Desacuerdo controlado: revisión adversarial automatizada con un segundo asistente de código en el desarrollo de software

Informe de experiencia

Nicolas Rocchia
San Francisco, Córdoba, Argentina
nicolasrocchia@gmail.com

Julio de 2026

Resumen

Este informe de experiencia documenta una metodología de desarrollo operada durante 52 días distribuidos en ocho semanas calendario sobre cinco proyectos profesionales: un asistente de código (Claude Code, que ejecutó dos modelos de Anthropic a lo largo del período) genera planes e implementaciones; un segundo asistente de otro proveedor (Codex CLI, de OpenAI) los ataca con una consigna adversarial; el primero recibe la instrucción de verificar cada hallazgo contra el repositorio; y cada evento de revisión termina cuando cada hallazgo ha quedado resuelto, refutado con evidencia o transferido explícitamente al desarrollador (la transferencia termina el evento; el hallazgo puede quedar abierto), un criterio de corte aquí denominado *desacuerdo controlado*. El informe explica por qué el diseño es plausible (límites documentados de la autorrevisión, diversidad de errores, la ejecución como árbitro), cómo se opera (mecanismo, receta mínima de adopción) y narra un flujo completo real de punta a punta, con la consigna, los 20 hallazgos del revisor, su disposición y los tiempos y consumos de tokens disponibles. Se observaron dos mecanismos de intervención: la detección de defectos existentes y la conformación del diseño en la compuerta de plan; ambos se ilustran con casos trazables, junto con los falsos positivos, los defectos que el método no detectó y una regresión introducida por una corrección. Sobre el subconjunto auditable del corpus, al menos el 64,9 por ciento de los hallazgos fue incorporado y el 3,6 por ciento fue refutado con evidencia. La contabilidad disponible y reconciliada (91 eventos de compuerta documentados, unidades, universos de cobertura y reconciliaciones) se presenta en el Apéndice A. La atribución causal del beneficio a la diversidad de proveedor, y el balance neto de costos, quedan explícitamente como hipótesis: no se realizó comparación controlada, y el experimento pendiente se especifica.

1 Introducción

La adopción de asistentes de código basados en modelos de lenguaje (LLM) se ha extendido rápidamente; su verificación sigue siendo un problema abierto. Una respuesta habitual, pedirle al mismo modelo que revise su propia salida, tiene limitaciones documentadas en la literatura, que motivan explorar revisores externos.

Este informe documenta una alternativa operada en desarrollo comercial real: la revisión adversarial automatizada por un segundo asistente de otro proveedor, con la instrucción de verificar cada hallazgo contra el repositorio y el desarrollador como responsable final. El propósito del informe no es inventariar defectos: es documentar una metodología operable (en el sentido mínimo de que fue aplicada en los 91 eventos de compuerta documentados), con su fundamento, su mecanismo, una receta mínima de adopción, un flujo completo real narrado por dentro, y su comportamiento observado, incluidos sus fallos.

Para evitar el equívoco más frecuente en este tipo de textos, se separan tres afirmaciones de distinto estatus epistemológico: (a) la afirmación de que la autocorrección intrínseca de los LLM tiene límites está respaldada por la literatura; (b) que un segundo asistente detecta defectos que el generador omitió, y que su crítica sobre el plan modifica el diseño antes de la implementación, son hechos observados y documentados en la operación reportada, de la que se extraen frecuencias y ejemplos; (c) que la *diversidad de proveedor* sea la causa de esa detección, y que el balance neto sea favorable, son hipótesis: no se realizó la comparación controlada (mismo flujo con revisor del mismo proveedor, o con autorrevisión) que permitiría sostenerlas. El centro de gravedad de la contribución está, por eso, en el mecanismo (la revisión en compuerta de plan y la regla de terminación por disposición individual), no en la identidad del revisor: el proveedor distinto fue una implementación conveniente de señal externa, no una palanca aislada y probada.

2 Por qué es plausible que funcione

El diseño descansa sobre cuatro piezas de evidencia, que conviene leer como un argumento y no como un catálogo.

El motor: el punto ciego de la autorrevisión. En tareas de razonamiento, la autocorrección *intrínseca* (sin señal externa) no mejora de forma confiable y puede degradar: en una muestra de 200 problemas de GSM8K con el GPT-4 de agosto de 2023, la exactitud cayó de 95,5% a 91,5% tras una ronda de autorrevisión y a 89,0% tras dos [3]. El cuello de botella es la detección: los modelos fallan en localizar errores lógicos pero los corrigen cuando se les indica la ubicación [4]. Y existe un punto ciego específico: errores que el modelo no corrige en su propia salida sí los corrige cuando se le presentan como entrada externa (medido con errores inyectados de forma controlada en modelos abiertos) [5]. Ese hallazgo aporta evidencia compatible con el mecanismo que el flujo explota: el defecto que el generador no ve en su propio conjunto de cambios es visible para un revisor al que le llega desde afuera. Complementariamente, hay evidencia inicial de un sesgo de autopreferencia en evaluadores LLM [6]: el generador es un juez parcial de sí mismo. Tres precisiones de alcance: estos resultados provienen de razonamiento y evaluación de texto, no de revisión de código; justifican la necesidad de señal externa, no que deba provenir de otro proveedor; y existen técnicas que mejoran la autocorrección en tareas puntuales. La afirmación defendible es que la autorrevisión no constituye un control *independiente*.

El multiplicador: diversidad de errores. La intuición proviene de los métodos de conjunto: bajo condiciones que involucran la competencia individual y el mecanismo de combinación, la diversidad de errores contribuye a la mejora del conjunto [1]; la teoría moderna lo formaliza como un compromiso, no una ganancia incondicional [2]. En generación general, combinar modelos heterogéneos puede superar al mejor individual: Mixture-of-Agents alcanzó 65,1% de tasa de victoria controlada por longitud en AlpacaEval 2.0 contra 57,5% de GPT-4o usando solo modelos abiertos [8], y el debate multiagente reduce errores en benchmarks de razonamiento [7], aunque no supera consistentemente a técnicas de muestreo del mismo modelo a igual presupuesto [3]. El flujo aquí descrito no promedia ni vota: traslada la intuición de diversidad a un esquema de *detección* secuencial sobre una única implementación, configuración que esa teoría no cubre directamente.

