosión cámbrica". Era caótico, pero tenía sentido: la simplicidad inicial del prompt de texto estaba dando paso a la ingeniería de software real.

Este es el punto que más nos importa subrayar: la tecnología madura cuando deja de ser magia y se convierte en disciplina. Y eso es exactamente lo que estamos viendo ahora en 2026 con los agentes, con el context engineering, con los patrones de evaluación de LLMs.

Pero también hemos visto lo que no funcionó como esperábamos. Fuimos inicialmente optimistas con la autonomía total de los agentes de código, y hemos tenido que ser honestos: el enfoque supervisado sigue siendo imprescindible. La complacencia con el código generado por IA es un antipatrón real, documentado, con datos que lo respaldan.

One of the patterns we keep seeing es que la velocidad de las herramientas supera la madurez de las prácticas. Y eso tiene un coste: deuda cognitiva, shadow IT acelerado por IA, arquitecturas que derivan sin control.

El sector está en un momento apasionante, pero los fundamentos de ingeniería — testing, observabilidad, diseño arquitectónico — nunca han sido más relevantes que ahora.

— Me parece muy interesante lo que comentas de que "los fundamentos de ingeniería nunca han sido más relevantes qué ahora". ¿Por qué ahora son una pieza clave? ¿No lo han sido siempre?

Sí, siempre lo han sido. Pero hay una diferencia crucial entre que algo sea importante y que su ausencia sea inmediatamente catastrófica.

Antes, si un equipo tenía poca cobertura de tests o una arquitectura mediocre, el daño se acumulaba lentamente. Había fricción, deuda técnica, pero el sistema degradaba de forma gradual y visible. Ahora, con agentes de código generando changesets masivos en minutos, esa misma ausencia de disciplina puede destruir la coherencia de un sistema en horas. La velocidad amplifica todo, lo bueno y lo malo.

This is where things get interesting. Lo que estamos documentando en el Radar como *codebase cognitive debt* no es simplemente deuda técnica tradicional. Es algo más sutil y más peligroso: es la pérdida del modelo mental colectivo sobre cómo funciona el sistema. Cuando el código lo genera la IA y nadie lo revisa en profundidad, el equipo empieza a operar sobre una caja negra que ellos mismos construyeron sin entender. Los datos de GitClear son bastante elocuentes al respecto: más código duplicado, más churn, menos refactoring.

Y aquí está la paradoja que más nos preocupa: los agentes de código funcionan *mejor* en codebases bien estructuradas, bien testeadas, con diseño limpio. Es decir, las prácticas que algunos equipos estaban tentados a relajar porque "la IA lo arregla" son exactamente las que hacen que la IA funcione bien. El buen diseño para humanos también es buen diseño para agentes.

Por eso en el último Radar hemos puesto en Adopt cosas como DORA metrics, context engineering, zero trust, OpenTelemetry. No son novedades. Son recordatorios de que sin esa base, la velocidad que promete la IA se convierte en deuda compuesta.

Lo que ha cambiado no es la importancia de los fundamentos. Es el coste de ignorarlos.

— Viendo la evolución de estos últimos años, ¿qué puntos consideras clave para que la incorporación de la IA sea un éxito hoy día en una empresa?

Lo primero que diría es que el fracaso más común que vemos no es técnico. Es organizacional. Las empresas que están fallando con IA no fallan porque eligieron el modelo equivocado o el framework incorrecto. Fallan porque intentaron incorporar IA sobre una base que ya era frágil.

Dicho esto, hay varios patrones que hemos documentado como críticos.

Primero, tratar el contexto como un activo de ingeniería, no como un detalle de implementación. Context engineering está en Adopt en el último Radar precisamente porque hemos visto que las organizaciones que lo hacen bien —que diseñan el contexto como una pipeline dinámica y gestionada— obtienen resultados cualitativamente distintos. Las que lo ignoran acaban con agentes alucinando o comportándose de forma inconsistente.

Segundo, invertir en las señales de feedback antes de escalar. DORA metrics en Adopt no es nostalgia. Es un recordatorio de que si no mides lead time, deployment frequency y rework rate, no sabrás si la IA está generando valor real o simplemente más código. Y el rework rate específicamente es el canario en la mina para detectar deuda cognitiva acumulada.

Tercero, gobernar los agentes desde el principio. Zero trust architecture en Adopt para agentes no es opcional. El "lethal trifecta" que describe Simon Willison —datos privados, contenido no confiable, acción externa— describe a la mayoría de los agentes útiles por defecto. Si no tienes least privilege y monitorización continua desde el día uno, estás construyendo sobre arena.

