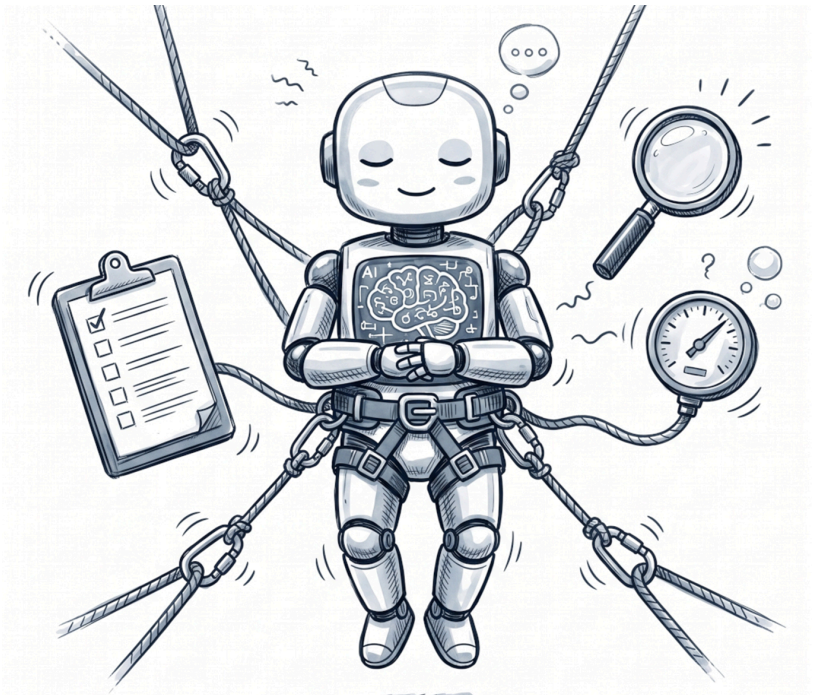
Harness engineering



State-of-the-art

Mayo 2026

Harness engineering

State-of-the-art

Mayo 2026

© Autor: Juan Palacio Asistente de IA: Anthropic Opus 7

Edita: Scrum Manager (scrummanager.com)

Los contenidos de esta guía didáctica están sujetos a revisión y actualización.

Consulta siempre la versión más reciente en scrummanager.com.

Tabla de contenidos

Sobre esta guía

Esta guía te lleva desde los fundamentos del Harness engineering hasta los patrones que la disciplina ha consolidado en mayo de 2026. Está dirigida a **cualquier profesional que quiera adoptar agentes de IA con rigor en su trabajo, sea cual sea su disciplina:** programación, análisis, consultoría, investigación, escritura profesional, asesoría legal, periodismo, gestión, enseñanza.

La premisa que la atraviesa es simple: cuando el resultado de un agente oscila entre lo deslumbrante y lo desesperante, esa oscilación no es un defecto del modelo, sino el síntoma inequívoco de que falta arnés alrededor. El Harness engineering es el **conjunto de prácticas, decisiones y artefactos con los que recubrimos un modelo para que se comporte de forma fiable, verificable y útil.**

La guía cubre exclusivamente **el arnés externo:** el que tú, como usuario de una plataforma de IA, construyes sobre la infraestructura que el fabricante ya te da hecha. El arnés interno (los mecanismos que los productos usan internamente para gestionar contexto, memoria, herramientas, búsqueda, orquestación, etc.) existe, es fascinante, y queda fuera del alcance de esta guía. Lo bueno es que el 99% del valor que extraerás de los agentes en tu trabajo diario proviene de hacer bien el arnés externo, no de comprender el interno.

Cómo leerla

- **Si eres nuevo en agentes:** léela en orden, sin saltarte la Parte I. Los fundamentos no son adorno: cuando algo falle en la práctica (y fallará), volver a ellos te ahorrará horas de frustración.
- **Si ya usas agentes "a ojo":** puedes saltar a la Parte III para entender lo que estás haciendo bien sin saberlo, y lo que estás haciendo mal sin saberlo.
- **Si vienes a buscar artefactos o checklists:** ve directo a los apéndices. Pero vuelve a la Parte III después; las plantillas sin contexto se aplican mal.

Convenciones

Las áreas que cambian con frecuencia (nombres exactos de modelos, características experimentales, planes y límites de productos) llevarán este recuadro:

> ⚠️ **Consulta la documentación oficial**
>
> Este detalle cambia con cada actualización. Verifica el estado actual antes de basar decisiones operativas en él.

Cuando un párrafo describe **cómo se comporta un agente bien arnesado** frente a uno que no lo está, aparecerá el contraste ✅ **Con arnés** / ❌ **Sin arnés**, porque ese contraste es el núcleo pedagógico de toda la guía.

Una nota terminológica: usaremos "agente" tanto para hablar del sistema completo (modelo + arnés) como del comportamiento que tiene un asistente cuando se le configura para actuar con cierta autonomía. Cuando importe distinguir entre "el modelo" (lo que entrena el fabricante) y "el agente" (modelo + arnés), se hará explícito.

Formación y acreditación

Los contenidos de esta guía están disponibles en Skill Arena, la plataforma de autoformación y entrenamiento de Scrum Manager. Allí puedes realizar ejercicios de práctica, refuerzo y evaluación, y también obtener un diploma acreditativo del conocimiento adquirido. Como la aplicación de SDD con IA evoluciona rápidamente, Skill Arena mantiene el contenido actualizado, por lo que es la referencia recomendada para consultar la versión más reciente.

PARTE 1. Fundamentos

1. El despertar de la agencia: por qué el modelo no basta

Existe una distinción crítica entre **inteligencia y agencia.** La primera es una propiedad entrenada del modelo: su capacidad de razonar sobre el lenguaje, conectar conceptos, generar texto coherente, sintetizar información, planificar, intuir patrones. La segunda es su capacidad de **actuar sobre el mundo:** leer documentos, ejecutar acciones, modificar artefactos, llamar a herramientas externas, mantener coherencia entre sesiones, producir entregables verificables.

Un modelo sin entorno operativo es como un cerebro flotando en un tarro: brillante, sí, pero incapaz de hacer nada útil. Razona, especula, opina, pero no puede comprobar nada. Para que esa inteligencia se convierta en un **agente útil para trabajo profesional**, necesita un cuerpo. Ese cuerpo es el **arnés**.

La ecuación canónica, popularizada por **Birgitta Böckeler (Thoughtworks)** y adoptada por toda la industria desde principios de 2026, es:

Agente = Modelo + Arnés

Donde el arnés engloba **todo lo que no es el modelo:** las instrucciones que lee al empezar, los documentos a los que tiene acceso, las herramientas que puede invocar, las políticas de permisos que defines para él, los bucles de orquestación, la memoria que persiste entre sesiones, los sensores de verificación y las interfaces de control humano.

La intuición clave es ésta:

- ❌ **Sin arnés:** un modelo de frontera al que se le encarga una tarea sustantiva se comporta como un conductor brillante en un coche sin frenos ni dirección. Empieza rápido, dice cosas inteligentes, y termina estrellado contra una pared porque no había forma de corregir el curso a tiempo. En producción profesional: resultados pulidos, articulados y persuasivos, pero plagados de fuentes inventadas, cifras aproximadas presentadas como exactas, decisiones que no sobreviven a una revisión seria, o código que parece funcionar y rompe cosas que no se ven.
- ✅ **Con arnés:** el mismo modelo, con guías que orientan su acción antes de actuar y sensores que detectan errores después de actuar, completa tareas reales con calidad de producción. Es más lento turno a turno, pero entrega valor real y aguanta la revisión de un experto humano.

La promesa fundamental del Harness Engineering es por tanto sencilla y agresiva: **el modelo es un commodity; el arnés es la ventaja competitiva.** Cuando dos profesionales

usan el mismo modelo y uno entrega trabajo riguroso mientras el otro entrega humo articulado, la diferencia siempre está en el arnés.

2. El marco de Böckeler: guías y sensores

En abril de 2026, Birgitta Böckeler publicó el artículo Harness engineering for coding agent users en martinfowler.com. Aunque el título habla de codificación, el marco conceptual aplica directamente a cualquier trabajo asistido por agentes, y se ha convertido en el lingua franca de la disciplina.

Böckeler descompone el arnés externo en dos tipos de control que actúan en momentos distintos del ciclo de trabajo del agente.

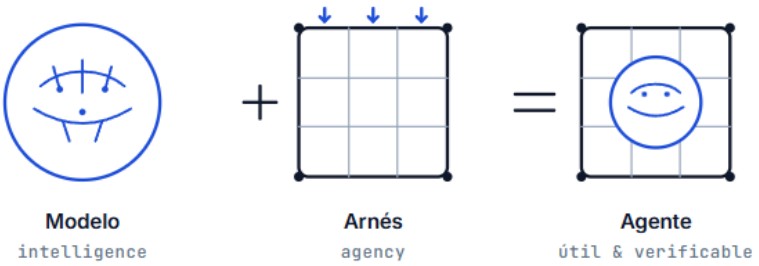


Imagen 1. *Ecuación canónica*

2.1. Guías (controles feedforward, antes del acto)

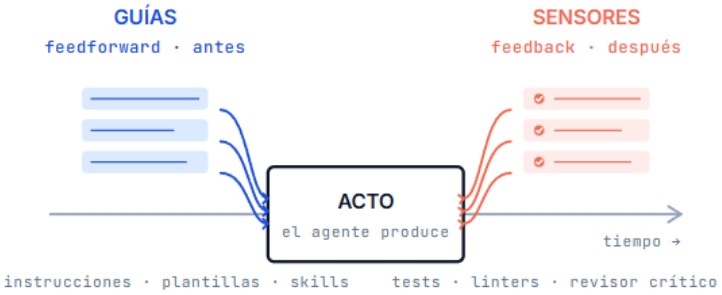


Imagen 2. *Guías y sensores*

Las **guías** anticipan el comportamiento del agente e intentan dirigirlo **antes de actuar.** Aumentan la probabilidad de que el agente acierte a la primera. Son lo que el agente "lee" al empezar su trabajo y lleva consigo mientras opera.

