El Ciclo de Ejecución Semanal (Sprint)

Capítulo 3 de «Soberanía de la Intención» · Marcos Fernández Otero

CAPÍTULO DE MUESTRA

✳ Metodología Designio · designio.dev

3. El Ciclo de Ejecución Semanal (Sprint)

Cuando pensaba en una metodología agéntica y en cómo podía afectar en los tiempos de un proyecto y en los tiempos de un sprint de un Agile clásico (me parece increíble, pero desde mi punto de vista, la metodología Agile ya es una metodología clásica), me di cuenta de que el desarrollo era tan rápido que el embudo del tiempo era las tareas humanas a realizar en el proyecto.

Me quedó claro desde el principio un axioma para definir la metodología Designio, se puede usar sprints en las implantaciones, pero tienen que ser tan cortas como el QA humano que valida UAT tenga la capacidad de validar y dar el go al pase a producción y que el Intent Director y BO sean capaces de crear y/o validar las EIS que se generen para un sprint.

Este Sprint lo definimos en una semana, porque al menos se necesitará un día con el BO para refinar los requisitos y generar las EIS con el IAR bajo, otro día de Intent Director para aprobarlo, otro día de construcción y otro día para las pruebas por parte de QA, por lo que una semana parecía justo y necesario lo que se necesita para tener garantías de una entrega productiva y mostrar muy rápidamente una primera versión o MVP de lo que se está construyendo.

Sin embargo, el objetivo de esta metodología es dar las herramientas necesarias para que cada vez sea más rápido y sobre todo que en cada sprint tengamos mayor capacidad de hacer más cosas y por tanto desplegar más líneas de código, dicho esto, Designio es un marco de trabajo elástico, donde se puede recortar los tiempos de la semana, o bien incrementar el número de EIS por sprint.

Este ciclo sustituye la ahora arcaica estimación por "puntos de historia" por métricas de rendimiento real, capacidad de procesamiento de tokens y validación de la intención construida.

Como hemos comentado anteriormente esta metodología tiene un cumplimiento regulatorio por diseño y se garantiza la trazabilidad total y el cumplimiento de estándares de grado empresarial, el ciclo se divide en fases críticas que cubren desde la captura de la intención hasta el despliegue en producción.

Se optimiza el tiempo humano hacia la supervisión estratégica, el arbitraje de diseño y el ajuste de precisión, sin olvidarnos que en dichas tareas también se cuenta con la ayuda de IA, eliminando las tareas repetitivas de codificación base.

Fase 1: Lunes (AM) - Ingesta Crítica y Protocolo de Interrogación (AI-Refiner)

El objetivo del primer día del sprint es ejecutar la captura de la intención de negocio para transformarla en una especificación técnica ejecutable denominada EIS (Executable Intent Specification).

El lunes es el día de mayor densidad de trabajo humano. Es el punto crítico que determina el éxito o el fracaso del sprint. Todo lo que se defina en el EIS y se decida construir durante la semana deberá ser validado después por el equipo de QA.

Antes de llegar a QA, el Intent Director también realiza pruebas en el entorno de desarrollo para asegurarse de que lo construido cumple exactamente con lo especificado en el EIS. No debemos intentar "validar de más" ni ampliar tiempos de QA hasta que hayamos completado varios sprints y demostrado que podemos reducir los tiempos de pase a producción. Ese momento llega cuando los agentes de QA ya son capaces de cubrir alrededor del 80% del trabajo de detección y corrección de bugs.

El lunes va a marcar el éxito del sprint, según el contenido y densidad que se quiera.

En este primer día el BO, junto al Intent Director tienen el protagonismo total. Primero el Intent Director activará al agente AI Interrogador, el cual confrontará los requisitos contra una posible ambigüedad, obligando al BO a contestar todo lo necesario para que las EIS que se generen tengan la mínima ambigüedad.

Una decisión que conviene explicar es hasta dónde refinar el requisito antes de dejar construir al agente. En mis proyectos trabajo con un margen de ambigüedad residual de en torno al 5%. No es una regla fija ni un número mágico: es el punto que a mí me ha funcionado. Y, sobre todo, no busco bajar ese margen a cero a propósito. Dejar un pequeño espacio de interpretación tiene una ventaja: el agente, para rellenar ese hueco, tira de buenas prácticas, y a veces da con una solución mejor que la que tú tenías en la cabeza.

Me pasó en un proyecto. Pedí que dentro de la ficha de cliente aparecieran las noticias de ese cliente, recogidas de un proveedor de noticias externo. Yo tenía una idea más o menos convencional de cómo mostrarlas. El agente lo interpretó a su manera y me lo presentó como un banner en movimiento que cruzaba la pantalla, igual que el teletipo de cotizaciones que sale en la parte baja de Bloomberg TV. No era lo que yo había imaginado. Pero era mejor. Gustó más al Business Owner, gustó más al cliente, y se quedó tal cual.

Esa es la verdadera razón del margen. No se trata solo de ahorrarte trabajo de especificación: se trata de que, si aprietas el requisito hasta dejarlo sin un milímetro de holgura, le quitas al agente la posibilidad de aportar lo que tú no has visto. La clave está en calibrar ese margen: el suficiente para que el agente pueda sorprenderte para bien, el justo para que no se vaya por las ramas. Ese equilibrio, en mi experiencia, está alrededor de ese 5% de ambigüedad que te permite dormir tranquilo sin atar al agente de pies y manos.

Estas sesiones de trabajo junto al Business Owner del cliente están dirigidas a partir de la plantilla maestra de las EIS que contiene todos los puntos necesarios que hay que despejar de un requisito para generar las "n" especificaciones atómicas de intención ejecutables (EIS).

El agente interrogador, además, tiene otra habilidad y tarea a realizar que es el cruce del requisito con el mapa del proyecto, donde tenemos representados los módulos del sistema, los microservicios y las dependencias entre ellos, para responder a la pregunta: ¿qué partes del sistema se ven afectadas por la implantación de esta EIS?