Los vecinos: revisión de código con agentes. La estructura generador-revisor con LLMs no es nueva y este informe no la propone. CodeAgent organiza múltiples agentes comunicativos con un supervisor para revisión de código [9]. PairCoder, el vecino conceptual más cercano, separa un agente Navegante (planificación y dirección por retroalimentación) de un agente Conductor (implementación), con mejoras relativas de pass@1 de 12,00% a 162,43% sobre la generación directa en benchmarks [10]. Y el reexamen de la autodepuración con pruebas autogeneradas, en tareas autocontenidas de Python, muestra que estas introducen sesgo (etiquetas falsas negativas que llevan al modelo a "corregir" programas correctos), exactamente la clase de problema de oráculo que este informe discute [11]. Frente a esa línea, evaluada sobre benchmarks, la contribución diferencial de este informe es documentar la automatización sostenida de la estructura sobre proyectos profesionales reales, la revisión de planes además de código, y una regla de terminación basada en la resolución individual de hallazgos. Tres trabajos contemporáneos al cierre de este informe delimitan mejor el espacio. Un experimento controlado con el mismo par de proveedores (116 tareas de LiveCodeBench, seis condiciones de escritor y revisor) encontró que el valor de la revisión cruzada es asimétrico y depende de la brecha de capacidad entre los modelos: revisar borradores del modelo más débil mejoró la corrección hasta 18,1 puntos, mientras que el revisor más débil empeoró los borradores del más fuerte, con las revisiones dañinas concentradas en reescrituras que descartan una solución funcional y las útiles en ediciones locales [12]. Ese resultado no evalúa la arquitectura de este informe, pero respalda dos de sus decisiones de diseño: aquí el revisor no reescribe (propone hallazgos con evidencia que el generador verifica y un humano arbitra), y la compuerta de plan actúa antes de que exista código, un escenario que ese experimento no cubre. En la misma dirección, un protocolo revisor-crítico documentó un modo de fallo de falso consenso (agentes que convergen en acuerdo sin evidencia suficiente) y concluye que la cooperación confiable exige desacuerdo estructurado y fundado en evidencia [13]; y una metodología adversarial por etapas con crítico de otro modelo, motivada por la crisis de precisión de los reportes generados por LLMs, registra que su fallo más instructivo lo eliminó una única prueba empírica y no un mejor encuadre adversarial [14], consistente con el rol de los oráculos en este informe.

El límite: la lección de las N versiones. La confiabilidad por diversidad de implementaciones tiene su antecedente en la programación de N versiones [15], y su historia aporta el límite central. Knight y Leveson sometieron 27 versiones desarrolladas independientemente desde una misma especificación, en dos universidades, a un millón de casos de prueba, y rechazaron estadísticamente la independencia de fallas: personas distintas malinterpretan la misma especificación de la misma manera [16, 17]. El experimento fue revisitado en 2026 con agentes de código, variando modelo, agente y lenguaje: 429 casos de prueba con fallas coincidentes donde la independencia predecía 115,36, sobre 48 configuraciones, junto con reducciones importantes de fallas mediante votación [18]. La consecuencia de diseño: cualquier decorrelación entre asistentes debe asumirse parcial, la especificación puede

constituir una fuente común de error, y la decisión final sobre la intención del producto requiere una autoridad externa al par de asistentes. Por eso el flujo termina en un desarrollador, no en un consenso.

3 El método

3.1 Roles y compuertas

Roles fijos que no se mezclan: un asistente principal (generador) planifica e implementa; un asistente de otro proveedor (revisor) ataca con consigna explícitamente adversarial; la ejecución real (compilación, pruebas, ejecución funcional) arbitra todo lo que puede arbitrar; el desarrollador es dueño del resultado. En la instanciación reportada los roles se encarnan así: Claude Code es el entorno de orquestación; el modelo de Anthropic activo en la sesión es el generador, que planifica, implementa, verifica hallazgos y reconcilia (en este informe "orquestador" y "generador" nombran a esa misma unidad operacional); Codex, por CLI o plugin, es el revisor; la compilación y las pruebas son un oráculo parcial; el desarrollador es la autoridad final. El método contempla dos compuertas: sobre el *plan* de implementación, antes de escribir código, y sobre el *diff* (el conjunto de cambios) resultante, antes de confirmar. La consigna (la instrucción de entrada o *prompt* del revisor) pide no aprobar sino atacar: errores funcionales, regresiones, concurrencia, seguridad, persistencia, supuestos de dominio, citando impacto y evidencia por hallazgo. El orquestador recibe la instrucción de verificar cada hallazgo contra el repositorio antes de consolidar el resultado; el corpus contiene casos en los que esa verificación funcionó, un fallo de verificación del hallazgo (la autorización dada por verificada, Sección 5.1) y un fallo posterior de diagnóstico y validación de una corrección, no del hallazgo (la regresión de la Sección 5.5, atribuida primero a otra causa); son categorías distintas.

3.2 La regla de corte

El evento de revisión no termina en consenso: el acuerdo entre asistentes que comparten datos de entrenamiento no vuelve verdadera una conclusión, y ambos pueden compartir la misma malinterpretación de una especificación ambigua [16, 18]. El evento termina cuando cada hallazgo ha quedado (i) resuelto, (ii) refutado o descartado con evidencia verificable, o (iii) transferido explícitamente al desarrollador; en el caso (iii) el evento termina, pero el hallazgo puede permanecer abierto. La derivación (iii) tiene estados terminales propios, observados en la operación: *deuda aceptada* (registrada con identificador y justificación), *riesgo aceptado* (decisión explícita del desarrollador), *mitigación parcial* (parte aplicada, resto a deuda) y *escalado pendiente* (derivado sin decisión registrada). El generador propone y registra el cierre operativo de (i) y (ii) con evidencia verificable (cita de archivo y línea, medición, o comportamiento documentado del entorno); el desarrollador conserva autoridad de revisión y responsabilidad final sobre todos los cierres. Los estados de deuda aceptada, riesgo aceptado y mitigación parcial requieren una decisión explícita del desarrollador; el escalado pendiente requiere una transferencia registrada y un responsable asignado, pero no constituye una decisión ni un cierre del hallazgo: es una derivación abierta, y el corpus contiene una (la objeción de costo de la Sección 3.4). En la operación real la norma fue una sola corrida del revisor por compuerta (la reanudación, es decir reutilizar la sesión del revisor, aparece 3 veces en todo el corpus, siempre para confirmar una corrección); la señal de corte adoptada es "severidad decreciente y repetición sobre el mismo archivo", una heurística surgida de un episodio único (Sección 6), no un criterio validado.

3.3 Adopción mínima

La instanciación completa (Apéndice A) es una implementación posible, no un requisito. La receta mínima: (1) elegir un asistente revisor separado del generador (para reproducir la configuración reportada, de un proveedor distinto); (2) definir una política explícita de activación y someter el plan de las unidades de trabajo que la cumplan, y luego su diff, al revisor con consigna adversarial (la política de la operación reportada: Sección 4.1 y Apéndice C); (3) hacer que el generador verifique cada hallazgo contra el repositorio antes de presentar el resultado; (4) terminar cada evento cuando todo hallazgo haya sido resuelto, refutado con evidencia o transferido explícitamente al desarrollador, nunca por consenso (la transferencia termina el evento; el hallazgo puede quedar abierto); (5) conservar una bitácora de veredictos, que calibra el proceso y es la memoria que sobrevive a la retención de los historiales.

3.4 Escalamiento y decisión humana

Tres escalamientos documentados ilustran el reparto de responsabilidades (no constituyen un censo): la semántica de deduplicación de notificaciones la decidió el desarrollador; una objeción de costo operativo quedó explícitamente sin resolver en la sesión; y en P5 el revisor emitió un veredicto negativo global con una objeción mayor que el desarrollador decidió mantener por razones de producto, dejando constancia en la memoria del proyecto. Este último caso documenta el principio en su forma más incómoda: la última palabra es humana incluso contra el revisor.