Ejemplos típicos:

- Archivos de instrucciones que describen el proyecto, sus convenciones y sus reglas.
- Documentación de arquitectura, de método o de proceso.

- Glosarios terminológicos que fijan vocabulario.
- Guías de estilo (de código, editoriales, de citación).
- Plantillas estructurales para los entregables (la forma canónica de un módulo, un informe, un memorando, una pieza).
- Corpus curados de referencia (aplicaciones modelo, bibliografías acotadas, ejemplos canónicos).
- "Skills" reutilizables que el agente puede invocar cuando el contexto lo pide.

2.2. Sensores (controles feedback, después del acto)

Los **sensores** observan al agente **después de que ha producido algo** y le permiten autocorregirse. Lo más potente de los buenos sensores es que producen señales **accionables y optimizadas para que las consuma otra IA:** en lugar de fallar con un mensaje críptico, devuelven algo como "la afirmación X falla la verificación Y; en concreto, el valor esperado era A y el observado es B; la causa probable está en Z; corrige así o así".

Ejemplos típicos:

- Suites de verificaciones automáticas (linters, tests, type checkers, comprobadores de coherencia).
- Listas de comprobación pre-entrega (¿está todo lo requerido? ¿están las afirmaciones soportadas? ¿hay coherencia interna?).
- Fact-checking asistido por un segundo agente.
- Verificación de citas o referencias contra fuentes reales.
- Análisis estructural o estilístico automático (análisis de complejidad, métricas de legibilidad, detección de duplicados).
- Revisores adversariales: un segundo agente cuyo único trabajo es poner pegas al primero.
- Verificaciones end-to-end (un test que ejerce el sistema completo; una revisión humana en la última milla).

2.3. Por qué hacen falta los dos

Esta es la lección que más cuesta interiorizar:

- **Solo guías:** el agente codifica las reglas pero nunca se entera de si su producción las cumple. Repite errores estructuralmente válidos pero funcionalmente rotos. Las alucinaciones que parecen rigurosas son el ejemplo paradigmático.
- **Solo sensores:** el agente recibe feedback pero sin contexto sobre qué se esperaba. Da bandazos buscando "qué pasa la verificación", a veces sobrecorrigiendo, otras infrarrespondiendo, agotando tiempo y recursos.
- **Guías + sensores:** el agente sabe a dónde va, intenta llegar bien a la primera, y cuando se desvía se autocorrige porque tiene señales claras de qué falló.

2.4. Computacional vs. Inferencial

Cualquier control —guía o sensor— puede ser de dos tipos según cómo se ejecuta:

Tipo	Quién lo ejecuta	Velocidad	Coste	Fiabilidad
Computacional	Una herramienta determinista (un linter, un test, un script, una búsqueda en una base de datos)	Rápido (ms-s)	Mínimo	Total dentro de su alcance
Inferencial	Otro modelo de lenguaje (otro agente como juez o evaluador, LLM-as-judge)	Lento (s-min)	Alto	Probabilística

La regla pragmática es simple: **siempre que puedas resolver algo con un control computacional, no uses uno inferencial.** Los controles computacionales son rápidos, baratos y no mienten. Los inferenciales son potentes para semántica (¿este resultado está sobreingeniado? ¿este texto matiza correctamente?) pero caros y probabilísticos.

Imagen 3. *Computacional vs inferencial*

Un arnés maduro tiene los dos tipos. Un arnés inmaduro tiene solo controles inferenciales porque parece más sofisticado, y termina costando una fortuna sin entregar fiabilidad.

3. Los círculos concéntricos: arnés interno vs arnés externo

Una confusión frecuente en la literatura: hay (al menos) dos cosas que se llaman "arnés" según el contexto. Böckeler las dibuja como **círculos concéntricos:**

Imagen 4. *Círculos concéntricos*

- **El modelo:** lo que entrenan los laboratorios de IA. Tú no tocas esto.
- **El arnés interno:** lo construyen los fabricantes alrededor del modelo. Incluye la interfaz de chat, el sistema de proyectos, los conectores nativos, la memoria, la búsqueda integrada, los modos especializados, el bucle de orquestación, los ejecutores de herramientas, los sistemas de permisos. Tú usas este arnés, pero no lo construyes.
- **El arnés externo:** lo construyes tú alrededor del arnés interno que te dan hecho. Es lo que escribes en los archivos de instrucciones del proyecto, los documentos que pones a su disposición, las habilidades especializadas que defines, los conectores que enchufas, las verificaciones y rutinas que aplicas, las plantillas que diseñas.

⚠️ Aclaración importante

Esta guía cubre **solo el arnés externo.** Lo que las plataformas hacen por dentro para gestionar contexto, dividir tareas o invocar herramientas es un mundo apasionante pero queda fuera del alcance. La buena noticia: el 99% del valor que extraerás de los agentes en tu día a día viene de hacer bien el arnés externo, no de reinventar el interno.

4. Las tres dimensiones de regulación

Böckeler propone que el arnés actúa como un **gobernador cibernético** que regula la producción del agente hacia un estado deseado. Pero "el estado deseado" no es una sola dimensión: son al menos tres.

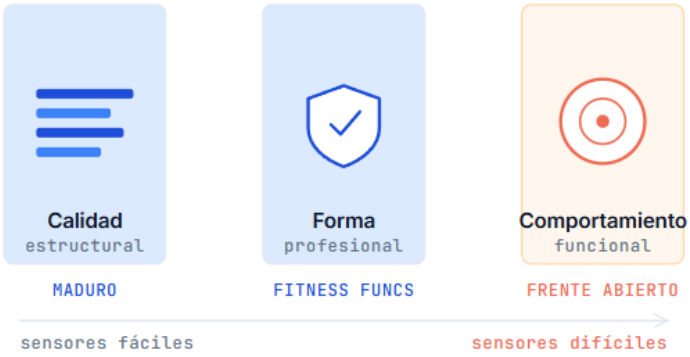


Imagen 5. *Dimensiones de regulación*

4.1. Calidad estructural (mantenibilidad)

Regula la **calidad interna** del producto: legibilidad, estructura, coherencia, consistencia, ausencia de redundancias, integridad de las partes. Es la dimensión **más madura** porque heredamos décadas de tooling y manuales de estilo. Los sensores aquí pueden ser:

- **Computacionales:** verificación de consistencia, búsqueda de patrones prohibidos, análisis de complejidad o legibilidad, detección de duplicados, verificación de que todas las secciones requeridas existen, comprobación de que las referencias se resuelven.
- **Inferenciales:** revisión por un agente, evaluación de tono, detección de soluciones semánticamente redundantes.

4.2. Forma física profesional (estándares del oficio)

Regula estándares transversales del trabajo: rigor metodológico, neutralidad, gestión de la incertidumbre, transparencia sobre fuentes y supuestos, observabilidad, seguridad, cumplimiento normativo si aplica. Es la idea clásica de ***fitness functions*** aplicada al trabajo agéntico.

Ejemplos:

- Una habilidad que verifica que cada afirmación cuantitativa va acompañada de su fuente, periodo y método.
- Una plantilla que obliga a incluir una sección de limitaciones o riesgos en cualquier entregable.
- Tests de rendimiento o de carga que retroalimentan al agente sobre si su trabajo mejoró o degradó propiedades transversales.
- Verificaciones de que la evidencia contradictoria se expone, no se esconde.

4.3. Comportamiento funcional (¿hace lo que debía?)

Aquí está el elefante en la habitación. ¿Cómo verificas que el entregable realmente resuelve la pregunta del cliente? ¿Que el código realmente hace lo que necesitas? ¿Que la conclusión es defendible y las recomendaciones son ejecutables?

Hoy, la mayoría de los equipos hace esto:

- **Antes:** una especificación funcional (desde un prompt corto hasta un dossier multi-archivo).
- **Después:** confiar en una suite de verificaciones que a menudo genera el propio agente, complementada con revisión humana experta.

Eso pone demasiada fe en verificaciones generadas por la misma IA que produjo el trabajo. Hay patrones emergentes —los ***approved fixtures*** (que verás en §15) y los **revisores adversariales**— que prometen, pero estamos aún lejos de una solución general.

La consecuencia práctica: cuanto más crítica sea la **corrección sustantiva** de un entregable, **menos autonomía debes dar al agente** y más rol humano debe haber. El behaviour harness es la frontera abierta.

5. Casos de estudio fundacionales

Dos publicaciones de 2025-2026 son lectura obligatoria para entender por qué este campo está donde está.

5.1. OpenAI: el experimento del millón de líneas

En febrero de 2026, **Ryan Lopopolo** publicó *Harness engineering: leveraging Codex in an agent-first world* en el blog de ingeniería de OpenAI. Documentó cinco meses de un experimento interno en el que un equipo pequeño construyó un producto completo asistido por agentes:

- 3 ingenieros iniciales (luego 7).
- Aproximadamente **1 millón de líneas de código** generadas.
- Aproximadamente **1.500 pull requests fusionados.**
- **Cero líneas escritas a mano.**
- Aproximadamente **una décima parte del tiempo** que habría llevado a un equipo humano tradicional.

El principio operativo del equipo era explícito: "humanos diseñan entornos, especifican intención y construyen bucles de feedback; agentes producen".