Fase 2: Lunes (PM) - Certificación del EIS y Alineación de Negocio

La fase 2 también se realiza el mismo lunes, es donde se asegura el trabajo que se va a realizar. Es importante cerrar la ambigüedad y la priorización o selección de las EIS que se van a construir, ya que por metodología no se permite continuar a construcción sin una validación explícita y firmada de las Especificaciones de Intención Ejecutables (EIS) que se quieren implementar.

Esta fase consiste en la certificación de las EIS por parte del Delivery Lead y del Intent Director (algunas veces la misma persona tendrá los dos roles). Estos roles deben validar que las respuestas del cliente, en la figura de Business Owner, a las preguntas del agente interrogador son satisfactorias y viables. Aunque por metodología no es algo necesario o marcado, es muy buena práctica en mi opinión, que los BO también acepten estas EIS y las certifiquen, de este modo no tendremos problemas de gestión de alcance durante el proyecto, sin embargo, para esto es necesario que el BO tenga las capacidades técnicas necesarias para entender la disgregación de su intención en especificaciones más atómicas y en algunas ocasiones algo más técnicas. En el caso de que el BO no tenga estas capacidades, lo que sí recomiendo es al menos informar sobre las EIS que se van a construir y la relación con sus requisitos completos (después de la sesión de refinamiento) antes de comenzar con la construcción.

Conviene aclarar con qué criterio decide el Business Owner si una EIS merece aprobarse, porque no todo el valor de una intención es del mismo tipo. En la práctica, una EIS se puede justificar por tres clases distintas de retorno. La primera es el ROI directo: un valor funcional inmediato y medible, como reducir el tiempo de alta de un cliente de cinco minutos a treinta segundos. La segunda es el ROI indirecto: una EIS que no da valor por sí misma hoy, pero que habilita valor futuro, como crear una tabla de perfiles de riesgo que permitirá más adelante un módulo de scoring de fraude. Y la tercera es el ROI de riesgo o gobernanza: una EIS que reduce deuda técnica, riesgo operativo o coste futuro, como añadir registros estructurados para cumplir una auditoría regulatoria. Mientras una EIS se pueda justificar en al menos una de estas tres categorías, el Business Owner tiene base para aprobarla; si no encaja en ninguna, conviene preguntarse si de verdad merece construirse.

Hay un mecanismo más que se define en este momento de certificación y que conviene explicar, porque es la frontera última entre la autonomía de la IA y la responsabilidad humana: los puntos de control de soberanía. Son hitos obligatorios en los que la IA, a través del Agente Guardián, tiene que detenerse y pedir confirmación humana antes de proceder, en lugar de decidir por su cuenta. No se activan para cualquier cosa, sino para los cambios que pueden comprometer la visión a largo plazo, el coste o la seguridad: tocar el modelo de datos, alterar la persistencia, disparar el gasto de infraestructura o modificar protocolos de seguridad críticos. En esos casos, el humano actúa como juez de riesgo, y la IA no avanza sin su firma.

Lo veo más claro con ejemplos reales de situaciones en las que el sistema debe pararse en seco. Si la IA propone eliminar un campo legacy_id porque no lo ve usado en el código, STOP: puede ser una clave de migraciones o de auditoría que no se usa en el código pero que sostiene cosas fuera de él. Si propone escalar el clúster de tres a doce nodos para ir más rápido, STOP: tiene impacto directo en la factura mensual. Si propone desactivar el doble factor de autenticación para agilizar las pruebas automatizadas, STOP: prohibido. Si propone migrar de una arquitectura monolítica a microservicios,

STOP: es una decisión estratégica, no técnica. Si propone cambiar el formato de respuesta del endpoint de pagos, STOP: puede romper a clientes externos que dependen de él. Si propone cambiar la lógica de cálculo de comisiones, STOP: tiene impacto financiero directo. En todos esos casos, el patrón es el mismo: la IA puede proponer, pero quien decide es el humano, porque son cambios cuyo coste de equivocarse es demasiado alto para delegarlo.

Técnicamente, esto se implementa dotando al Agente Guardián de una serie de disparadores que vigilan precisamente esos territorios sensibles: cambios de esquema de datos, cambios en políticas de seguridad, incrementos de coste por encima de un umbral, modificaciones de contratos con sistemas externos o cambios en reglas de negocio críticas. Cuando uno de esos disparadores salta, la IA se detiene y envía una solicitud de permiso al Intent Director, que es quien autoriza o deniega.

Y aquí va una recomendación personal, nacida de la experiencia. En los primeros proyectos que dirija un Intent Director, mi consejo es que sea él mismo quien asuma el papel del Agente Guardián, operándolo a mano en lugar de delegarlo del todo. Hacerlo así, aunque al principio sea más lento, tiene un valor enorme: el Intent Director entiende desde dentro qué tareas y qué orquestación le corresponden al Guardián, dónde están los puntos de control que de verdad importan y cómo se comporta el sistema cuando algo se detiene. Esa comprensión profunda no se adquiere leyendo un manual, se adquiere haciéndolo. Y una vez interiorizada, los proyectos siguientes se ejecutan mucho mejor y más rápido, porque quien dirige sabe exactamente qué está automatizando y por qué.

Bien, ya sabemos todo lo que tenemos que hacer el lunes en un sprint de Designio, o por lo menos tenemos la intuición de todo lo que hay que realizar. Pero ¿cómo? Voy a intentar detallarlo a continuación.

Me gustaría contaros una experiencia personal de este primer día con esta metodología, pero la verdad es que siempre me he encontrado con cosas muy diferentes según el cliente y según la persona que hacía la labor de BO. Es aquí donde el Intent Director se consagra y se gana la credibilidad necesaria para el resto del proyecto. Mi consejo, después de varias iteraciones, es claro: mejor empezar con menos requisitos y cumplir la promesa de entrega en una semana, que abarcar de más y acabar pidiendo dos. Recordemos que la capacidad agéntica debe reducir los días de los sprints clásicos, no añadirles trabajo.