4 Un flujo completo por dentro

Para que el mecanismo no quede abstracto, esta sección narra un flujo completo real (una unidad de trabajo que atravesó la compuerta de plan, la implementación y la

compuerta de diff), reconstruido de los historiales de sesión y sus artefactos persistidos: la funcionalidad F6 de P1 (una pantalla de fiscalización con evidencia georreferenciada, notificaciones y una declaración jurada digital), 20 de julio de 2026.

4.1 La decisión de correr la ronda

Las compuertas no se disparan a ciegas. Con el plan escrito, el orquestador planteó al desarrollador, textualmente: "F6 cruza la barra de alto impacto (contrato API-Web nuevo + autorización anti-IDOR + dominio de fiscalización). ¿Corro la ronda Codex red-team sobre el plan antes de cerrarlo?", con la recomendación: "Cuesta tokens y unos minutos. Calibración histórica: caza sistemáticamente auth/concurrencia/consumidores que no miré" (en la jerga de la bitácora: encuentra de manera recurrente defectos de autorización, concurrencia y consumidores no revisados). El desarrollador aprobó la ronda y, en el mismo paso, tomó una decisión de producto que el plan necesitaba (la semántica de deduplicación: "solo la primera" notificación por anomalía). La política de activación consta formalizada en las instrucciones globales del orquestador vigentes al cierre del período (reproducidas en el Apéndice C): se propone una ronda cuando la unidad cruza una barra de impacto alto definida por al menos uno de cinco criterios (cambio de contrato o API entre componentes; cambio de esquema de base de datos o migración; seguridad, autenticación o autorización; concurrencia, condiciones de carrera o transacciones; lógica difícil de revertir o crítica para el negocio), o cuando hay incertidumbre real (enfoques viables sin claro ganador, baja confianza en la lectura del código, requerimiento ambiguo y caro de equivocar); se omite en cambios locales, reversibles, triviales o por plantilla; y cada corrida requiere la aprobación explícita del desarrollador, nunca automática. La aprobación humana gobierna la activación lógica del evento; el permiso permanente de la configuración (Apéndice C) elimina únicamente la segunda confirmación técnica de la herramienta, y la separación es procedimental, impuesta por las instrucciones del orquestador, no por un control técnico. Dos límites: el historial de cambios de ese archivo de instrucciones durante el período no fue reconstruido, y cuántas unidades elegibles no fueron revisadas no se registró, de modo que los 91 eventos de compuerta documentan la actividad revisada, no toda la actividad del período.

4.2 La consigna

El orquestador escribió la consigna a un archivo (deliberadamente sin tildes, por robustez de codificación en el paso por shell) y lanzó al revisor. El encabezado, textual:

```
ATENCION - SIN SANDBOX: NO
escribas/edites/borres/muevas archivos
ni corras git que modifique estado.
SOLO lee y analiza.

Actua como RED-TEAM de este plan de
implementacion:  ataca supuestos,
agujeros de seguridad, riesgos y casos
borde.  NO lo apruebes - busca lo que
esta mal o falta.  El codigo esta en
este repo:  verifica los anclajes del
plan contra el codigo real antes de
afirmar hallazgos, y cita archivo:linea.
```

Siguen unas quince líneas de contexto del sistema, el plan completo en nueve fases, y siete pedidos específicos de ataque (autorización e IDOR, control de acceso inseguro a nivel de objeto, traducibilidad de las consultas a SQL, contrato y consumidores, fallas de diseño de la deduplicación, pérdida de campos en cadenas de error, casos borde del dominio, "y cualquier otra cosa que el plan asuma mal o le falte"). Cierre: lista numerada de hallazgos con severidad, evidencia archivo:línea y qué cambiaría del plan.

4.3 La corrida y los 20 hallazgos

El revisor (modelo gpt-5.6-sol, esfuerzo de razonamiento alto, Codex CLI v0.144.3) exploró el repositorio durante 12 minutos 22 segundos y consumió 283.638 tokens según su propio registro. Tras la corrida, el orquestador verificó con `git status` que el repositorio estuviera intacto (el revisor corre sin aislamiento técnico en este entorno). El resultado: 20 hallazgos (7 de severidad alta, 11 media, 2 baja), cada uno con evidencia archivo:línea. Cuatro ejemplos que muestran el rango:

El hallazgo que el autor consideró más valioso (alta, incorporado). La deduplicación planificada era global por tipo y entidad; el revisor demostró que eso silencia alertas legítimas: "un empleado agregado después nunca recibirá la alerta; un cambio de jurisdicción quedará silenciado por las filas de la anterior". Propuso deduplicar por destinatario. El orquestador lo verificó y lo incorporó al plan: "es una mejora real y casi sin costo".

El hallazgo legal (alta, incorporado con reemplazo del diseño original). El plan previa caer a un diálogo de confirmación genérico si el texto de la declaración jurada no llegaba al cliente. El revisor lo objetó de raíz: "estaría registrando aceptación de un texto que nunca se mostró"; funcional, pero comprometía el valor probatorio de la aceptación: un riesgo jurídico que justificó rechazar de manera segura (*fail closed*). El plan pasó a ese comportamiento: sin texto, no se permite completar ni se envía aceptación.

La derivación a deuda técnica (alta). El revisor demostró que desactivar a un usuario no siempre revoca su acceso, por un vínculo heredado (*legacy*). La verificación del orquestador confirmó el problema y su alcance: "el mismo resolver se usa en toda la autorización del portal; F6

hereda el patrón sin empeorarlo; cambiarlo acá saldría del alcance". Se registró como deuda técnica con identificador, no se ignoró.

La refutación (alta, refutado con evidencia). El revisor objetó que una lista de campos permitidos en la capa web "no constituye una frontera de privacidad" porque el rol puede llamar a la API directamente. La verificación mostró que ese rol ya accede legítimamente al expediente completo por otra vía oficial: la lista es higiene de lo incrustado en HTML, no una frontera de autorización. Refutado, con la nota de propósito agregada al plan para que la discusión no se repita.

4.4 Disposición y cierre de la compuerta de plan

La disposición quedó registrada en el plan endurecido, con una ambigüedad declarada: 11 hallazgos incorporados (el mensaje de confirmación dice once; una entrada de sesión clasifica como incorporado el hallazgo mitigado parcialmente, lo que daría doce; se reporta el piso), 4 derivados a registro de deuda con justificación (todos preexistentes o transversales), 4 refutados con evidencia, y 1 mitigado parcialmente (mitigación aplicada, resto a deuda). Una sola corrida; sin reanudación. El procesamiento (leer, verificar los 20 contra el código, reescribir el plan) tomó unos 6 minutos; el desarrollador aprobó el plan endurecido y la implementación tomó 33 minutos con pruebas automatizadas y prueba de humo exitosas.