Lo más interesante: el progreso inicial fue lento. No porque el agente fuera incapaz, sino porque el entorno estaba sub-especificado. Cuando algo fallaba, la

respuesta nunca era "esfuérzate más, agente"; era siempre: "¿qué capacidad falta, y cómo la hacemos a la vez legible y verificable para el agente?".

Los pilares de su arnés (generalizables a cualquier disciplina):

1. **Instrucciones jerárquicas y legibles** por máquina que orientan al agente desde lo más general a lo más específico.
2. **Entornos reproducibles** con un único punto de entrada que pone el contexto en estado conocido, y aislamiento por workspace para evitar contaminación cruzada entre tareas.
3. **Invariantes mecánicos** que hacen cumplir reglas en cada paso (fronteras estructurales, formato, validaciones).
4. **Estándares estrictos:** lo que parecería pedante en un flujo humano-primero, se vuelve crítico cuando el lector es un agente. Lo que para un humano es ruido, para un agente es señal.
5. **Limpieza continua del entorno:** tareas programadas que detectan y corrigen la entropía que el trabajo continuado genera (referencias huérfanas, duplicados, deriva entre módulos).

Traducido a cualquier disciplina del conocimiento: si tienes un equipo trabajando con agentes, **invertir tiempo en construir manuales metodológicos, glosarios, plantillas y verificaciones produce más output a medio plazo que pedirle al equipo que aproveche mejor su tiempo.**

5.2. Anthropic: arneses para tareas largas

En noviembre de 2025, Anthropic publicó *Effective harnesses for long-running agents*, ampliado en marzo de 2026 con *Harness Design for Long-Running Application Development.*

El problema que abordaban: aun con el mejor modelo y la mejor interfaz, los agentes fallaban en tareas largas porque:

1. Intentaban resolver todo en un único turno y declarar victoria prematuramente.
2. Olvidaban entre sesiones lo decidido en sesiones anteriores.
3. Sub-verificaban porque marcaban entregables como completos sin evidencia real.

Su solución, hoy considerada **el patrón canónico para tareas largas**, es directamente aplicable a programación, investigación, análisis y cualquier trabajo del conocimiento:

- ***Initializer agent:*** una sesión inicial que se ejecuta una sola vez. Lee el brief o prompt del usuario, lo expande en una **lista estructurada de entregables o features** (cada uno con un estado `pendiente/hecho` y criterios de aceptación claros), monta el espacio de trabajo (estructura de archivos, plantillas, scripts de bootstrap, fuentes iniciales), y prepara las **rutinas de verificación**.
- ***Working agent* (o *coding agent*):** se despierta una y otra vez. Cada sesión:

1. **Ejecuta el bootstrap** o se "re-fundamenta" leyendo los archivos del proyecto.
2. **Lee el registro de progreso** para entender qué pasó en la sesión anterior.
3. **Elige una entrada** de la lista con estado pendiente.
4. **La produce, la verifica** contra sus criterios, somete su trabajo a los sensores definidos.
5. **Actualiza el registro de progreso** con notas para el próximo turno.
6. **Guarda**, hace checkpoint y termina.

La clave: la **memoria persistente vive en archivos del proyecto, no en la conversación ni en el contexto del modelo.** Cada nueva sesión "se re-fundamenta" leyendo esos archivos. Es exactamente lo que un profesional humano haría al volver el lunes después de un fin de semana sin ver el proyecto.

En abril de 2026, Anthropic dio un paso más con la **arquitectura *Brain/Hands/Session*:**

- **Brain:** el modelo más el bucle del arnés que lo invoca.
- **Hands:** entornos sandboxed efímeros donde las herramientas se ejecutan.
- **Session:** un log append-only de cada pensamiento, llamada a herramienta y observación.

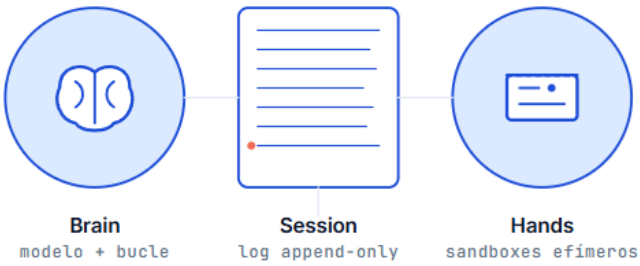


Imagen 6. *Brain / hands / session – Anthropic 2026*

La motivación: cada componente del arnés codifica una asunción sobre lo que el modelo no puede hacer solo. Acoplar las tres cosas significa que cuando una asunción envejece (el modelo nuevo ya puede hacer algo que antes requería un componente del arnés) hay que cambiar todo el sistema. Desacoplarlas hace al arnés *stateless*, a los *sandboxes* ganado en lugar de mascotas, y un crash del Brain no pierde la sesión.

📝Importante

Estas publicaciones son lectura obligatoria. Los enlaces están en el Anexo B. Aunque su contexto original es ingeniería de software, la sustancia conceptual aplica a cualquier trabajo del conocimiento.

6. Conceptos transversales que vertebran todo

Antes de entrar en práctica, hay seis ideas que aparecen una y otra vez en el resto de la guía. Conviene tenerlas claras.

6.1. WIP = 1

Work-in-Progress igual a uno: cada sesión del agente trabaja en **una sola tarea o entregable** y no avanza al siguiente hasta que el actual está verificado.

¿Por qué? Porque el contexto del modelo es finito y, peor, su parte **utilizable** es mucho menor que la anunciada. El benchmark **RULER** de NVIDIA encontró que el contexto efectivo real es típicamente del **50-65%** de la ventana anunciada. Y el efecto ***Lost in the Middle*** (Liu et al., 2023) demuestra que los modelos pierden eficacia procesando instrucciones situadas en el centro de contextos largos.

Imagen 7. *Lost in the middle*

WIP = 1 protege el presupuesto de contexto y reduce la carga cognitiva del agente. Datos empíricos de implementaciones reales muestran tasas de éxito notablemente más altas bajo WIP = 1 (del orden del 80-100%) frente a sesiones sin restricciones (del orden del 20%).

Imagen 8. *Una tarea por sesión, hasta verificar*

En la práctica: no intentes que el agente haga tres tareas relacionadas en una sola sesión. Cierra y abre una nueva. Sí, tendrás que "re-fundamentarlo" leyendo los archivos

del proyecto, pero ese coste es minúsculo comparado con el de un agente confundido en mitad de un contexto saturado.

6.2. La brecha de verificación

Existe una distancia peligrosa entre el "sentimiento de confianza" que produce un resultado bien presentado y su corrección real. Los agentes son **optimistas estructurales:** tienden a declarar entregables "completos" cuando solo son "plausibles".

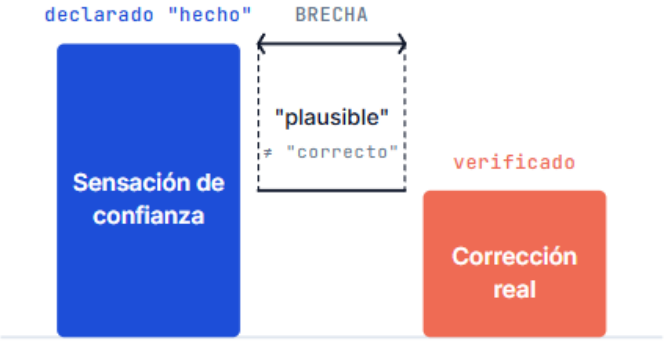


Imagen 9. *Brecha de verificación*

La regla de oro:

"Hecho" no es un adjetivo; es un resultado de verificaciones.

Esto se materializa en el patrón ***pass-state gating*** (o pase verificado): el arnés prohíbe al agente avanzar al siguiente entregable hasta que el actual haya superado un conjunto explícito de verificaciones (tests verdes, *fact-check* pasado, citas verificadas, coherencia interna OK, todas las secciones requeridas presentes, *build* OK, *smoke test* OK, lo que sea pertinente a tu tipo de trabajo).

6.3. El *cold-start test*

La métrica más poderosa para evaluar la calidad de tu arnés: **abre una sesión totalmente nueva, sin historial, sin contexto previo.** ¿Puede el agente, leyendo solo los archivos del proyecto, identificar qué es el sistema, cuál es el progreso actual y qué falta por hacer?

- Si la respuesta es **sí en menos de 3 minutos:** tu arnés es de élite.
- Si la respuesta es **sí pero tarda 20 minutos:** tu arnés está documentando demasiado o lo equivocado.
- **Si la respuesta es no:** tu conocimiento está en la cabeza de un humano, no en el proyecto, y tu arnés está roto.

Imagen 10. *Sesión vacía → trabajo productivo*

Para proyectos largos —que viven semanas o meses con sesiones intercaladas— este indicador es especialmente importante. Sin *cold-start passing*, cada sesión empieza con 20-30 minutos de "ponerse al día" del agente.

6.4. Lo que no está en el proyecto no existe para el agente

Corolario del cold-start test. El conocimiento tácito (lo que se habló en una reunión, lo que está en una conversación de chat informal, los porqués que solo conoce el senior, los gotchas que nadie ha escrito) es **invisible para el agente**.

El trabajo del harness engineer es codificar ese conocimiento tácito en artefactos explícitos: documentos de brief, registros de decisiones, glosarios, plantillas, comentarios en código, tests de regresión que documentan casos límite. Si se decidió que "no tocamos X" o "siempre hacemos Y", ese acuerdo tiene que estar en algún archivo del proyecto, no solo en tu memoria.

6.5. Escritura atómica y versionado

Cuando un agente edita un artefacto, hay un momento en el que el artefacto está **parcialmente reescrito**. Si el proceso muere ahí, o si el agente se equivoca a mitad de camino, dejas el proyecto en estado inconsistente.