La semana operativa arranca con la recogida de peticiones. El Agente Guardián actúa como punto único de entrada para todo lo que llega desde los canales autorizados: Jira, Slack, correo, voz, lo que use el cliente. Su trabajo en este primer momento es sencillo de enunciar y complicado de hacer bien: filtrar el ruido y dejar pasar solo aquello sobre lo que merece la pena que la IA piense.

Cada petición pasa por un proceso de normalización. El agente traduce el lenguaje del cliente, que puede venir en formato muy variado (un ticket de Jira con cinco líneas, un audio de WhatsApp, un correo largo), a una estructura común que el resto del sistema sabe leer. Durante esta normalización se descartan duplicados, peticiones que en realidad ya estaban cubiertas por otra anterior, y solicitudes que quedan fuera del alcance contratado con el cliente. Esta limpieza es importante porque cada petición que pasa este filtro va a consumir tokens más adelante, y dejar entrar ruido sale caro.

Una vez normalizada la petición, el agente la clasifica. Designio distingue cuatro tipos según el flujo de trabajo que vayan a requerir:

- **Tipo A** son los evolutivos críticos: cambios en lógica de negocio, modificaciones en base de datos, integraciones nuevas con sistemas externos.
- **Tipo B** son los correctivos: bugs, parches de seguridad, fallos de rendimiento que necesitan análisis de causa raíz.
- **Tipo C** son las optimizaciones y refactorizaciones: mejoras internas que no cambian lo que el sistema hace de cara al usuario, pero sí cómo lo hace; reducción de deuda técnica, actualización de dependencias.
- **Tipo D** son las consultas y peticiones de documentación, que muchas veces se resuelven con un RAG sobre la documentación del propio proyecto sin tocar código.

Esta clasificación no es decorativa. Cada tipo activa un flujo distinto, consume un presupuesto distinto, y requiere un nivel distinto de revisión humana. Tratar igual un bug menor que una mutación crítica de base de datos es uno de los errores más caros que se pueden cometer en una operación agéntica.

A continuación, cada petición recibe un índice de criticidad que combina el tipo con su impacto de negocio. Este índice determina tres cosas: cuánto presupuesto de tokens se le asigna, cuánto va a apretar el agente interrogador en la sesión de refinamiento, y cuántos agentes especializados van a participar en el análisis. Cuanto más alto el índice, más profundidad en el razonamiento, más rigor en las pruebas y más exhaustividad en el análisis de riesgos.

El último paso de esta sub-etapa es la detección de colisiones. El Agente Guardián compara cada nueva petición contra el backlog activo y contra el histórico de sprints anteriores, buscando solicitudes que se solapen o que pretendan modificar lo mismo en direcciones distintas. Cuando encuentra una colisión, las peticiones implicadas quedan bloqueadas hasta que el Intent Director decida si una tiene prioridad o si hay que fusionarlas. Esto evita que dos cambios incompatibles entren al mismo sprint y choquen durante la construcción.

Fase 3: Martes - Aceleración de Entorno y Pre-vuelo Técnico (Modo General)

La fase 3 es el preludio de la construcción de alta velocidad por los agentes IA. Es una fase de preparación con un objetivo claro: dejar el entorno listo y eliminar cualquier fricción de infraestructura antes de que arranque la construcción masiva.

Como ya os habréis dado cuenta, esta fase tiene mucha relevancia en el primer sprint del proyecto, pero solo tendrá sentido en los demás sprints si hace falta algo nuevo de infraestructura o configuración. Es decir, esta fase no siempre aparece, y cuando no aparece, la fase 4 se puede adelantar al lunes por la tarde o al martes por la mañana sin esperar.

En un sprint normal, en esta fase se realizan dos tipos de tareas. Por un lado, la configuración del entorno: preparar las ramas de git, levantar los contenedores Docker, instalar las dependencias necesarias y dejar listos los stubs y mocks de servicios externos para que el agente de construcción

pueda trabajar en aislamiento. Por otro lado, una validación del entorno construyendo una EIS de referencia como prueba, para confirmar que la infraestructura está estable antes de empezar la construcción real.

Solapamiento de diseño y construcción

Una de las decisiones más importantes que tomamos al definir Designio fue permitir que el diseño y la construcción se solapen cuando es posible. Esto significa que el martes por la mañana, mientras se prepara el entorno y se cierran los últimos detalles arquitectónicos, los agentes constructores ya pueden empezar a generar los primeros artefactos (esquemas de datos, contratos de API, interfaces) sobre las EIS que ya están certificadas y no dependen de decisiones pendientes.

Esto nos permite ganar varias horas de ejecución sin sacrificar control, siempre que la coordinación entre la generación de ADRs y el arranque de la construcción esté bien controlada.

Excepción: protocolo de re-arquitecturación

Este protocolo se activa en dos situaciones:

- La primera es en el primer sprint del proyecto, porque por definición todavía no hay arquitectura asentada ni blueprint firmado.
- La segunda es en sprints posteriores cuando un EIS introduce cambios estructurales: modificaciones en el núcleo del sistema, nuevas integraciones críticas con terceros, o cualquier decisión que afecte al blueprint vigente.

Cuando el protocolo se activa, entra en juego un agente que hasta ahora no había aparecido en el libro: el Shadow Architect. Es un agente que trabaja en paralelo al arquitecto humano, no en su lugar.

La metáfora del "shadow" es deliberada: el agente acompaña al humano, le ofrece alternativas, le advierte de riesgos y le simula impactos, pero la firma final siempre la pone una persona. Sigue la misma lógica que el Shadow Sandbox del paso 5: un elemento que corre en paralelo a lo principal sin sustituirlo.

El martes por la mañana, el Shadow Architect genera varias propuestas de ADRs (Architecture Decision Records), no una sola. Cada propuesta optimiza para un criterio distinto (rendimiento, simplicidad, escalabilidad u otro relevante según el caso), y cada una incluye su justificación técnica, sus consecuencias y sus trade-offs.