4.5 La compuerta de diff

Antes de confirmar los cambios, el flujo de validación previo a la confirmación (*pre-commit*) del proyecto corrió tres pasos: revisión interna del orquestador (5 hallazgos menores, corregidos), revisión de seguridad (limpia), y la ronda del revisor externo sobre el árbol de trabajo, esta vez vía el plugin (modelo gpt-5.5: evidencia directa de que los dos caminos de invocación pueden ejecutar modelos distintos). Veredicto: "needs-attention" (requiere atención), 2 hallazgos. El de severidad alta produjo el intercambio que el autor consideró más valioso del flujo: el revisor sostuvo que la guarda (validación) de versión de la declaración jurada, incorporada en la compuerta de plan, podía bloquear en el campo a un cliente futuro defectuoso. El orquestador primero lo defendió ("el único cliente actual que acepta envía siempre la versión"), verificó, y después reconoció el riesgo real: "si se publica una aplicación con un defecto que no envía versión, el fallo ocurriría al completar, dejando al operario en campo bloqueado". El cierre adoptó la recomendación del revisor, rediseñada como degradación suave: sin versión y con el control desactivado, el trabajo se completa pero la declaración no se ancla ("el anclaje nunca debe apuntar a un texto que el operario no vio: una aceptación sin versión ecoada [devuelta por el cliente] es evidencia sin valor probatorio"), y con el control activo, el error estructurado estándar. El segundo hallazgo resultó un vacío preexistente de otra funcionalidad, verificado y documentado en el registro de deuda. La corrección del primero se aplicó con su prueba renombrada y un caso nuevo; pruebas exitosas, nueve confirmaciones de código cuyos propios mensajes llevan el resultado de las dos rondas.

4.6 Los números del flujo y lo no registrado

El flujo completo, del pedido del plan a la última confirmación, tomó 2 horas 57 minutos, con esta cronología (horas UTC de los historiales): redacción del plan, 19:00 a 19:24 (24 minutos); decisión humana de correr la ronda y semántica de la deduplicación, 2 minutos; ventana de la compuerta de plan, 19:27 a 19:42 (15 minutos: un lanzamiento fallido por sintaxis, el reintento, y 12 minutos 22 segundos de ejecución efectiva del revisor); verificación de los 20 hallazgos y aprobación del plan endurecido, 6 minutos; implementación, 19:48 a 20:21 (33 minutos); pruebas, humo y los dos primeros pasos del flujo previo a la confirmación, 20:21 a 21:48 (unos 87 minutos); corrida de diff, 5 minutos; verificación, corrección final y nueve confirmaciones de código, 21:53 a 21:58 (5 minutos). La ejecución efectiva acumulada de los dos revisores fue de unos 17 minutos de los 177 del flujo. Piezas explícitamente no registradas en los historiales: el prompt interno que el plugin construye para su revisión, los tokens de la segunda ronda, y la verificación de estado del repositorio tras esa segunda corrida. Un flujo, además, no es todos: este tuvo dos eventos de compuerta, cuatro hallazgos refutados con evidencia en la compuerta de plan (la Sección 4.3 muestra uno) y ninguna refutación en la de diff; el corpus contiene eventos con hallazgos de contexto desactualizado y con cero hallazgos (Sección 5).

5 Qué se observó en el corpus

La operación completa (91 eventos de compuerta sobre cinco proyectos; unidades, universos y reconciliaciones en el Apéndice A) mostró dos mecanismos de intervención de naturaleza distinta: la *detección* corrige algo que ya existía; la *conformación* modifica el diseño antes de materializarse. La conformación actúa cuando cambiar es plausiblemente más barato; ni esa ventaja de costo ni la superioridad de los diseños resultantes fueron medidas, y así debe leerse todo lo que sigue. Con la línea de tiempo de modelos del Apéndice A: los casos de detección 1 y 3 corrieron con Opus 4.8 como generador, y el hallazgo de la compuerta de diff del flujo trazado (Sección 4.5) con Fable 5; el modelo del caso 2 no es atribuible con certeza porque su sesión no sobrevivió. De los cinco planes con conteo explícito, cuatro corrieron con Fable 5 y uno con Opus 4.8. Ambos mecanismos se observaron, así, bajo ambos modelos del

orquestador. Sobre el subconjunto auditable del corpus (los 35 eventos de sesiones sobrevivientes con registro numérico), al menos el 64,9 por ciento de los hallazgos fue incorporado (180 de 277) y el 3,6 por ciento fue refutado con evidencia (10 de 277); las reglas de cómputo, las exclusiones y una verificación de consistencia están en el Apéndice A.4.

5.1 Detección: tres casos

Cobro sin comprobante (P2). En la compuerta de diff de un flujo de emisión contra un servicio SOAP de terceros, el revisor detectó que una corrección previa del propio orquestador dejaba responder éxito (HTTP 200) con la llamada SOAP fallida: el sistema confirmaba el cobro de una operación cuyo comprobante nunca se emitía. El caso es compatible con la hipótesis de que una señal externa puede revelar errores omitidos durante la revisión del propio generador, sin que esta experiencia haya medido probabilidades.

Error de dominio (P3). El catálogo de perfiles de acero contenía 0,88 kg/m para el cuadrado macizo de 12 mm, valor del perfil redondo; el correcto es 1,13 kg/m, con la densidad convencional del acero de 7.850 kg/m^3. El código era correcto; el dato era falso. El revisor lo marcó como inconsistente; la corrección se validó por dos vías independientes del código (el peso del producto de referencia cayó en el rango físicamente esperado, y el catálogo se contrastó con datos del fabricante). Una prueba con oráculo independiente del catálogo habría podido detectarlo; las verificaciones cuyo oráculo deriva del mismo dato erróneo, no.

Autorización verificada mal (P1). El revisor detectó una falla de control de acceso a nivel de objeto en una validación que el orquestador había dado por verificada: la segunda opinión corrigió al verificador, no al código.

5.2 Conformación del diseño

Además del flujo narrado en la Sección 4 (11 de 20 hallazgos de plan incorporados; límite inferior cuya ambigüedad se explica en la Sección 4.4), los eventos de plan con conteo explícito registran 7 de 8, 8 de 8, 14 de 14 y 15 de 15 hallazgos incorporados; uno de ellos terminó con el diseño original del generador reemplazado por completo, y otro endureció una migración destructiva de base de datos (validación de la definición real del índice, verificación previa de duplicados y cancelación segura de la reversión) antes de tocar producción. No se agrega una proporción total del corpus: varios ciclos registran conteos no numéricos, y su formalización integra el artefacto pendiente. La bitácora registra la percepción contemporánea del autor (“la ronda de plan fue la de mayor valor; las de diff validaron”); es una apreciación, no una métrica. La conformación es contrafactual por naturaleza: no puede probarse qué defectos habría tenido el diseño original. El corpus documentó por accidente una observación longitudinal, la única observación longitudinal disponible sobre las consecuencias posteriores de una decisión tomada en la compuerta: una semántica aceptada como decisión de producto en una compuerta de plan reapareció cinco rondas después como defecto bloqueante, cuando una interfaz nueva volvió visible la contradicción. Lo que la compuerta dejó pasar tuvo fecha de vencimiento; no se realizó seguimiento longitudinal del resto.