En el mundo del software, el patrón canónico es **`tmpfile + rename`:** el agente escribe a un archivo temporal, verifica que el contenido es correcto, y solo entonces hace un rename atómico al destino. Las plataformas profesionales lo hacen por ti en su arnés interno; tú no tienes que implementarlo, pero debes verificar que tus hooks o automatismos personalizados no lo rompan.

El equivalente para trabajo de conocimiento es:

- **Versionar entregables.** Cada vez que una sección o entregable pasa todas las verificaciones, guarda un snapshot con fecha. Esto te permite volver atrás si descubres que una decisión posterior rompió algo de antes.
- **No sobrescribir sin red de seguridad.** Si el agente va a reescribir una sección completa, guarda primero la versión anterior. Herramientas con historial (Git para

código, Google Docs/Notion para texto, cualquier sistema de carpetas con fechas) sirven.

6.6. La Ley de Ashby

La Ley de la Variedad Requisita de W. Ross Ashby (cibernética, 1956): **un regulador debe tener al menos tanta variedad como el sistema que gobierna.**

Aplicado al arnés: si quieres regular efectivamente lo que produce un LLM (que puede producir casi cualquier cosa), tu arnés tendría que tener variedad casi infinita. **Imposible.**

¿La solución? Reducir la variedad del sistema. Si comprometes al agente con una topología concreta (una arquitectura conocida, un stack fijo, una plantilla cerrada de informe, un proceso metodológico definido, un corpus de fuentes acotado), reduces el espacio de cosas que puede producir, y hacer un arnés comprehensivo se vuelve posible.

Esto es lo que están haciendo las **plantillas de arnés** que veremos en §14: bundles pre-configurados de guías y sensores que atan al agente a una estructura predefinida para un tipo de proyecto o entregable concreto.

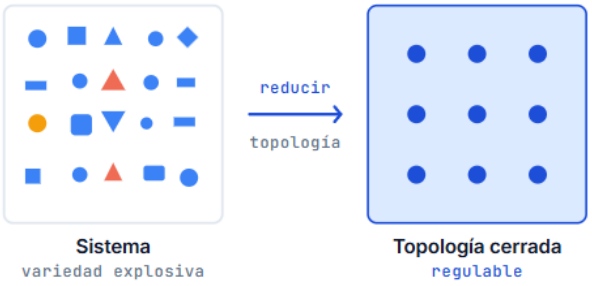


Imagen 11. *Variedad requisita*

PARTE 2. Preparación del entorno

Antes de tocar cualquier herramienta concreta, hay cosas que tienes que tener listas. Esta parte es independiente del producto que uses.

7. Pre-flight: espacio de trabajo, confidencialidad y materiales

7.1. Un espacio de trabajo claro, no una conversación suelta

El agente "necesita" un espacio donde vivir el proyecto. Tanto si trabajas con código como con texto, no es lo mismo una conversación suelta que un proyecto con vida propia.

Asegúrate de que para cada proyecto tienes:

- **Un contenedor dedicado** en la plataforma que uses (un Project, un repositorio, un workspace), no una conversación dispersa.
- **Una carpeta correspondiente** en tu sistema donde guardas los outputs y los inputs.
- **Un sistema de versionado de los entregables:** Git si trabajas con código, un sistema con historial (Google Docs, Notion, control de carpetas con fechas) si trabajas con documentos.

Una conversación suelta no es un proyecto. Es una mesa de trabajo provisional.

7.2. Confidencialidad y secretos

Los modelos comerciales tienen políticas explícitas sobre uso de datos, pero las condiciones cambian y los riesgos no son cero. Y un agente con permisos para ejecutar acciones tiene, en el peor caso, la misma capacidad que tu cuenta de usuario en cualquier sistema al que tenga acceso.

Reglas mínimas universales:

- **Nunca pegues credenciales, claves de acceso o secretos en el chat de un agente.** Aunque tu herramienta los maneje localmente, hay servicios conectados, hooks o subagentes que podrían leerlos y filtrarlos.
- **Nunca subas datos altamente confidenciales** (información personal sensible, secretos comerciales, datos bajo NDA estricto) a las versiones de consumidor de productos generales. Usa los planes empresariales o despliegues controlados vía API.
- **Anonimiza datos** antes de subirlos cuando puedas hacerlo sin perder utilidad.
- **Documenta en cada proyecto qué nivel de confidencialidad tiene**, y atente a las restricciones correspondientes.

- Prefiere **autenticación gestionada por la CLI o la plataforma** (que guarda credenciales con permisos restrictivos) sobre pegar secretos en archivos del proyecto.

Patrón canónico para secretos en proyectos de código:

```
proyecto/
├── .env                # contiene secretos REALES (en .gitignore)
├── .env.example             # plantilla con nombres pero sin valores (sí versionada)
└── instrucciones.md     # menciona "lee secretos desde .env" pero NUNCA los valores
```

> ⚠️ **Importante**
>
> Consulta las políticas vigentes del fabricante antes de subir cualquier información sensible, especialmente si manejas datos regulados.

7.3. Los materiales: el cimiento de cualquier trabajo

En programación los "materiales primos" son el código existente y las librerías; en investigación o análisis, son fuentes, datos y documentos; en escritura profesional, son briefs, transcripciones y referencias.

Antes de empezar, **acota y cura los materiales:**

- **Cura un corpus de referencia confiable** y ponlo al alcance del proyecto. Esto es radicalmente mejor que dejar al agente "buscar lo que encuentre".
- **Documenta qué tipo de materiales son aceptables y cuáles no** (qué librerías son admitidas, qué tipo de fuentes consideramos válidas, qué tipo de datos podemos usar).
- **Lleva un registro desde el principio:** dependencias declaradas, bibliografía citable, decisiones documentadas.

Un agente con materiales acotados y curados produce trabajo verificable. Un agente que "se va a buscar lo que necesite" produce trabajo plausible pero no verificable.

8. Diseñar un entorno agent-legible

Ryan Lopopolo lo llamó ***agent legibility***. La idea: un agente, como un colaborador nuevo que se incorpora un lunes por la mañana, debe poder **navegar el proyecto sin pedir indicaciones.**

Principios prácticos:

8.1. Estructura predecible

Si tu proyecto sigue las convenciones de su disciplina, el agente lo navega solo. Si tienes una estructura "creativa" o nombres crípticos (`borrador 2 v3 final BUENO.docx`, carpeta `temp/` con cosas importantes, archivos con nombres internos), el agente tarda más en orientarse y consume más recursos.

Tu instinto de organización es el mejor sensor. **Si un colega nuevo entendería de un vistazo qué hay en cada carpeta y qué hace cada archivo, el agente también.**

8.2. Fronteras explícitas

En un proyecto de software, eso significa una arquitectura clara con módulos, capas y reglas de dependencia. En un proyecto de conocimiento, una separación clara entre brief, metodología, fuentes, borradores, entregables y registros.

Un sensor que falle cuando alguien (o el agente) rompe una frontera (un módulo que importa de donde no debe, una sección que cita una fuente prohibida, un entregable que mezcla brief con conclusiones) es un control poderosísimo.

8.3. Tooling y convenciones estándar

Cuanto más estándar sea tu toolchain, tu plantilla o tu convención, mejor. El agente "conoce" los estándares de su disciplina; las cosas exóticas hay que documentárselas. Si usas un proceso custom, **documéntalo** en los archivos del proyecto.

9. El contrato de bootstrap

El contrato de bootstrap es el conjunto de pasos (idealmente automatizables) que pone el entorno en estado conocido. Su existencia es lo que permite el cold-start test.

En un proyecto de software, suele ser un script ejecutable e idempotente que:

1. Verifica versiones de dependencias.
2. Crea o activa entornos virtuales.
3. Instala lo necesario.
4. Inicializa bases de datos locales si las hay.
5. Lanza una verificación de sanidad (lint + tests rápidos).
6. Reporta "entorno listo".

Y un script complementario de **verificación de cierre** que actúa como *pass-state gate*: *lint, format check, type check, tests, build*. Devuelve 0 si todo está OK, distinto de 0 si no.

En un proyecto de conocimiento, el "bootstrap" puede ser una **rutina documentada:** leer el brief, leer la lista de entregables, leer el registro de progreso, leer la última versión de los archivos vivos, confirmar contexto antes de producir. Aunque no sea

un *script* automatizable, debe estar escrito de forma que cualquier sesión pueda ejecutarlo sin necesidad de preguntar.

Ambos —script o procedimiento— deben ser invocables tanto por humanos como por el agente. Y deben estar referenciados explícitamente en las instrucciones del proyecto.

10. Anatomía de un arnés mínimo viable

Un arnés externo mínimo viable —que sirva como cimiento sobre el cual añadir sofisticación— tiene **cinco piezas:**

Pieza	Para qué	Archivos típicos
Instrucciones	Guía maestra que lee el agente al empezar	Archivo de instrucciones del proyecto (instrucciones de proyecto, AGENTS.md, CLAUDE.md, manual del proyecto)
Estado	Memoria persistente entre sesiones	Lista de entregables o features con estado, registro de decisiones, registro de progreso, sistema de versiones
Verificación	Sensores que confirman corrección	Linter, tests, type checker, fact-checker, listas de comprobación, smoke tests
Alcance	Reglas que restringen qué hace el agente en cada sesión	Políticas declaradas en las instrucciones, WIP=1, pass-state gating
Ciclo de vida	Bootstrap, persistencia y limpieza del entorno	Scripts de bootstrap, devcontainers, plantillas, rutinas de cierre

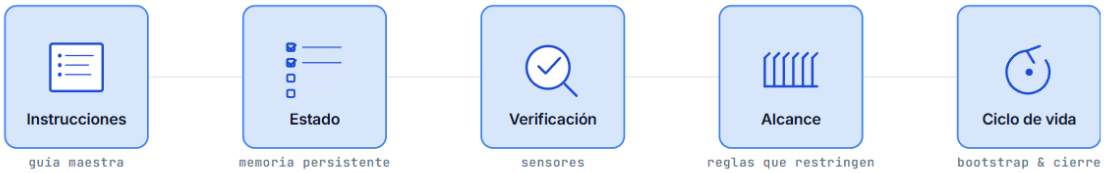


Imagen 12. *Arnés mínimo viable*

Si tienes estas cinco piezas, ya estás haciendo Harness Engineering. Lo que viene a partir de aquí es sofisticación: hooks, skills, subagentes, conectores, plantillas pre-empaquetadas. Todo eso amplifica el efecto del arnés básico, pero no lo sustituye.