Junto a los ADRs, el Shadow Architect genera un informe de coste estimado de tokens y de complejidad de implementación, para que el Financial Guard que definimos en el paso 3 pueda dar luz verde al gasto antes de construir.

Una vez generadas las propuestas, el Agente Guardián entra a hacer su trabajo habitual de vigilancia: valida automáticamente que cada propuesta cumple con los estándares del cliente. Comprueba librerías permitidas en la whitelist, versiones soportadas, estrategia de branching y reglas de despliegue. Las opciones que no cumplen quedan marcadas para revisión humana o directamente descartadas.

Por la tarde, el arquitecto humano trabaja codo con codo con el Shadow Architect. Revisa las propuestas que han pasado el filtro del Guardián, ajusta parámetros si lo cree necesario, y finalmente firma la propuesta elegida. A partir de ese momento, el ADR firmado se convierte en la referencia inmutable de la arquitectura para el resto del sprint, y el Agente Guardián pasa a vigilar que los constructores no se desvíen de él. Cualquier intento de salirse del ADR firmado levanta una alerta, igual que ocurre con cualquier otro intento de saltarse las restricciones definidas, según vimos en el paso 6.

En el primer sprint, este protocolo ocupa todo el martes, lo cual desplaza el inicio de la construcción al miércoles. En sprints posteriores sin cambios arquitectónicos, la fase 3 desaparece o se reduce a un pre-vuelo rápido, y la fase 4 puede empezar incluso el lunes por la tarde si el EIS está bien certificado.

Fase 4: Martes (PM) y Miércoles - Construcción

El martes por la tarde, una vez validado el entorno en la fase anterior, arranca la construcción.

Adelantar el inicio de esta fase respecto a metodologías clásicas (donde la construcción no empezaba hasta el día siguiente) nos permite ganar varias horas de ejecución útil. El humano que supervisa puede revisar las primeras decisiones del agente antes de cerrar la jornada, y los agentes pueden seguir trabajando en tareas de baja supervisión durante la noche si lo necesitan.

El miércoles es el día más intensivo del sprint. Es donde el código toma forma de verdad, donde la lógica de negocio compleja se materializa, y donde la capacidad de construcción masiva del agente se cruza con la supervisión humana en tiempo real. Lo cuento por bloques, no porque sean fases secuenciales (en realidad pasan en paralelo a lo largo del día), sino porque conviene entender cada parte por separado antes de ver cómo encajan.

Lo que hace el agente constructor durante el miércoles

- El Agente Constructor opera a plena capacidad desde el primer minuto del miércoles, con todo el contexto cargado desde el martes: dependencias, esquemas de base de datos, contratos de interfaz, el histórico de decisiones de la EIS, y el ADR firmado en la fase 3 si hubo re-arquitecturación.
- Su producción es múltiple. Por un lado, escribe el código fuente siguiendo el patrón arquitectónico elegido (Clean Architecture, Hexagonal, o el que corresponda al proyecto). Implementa la lógica funcional, los scripts de migración de base de datos, los esquemas OpenAPI/Swagger (las definiciones que describen cómo se comunica el sistema con el exterior) y los archivos de configuración (.env, YAML), de forma que el sistema sea reproducible y portable en cualquier entorno.
- Por otro lado, y esto es importante porque rompe el orden tradicional de desarrollo, el agente diseña los casos de prueba antes de escribir el código. La práctica está inspirada en el clásico Test-Driven Development, pero adaptada al contexto agéntico: la IA lee la EIS, extrae los criterios de aceptación, traduce esos criterios en pruebas concretas, y solo entonces se pone a escribir el código que tiene que pasar esas pruebas. La condición para considerar el código válido es que supere una cobertura superior al 90% en pruebas unitarias. Por debajo de esa cifra, no se considera entregado.

- Junto al código y los tests, el agente genera la documentación técnica inline (JSDoc, Pydoc, etc., según el lenguaje), un README técnico del módulo, y un log de decisiones donde se explican los trade-offs que ha tomado durante la construcción. Esto último es lo que diferencia un código bien generado de un código mantenible: no basta con que funcione, hay que poder entender por qué se construyó así cuando alguien lo lea dentro de seis meses.

Lo que hace el humano en paralelo: las pruebas de intención

- Mientras el agente construye, el Delivery Lead no está parado. Tiene acceso a un entorno de pruebas en caliente, un staging que se actualiza con cada commit exitoso del agente. A medida que se completan submódulos, el humano entra al entorno y los prueba.

- Es importante entender qué tipo de prueba se hace aquí, porque no es una QA técnica. El Delivery Lead no busca errores de código (de eso se encarga la suite de tests que diseñó el propio agente). Lo que busca es confirmar que la intención del producto se está materializando según lo esperado por el negocio. Que el flujo de checkout tiene el tono adecuado, que los cálculos financieros se comportan como espera el cliente, que la usabilidad es la que se prometió en el EIS, que los permisos de seguridad se aplican donde toca. A esto le llamamos pruebas de intención.

- Cuando el Delivery Lead detecta una desviación, no abre un ticket ni espera a la siguiente fase. Inyecta un micro-ajuste directamente, lo que viene a ser un prompt corto y dirigido que reorienta al agente sin detener el flujo general. Esto es importante: la mayoría de las desviaciones que un humano detecta en este momento son sutiles (un botón que no está donde debería, un texto que no suena natural, una validación que es demasiado estricta o demasiado permisiva), y corregirlas en caliente cuesta segundos. Si esas mismas desviaciones se acumulan y se descubren días después, corregirlas cuesta horas o supone refactorizaciones masivas y deuda técnica.

Cómo se controla la seguridad: el protocolo Allow/Run

El Agente Constructor no tiene autonomía total. Antes de ejecutar cualquier comando que pueda afectar al sistema, al entorno o a recursos externos, el Agente Guardián se interpone y verifica.