5.3 Falsos positivos y un modo de fallo de contexto

Se documentaron refutaciones duras: nueve fuera del flujo trazado, cuatro dentro de su compuerta de plan (Sección 4.4) y un caso adicional de clasificación ambigua (la regla de conteo se discute en el Apéndice A.4); no se calcula una tasa porque numerador y denominador provienen de universos distintos (Apéndice A). Dos refutaciones comparten un mecanismo que no es un error de razonamiento sino de contexto: el revisor objetó contra una versión desactualizada de su propio prompt (una ventana temporal ya cambiada por decisión de producto posterior; una migración “faltante” que existía desde semanas antes y buscó en el diff, donde no podía estar). Mitigación adoptada: regenerar la instrucción del revisor tras cada decisión intermedia. El modo de fallo es consecuencia directa de una restricción de diseño documentada en las instrucciones (Apéndice C): el revisor ve el repositorio pero no la conversación del orquestador, de modo que todo el contexto le llega, y solo le llega, por la consigna. La refutación corrió en ambas direcciones: el revisor también refutó hipótesis del orquestador en al menos cuatro ocasiones documentadas.

5.4 Lo que el método no detectó

Los defectos no detectados que quedaron documentados siguen un patrón: casi ninguno es lógica del diff revisado. (i) Contrato externo: cinco diferencias entre lo planificado y el XML real de un servicio de terceros, inverificables con los artefactos disponibles en el repositorio, que solo el ejemplo real reveló. (ii) Entorno de ejecución: una declaración faltante en el manifiesto Android que rompía la cámara en todos los dispositivos; la detectó únicamente la prueba de humo en emulador, tras pasar por el revisor y tres revisiones internas. (iii) Estado real del sistema: una prueba integral posterior a cinco sprints reveló 23 hallazgos que ninguna revisión tenía. (iv) Decisiones de producto, que requieren autoridad y responsabilidad humanas. Este patrón habilita una hipótesis rival que conviene confrontar de frente: que los defectos escapados delatan pruebas faltantes antes que revisores faltantes. La respuesta de esta experiencia es que ambas afirmaciones son ciertas y no compiten, porque la revisión estática y la

ejecución real actúan sobre clases de defecto y momentos distintos: las pruebas no pueden revisar un plan (la compuerta donde se concentraron los hallazgos incorporados actúa antes de que exista código que probar), y el revisor no puede ejecutar el entorno real. El caso de la Sección 5.5 muestra las dos caras a la vez: la regresión la cazó un dispositivo real, y su causa habilitante fue la falta de pruebas del componente, no la falta de revisión. En la configuración observada, ambos controles aportaron en momentos y sobre clases de defecto diferentes; esta experiencia no evaluó cuánto del aporte del revisor podría reproducirse mediante especificaciones o pruebas más fuertes.

5.5 La regresión introducida por una corrección

El caso negativo más instructivo: un hallazgo válido del revisor (pérdida de archivos locales ante una caída de la aplicación) recibió una corrección cuyo mecanismo de mitigación introdujo una regresión distinta (duplicación), que sobrevivió al cierre del ciclo, fue atribuida erróneamente a otra causa por el orquestador, y la detectó el desarrollador en un dispositivo real dos días después. La causa habilitante: el componente carecía de pruebas por depender de API de plataforma. La lección consignada en la bitácora: una corrección derivada de un hallazgo es también código nuevo sin cobertura, y merece su propio ciclo o sus propias pruebas antes de darse por cerrada. Es la única regresión de este tipo documentada; pueden existir otras no observadas.

6 Lecciones operativas preliminares

Hipótesis de operación surgidas del corpus, no propiedades establecidas; varias descansan en observaciones únicas. (1) Los episodios consignados como más valiosos en la bitácora se concentraron en la compuerta de plan; las de diff aportaron mayormente validación y detección puntual. Es experiencia reportada, no comparación cuantitativa. (2) El método no produjo valor observable en el único ciclo sobre una refactorización mecánica ya verificada por ejecución. (3) Una secuencia de siete eventos consecutivos sobre cambios sucesivos de una misma funcionalidad mostró indicios de rendimientos decrecientes: el séptimo produjo el único falso positivo de la serie; la observación es única y no permite atribución causal a la repetición. (4) Las instrucciones del revisor caducan con cada decisión de producto intermedia. (5) Los descartes y decisiones diferidas tienen fecha de vencimiento. (6) La bitácora de calibración versionada junto al código resultó imprescindible: es la memoria que sobrevivió a la retención de los historiales, y los once ciclos que no se registraron contemporáneamente requirieron reconstrucción posterior.

7 Propuesta organizacional (no evaluada)

Separadamente de la experiencia reportada, el autor propuso a su organización un protocolo que extiende el principio al ciclo de vida completo (relevamiento con captura de audio consentida, arquitectura comparada con ataque cruzado y registro de decisión, las dos compuertas descritas, y pruebas con los controles preexistentes intactos), con niveles de criticidad y reglas de gobernanza de datos. Se encuentra en discusión para un piloto: es una propuesta de diseño informada por la experiencia individual, no una práctica organizacional evaluada.

8 Limitaciones

(1) *Diseño*: sin comparación controlada, la atribución causal a la diversidad de proveedor es hipótesis; "familia" se usa en su sentido operacional más débil (proveedor distinto, sin linaje de destilación conocido en común). (2) *Datos*: conteos de historiales generados por el propio sistema y de una bitácora escrita por el autor; parte del período reconstruido retrospectivamente; un solo practicante, dominios específicos. (3) *Oráculos*: la ejecución arbitra solo aquello para lo que existe un oráculo independiente; compilar y pasar pruebas escritas desde la misma especificación compartida no garantiza corrección. (4) *Balance neto*: se documentaron las refutaciones duras del Apéndice A.4 y una única regresión inducida por una corrección, pero no se midió el tiempo de refutación ni el costo económico total; la degradación benigna por omisión coincidente vale solo para omisiones: los cambios inducidos por la revisión pueden empeorar la línea de base, sea por un hallazgo incorrecto que induce una modificación perjudicial (no documentado en este corpus) o por la corrección mal implementada de un hallazgo válido (este corpus contiene un caso). (5) *Extracción*: minería auto-reportada sin auditoría manual sistemática ni reglas de deduplicación formalizadas; el modelo del orquestador varió y su línea de tiempo se reconstruyó por sesión (Apéndice A), aunque no toda corrida es atribuible a un modelo; el del revisor solo está registrado donde su salida lo persistió (en el flujo trazado: gpt-5.6-sol y gpt-5.5, distintos entre las dos vías de invocación). El informe no describe las condiciones contractuales de procesamiento del código por los proveedores. (6) *Correlación residual*: los modelos comparten datos de entrenamiento y la evidencia indica dependencia de fallas incluso variando modelo, agente y lenguaje [18]; el patrón de defectos no detectados (contrato externo, entorno real, producto) es consistente con límites que ninguna diversidad de asistente elimina.