PARTE 3. Patrones avanzados

A partir de aquí, conceptos que aplican igual sea cual sea la herramienta que uses. Son ideas, no comandos.

11. Diseñar guías de alta señal

Unas instrucciones de proyecto mediocres (un archivo inflado, genérico o desactualizado) **son peores que no tener instrucciones:** ocupan contexto, dispersan la atención del modelo y le hacen ignorar las reglas que sí importan.

11.1. Principio "Toolchain First"

Antes de escribir una regla, pregúntate: **¿no la captura ya otra parte del sistema?**

- "Usa 4 espacios de indentación" → si tu formatter lo enforza, **borra la regla.**
- "Cita en estilo APA" → si tienes una skill `citation-formatter` con APA configurado, **la regla está en otro sitio.** No la repitas.
- "Type hints obligatorios" → si tienes type-check estricto en CI, ya lo enforza. **Borra.**
- "Términos del glosario" → si tienes un glosario y un verificador, la regla está allí. **No la dupliques.**

La regla solo sobrevive si captura algo **no enforzable mecánicamente o por skill:** una decisión arquitectónica del proyecto, una restricción específica que solo aplica aquí, un patrón semántico que ningún linter o verificador puede atrapar.

11.2. Principio "Operativo, no informativo"

- ❌ "El proyecto usa metodología comparativa entre cinco países".
- ✅ "Antes de incluir datos de un país, verifica que tienes la misma métrica para al menos otros 3 para permitir comparación".
- ❌ "El proyecto usa pytest para tests".
- ✅ "Antes de declarar terminada una tarea, ejecuta `pytest -x --tb=short` y verifica que pasa".

La primera versión de cada par es información (el agente puede deducirla del proyecto). La segunda es una **instrucción accionable.** El agente lee mejor lo segundo.

11.3. Principio "Negative space matters"

Las listas de "no hagas" tienen relación señal/ruido más alta que las listas de "haz":

- "No uses `print()`; usa `logger`".

- "No marques tests como xfail/skip para hacer pasar el suite".
- "No uses 'nosotros' para referirte al autor".
- "No cites a la consultora X salvo en el anexo".
- "No produzcas recomendaciones hasta que el bloque analítico esté cerrado".

Las prohibiciones específicas son más útiles que principios genéricos como "escribe código limpio" o "sé riguroso".

11.4. Principio "Progressive disclosure"

No metas todo en el archivo de instrucciones raíz. Si tu proyecto tiene módulos, secciones o áreas con convenciones distintas, **delega:**

- **Instrucciones raíz:** reglas globales (5-15 reglas).
- **Archivos referenciados:** lo detallado (metodología, glosario, guía de estilo, plantillas).
- **Skills o subagentes:** procedimientos especializados o roles concretos.

El agente carga lo correcto en cada momento; tú mantienes las reglas globales legibles.

11.5. Límite duro de tamaño

Si tu archivo de instrucciones raíz supera ~150-200 líneas (o ~2.000 palabras), está **demasiado largo.** El modelo no las lee con igual atención. Es síntoma de que estás metiendo en él cosas que pertenecen a archivos del proyecto, skills o subagentes. Trocéalo.

12. Diseñar sensores accionables

Un sensor (*test, linter, fact-checker*, revisor, lista de comprobación) **comunica con dos audiencias:** el humano y el agente. Si lo optimizas solo para el humano, el agente desperdicia tokens entendiéndolo. Si solo para el agente, el humano sufre. Los buenos sensores hablan a los dos.

12.1. Mensajes de error como prompts

Un mensaje de error es un prompt que estás inyectando al agente para que se autocorrija. Diseñarlo bien multiplica la eficacia del bucle de feedback.

Mal mensaje:

```
TypeError: Object of type 'datetime' is not JSON serializable
```

Buen mensaje:

```
TypeError: datetime no es JSON-serializable directamente.
```

```
Archivo: routes/tasks.py, línea 47
Por qué: estás devolviendo Task.created_at (datetime) sin convertir.
Solución típica: usa response_model en el decorador del endpoint,
o convierte explícitamente con .isoformat().
```

Mal reporte:

> La sección tiene varios problemas. Revisar atribuciones y cifras.

Buen reporte:

> BLOQUEANTE — Párrafo 3, frase 2: 'el 47% reportan haber adoptado X en 2025'. La fuente citada no contiene esta cifra; la cifra real en esa fuente es 23%, para un alcance distinto. Correcciones posibles: (a) cambiar la cifra al 23% y matizar; (b) localizar otra fuente; (c) eliminar la cifra y reformular cualitativamente.

Las dos versiones consumen un número similar de líneas. La segunda **enseña al agente a arreglarlo.**

12.2. Verificadores personalizados

Cuando detectes un patrón específico de tu trabajo que se viola repetidamente, escribe un verificador custom y conéctalo como hook o skill:

- **En código:** un linter que prohíbe importar tal módulo desde tal capa, un script que detecta `TODO` sin issue asociada, una regla que veta llamadas a APIs sin manejo de errores.
- **En conocimiento:** un verificador que comprueba que cada cifra del texto tiene fuente adyacente, un verificador de coherencia que busca contradicciones entre secciones, un verificador de glosario que marca usos fuera de definición, un verificador de alcance que marca aserciones fuera del brief.

Conectados al momento adecuado del flujo, te avisan **cuando el agente comete la violación**, no días después en revisión.

12.3. Tests estructurales

Los tests estructurales validan propiedades, no comportamiento puntual. Funcionan tanto en código (ArchUnit-style: las rutas no importan de la base de datos directamente, los módulos respetan capas) como en trabajo del conocimiento (toda sección tiene encabezado de criterios de aceptación; toda afirmación cuantitativa va acompañada de fuente; toda recomendación tiene al menos un riesgo asociado).

> Falla rápido, mensaje claro, agente aprende.

12.4. Revisores adversariales

El sensor más infrautilizado: un agente cuyo único trabajo es **objetar.**

> Eres un revisor hostil con incentivos para encontrar fallos. Lee este trabajo y produce el peor memorando de objeciones que un par crítico podría escribirte. No te suavices. Es preferible una objeción dura que un consenso amable que se rompe en la presentación al cliente.

Hazlo correr sobre cada entregable importante. Las objeciones que sobrevivan a tu juicio son trabajo extra; las que descartes te enseñan dónde tu propio razonamiento ya estaba sólido.

12.5. Sensores end-to-end

Para verificar **comportamiento real** (no solo lógica o forma), un sensor end-to-end es el más fiable que hay. En software: un smoke test con browser headless que verifica que la UI realmente se renderiza, los endpoints responden bien, los flujos completos funcionan. En investigación o análisis: un protocolo en el que un experto humano lee el documento completo, verifica una muestra de citas críticas tirando del enlace original, y aplica el "test del cliente exigente" (¿qué preguntaría el cliente más sofisticado?).

Para un agente que produce trabajo sin "verlo" directamente, los sensores end-to-end son sus ojos.

13. El bucle de dirección

Böckeler lo llama ***steering loop***. Es la práctica humana de **iterar el arnés** cuando algo falla, en lugar de iterar el prompt o regañar al modelo.

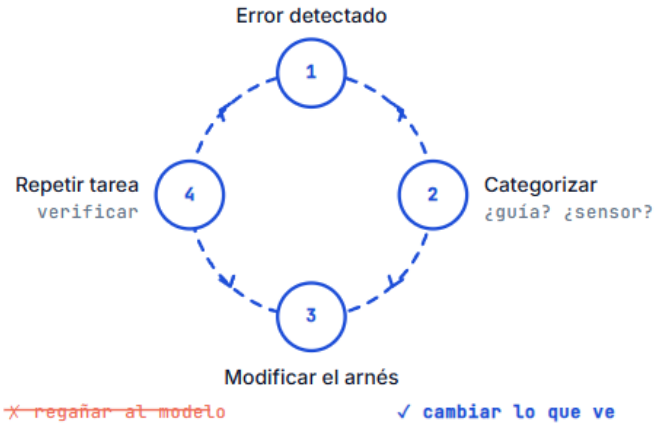


Imagen 13. Iterar el arnés

13.1. El procedimiento

Cuando un agente comete un error:

1. **No le riñas al modelo.** Es ruido en su contexto y no resuelve nada estructural.
2. **Categoriza el error:**
   - ¿Falta una guía (no sabía la convención)? → Va a las instrucciones, glosario, guía de estilo, plantilla.
   - ¿Faltaba un sensor (no detectó que estaba mal)? → Linter, test, skill verificadora, lista de chequeo.
   - ¿Falta scope (intentó hacer demasiado a la vez)? → Más WIP=1, mejor descomposición.
   - ¿Falta legibilidad del entorno? → Mejor estructura, más documentación operativa.
3. **Modifica el arnés** consecuentemente.
4. **Repite la tarea** y verifica que el cambio funciona.
5. **Si vuelve a fallar el mismo tipo de error** después del cambio, el cambio no era el correcto. Itera.