Cada vez que el constructor intenta lanzar un comando (instalar una librería, levantar un contenedor, aplicar una migración de infraestructura), la petición pasa por la whitelist que vimos en el paso 6. Si el comando está autorizado, se ejecuta. Si no lo está, el Guardián lo bloquea instantáneamente y obliga al constructor a buscar una alternativa permitida o a solicitar elevación de privilegios. Este patrón es el que en la industria se conoce como Allow/Run: por defecto se deniega, y solo se permite lo explícitamente autorizado.

Las credenciales y secretos siguen el patrón que ya tratamos en el paso 4: el Agente Guardián los inyecta en memoria volátil en el momento exacto de la ejecución, y el constructor los usa pero nunca los visualiza ni puede extraerlos en texto plano. Toda acción, todo comando, todo intento bloqueado queda registrado en un log inmutable de trazabilidad, vinculado al coste exacto en tokens. Esto no es burocracia: es lo que permite, días o meses después, auditar exactamente qué hizo la IA y por qué.

Qué pasa cuando los tests fallan

Aquí es donde la construcción agéntica se diferencia más de la construcción humana clásica. Cuando un test falla, el Agente Constructor entra automáticamente en un bucle de diagnóstico. Lee el stack trace (la traza del error), identifica la causa probable, propone un parche, lo aplica, vuelve a ejecutar el test, y repite hasta que el test pasa.

Este bucle puede entrar en una situación incómoda: que el agente no consiga arreglar el fallo después de varios intentos. Aquí es donde aplica lo que dijimos en el paso 3 sobre la detección de bucles de inferencia. Si el constructor envía esencialmente el mismo prompt y recibe el mismo error más de un número determinado de veces, sin variaciones significativas en el código propuesto, el sistema mata el proceso de razonamiento y notifica al humano. No tiene sentido seguir quemando tokens en un problema que el agente no es capaz de resolver por sí solo: lo más probable es que haya una ambigüedad en el EIS que el constructor está intentando interpretar y no llega.

Control financiero y velocidad de avance

El Financial Guard del paso 3 sigue activo durante todo el miércoles. El consumo de tokens se monitoriza en tiempo real, y las alertas de tres niveles (50%, 80%, 100% del presupuesto) operan tal como las definimos en aquel paso. Lo destacable del miércoles es que es el día de mayor consumo del sprint, así que es habitual que los avisos de nivel 1 o 2 salten aquí. No es una señal de alarma por sí misma: es el día donde se construye más, así que se gasta más.

Una métrica adicional que está empezando a usarse, y que considero útil aunque todavía no la apliquemos de forma sistemática en todos los proyectos, es la densidad de progreso. Compara la velocidad de generación de código del agente con la complejidad restante del backlog del sprint. Si la densidad cae (el agente avanza despacio respecto a lo que queda por hacer), es señal de que algo está atascado: un módulo más complejo de lo previsto, una ambigüedad que está consumiendo iteraciones, un patrón nuevo que el agente no domina. Detectarlo a tiempo permite intervenir antes de que el viernes lleguemos con la EIS a medias.

Sobre la autonomía del agente

Una aspiración de esta metodología es alcanzar un alto grado de autonomía del agente constructor, idealmente por encima del 95%, lo que significaría que solo el 5% o menos de las decisiones requerirían intervención humana. En proyectos maduros, con equipos que ya han ajustado los prompts, las whitelists y las plantillas de EIS a su contexto, esa cifra es alcanzable.

Conviene ser honesto en este punto: en los primeros sprints de un proyecto, esa autonomía no se alcanza. Es habitual que los primeros sprints tengan ratios mucho menores, con el Delivery Lead interviniendo constantemente para reorientar al agente, ajustar prompts y refinar EIS. No es un fracaso: es parte natural del aprendizaje del sistema sobre el proyecto concreto, sobre las convenciones del cliente, sobre los patrones que funcionan y los que no. La autonomía del 95% es un objetivo a perseguir con la madurez del equipo, no un umbral que se exige desde el primer día.

Transición al jueves

Al final del miércoles, si todo ha funcionado, el software tiene que estar técnicamente terminado: el código compila sin errores en el entorno sandbox, la suite de tests del agente pasa con la cobertura superior al 90%, los controles de seguridad del Guardián no han detectado intentos pendientes de saltarse restricciones, y el Delivery Lead ha dado el visto bueno de intención a los módulos principales. A partir de ahí, el jueves se dedica a la auditoría profunda de calidad: revisión humana del código, análisis estático más exhaustivo, validación de seguridad por parte del responsable correspondiente, y preparación para el despliegue. Pero esa es la fase 5, y la veremos en su momento.

Fase 5: Jueves - Auditoría de Salud Técnica y Despliegue en Pre con Certificación de Seguridad

El objetivo del jueves es asegurarse de que el código generado el miércoles es impecable antes de mezclarlo con el resto del proyecto. Aquí no hablamos de probar funcionalidad (eso es trabajo del viernes con la validación de aceptación), sino de revisar la calidad interna del código: su seguridad, su mantenibilidad, su fidelidad respecto a la EIS y su salud técnica general.

Es una auditoría de Caja Blanca. La IA abre el código, lo analiza por dentro, y detecta cualquier problema antes de que llegue a producción. La idea es eliminar la deuda técnica antes de que nazca, no después.

Análisis estático de seguridad

Aquí entra en juego una especialización del Agente Guardián que llamamos Security Sentinel. Conviene aclarar una decisión metodológica de Designio: el Agente Guardián no es un agente único monolítico, es un agente con varios subagentes a su cargo, cada uno especializado en un tipo de control. El Security Sentinel es uno de ellos, el especializado en seguridad y calidad del código. De esta forma el Guardián mantiene la visión global de todo lo que sale del sistema, pero delega la profundidad técnica de cada control a un subagente experto.