9 Trabajo futuro

Primero, instrumentación prospectiva y un artefacto de replicación anonimizado: registrar por ciclo hallazgos, veredictos, el modelo efectivo de cada asistente, tiempo de refutación y destino final, y publicar la matriz completa de resultados por compuerta y categoría, con la taxonomía, las consignas, el procedimiento de minería y una muestra auditada manualmente; el total de hallazgos válidos, hoy no agregado, es su primera pieza. Segundo, el experimento que la tesis causal requiere: un mismo conjunto de cambios en cuatro condiciones (sin revisión, autorrevisión, revisor del mismo proveedor, revisor de proveedor distinto), con oráculos independientes, midiendo por separado detección y conformación. Una primera versión de ese experimento sobre tareas autocontenidas ya existe [12], con condiciones solas, de autorrevisión y de cruce en ambas direcciones; quedan pendientes la escala de repositorio, la compuerta de plan y la separación de mecanismos: es concebible que la diversidad importe de manera distinta en cada mecanismo, y la literatura citada aporta evidencia solo indirecta, y únicamente sobre el primero. Tercero, el piloto organizacional, si su discusión prospera, con estas métricas desde el primer día.

10 Conclusión

En 52 días de operación real sobre cinco proyectos, la revisión adversarial automatizada por un segundo asistente operó mediante dos mecanismos. Se documentaron casos de detección de defectos omitidos o introducidos por el generador, incluidos errores de dominio y de autorización; separadamente, la enumeración disponible registra trece hallazgos refutados con evidencia (nueve fuera del flujo trazado y cuatro dentro; procedencia y límites en el Apéndice A.4) y un caso adicional ambiguo, mientras que el total de hallazgos válidos no fue agregado y constituye la primera pieza del artefacto pendiente. Y conformó el diseño en la compuerta de plan, llegando en un caso a reemplazar por completo la arquitectura propuesta; si esos diseños fueron objetivamente superiores no puede determinarse con esta experiencia. Los defectos no detectados documentados se concentraron en contratos, estados u oráculos no disponibles para la revisión estática del conjunto de cambios, y en decisiones de producto que requieren autoridad humana. El criterio de terminación por disposición individual y evidencia, en lugar de consenso, se aplicó en los 91 eventos de compuerta documentados; no se identificaron abortos entre los eventos contabilizados, aunque el procedimiento no reconstruyó sistemáticamente intentos iniciados y no registrados; el caso más instructivo del corpus fue una regresión introducida por la corrección de un hallazgo válido, que recuerda que el método no exime de una cobertura de pruebas independiente. Si la diversidad de proveedor es la causa de la detección observada, y si el balance neto es favorable una vez contados todos los costos, son preguntas que esta experiencia motiva y deja abiertas; el diseño experimental para responderlas está especificado.

Referencias

[1] L. K. Hansen y P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[2] D. Wood, T. Mu, A. M. Webb, H. W. J. Reeve, M. Luján y G. Brown. A unified theory of diversity in ensemble learning. *Journal of Machine Learning Research*, 24(359):1–49, 2023.

[3] J. Huang et al. Large language models cannot self-correct reasoning yet. En *ICLR*, 2024. arXiv:2310.01798.

[4] G. Tyen et al. LLMs cannot find reasoning errors, but can correct them given the error location. En *Findings of ACL*, páginas 13894–13908, 2024. arXiv:2311.08516.

[5] K. Tsui. Self-correction bench: Uncovering and addressing the self-correction blind spot in large language models. Preprint, 2025. arXiv:2507.02778.

[6] A. Panickssery, S. R. Bowman y S. Feng. LLM evaluators recognize and favor their own generations. En *NeurIPS*, 2024. arXiv:2404.13076.

[7] Y. Du, S. Li, A. Torralba, J. B. Tenenbaum e I. Mordatch. Improving factuality and reasoning in language models through multiagent debate. En *ICML*, PMLR 235:11733–11763, 2024. arXiv:2305.14325.

[8] J. Wang et al. Mixture-of-agents enhances large language model capabilities. En *ICLR*, 2025. arXiv:2406.04692.

[9] X. Tang, K. Kim, Y. Song, C. Lothritz, B. Li, S. Ezzini, H. Tian, J. Klein y T. F. Bissyandé. CodeAgent: Autonomous communicative agents for code review. En *EMNLP*, páginas 11279–11313, 2024. arXiv:2402.02172.

[10] H. Zhang, W. Cheng, Y. Wu y W. Hu. A pair programming framework for code generation via multi-plan exploration and feedback-driven refinement. En *ASE*, páginas 1319–1331, 2024. arXiv:2409.05001.

[11] X. Chen et al. Revisit self-debugging with self-generated tests for code generation. Preprint (trabajo en progreso), 2025. arXiv:2501.12793.

[12] Z. Xiang, Y. Zhang, Y. Zhang y H. Xu. Cross-model LLM code review: Should you use Claude to review Codex or vice versa? En *Agentic Software Engineering (SE 3.0) Workshop*, 2026. arXiv:2607.21656.

[13] E. S. Qiu y J. Gill. Adversarial review: Cooperative code review through structured disagreement. En *AI4GOOD Workshop, ICML*, 2026.

[14] A. Agarwal. Refute-or-promote: An adversarial stage-gated multi-agent review methodology for high-precision LLM-assisted defect discovery. Preprint, 2026. arXiv:2604.19049.

[15] A. Avižienis. The N-version approach to fault-tolerant software. *IEEE Transactions on Software Engineering*, SE-

11(12):1491–1501, 1985.

[16] J. C. Knight y N. G. Leveson. An experimental evaluation of the assumption of independence in multiversion programming. *IEEE Transactions on Software Engineering*, SE-12(1):96–109, 1986.

[17] J. C. Knight y N. G. Leveson. A reply to the criticisms of the Knight & Leveson experiment. *ACM SIGSOFT Software Engineering Notes*, 15(1):24–35, 1990.

[18] J. Ron, B. Baudry y M. Monperrus. N-version programming with coding agents. Preprint, 2026. arXiv:2606.20158.

A Datos de operación

Este apéndice concentra la contabilidad de la operación: unidades, universos de cobertura, reconciliaciones y procedencia. Existe para que las afirmaciones del cuerpo sean auditables; sus límites se declaran en la Sección 8 del cuerpo.

A.1 Contexto y procedencia de las fuentes

La experiencia corresponde a un único desarrollador (el autor), entre el 2 de junio y el 23 de julio de 2026 (52 días, ocho semanas calendario), sobre cinco proyectos .NET y TypeScript identificados como P1 a P5: una plataforma de trazabilidad para el sector agroindustrial, en producción (P1); una aplicación web comercial con integración SOAP a servicios de terceros (P2); una aplicación pública de cálculo técnico, en producción (P3); un sistema de gestión para una institución deportiva (P4); y un prototipo de gestión de socios (P5). **Nota de confidencialidad**: los proyectos no se identifican por nombre y los detalles que permitirían identificar sistemas u organizaciones de terceros fueron generalizados; los datos operativos no fueron alterados.

Fuentes: (i) minería de 345 archivos de transcripción de sesión del orquestador (JSONL, parseados defensivamente) más artefactos persistidos de sesión (consignas y salidas del revisor); (ii) una bitácora de calibración que registra hallazgos, veredictos y refutaciones, escrita contemporáneamente al cierre de cada ciclo para 54 rondas de P1; otros 11 ciclos de P1 se reconstruyeron después desde los historiales. La retención por defecto de los historiales es de 30 días: las sesiones anteriores al 16 de junio ya no existían al momento del análisis (una única excepción del 12 de junio, sin corridas del revisor); para ese período la fuente es la bitácora. El informe de minería fue generado por el propio orquestador bajo la regla explícita de responder "no encontrado" ante ausencia de datos; su carácter auto-reportado se discute en las Limitaciones del cuerpo.