13.2. Por qué funciona

El modelo no aprende de tus regaños (no actualiza sus pesos). Pero sí cambia su comportamiento si cambian:

- Las instrucciones que lee.
- Los errores que ve cuando se autocorrige.
- Las herramientas y materiales a los que tiene acceso.

Estas son las cosas que tú controlas. Cambiarlas es el único punto de palanca real que tienes.

13.3. Ejemplos concretos

Caso 1: el agente mete un `print()` para debug en producción.

- ❌ **Mala respuesta:** "no uses print, te lo dije en las instrucciones".
- ✅ **Buena respuesta:** añadir un hook que detecte `print(` en el código de producción y lo reporte. Ahora es **imposible** que el agente cometa ese error sin que se entere.

Caso 2: el agente menciona repetidamente una cifra sin fuente y, cuando se la pides, se la inventa.

- ❌ **Mala respuesta:** "no inventes cifras, ya te lo he dicho".
- ✅ **Buena respuesta:**
  - Añade a las instrucciones: "Toda cifra cuantitativa debe ir acompañada de fuente entre paréntesis en el momento de redactarla. Si no tienes fuente disponible, escribe `[CIFRA PENDIENTE — verificar]` y sigue".

    - Crea una skill `verificador-de-cifras` que escanee el texto buscando dígitos y reporte cuáles van sin fuente adyacente.
    - Pásala como sensor automático antes de cualquier cierre de sección.

Caso 3: el agente olvida actualizar el registro de progreso al cerrar.

- ❌ **Mala respuesta:** "tienes que actualizarlo, está en las instrucciones".
- ✅ **Buena respuesta:** hook al cierre de sesión que verifique si la sesión cambió artefactos pero no el registro de progreso, y avise.

Después del cambio, el patrón cae a casi cero. Si vuelve a aparecer en otro contexto, refinas. Eso es el bucle de dirección.

14. Plantillas de arnés y topologías

Una idea emergente: si reduces la variedad del sistema (Ley de Ashby, §6.6), puedes construir un arnés que lo regule de verdad.

Las plantillas de arnés por tipo de proyecto o entregable son bundles pre-configurados (instrucciones, plantillas, skills, sensores, hooks) para una topología concreta:

En software:

- CRUD business service en un framework concreto.
- Event processor en otro lenguaje.
- Dashboard de datos en un stack frontend específico.
- Microservicio con esquema de eventos predefinido.

En trabajo del conocimiento:

- Informe estratégico/sectorial.
- Memorando legal con estructura predefinida.
- Brief de mercado con secciones canónicas.
- Paper académico con estructura IMRyD.
- Artículo de investigación periodística con sus verificaciones específicas.
- Análisis financiero con sensibilidad de hipótesis obligatoria.

Cada plantilla incluye:

- Archivos de instrucciones preescritos con las convenciones de la topología.
- Hooks o rutinas configuradas (formatter, linter, verificadores específicos).
- Skills relevantes (test-writer, fact-checker, verificador especializado).
- Subagentes pre-definidos (reviewer, e2e-tester, revisor crítico).
- Plantillas estructurales del entregable.
- Lista de fuentes o dependencias válidas por defecto.

- Tests o verificaciones estructurales.

Hoy (mayo 2026) están empezando a aparecer como repos públicos y como recursos compartidos en comunidades profesionales. El movimiento es claro: **los próximos años verán explosión de plantillas de arnés** como ya pasó con los service templates en software.

Recomendación pragmática: cuando empieces un proyecto nuevo, **mira si hay una plantilla para tu topología.** Si no, considera construir una propia para tu equipo a partir del segundo proyecto similar que hagas.

15. El behaviour harness y los approved fixtures

El *behaviour harness* —cómo verificar que el entregable realmente **hace lo que tenía que hacer**— sigue siendo el frente abierto del campo. Pero hay patrones prometedores.

15.1. El problema

Un entregable bien construido, bien estructurado, sin errores de superficie, **puede aún así no responder la pregunta que se planteaba.** O responderla en un nivel inútil de generalidad. O resolver técnicamente algo que no era el problema. Los sensores agénticos detectan errores de superficie; los errores de **sustancia** son más difíciles.

A esto se suma un problema circular: las verificaciones generadas por el propio agente tienden a ser circulares. El agente escribe la verificación que valida lo que el agente acaba de producir. Si entendió mal el requisito, su verificación valida el malentendido.

15.2. Approved fixtures

Patrón propuesto por equipos como Thoughtworks: el humano (o un experto) produce ***fixtures* aprobados** —pares (input, output esperado) o (pregunta, respuesta mínima aceptable)— que el agente **no puede modificar.**

- **En software:** un directorio `tests/fixtures/approved/` con archivos JSON o YAML que documentan pares input/output. Los tests son simplemente "para este input, ¿produces este output?". Si el agente cambia el output esperado para hacer pasar el test, está rompiendo el contrato; el humano (o un sensor inferencial) lo detecta.
- **En trabajo del conocimiento:** antes de escribir un informe, decides con tu cliente las 5 preguntas que el informe debe responder, con sus respuestas mínimas aceptables. Esas son tus *fact-fixtures*: el agente las debe responder, no las puede ignorar, y debe demostrar respuesta para declarar la sección cerrada.

Imagen 14. *Approved fixtures. Contrato read-only*

En ambos casos, los fixtures están **marcados como *read-only***, y un *hook* o regla del proyecto veta cualquier *edit* sobre ellos.

15.3. *Property-based testing*

Otra dirección, especialmente útil en software: en lugar de fixtures puntuales, **propiedades invariantes** que deben cumplirse para cualquier input válido. Herramientas como Hypothesis (Python), fast-check (JS) o QuickCheck (Haskell) generan cientos de inputs aleatorios y verifican que la propiedad se cumple. Es muy difícil que un agente "haga trampa" con un property-test bien escrito.

15.4. El test del lector escéptico

Antes de cerrar un entregable importante, ejecuta este protocolo:

1. **Identifica los 3 lectores más sofisticados** que podría tener (un experto sectorial, un par crítico, un competidor, un revisor hostil).
2. **Por cada uno:** ¿qué objeción específica plantearía? ¿qué dato pediría ver? ¿qué contraejemplo le vendría a la cabeza?
3. **Verifica que el entregable anticipa**, o al menos no es vulnerable, a esas objeciones.

Se puede automatizar parcialmente como skill ("Lector escéptico triple"), pero el juicio fino sigue siendo humano.

15.5. LLM-as-judge

Un sensor inferencial que evalúa el output del agente principal. Eficaz pero caro y probabilístico. Úsalo como **segunda línea** después de los computacionales, no como única defensa. El patrón típico: cuando cierras un entregable, abres una sesión con un agente "evaluador" en contexto fresco (que no vio cómo se produjo el entregable) y le pides que aplique unos criterios de aceptación claros.

16. Limpieza periódica: garbage collection agéntica

Inspirado en el patrón que OpenAI documentó: **agentes en segundo plano que limpian la deriva.**

16.1. La idea

Cualquier proyecto acumula entropía con el tiempo: código muerto, dependencias huérfanas, tests obsoletos, documentación desactualizada, naming inconsistente, fuentes que ya no se usan, decisiones revertidas no anotadas, glosario que ha crecido sin podar, borradores antiguos.

Un equipo humano lo arregla en sprints de "limpieza" cada cierto tiempo (es decir, casi nunca, o cuando duele demasiado). Un equipo con agentes puede arreglarlo **continuamente**, con jobs programados que disparan agentes con misiones específicas:

- "Encuentra código sin referencias y elimínalo".
- "Encuentra tests redundantes y consolídalos".
- "Detecta fuentes que ya no se citan y muévelas a archivo".
- "Detecta términos del glosario sin uso reciente y propón retirada".
- "Detecta inconsistencias terminológicas entre secciones".

16.2. Reglas para que no se descontrole

- **El humano aprueba cada eliminación importante.**
- **Archivar antes de borrar:** nada se elimina; todo se mueve a un histórico o se marca como deprecado.
- **Bitácora de la limpieza:** una entrada en el registro del proyecto resume qué se ha movido y por qué.

16.3. Rutina periódica

Cada mes (o al cierre de cada fase del proyecto):

1. **Auditoría de materiales:** ¿qué está pendiente de validación? Decisión: promover o archivar.
2. **Auditoría de glosario o convenciones:** ¿qué se ha usado este mes? ¿algo nuevo merece entrar? ¿algo definido se ha quedado sin uso?
3. **Auditoría del registro de decisiones:** ¿hay decisiones revertidas sin anotar? ¿conflictos entre decisiones?
4. **Auditoría de versiones:** archiva las antiguas, deja accesibles las dos o tres más recientes.
5. **Auditoría del registro de progreso:** archiva entradas antiguas en un histórico para mantener el archivo activo manejable.

Esto se puede pedir a un agente: "limpia este proyecto, propón los cambios, ejecútalos cuando los apruebe".

17. Observabilidad y telemetría

Si no observas, no mejoras.

17.1. Qué medir

Métricas de proceso:

- Tiempo medio por sesión o tarea.
- Número de iteraciones hasta cerrar una tarea.
- Tasa de issues bloqueantes detectados por los sensores (debería bajar con el tiempo si tu arnés mejora).
- Tasa de objeciones MAYORES del revisor crítico que sobreviven a tu juicio.

Métricas de resultado:

- Tasa de objeciones del destinatario al recibir entregables (debería estar al mínimo).
- Tasa de aceptación a la primera de los pull requests, o de los entregables.
- Tiempo de cold-start al volver a un proyecto tras una semana sin tocarlo.