El Security Sentinel revisa el código en varias dimensiones:

- Vulnerabilidades conocidas según el OWASP Top 10 (inyecciones SQL, Cross-Site Scripting, control de acceso roto, etc.). No se limita a buscar firmas conocidas, sino que aplica Taint Analysis, una técnica de rastreo del flujo de datos desde fuentes no confiables (lo que introduce el usuario, lo que viene de APIs externas) hasta los puntos donde esos datos se usan de forma sensible (consultas a base de datos, ejecución de comandos, etc.). Esto permite detectar mitigaciones cosméticas que parecen seguras pero no lo son.
- Malas prácticas de código (code smells): bucles ineficientes, complejidad ciclomática excesiva, código muerto, gestión incorrecta de memoria o conexiones de base de datos que no se cierran como deben.
- Secretos expuestos: tokens, claves API, contraseñas escritas directamente en el código (lo que se conoce como hardcoded credentials). Esto enlaza con el paso 4 del libro: si algo se ha escapado pese al Filtro de Salida de Secretos, aquí debe quedar detectado.

- Cumplimiento de las convenciones del cliente: nomenclatura (CamelCase, snake_case), estructura de carpetas, tipado estricto, estilo de comentarios. Lo que en otros pasos hemos llamado el ADN técnico del proyecto.
- Seguridad de la cadena de suministro: cada dependencia que el agente constructor instaló durante el miércoles se valida contra bases de datos de vulnerabilidades activas. Se genera además un inventario completo de dependencias (lo que en la industria se llama SBOM o Software Bill of Materials) y se verifica que las licencias de las librerías sean compatibles con la política legal del cliente. Esto importa más de lo que parece: una librería con licencia GPL incluida sin querer en un proyecto comercial cerrado puede comprometer la propiedad intelectual entera del cliente.

Es como tener un arquitecto senior y un experto en ciberseguridad revisando cada línea del código, pero sin cansarse nunca y sin pasar nada por alto.

Sandbox de Validación de Fidelidad

Antes de proceder al despliegue, el código pasa por un sandbox específico que mide algo distinto a lo que mide el sandbox de construcción del miércoles. Aquí se prueba, si lo construido resuelve realmente lo que la EIS pedía.

Esto es importante porque uno de los riesgos más sutiles de la construcción agéntica es lo que llamamos alucinación lógica: código que funciona técnicamente (no da error, los tests pasan), pero que no resuelve el problema de negocio planteado, o que introduce funciones no solicitadas que expanden el alcance sin que nadie lo haya pedido. Lo que el Intent Director había firmado el lunes y lo que el agente ha construido el miércoles tienen que coincidir milimétricamente, no aproximadamente.

El sandbox hace tres cosas:

- Compara el comportamiento real del código bajo entradas controladas con los criterios de aceptación definidos en la EIS. Mide la desviación entre intención original y resultado ejecutado.
- Estresa el código con casos de borde y entradas malformadas o aleatorias (técnica conocida como fuzzing) para verificar que el manejo de errores responde según lo previsto, sin filtrar información sensible en los mensajes de error (trazas de stack, versiones de servidor, estructura interna del sistema).
- Si detecta desviaciones significativas respecto al EIS, recalibra automáticamente los Railguards del agente constructor, igual que vimos en el paso 5 del libro. Esto cierra el bucle: lo que se aprende en una auditoría se traduce en restricciones más finas para los siguientes sprints.

El Health Score

Una vez completada la auditoría del Security Sentinel y las pruebas del sandbox de fidelidad, el Agente Guardián consolida todos los hallazgos en una métrica única: el Health Score, una nota del 0 al 100 que mide la salud técnica del código.

Para considerar que el código está listo para despliegue, exigimos un Health Score igual o superior a 98 sobre 100. Por debajo de esa cifra, el Agente Constructor recibe el reporte detallado del Security Sentinel, con cada hallazgo clasificado por severidad (crítica, alta, media o baja) y con la instrucción exacta de corrección, y refactoriza el código automáticamente sin intervención humana. Este ciclo se repite hasta que el código alcanza el umbral.

El Health Score se calcula a partir de cinco vectores principales:

- Seguridad: ausencia de vulnerabilidades críticas, cumplimiento de políticas de cifrado.
- Fidelidad: alineación entre el código generado y la intención original definida en la EIS.
- Mantenibilidad: deuda técnica, calidad de la documentación interna, legibilidad y modularidad.
- Rendimiento: eficiencia bajo carga simulada (uso de CPU, memoria, operaciones de entrada/salida).
- Cobertura de tests: porcentaje de líneas y ramas lógicas validadas por pruebas unitarias y de integración.

Lo que sí varía entre proyectos es el peso que se da a cada vector. Una calibración que en mi experiencia personal me ha funcionado bien para proyectos donde la seguridad y la fidelidad son críticas es: 35% seguridad, 25% fidelidad, 20% mantenibilidad, 10% rendimiento, 10% cobertura. Pero esos porcentajes no son universales.

He trabajado en proyectos donde esa ponderación cambiaba radicalmente. En una aplicación muy reglada, sin previsión de cambios a futuro, donde lo que de verdad importaba era que estuviera todo cubierto al milímetro y bien probado, la cobertura subía mucho y la mantenibilidad bajaba: tenía poco sentido optimizar para futuras refactorizaciones de una app que iba a quedarse congelada. En otros proyectos con requisitos extremos de latencia, el rendimiento llegaba al 30% del peso. Cada cliente, cada producto y cada momento del ciclo de vida pide su propia ponderación. Lo que no cambia es la lógica: hay cinco vectores, todos importan, y la suma tiene que llegar al 98 para que el código salga.

Esta calibración la hace el Arquitecto en la primera definición del proyecto, junto con el cliente. No es decisión del Agente Constructor ni del Guardián: es una decisión humana sobre qué tipo de calidad nos importa más en este proyecto concreto.

Junto al Health Score, que mide la salud del código construido, conviene seguir una segunda métrica que mide algo distinto: la calidad de la propia intención, es decir, lo bien o mal que se capturó lo que el negocio quería. La llamo Precisión de la Intención (PA), y se calcula con una fórmula sencilla:

$$PA = (C / I) \times 100$$

Donde C es la complejidad de la EIS (una ponderación de su impacto arquitectónico, la criticidad del proceso de negocio que toca y sus requisitos de seguridad, calculada a partir del grafo de dependencias de la EIS) e I es el número de iteraciones, es decir, los ciclos de generación, fallo y

corrección que el sistema necesitó antes de alcanzar el estado deseado. La idea de fondo es intuitiva: si una EIS sencilla necesita muchas iteraciones para quedar bien, algo falla en cómo se definió la intención, no en la capacidad de construir.