A.2 Instanciación y línea de tiempo de modelos

Orquestador: Claude Code (Anthropic), que ejecutó dos modelos durante el período. Línea de tiempo reconstruida del campo de modelo de los historiales sobrevivientes: del 16 de junio al 1 de julio todas las sesiones registran Claude Opus 4.8, período coincidente con la suspensión global de Claude Fable 5 (lanzado el 9 de junio de 2026, suspendido el 12 y restablecido el 1 de julio); el retorno quedó registrado dentro de una sesión iniciada el 30 de junio, con 421 mensajes de Opus 4.8 seguidos de 71 de Fable 5. Del 2 al 23 de julio predomina Fable 5, con excepciones documentadas: una sesión del 7 y 8 de julio que volvió a Opus 4.8 a mitad de trabajo, tres sesiones de mediados de julio con mensajes de ambos modelos, y el único ciclo de P5 (23 de julio) ejecutado íntegramente sobre Opus 4.8. Para los ciclos del 2 al 15 de junio no hay sesiones sobrevivientes que registren el modelo, y los anteriores al 9 de junio corrieron necesariamente sobre un modelo previo al lanzamiento de Fable 5. Ninguna afirmación del informe distingue efectos por modelo.

Revisor: Codex CLI (OpenAI), integrado por el plugin oficial `openai/codex-plugin-cc`. Corridas directas con `codex exec` bajo un permiso permanente con aprobación automática; cinco corridas vía los comandos del plugin, correspondientes a cinco eventos con una corrida cada uno (configuración e instrucciones reproducidas en el Apéndice C). Ambos caminos comparten la consigna adversarial, pero ejecutan modelos potencialmente distintos (en el flujo trazado en el cuerpo: gpt-5.6-sol por CLI y gpt-5.5 por plugin) y se agregan aquí de forma descriptiva. Se instruyó al revisor para no modificar el repositorio y se inspeccionó el estado de Git tras las corridas; este control detectó un incidente (reescritura de terminadores de línea de cinco archivos de instantáneas, contenido intacto, revertida) y no constituye aislamiento técnico completo: no impide escrituras temporales, efectos sobre archivos ignorados ni accesos de red.

A.3 Unidades, universos y conteos

Unidades: una *corrida* es una invocación efectiva del revisor; un *ciclo* (o *evento de compuerta*) es la aplicación de una compuerta, plan o diff, a una unidad de trabajo, y puede constar de una o más corridas; una *unidad de trabajo* es una funcionalidad, corrección o refactorización; un *flujo completo* es una unidad que atravesó ambas compuertas; una *ronda* es una entrada de la bitácora de P1. Los 91 conteos de este informe son eventos de compuerta, no flujos completos; cuántas unidades pasaron por solo plan, solo diff o ambas compuertas no fue agregado e integra el artefacto pendiente. Universos: las corridas se contaron sobre las sesiones sobrevivientes (posteriores al 2026-06-16): los historiales registran 81 eventos de invocación directa, 79 únicos tras deduplicar dos sesiones bifurcadas;

Proyecto	Eventos	Período	Corridas
P1	54 + 11	02/06 a 22/07	48 + 5
P3	11	01/07 a 17/07	14
P4	10	15/07 a 17/07	10
P2	4	16/06 a 08/07	4
P5	1	23/07	1

Tabla 1: Eventos de compuerta por proyecto (fechas de 2026, formato día/mes). En P1, 54 rondas de bitácora más 11 eventos reconstruidos de sesión; corridas vía CLI más plugin, contadas solo en sesiones sobrevivientes; la distribución asigna 77 de las 79 corridas directas.

la reconstrucción por proyecto asigna 77 de esas corridas, más 5 vía plugin (Tabla 1), quedando 2 corridas directas sin asignación de proyecto. Los ciclos se contaron sobre el corpus completo (bitácora más sesiones): 91, de los cuales 65 en P1 (54 rondas de bitácora, que incluyen ciclos cuyas sesiones ya no existen, más 11 reconstruidos) y 26 en los restantes. Hay más ciclos que corridas porque los ciclos anteriores al 16 de junio carecen de corridas sobrevivientes. El conteo de hallazgos, aproximadamente 330, cubre solo las sesiones sobrevivientes; su valor exacto depende de reglas de deduplicación aún no formalizadas.

A.4 Observaciones cuantitativas

Con sus límites: (1) Refutaciones duras documentadas: nueve fuera del flujo trazado, cuatro dentro de su compuerta de plan, y un caso adicional de clasificación ambigua. La regla de conteo no fue formalizada, y la fuente contiene una inconsistencia interna que se declara: el artefacto de extracción encabeza su sección de refutaciones con "9 o 10" mientras su propia enumeración lista trece hallazgos individuales; versiones previas de este informe propagaron el encabezado. El conteo aquí reportado (nueve más cuatro, trece por hallazgo individual, más el ambiguo) proviene de la enumeración, y su auditoría definitiva integra el artefacto pendiente. No se calcula tasa por la asimetría de universos, y la categoría "refutación dura" es estrecha: los hallazgos no accionables por otras vías (preexistencia, alcance, decisión de producto, contexto desactualizado) fueron más numerosos, y la matriz completa por compuerta y categoría queda como artefacto pendiente. (2) Un evento de compuerta sobre una refactorización mecánica ya verificada por prueba de humo produjo cero hallazgos; observación única. (3) Costo, como indicador grueso: las corridas con registro de tokens consumieron entre 58.000 y 284.000 tokens totales según el propio registro del revisor (sin desglose de entrada y salida; el extremo superior corresponde a la corrida de plan del flujo trazado, con esfuerzo de razonamiento alto); el tiempo de refutación de falsos positivos no fue medido. En el único flujo con cronología completa (Sección 4.6), el tiempo de pared atribuible a la revisión (las ventanas de las dos corridas más la verificación de hallazgos y el cierre) fue de entre 26 y 31 de los 177 minutos, un 15 a 18 por ciento según se impute o no el cierre; es una observación única, no una medición del corpus, y su traducción a costo económico requiere tarifas que este informe no fija. (4) La norma fue una corrida por compuerta; la reanudación (reutilizar la sesión del revisor) aparece 3 veces en el corpus, siempre para confirmar una corrección. Las "pasadas" de los extremos (8 en un día sobre una misma funcionalidad; una serie de 7 consecutivas de diff cuya séptima produjo el único falso positivo de la serie) fueron eventos de compuerta distintos, con corridas nuevas sobre lotes de cambios sucesivos, no reanudaciones ni corridas repetidas sobre el mismo diff, por lo que no contradicen la norma. (5) Tasas sobre el subconjunto auditable. Para dar un piso de la relación señal-ruido con numerador y denominador del mismo universo, se agregaron las filas por evento del informe de minería correspondientes a sesiones sobrevivientes con conteos numéricos: 35 de los 37 eventos de ese universo. Reglas: en las dos filas con rangos se tomó el piso del numerador y el techo del denominador; se excluyeron y declaran dos eventos con registros no numéricos (uno con 20 hallazgos y disposición registrada como "mayoría", otro con 12 y disposición integrada sin conteo); los anotados como incorporados más tarde o ya cubiertos no se contaron en el piso. Resultado: 277 hallazgos, al menos 180 incorporados (64,9 por ciento), 10 refutados con evidencia (3,6 por ciento); el resto corresponde a derivaciones, riesgos aceptados y descartes con evidencia, cuya matriz completa sigue pendiente. Verificación de consistencia independiente: los 10 refutados de este subconjunto, más los 3 documentados únicamente en la bitácora (fuera de este universo), suman exactamente los 13 de la enumeración del punto (1). La diferencia con el conteo aproximado de 330 corresponde principalmente a los dos eventos excluidos y a reglas de deduplicación no formalizadas. Es un piso computado sobre el artefacto auto-reportado, no una auditoría; hereda sus límites.