17.2. Cómo medir

No necesitas un sistema sofisticado. Una hoja de cálculo simple basta:

Fecha	Proyecto	Tarea	Duración	Iteraciones	Bloqueantes	Mayores	Notas
...	...	...	...	...	...	...	...

Después de 20-30 entradas, los patrones empiezan a verse.

17.3. El KPI norte: *rebuild cost*

Una sesión nueva, sin contexto previo: **¿en cuánto tiempo está produciendo trabajo útil?**

- **< 3 minutos:** arnés de élite.
- **3-10 minutos:** aceptable, hay margen.
- **> 10 minutos:** tu arnés no cumple su función. Probablemente requiere demasiado descubrimiento por parte del agente. Documenta más estado y más explícitamente.

PARTE 4. Evaluación y mejora continua

18. KPIs del arnés

Resumen ejecutivo de indicadores con valores objetivo:

KPI	Qué mide	Objetivo
Rebuild cost	Tiempo desde sesión vacía a trabajo productivo	< 3 min
Tasa de cierre primer intento	% de tareas que cierran sin más de 1 iteración mayor	> 60%
Tasa de objeciones del destinatario	% de entregables que reciben objeciones sustantivas externas	> 15%
Cobertura de sensores	% de cierres que han pasado por todos los sensores definidos	1
Latencia de mejora del arnés	Tiempo desde detectar un patrón de fallo a cerrarlo en el arnés	< 1 sesión
Saturación del proyecto	Cuán "obvio" es para un colaborador nuevo qué hay que hacer al entrar	Cold-start passing

Revisa estos KPIs **mensualmente** en proyectos largos. Si alguno empeora, identifica qué cambió.

19. Antipatrones frecuentes

19.1. Las instrucciones sobre-especificadas

Síntoma: tus instrucciones crecen sin parar. Cada vez que algo falla, añades una regla.

Por qué falla: el modelo deja de leerlas con atención. Las reglas críticas se pierden entre el ruido.

Antídoto: poda agresiva. Si una regla no se ha violado en las últimas 5 sesiones, considera quitarla. Si una regla es enforzable mecánicamente, conviértela en hook/linter/skill y bórrala del archivo.

19.2. La exploración infinita

Síntoma: le pides al agente "investiga X" o "explora Y" sin acotar y termina leyendo cientos de archivos/fuentes, agotando el contexto.

Antídoto: acota explícitamente al pedir ("investiga X mirando solo Y y Z", "3-5 fuentes tier-1 que respondan a esta pregunta específica"). Para investigaciones grandes, usa un modo dedicado de investigación con brief estructurado.

19.3. La confianza prematura

Síntoma: el agente dice "hecho" y tú aceptas sin verificar. Días después descubres que media implementación estaba rota, o que el entregable tenía errores de atribución.

Antídoto: pass-state gating estricto. Sin verificación verde, no hay "hecho". Sin evidencia documentada, no hay "hecho".

19.4. El bypass por desesperación

Síntoma: tras una sesión frustrante, activas un modo permisivo ("yolo", "bypass") para "que termine ya". El agente hace algo irreversible.

Antídoto: el bypass solo en entornos sandboxed (devcontainer, VM efímera, copia de trabajo segregada). Y si te encuentras tentado a bypassear en tu máquina principal, vete a dormir. Mañana revisas el arnés.

19.5. La carrera de subagentes

Síntoma: cada vez que aparece un problema, creas un nuevo subagente o una nueva skill. Acabas con 15 que se solapan, nadie sabe cuál hace qué, los costes se disparan.

Antídoto: pocos subagentes y pocas skills, bien definidos. Si tienes más de 5 activos en un proyecto, probablemente puedes consolidar. Son martillo; no todo es clavo.

19.6. La fe en el revisor inferencial

Síntoma: descansas toda la verificación en otro LLM. Costes altos, falsos positivos y negativos aparecen.

Antídoto: el revisor inferencial es segunda línea, no única. La primera son verificaciones computacionales o procedimentales; el revisor inferencial cierra lo que esas no pueden alcanzar.

19.7. La dependencia oculta de una sola fuente o herramienta

Síntoma: revisando un proyecto terminado, descubres que un porcentaje desproporcionado del trabajo descansa sobre una sola fuente, librería o servicio.

Antídoto: documenta una regla de diversificación en las instrucciones. Construye un sensor que reporte la distribución de fuentes/dependencias al cierre de cada entregable.

20. Cuándo NO usar un agente

Hay casos donde la respuesta correcta es cerrar el chat, abrir el editor o el documento, y trabajar tú.

- **Tareas cortas donde el contexto está solo en tu cabeza:** el tiempo de explicarle al agente excede al de hacerlo tú.
- **Decisiones críticas con responsabilidad organizacional:** el agente no carga con la responsabilidad. Tú sí. Tu juicio es lo que se está pagando.
- **Cuando el cliente paga por tu juicio, no por tu output:** hay piezas donde lo que se factura es que lo hayas pensado tú. Usarlo y no decirlo es mentir; decirlo y haberlo usado a fondo desvirtúa el servicio. Acuerda con el cliente dónde puede y no puede entrar IA.
- **Refactors o cambios profundos con dependencias sutiles:** el agente tiende a romper invariantes que no ve. Mejor humano + agente como pareja, no agente solo.
- **Cuando la confidencialidad es máxima:** ciertas zonas (secreto profesional estricto, datos sensibles regulados, materiales gubernamentales) donde el riesgo no se justifica aun con planes empresariales.
- **Cuando tú no sabes la respuesta:** si tú no sabes qué pregunta hacer ni cómo evaluar la respuesta, el agente alucinará una respuesta plausible y tú la firmarás. Antes de usar el agente, ten al menos una hipótesis informada propia que el agente pueda confirmar o desafiar.
- **Cuando el sistema está roto y no sabes por qué:** déjate guiar por intuición y debug humano; el agente alucinará causas plausibles que te despistan.

21. El rol del humano

El arnés bien diseñado no elimina al humano: lo concentra donde más vale. Tu trabajo profesional ya no es escribir cada función o cada párrafo. Tu trabajo es:

- **Diseñar el entorno (el arnés).**
- **Especificar la intención:** el brief, los criterios de aceptación, las preguntas que el entregable debe responder, las invariantes que no se pueden violar.
- **Construir bucles de feedback:** sensores, verificaciones, listas de comprobación, revisiones críticas.
- **Aplicar juicio en los puntos donde la máquina no debe decidir sola:** la tesis, la recomendación final, la decisión arquitectónica, la matización política o legal, la decisión de qué cabe y qué no.
- **Iterar el arnés cuando algo no funciona.**

Esto es trabajo intelectual real. No es menos exigente que escribir cada palabra o cada línea; es exigente de otra manera. Y es más *leveraged*: un buen arnés produce semanas de output de calidad mientras tú piensas en lo que de verdad merece tu pensamiento.

PARTE 5. De hobbyista a *harness engineer*

22. Rutas de carrera y especialización

El harness engineering está cristalizando como un rol diferenciado. En mayo de 2026 ya hay puestos con ese título exacto en algunas empresas (Anthropic, OpenAI, Stripe, varios scale-ups). En otras se solapa con títulos como *AI Platform Engineer, AI Tooling Engineer, Developer Experience Engineer* en la era de los agentes, *AI workflow designer, Knowledge engineer, Research operations specialist o AI editor / managing editor*.

Las habilidades que componen el perfil:

22.1. Habilidades sustantivas duras

- **Maestría en tu disciplina origen.** Sin esto, no hay arnés que arregle nada. La IA no sustituye experiencia; la amplifica. Vale para programación, análisis, derecho, investigación, escritura o gestión.
- **Metodología explícita:** capacidad de describir el "cómo se hace" del trabajo de tu disciplina con suficiente detalle como para que un agente lo siga sin que tú estés delante.
- **Gestión de materiales primarios:** curado, evaluación crítica, jerarquías de evidencia (bibliografías, dependencias, fuentes, librerías).
- **Pensamiento por restricciones:** capacidad de delimitar alcance, prohibir desviaciones, fijar criterios de aceptación.

22.2. Habilidades técnicas blandas

- **Diseño de instrucciones:** ingeniería de prompts y de instrucciones de proyecto. Concisión + completitud.
- **Razonamiento sobre incertidumbre:** los modelos son probabilísticos. Diseñar para "fallará el 5% de las veces, ¿qué pasa entonces?".
- **Diagnóstico de causa raíz en sistemas no deterministas:** distinguir entre "el agente no entendió la guía", "le faltaba un sensor", "el alcance estaba mal definido", "el modelo se equivocó este turno".
- **Iteración disciplinada:** el bucle de dirección requiere paciencia y método.

22.3. Habilidades no técnicas

- **Comunicación con clientes o stakeholders:** cuándo y cómo decir que has usado IA, qué disclaim, qué no negociar, cómo medir ROI.
- **Diseño de procesos y equipos:** equipos pequeños haciendo trabajo grande con agentes son posibles, pero requieren coordinación distinta. Liderar esa coordinación es un rol naciente.