Por eso establezco un umbral de aceptación, y una regla asociada que evita malgastar tokens: cualquier EIS que supere las tres iteraciones de corrección activa automáticamente una auditoría de ambigüedad. El sistema se pregunta entonces dónde está el problema real: si en la vaguedad de la instrucción humana, en la falta de contexto inyectado, o en las limitaciones del modelo elegido para esa tarea. Si el umbral no se alcanza, la EIS se devuelve al Intent Director para que la reescriba, en lugar de insistir a la fuerza con más y más prompts. Esto es importante, porque la tentación de "seguir insistiéndole a la IA hasta que salga" es cara en tokens y casi siempre es señal de que la intención estaba mal definida desde el principio. No se permite la fuerza bruta de prompts: si algo no sale en pocas iteraciones, el problema casi nunca es la máquina, es la claridad de lo que le pedimos.

Certificación humana del Arquitecto

El refactoring autónomo hasta llegar al 98 lo hace el Agente Constructor sin intervención humana. Pero antes de que el código se despliegue en pre-producción, hay un paso de certificación humana que es responsabilidad del Arquitecto de Proyecto (que en Designio, como vimos en la fase 3, es siempre una figura independiente del Intent Director y del Delivery Lead).

El Arquitecto no relee miles de líneas de código. Lo que hace es auditar el razonamiento del Security Sentinel y la coherencia del reporte:

- Revisa los hallazgos clasificados como críticos o altos que el Sentinel ha marcado como resueltos, verificando que la solución aplicada tiene sentido y no es un parche cosmético.
- Decide sobre los falsos positivos. El Sentinel a veces marca como problema algo que en realidad es una excepción justificada por la lógica de negocio o por una decisión de diseño consciente. El Arquitecto puede aceptar esos riesgos y dejarlos pasar, pero cada excepción queda registrada en el ADR del proyecto como decisión técnica argumentada.
- Valida que la documentación generada (manuales técnicos, diagramas, especificaciones de API en Swagger/OpenAPI) sea inteligible para un equipo de mantenimiento humano. Esto es importante: el cliente debe poder mantener el sistema sin depender perpetuamente de la IA que lo construyó.
- Firma digitalmente la certificación de salud. Esa firma cierra formalmente la rama de desarrollo del sprint y autoriza al sistema a iniciar el despliegue automatizado en pre-producción.

La firma del Arquitecto es lo que separa "el código está bueno técnicamente" de "el código puede salir del entorno de desarrollo". No hay despliegue sin esa firma.

Despliegue automatizado en PRE

Una vez firmada la certificación, el sistema dispara el pipeline de CI/CD (integración continua y despliegue continuo) para trasladar el código desde el sandbox de construcción al entorno de pre-producción del cliente.

El despliegue se hace de forma controlada y completamente automatizada:

- La infraestructura se levanta con scripts de Infraestructura como Código (herramientas como Terraform, Pulumi o CloudFormation) para asegurar que el entorno de pre-producción es un espejo exacto de producción. Esto incluye redes, grupos de seguridad, firewalls de aplicación, balanceadores de carga y servicios de base de datos. Lo importante de hacer esto con IaC es eliminar el error humano de configuración manual, que es una de las fuentes más comunes de bugs en despliegue.
- El despliegue de la aplicación se hace con estrategias como Blue-Green o Canary, que permiten ir trasvasando tráfico al nuevo código de forma progresiva en lugar de cambiar todo de golpe. Si algo falla durante el despliegue (un timeout, un error de conexión a base de datos, un health check que no responde), el sistema ejecuta un rollback automático al último estado estable conocido en menos de 30 segundos, y notifica al equipo. Nadie tiene que estar mirando la consola cruzando los dedos.
- Una vez completado el despliegue, el sistema ejecuta una batería final de smoke tests: pruebas básicas de disponibilidad, conectividad de endpoints, resolución de DNS, persistencia básica de datos. Estas pruebas confirman que la aplicación está viva y accesible dentro de la red del cliente.

Transición al viernes

Al final del jueves, el código está auditado, certificado, desplegado en pre-producción y respondiendo a las pruebas de humo. Lo que queda para el viernes es la validación funcional extensiva con el cliente: las pruebas de aceptación de usuario (UAT), la firma de intención final por parte del Business Owner, y el despliegue en producción.

En los proyectos en los que he tenido el placer de participar y donde hemos aplicado esta metodología, aproveché que teníamos detalladas todas las pruebas e2e de UAT y pasé el 100% de forma agéntica el jueves. Además, se resolvieron en PRE todos y cada uno de los bugs que aparecieron. De esta forma conseguíamos que UAT consumiera menos tiempo en validación (todo estaba pre-validado), y el equipo humano pudo dedicar más tiempo a pruebas de experiencia de usuario, de intención y exploratorias, que es donde aporta su mayor valor.

Fase 6: Viernes (AM) - Validación de Valor (UAT) y Refactorización Flash

El viernes por la mañana se hace la validación final del sprint: ¿lo que se ha construido cumple exactamente lo que el negocio pidió? Aquí se revisa el comportamiento, como lo haría un usuario real.

Es una validación de caja negra, y conviene precisar el término, porque en este libro he usado "caja negra" también para hablar del peligro del software indescifrable. Aquí lo uso en el sentido opuesto y deliberado: el validador trata el código como una caja que no necesita abrir, no porque sea opaca y peligrosa, sino porque su corrección interna ya está garantizada por las pruebas automáticas que pasó el jueves. No importa cómo está hecho por dentro, ni si usa una librería de moda o un patrón concreto; solo importa si se comporta como la intención exigía. Es lo que llamo aceptación por

comportamiento, no por inspección: el humano certifica que el software hace lo que el negocio pidió, no audita línea a línea cómo lo hace. La validación se ejecuta sobre el entorno de pre-producción que dejamos preparado el jueves, que es un espejo exacto de producción.