Y cuarto, y esto es quizás lo más contraintuitivo: las empresas que más éxito tienen son las que primero invierten en limpiar su codebase y sus prácticas de ingeniería, y *después* incorporan agentes. No al revés.

Lo que observamos repetidamente es que la IA amplifica lo que ya existe. Si tienes disciplina, la amplifica. Si tienes caos, también lo amplifica. Y a una velocidad que no da tiempo a reaccionar.

— A nivel de tecnologías, durante estos últimos años, ¿cuáles son las que se asentado totalmente y cuáles se han perdido por el camino o se van diluyendo?

Esta es una pregunta que me encanta porque el movimiento entre rings es donde realmente se ve si algo tiene sustancia o si era ruido.

Las que se han asentado de verdad:

OpenTelemetry es quizás el ejemplo más limpio. Pasó de Assess a Adopt de forma casi inevitable, porque resolvió un problema real —la fragmentación de observabilidad— con una abstracción correcta. Cuando Datadog, New Relic y Grafana adoptan el mismo protocolo, sabes que algo ha ganado. Eso no es hype, es consolidación genuina.

RAG —Retrieval-Augmented Generation— llegó a Adopt en tiempo récord. Lo que nos sorprendió no fue la velocidad, sino que la técnica sobrevivió al escrutinio de producción. Sigue evolucionando —GraphRAG, Corrective RAG, Self-RAG— pero el patrón central aguantó.

Y en tooling, cosas como Renovate o Vite: sin glamour, sin conferencias keynote, pero en Adopt porque simplemente funcionan y los equipos no quieren vivir sin ellos.

Las que se están diluyendo o nunca despegaron:

SAFe sigue en Hold, y cada edición el lenguaje se vuelve más duro. No es que no se use —se usa demasiado— sino que seguimos sin ver los resultados que promete.

Local coding assistants es interesante: llegaron con mucha expectativa por razones de privacidad, pero los pusimos en Hold porque los modelos locales simplemente no compiten con los cloud-based en capacidades agenticas. La realidad técnica ganó al deseo organizacional.

Y lo más revelador: "Replacing pair programming with AI" en Hold. Esto es casi una declaración de principios nuestra. No porque la IA no ayude, sino porque confunde velocidad individual con salud del equipo. Son cosas distintas.

El patrón que veo: lo que se asienta resuelve fricción real con abstracciones durables. Lo que se diluye suele ser o bien una solución buscando problema, o bien una optimización local que ignora consecuencias sistémicas.

— ¿Cuáles crees que son los principales riesgos o antipatrones que se han detectado durante estos últimos años y que pueden hacer mucho daño de no tomar medidas?

Esta es quizás la pregunta más importante que podemos hacernos ahora mismo, porque estamos en un momento donde la velocidad de adopción supera claramente la velocidad de comprensión de las consecuencias.

El primero y más peligroso: la complacencia con el código generado por IA. No es nuevo —lo llevamos poniendo en Hold desde hace varios volúmenes— pero se está agravando. Lo que antes era un desarrollador revisando sugerencias de Copilot, ahora es un agente modificando decenas de ficheros en paralelo mientras el desarrollador supervisa a alto nivel. El riesgo de automation bias se multiplica exponencialmente. Y lo que obtienes no es código malo —es código que *funciona* pero que nadie entiende del todo. Eso nos lleva directamente al segundo antipatrón.

La deuda cognitiva del codebase. Este término lo introdujimos recientemente en Caution y creo que es uno de los conceptos más importantes que hemos articulado en años. No es deuda técnica clásica —es la brecha entre lo que el sistema hace y lo que el equipo *comprende* que hace. Con agentes generando cambios a alta velocidad, esa brecha se ensancha silenciosamente hasta que un día un cambio pequeño rompe algo inesperado, y nadie sabe por qué.

Tercero: MCP by default. Lo pusimos en Caution explícitamente. El patrón que vemos es: "tengo una API, la convierto en MCP server, listo." Sin pensar en el modelo de seguridad, sin analizar los flujos tóxicos, sin preguntarse si realmente necesitas MCP o si un CLI script resuelve el problema igual de bien. La conversión naïve de APIs a MCP es una superficie de ataque que la mayoría de organizaciones no está gestionando.

Y cuarto, uno que me preocupa especialmente a nivel organizacional: el shadow IT acelerado por IA. No-coders construyendo workflows agenticos que procesan datos sensibles, sin governance, sin audit trail, sin que IT lo sepa. La diferencia con el shadow IT clásico es la escala y la velocidad. Lo que antes tardaba meses en convertirse en un problema crítico, ahora tarda semanas.

El patrón común en todos estos antipatrones es el mismo: **la IA amplifica la velocidad antes de que la organización haya construido los mecanismos de control necesarios.** Y cuando algo falla, falla rápido y a escala.