22.4. Rutas típicas

- **Desde DevOps / Platform Engineer:** la transición es natural en el lado software. Pasas de gestionar pipelines para humanos a gestionar arneses para agentes.
- **Desde Senior Engineer / Tech Lead:** aportas el ojo arquitectónico crítico. El reto es desaprender la tentación de escribir todo el código tú.
- **Desde ML / AI Engineer:** tienes la parte de modelos resuelta; te falta la parte de software systems.
- **Desde QA / SDET:** tu mundo es ya el de los sensores. Naturalmente bien posicionado.
- **Desde analista / consultor senior:** pasas de ejecutar análisis a diseñar el cómo se ejecutan análisis. Riesgo: la tentación de "hacerlo tú porque es más rápido" sin invertir en el arnés.
- **Desde investigador académico:** tienes la metodología en la cabeza pero implícita. La tarea es explicitarla. Inversión inicial grande, retorno alto.
- **Desde periodista de investigación:** tu mundo es ya el de las fuentes y la verificación. Naturalmente bien posicionado para construir buenos sensores.
- **Desde editor o manager:** ya piensas en procesos, plantillas y consistencia. Tu transición es a estructurar el trabajo del equipo entero.
- **Desde PM / product manager:** bien posicionado para "AI workflow designer", aunque conviene profundizar en alguna disciplina sustantiva para tener credibilidad.

24. Cómo seguir aprendiendo cuando esto cambia cada semana

Este campo avanza tan rápido que cualquier guía —esta incluida— envejece. Hábitos que mantienen la guía mental al día:

24.1. Lee los engineering blogs, no los anuncios de producto

Los laboratorios publican posts de ingeniería que describen cómo construyen ellos sus sistemas. Eso es más útil que los anuncios de producto.

24.2. Practica con casos públicos

Cuando salga un patrón nuevo, no esperes a entenderlo perfectamente. Clona un repo de referencia o replica el patrón en un proyecto propio pequeño. Aprendes más en una tarde manipulándolo que en una semana leyendo análisis.

24.3. Mantén un journal del arnés

En un documento propio, anota cada cambio que haces a tu arnés y el resultado. Cuando aparezca una técnica nueva en la industria, contrastas con tu experiencia.

24.4. No persigas todas las características nuevas

Cada release añade funcionalidades. La mayoría no las necesitas. **Una característica entra a tu arnés solo cuando tienes un problema concreto que resuelve**, no porque sea brillante o porque otros la usen.

24.5. Enseña

La mejor forma de consolidar lo aprendido es enseñárselo a otro. Sesión interna, post en redes, taller en una conferencia. Cada vez que articulas algo, descubres lo que no entendías del todo.

24.6. Cultiva la humildad técnica

Hace 18 meses, esta disciplina no existía. Dentro de 18 meses, la mitad de lo que sabes ahora habrá evolucionado. La **disposición a desaprender** vale más que cualquier certeza adquirida. Las verdades de este campo tienen vida corta.

Glosario

Agencia: capacidad del modelo de percibir, razonar y actuar sobre el mundo. Propiedad del modelo, no del arnés.

Agente: modelo + arnés. Un sistema completo capaz de realizar tareas sobre un entorno operativo.

Approved fixtures (fact-fixtures aprobados): patrón de verificación donde el humano produce pares (input, output esperado) o (pregunta, respuesta esperada) que el agente no puede modificar.

Arnés (harness): todo lo que no es el modelo. Engloba instrucciones, materiales accesibles, herramientas, permisos, orquestación, persistencia y sensores.

Arnés externo: el que construye el usuario sobre el producto que usa.

Arnés interno: el que construye el fabricante alrededor del modelo (bucle de orquestación, ejecutores de herramientas, gestión de contexto).

Behaviour harness: dimensión del arnés que regula la corrección sustantiva o funcional. Frente abierto.

Brain / Hands / Session: arquitectura desacoplada (Anthropic, 2026): Brain (modelo + bucle), Hands (sandboxes efímeros), Session (log append-only).

Cold-start test: métrica que mide si una sesión nueva del agente puede ser productiva leyendo solo los archivos del proyecto.

Computacional (control): control determinista, ejecutado por una herramienta. Linters, tests, verificadores procedimentales.

Context engineering: subdisciplina centrada en cómo se compone la información que el modelo ve en cada llamada.

Feedforward / Feedback: en cibernética y en Böckeler, control antes del acto (guías) vs después del acto (sensores).

Garbage collection agéntica: agentes en segundo plano que limpian la deriva del proyecto (código muerto, fuentes obsoletas, dependencias huérfanas).

Guía: control feedforward. Anticipa el comportamiento del agente y lo orienta antes de actuar.

Hook: script o rutina ejecutado deterministamente en momentos del ciclo de vida del agente.

Inferencial (control): control no-determinista, ejecutado por otro modelo. LLM-as-judge.

Initializer + Working agent (o Coding agent): patrón canónico de Anthropic para tareas largas. Un agente inicializa el entorno; otro va completando entregables.

***Lost in the Middle*:** efecto (Liu et al., 2023) por el cual los modelos pierden eficacia procesando información situada en el centro de contextos largos.

Pass-state gating (pase verificado): política que prohíbe avanzar a la siguiente tarea hasta que la actual está verificada.

Plantilla de arnés (harness template): bundle pre-configurado de guías + sensores para una topología específica.

Rebuild cost: tiempo desde una sesión vacía hasta que el agente está produciendo trabajo útil.

RULER benchmark: benchmark de NVIDIA que mide el contexto efectivo real de un LLM. Típicamente 50-65% de la ventana anunciada.

Sensor: control feedback. Observa al agente después de actuar y le permite autocorregirse.

Skill: paquete de instrucciones reutilizable que el agente carga cuando el contexto lo pide.

Steering loop (bucle de dirección): bucle humano de mejora del arnés. Cuando algo falla, se itera el arnés, no se regaña al modelo.

Subagente: agente secundario que el principal invoca para tareas acotadas. Contexto, permisos y herramientas propios.

Variedad requisita (Ley de Ashby): principio cibernético según el cual un regulador necesita al menos tanta variedad como el sistema que gobierna. En la práctica, se aplica al revés: para regular bien al agente, reduce la variedad del sistema (atándolo a una topología concreta).

WIP = 1: regla operativa: una sola tarea o entregable en curso por sesión, hasta verificación completa.

Anexos

Anexo A: checklist de auditoría de arnés

Lista mínima para auditar el arnés de un proyecto activo. Adapta los nombres concretos de los archivos a tu disciplina.

Instrucciones del proyecto

- ☐ Existe un archivo de instrucciones leído por el agente al empezar.
- ☐ Tiene una sección de "qué es el proyecto", "por qué priorizamos lo que priorizamos" y "cómo trabajamos".
- ☐ Es operativo (instrucciones accionables), no informativo (descripciones del proyecto).
- ☐ Está bajo el límite duro de tamaño (~150-200 líneas o ~2.000 palabras).
- ☐ Contiene reglas de scope (WIP=1, pass-state gating).
- ☐ Tiene una sección "no hagas" con prohibiciones específicas.

Estado persistente

- ☐ Existe una lista de entregables o features con su estado.
- ☐ Existe un registro de progreso entre sesiones.
- ☐ Existe un registro de decisiones con sus porqués.
- ☐ El versionado es claro (Git, historial nativo, sistema de carpetas con fechas).

Sensores

- ☐ Hay un punto único que dispara todas las verificaciones (un script verify.sh, una rutina pre-cierre, una lista de comprobación).
- ☐ Las verificaciones producen reportes accionables (no solo "falla / no falla").
- ☐ Hay al menos un revisor crítico (humano, skill o subagente).

Permisos y aislamiento

- ☐ Los permisos del agente están explícitos (lo que puede y no puede hacer).
- ☐ Hay un sandbox o aislamiento adecuado al riesgo del proyecto.
- ☐ Los secretos no entran en el contexto del agente.

Cold-start test

- ☐ Sesión nueva, sin contexto previo, el agente identifica proyecto + progreso + siguiente paso en < 3 min ✅

Bucle de dirección

- ☐ Tienes un journal del arnés o equivalente donde registras cambios.
- ☐ Revisión periódica (mensual) de KPIs.
- ☐ Los últimos 3 fallos del agente han generado un cambio en el arnés (no solo regaño al modelo).

Anexo B: bibliografía y lecturas recomendadas

Fuentes canónicas (lectura obligatoria)

Anthropic Engineering,

- (2025, noviembre). *Effective harnesses for long-running agents*. https://www.anthropic.com/engineering/effective-harnesses-for-long-running-agents
- (2026, marzo). *Harness Design for Long-Running Application Development.*
- (2026, abril). *Scaling Managed Agents: Decoupling the brain from the hands*.

Böckeler, B. (2026, abril). "Harness engineering for coding agent users". *Martin Fowler's blog*.

Lopopolo, R. (2026, febrero). "Harness Engineering: Leveraging Codex in an Agent-First World". *OpenAI Engineering Blog*.

Especificaciones y estándares relevantes

- AGENTS.md Project. https://agents.md/ (estándar abierto bajo Linux Foundation / Agentic AI Foundation).
- Model Context Protocol. https://modelcontextprotocol.io/

Lectura complementaria valiosa

Liu, N. F. et al. (2023). "Lost in the Middle: How Language Models Use Long Contexts". *arXiv:2307.03172*.

Mollick, E. *Co-Intelligence: Living and Working with AI y Substack One Useful Thing.*

Narayanan, A. & Kapoor, S. *AI Snake Oil* (Substack y libro). Anti-hype riguroso.

NVIDIA. *RULER benchmark y publicaciones derivadas sobre contexto efectivo*.

Stripe Engineering. Posts sobre agentes y hooks pre-push. https://stripe.dev/blog/

Comunidad

Discord oficiales de Anthropic Developers y OpenAI Developers.

Reddit: r/ClaudeAI, r/OpenAI.

LinkedIn y Slack: comunidades sectoriales (*AI in Consulting, AI for Lawyers*, etc.).

Conferencias: QCon, AI Engineer Summit, Code with Claude, OpenAI DevDay.

> El futuro del trabajo profesional asistido por IA no se encuentra en mejores prompts, sino en arneses que operen con precisión quirúrgica sobre la realidad operativa del proyecto.