B Receta operacional

Síntesis para reproducir el flujo sin reconstruirlo desde la narración.

1. **Activación.** Definir por escrito qué unidades se revisan; la operación reportada usó la barra de impacto formalizada en sus instrucciones globales (Sección 4.1 y Apéndice C).
2. **Consigna de plan.** Rol adversarial explícito con prohibición de aprobar, contexto del sistema, el plan completo, pedidos de ataque específicos del dominio, y formato de salida con severidad y evidencia archivo:línea por hallazgo; la consigna real del flujo trazado está en la Sección 4.2.
3. **Consigna de diff.** Mismo rol y formato aplicados al

conjunto de cambios del árbol de trabajo, regenerada tras cada decisión de producto intermedia (Sección 5.3).

4. **Registro mínimo por hallazgo.** Identificador, severidad, evidencia citada, veredicto del generador, estado de disposición (Sección 3.2), autoridad que resolvió o aceptó el riesgo, y fecha; en una transferencia abierta, el destinatario responsable.
5. **Repetición.** Una corrida por compuerta; reanudar solo para confirmar una corrección; cortar ante severidad decreciente con repetición (heurística de un episodio único: Sección 6).
6. **Refutación.** Exige evidencia verificable (cita de código, medición, o comportamiento documentado del entorno); sin evidencia, el hallazgo se transfiere, no se descarta.
7. **Sin oráculo ejecutable.** Los hallazgos que dependen de contratos externos o de intención de producto se transfieren al desarrollador.
8. **Aislamiento.** Instruir al revisor en solo lectura e inspeccionar el estado del repositorio tras cada corrida, sabiendo que esto no constituye aislamiento técnico completo (Apéndice A.2).

C Instanciación de referencia (julio de 2026)

Este apéndice reproduce la configuración que materializó el flujo, como registro histórico, no como tutorial: corresponde a las versiones de las herramientas de julio de 2026, los mecanismos de integración cambian con rapidez, y el lector debe verificar contra la documentación vigente. La instalación de las herramientas en sí queda fuera de alcance (documentación de los proveedores).

C.1 Configuración del orquestador

Fragmento relevante del archivo de configuración de Claude Code (`settings.json`), que materializa la automatización: el permiso permanente para invocar al revisor sin aprobación por llamada, y el plugin oficial de integración.[1]

```
{
  "permissions": {
    "allow": [ "Bash(codex exec:*)" ],
    "defaultMode": "auto"
  },
  "model": "claude-fable-5[1m]",
  "enabledPlugins": {
    "codex@openai-codex": true,
    "claude-security@claude-plugins-official": true
  },
  "extraKnownMarketplaces": {
    "openai-codex": {
      "source": {
        "source": "github",
        "repo": "openai/codex-plugin-cc"
      }
    }
  }
}
```

El segundo plugin habilitado provee la revisión de seguridad del flujo previo a la confirmación (el paso 2 del flujo descrito en la Sección 4.5).

C.2 Instrucciones del orquestador

Las reglas que gobiernan las rondas viven en el archivo de instrucciones globales del orquestador (transcripción curada, con ajustes tipográficos menores). La política de activación completa se reprodujo en la Sección 4.1; las reglas operativas restantes, textuales: el prompt debe pedir al revisor que ataque el plan o el diff, no que apruebe; el resultado se presenta endurecido, diciendo qué cazó el revisor y qué se hizo con cada punto; en cambios de seguridad convienen ambas compuertas; y los hallazgos se verifican contra el código, sin creerle a ciegas (“decorrelación en ambas direcciones”). El archivo documenta además la restricción estructural: el revisor ve el repositorio (mismo directorio de trabajo) pero no la conversación del orquestador, por lo que el plan, el diff y el objetivo deben pasarse completos en la consigna.

La vía de invocación en Windows, ejecutada a través de la herramienta Bash del orquestador (que provee un entorno Bash propio en Windows; el comando no es ejecutable directamente en PowerShell ni cmd.exe), verificada el 29 de mayo de 2026 (el sandbox del revisor no inicializa al ser invocado desde el orquestador; la mitigación es el aislamiento por instrucción más la verificación posterior, con los límites declarados en el Apéndice A.2):

```
codex exec \
  --dangerously-bypass-approvals-and-sandbox \
  > "$HOME/out.txt" 2>&1 <<'PROMPT'
AVISO - SIN SANDBOX: NO escribas/edites/
borres/muevas archivos ni corras git que
modifique estado. SOLO lee y analiza.
<plan o diff + objetivo en texto>.
Actua como red-team: ataca supuestos,
agujeros, riesgos y casos borde. El codigo
esta en este repo (mismo cwd), miralo.
PROMPT
```

Reglas acompañantes: lanzar en segundo plano y redirigir la salida a un archivo (incluye el registro de exploración y es grande; no confiar en el código de salida); ejecutar `git status` después de cada corrida para confirmar que el repositorio quedó intacto; y continuar una conversación existente con `codex exec resume` únicamente para confirmar una corrección. Cuando el paso del prompt por la shell falló (el caso del flujo trazado, Sección 4.2), la variante robusta fue escribir la consigna a un archivo y pasarla por sustitución de comando, es decir, entregando `"$(cat consigna.txt)"` como argumento de `codex exec` (las comillas evitan la separación por espacios y la expansión de comodines).

[1] El sufijo `[1m]` en `model` es la sintaxis de alias de Claude Code para la variante de contexto extendido de un millón de tokens.

C.3 Estructura de la bitácora de calibración

Un archivo de texto versionado junto al código del proyecto, con una entrada por ronda: número y fecha; unidad de trabajo y compuerta; veredicto global; cada hallazgo con severidad, cierre (estados de la Sección 3.2) y justificación; falsos positivos con su refutación; y una nota de calibración cuando un evento enseñó algo sobre el proceso mismo (las lecciones de la Sección 6 provienen de esas notas). Es el registro mínimo por hallazgo del Apéndice B, más la memoria de proceso que sobrevive a la retención de los historiales.