— Me parecen muy interesante estos puntos sobre todo cuando se está hablando de la desaparición del rol de programador pero sin embargo, detectáis muchos riesgos asociados con la generación automática de la mayor parte del código, sin revisar, aumentando la deuda técnica. ¿Cómo afecta esto realmente al rol de desarrollador? ¿Acabará desapareciendo este rol o tendrá mayor importancia de la que tiene?

La narrativa de "el programador va a desaparecer" es, francamente, una de las más irresponsables que circulan ahora mismo. Y lo digo con convicción porque los datos que vemos en el Radar apuntan exactamente en la dirección contraria.

Lo que estamos observando es una **transformación del rol, no su extinción**. Y esa transformación lo hace, si cabe, más crítico y más difícil de ejercer bien.

Piénsalo así: cuando un agente genera cientos de líneas de código en segundos, ¿quién decide si esa solución es la correcta arquitectónicamente? ¿Quién detecta que el agente ha resuelto el problema con treinta condicionales anidados en lugar de aplicar el patrón Strategy? ¿Quién entiende el acoplamiento oculto que ese cambio introduce? Eso requiere más criterio técnico, no menos.

Lo que sí desaparece —y esto es importante— es el desarrollador que solo sabe escribir código mecánico, sin entender diseño, sin entender sistemas, sin entender trade-offs. Ese perfil ya estaba en declive antes de la IA. Lo que emerge es un desarrollador que actúa más como **ingeniero de sistemas y director técnico de agentes**: alguien que sabe especificar intención con precisión, que puede evaluar output a alto nivel, que mantiene el modelo mental del sistema completo.

Y aquí está la paradoja que nos preocupa profundamente: **la IA está bajando la barrera de entrada al mismo tiempo que eleva el listón de lo que significa hacerlo bien**. Eso crea una brecha enorme. Tendrás más gente generando código, pero menos gente capaz de entender lo que se ha generado. Y esa brecha es exactamente lo que alimenta la deuda cognitiva del codebase que mencionábamos antes.

Lo que el Radar refuerza consistentemente —pair programming, TDD, fitness functions, code review riguroso— no son reliquias del pasado. Son los mecanismos que permiten que un equipo mantenga comprensión colectiva del sistema mientras la velocidad de generación se dispara.

El rol del desarrollador no desaparece. **Se bifurca**: los que entienden profundamente los sistemas se vuelven exponencialmente más valiosos; los que delegaron ese entendimiento a la IA se vuelven prescindibles. La IA no elimina la necesidad de criterio técnico. La hace más cara cuando falta.

— Por otro lado, has comentado también "MCP by default" y me parece muy interesante ya que hubo o hay un boom alrededor de este concepto, naciendo MCPs de debajo de las piedras. ¿Que riesgos reales tiene abusar de este concepto o usarlo cuando realmente no se necesita?

El boom de MCP es un caso de libro de lo que pasa cuando una tecnología resuelve un problema real pero la industria la sobreextiende hasta convertirla en un martillo que busca clavos.

MCP tiene valor genuino. Cuando necesitas contratos de herramientas estructurados, límites de autenticación OAuth y acceso multi-tenant gobernado, es la solución correcta. El problema es que estamos viendo equipos y vendors usarlo como capa de integración por defecto, incluso cuando un CLI bien diseñado con `--help` decente y respuestas JSON estructuradas daría exactamente lo mismo sin el overhead del protocolo.

El primer riesgo real es lo que llamamos el "abstraction tax". Cada capa de protocolo entre un agente y una API pierde fidelidad. Para APIs simples eso es tolerable; para APIs complejas, esas pérdidas se acumulan. El agente recibe una versión empobrecida de la interfaz original, y eso se traduce en comportamiento degradado o en prompts cada vez más elaborados para compensar.

El segundo riesgo es de seguridad, y este es el que más nos preocupa. Las APIs internas típicamente exponen datos sensibles o permiten operaciones destructivas. Cuando un desarrollador humano las consume, hay arquitectura, code reviews y contexto organizacional que mitigan esos riesgos. Cuando haces una conversión naïve de API a MCP y se la entregas a un agente autónomo, eliminas esas salvaguardas. No hay forma determinista de prevenir que el agente abuse de esos endpoints. Y aquí aparece de nuevo el lethal trifecta: datos privados, contenido no confiable, acción externa. La mayoría de los MCP servers útiles lo cumplen por defecto.

El tercer riesgo es de complejidad operacional innecesaria. MCP introduce overhead de mantenimiento, versionado y gobernanza. Si lo adoptas sin necesitarlo, estás pagando ese coste sin recibir el beneficio.

Lo que recomendamos es una pregunta previa obligatoria: ¿tu sistema realmente requiere interoperabilidad a nivel de protocolo? Si la respuesta no es un "sí" claro, un CLI bien diseñado o una función directa es mejor opción. MCP tiene su lugar, pero ese lugar no es "en todas partes".

— Tambien me gustaría hablar del último riesto que has apuntado "shadow IT acelerado por IA". ¿Cómo afecta realmente esto a una organizacion? ¿Qué implicaciones tiene en todos los sentidos, incluyendo los costes o la seguridad?

Este es uno de los riesgos que más nos preocupa precisamente porque no parece peligroso hasta que ya es demasiado tarde.

Lo que estamos observando es una versión turbocargada de algo que ya existía. Las hojas de cálculo que "silenciosamente gobiernan el negocio" llevan décadas siendo un problema. Pero ahora, con herramientas como Claude Cowork, n8n con integración de APIs de IA, o simplemente un product manager con acceso a un coding agent, el salto de "automatización informal" a "sistema crítico no gobernado" se produce en días, no en meses.

El impacto organizacional es multidimensional.

En seguridad, el problema central es que estos sistemas se construyen sin threat modeling, sin revisión de permisos, sin gestión de secretos. Un workflow de n8n que conecta Slack con un CRM vía OpenAI puede estar exfiltrando datos de clientes sin que nadie lo sepa. Y aquí vuelve el lethal trifecta: datos privados, contenido no confiable, acción externa. Ese workflow lo cumple por defecto.

En costes, el problema es la invisibilidad. Los equipos de finanzas no ven las llamadas a APIs de modelos que se están acumulando en tarjetas corporativas o cuentas cloud personales. Hemos visto organizaciones con docenas de integraciones paralelas haciendo llamadas redundantes a GPT-4 porque nadie sabía que el equipo de al lado ya había resuelto el mismo problema.

En deuda técnica y gobernanza, lo que empieza como un prototipo desechable se convierte en infraestructura crítica. Nadie lo documenta, nadie lo testea, y cuando el creador original se va, nadie sabe cómo funciona. Es exactamente el patrón de las macros de Excel, pero con capacidad de ejecutar acciones en sistemas externos.

Lo que recomendamos no es prohibir, sino canalizar. Sandboxes internos instrumentados donde los no-desarrolladores puedan experimentar con visibilidad. Un catálogo compartido de workflows existentes para evitar duplicación. Y criterios claros para determinar cuándo un prototipo necesita convertirse en una aplicación de producción con ingeniería real detrás.

La IA democratiza la construcción de software. Eso es genuinamente valioso. Pero democratizar sin gobernar es simplemente acelerar el caos.

— Muchas gracias por toda esta informacion tan valiosa que nos estás dando. Para terminar la entrevista y concluir, en pocas palabras, ¿qué crees que tenemos que tener en cuenta tanto los profesionales del sector como las empresas para sobrevivir a este tsunami que ya está aquí?

Lo que hemos aprendido de todos estos ciclos del Radar se puede condensar en algo que puede sonar paradójico: **la mejor forma de adaptarse a la velocidad de la IA es invertir en lo que no cambia.**

Los fundamentos de ingeniería —código limpio, testing, diseño deliberado, feedback loops cortos— no son nostalgia. Son exactamente lo que permite que la IA amplifique valor en lugar de amplificar caos. Lo hemos visto una y otra vez.

Para los **profesionales**, el mensaje es claro: no compitas con la IA en velocidad de generación de código. Compite en juicio. En capacidad de hacer las preguntas correctas, de detectar cuándo un agente está tomando un camino equivocado, de entender el sistema completo. Eso es lo que la IA no puede reemplazar todavía, y es lo que más valor tiene ahora mismo. Invierte en entender profundamente los sistemas que construyes, no solo en construirlos más rápido.

Para las **empresas**, el error más costoso que podemos ver es tratar la IA como una iniciativa de productividad individual en lugar de una transformación sistémica. Si introduces agentes sobre procesos rotos, sobre codebases sin disciplina, sobre equipos sin cultura de feedback, simplemente vas a llegar al desastre más rápido.

Y hay algo que me parece crítico para ambos: **aprender a distinguir entre velocidad y progreso.** Más PRs, más líneas de código, más workflows automatizados no son progreso si el rework rate sube, si nadie entiende lo que se ha construido, si la deuda cognitiva se acumula silenciosamente.

El tsunami ya está aquí, sí. Pero los que van a sobrevivir no son los que nadan más rápido. Son los que saben cuándo nadar, cuándo parar y cuándo cambiar de dirección.