Conviene reconocer algo antes de seguir: gracias al UAT agéntico que se ejecuta el jueves (las pruebas end-to-end pasadas al 100% antes de que el BO junto a QA se sienten a validar), esta fase del viernes suele ser rápida y, en proyectos maduros, casi una formalidad. La mayor parte de los errores funcionales ya se han detectado y corregido en PRE antes de que el cliente entre en escena.

Validación de aceptación del usuario (UAT)

El equipo de QA junto al BO acceden al entorno de pre-producción y prueban la funcionalidad, lo que intentará es confirmar 4 cosas, si hace lo que se pidió, si resuelve el problema de negocio, si se comporta como acordamos en la EIS y finalmente si la experiencia es la que se esperaba.

El tiempo del QA y del BO se debe invertir en lo que solo un humano puede hacer: experiencia de usuario, juicio sobre la intención, exploración de casos que no estaban en la EIS pero que se le ocurren al ver el producto en marcha.

Diferenciación entre Error de Intención y Evolutivo de Último Minuto

Aquí entra una distinción que en Designio tratamos de forma muy explícita, porque mezclar las dos cosas es una de las trampas más comunes del UAT.

Cuando el BO detecta algo que no le encaja, lo primero que hacemos es categorizar el hallazgo en una de estas dos categorías:

- **Error de Intención:** el sistema no cumple lo pactado en la EIS. Hay una desviación entre lo que se firmó el lunes y lo que se ha construido. Esto se corrige en el sprint, prioritariamente, porque es deuda directa del proveedor (la metodología, en este caso) respecto a lo prometido.
- **Evolutivo de Último Minuto:** el sistema cumple lo pactado, pero el BO, al verlo funcionando, identifica una mejora que no estaba en la EIS original. Es legítimo, es valioso (a veces solo se ven las oportunidades cuando el producto funciona), pero no es lo mismo que un error.

Esta distinción importa porque ambos casos se tratan distinto. El Error de Intención entra directamente al bucle de corrección rápida del viernes y se resuelve antes del release. El Evolutivo de Último Minuto solo se acepta en el sprint actual si el Intent Director da su visto bueno, ya que implica modificar el alcance original. Si además ese evolutivo introduce cambios arquitectónicos (nuevas integraciones, modificaciones del núcleo del sistema, decisiones que afecten al blueprint vigente), entonces se necesita además la firma del Arquitecto con un nuevo ADR, como vimos en la fase 3.

Si el Intent Director no autoriza el evolutivo, o si la complejidad lo desaconseja, ese hallazgo se registra como entrada para el siguiente sprint y no bloquea el release actual. Esta separación clara entre lo que se corrige ya y lo que se posterga es lo que evita que el viernes se convierta en un agujero negro donde caben todas las ideas nuevas del cliente. Sin esa frontera, el release nunca sale.

Bucle de corrección rápida (Flash Refactoring)

Cuando se confirma un Error de Intención, no se abre un nuevo sprint ni se aplaza una semana. El Agente Constructor aplica el cambio inmediatamente. La instrucción del BO se traduce en un prompt de corrección dirigido al módulo afectado, y el agente construye el parche en minutos.

Aquí es importante el papel del Security Sentinel. Cuando el constructor aplica el parche, el Sentinel re-valida automáticamente la seguridad y la integridad del cambio sobre el componente afectado. No se vuelve a hacer la auditoría completa del jueves (sería desproporcionado para un cambio pequeño), pero sí se verifica que el parche no rompe nada de lo que ya estaba certificado. Si la re-validación es satisfactoria, el pipeline despliega solo el componente afectado en PRE (despliegue ultra-dirigido, no completo del sistema), y el BO recibe notificación de que ya puede volver a probar.

Si la re-validación detecta un problema (el parche introduce una vulnerabilidad, rompe un test existente, contradice una decisión arquitectónica), el ciclo se reabre y el constructor itera hasta que el Sentinel da su OK. Esto puede repetirse varias veces, en ciclos de 15 a 30 minutos cada uno, hasta que el BO confirma "OK Final".

Es como tener un equipo de desarrollo trabajando en tiempo real con el cliente, sin tickets, sin esperas, sin colas. Pero con un guardia técnico (el Sentinel) verificando cada paso para que la velocidad no se cobre la calidad.

Para los Evolutivos de Último Minuto que sí se aceptan, el flujo es el mismo, con dos diferencias: requieren autorización previa del Intent Director (que es quien gestiona el alcance del sprint) y, si tocan arquitectura, firma del Arquitecto con nuevo ADR antes de que el constructor empiece a trabajar.

Excepciones: scope creep del viernes

Otra situación habitual es que el BO, viendo el producto funcionando, intente añadir funcionalidades nuevas que no estaban en la EIS y que tampoco son evolutivos menores. Aquí el Agente Guardián actúa de filtro: detecta automáticamente cuando una petición del BO excede los límites del EIS firmado el lunes, y alerta al Delivery Lead antes de que el constructor empiece a trabajar.

Si efectivamente es scope nuevo (no Error de Intención, no Evolutivo menor), la petición se deriva al backlog del próximo lunes para que entre en el siguiente ciclo de refinamiento del Intent Director con el BO. No se mezcla con el release actual.

Esto es importante por dos motivos. Primero, protege el release: lo que se ha validado el jueves no se contamina con cambios nuevos a última hora. Segundo, protege al BO: muchas veces lo que parece una buena idea el viernes a las once de la mañana se ve diferente cuando se piensa con calma el lunes siguiente. Forzar el paso por el refinamiento del próximo sprint filtra las ideas que no aguantan veinticuatro horas de reflexión.

Firma del BO y cierre formal del EIS

Cuando el BO da el "OK Final", se procede a formalizar el cierre del sprint. El Agente Guardián marca la EIS del sprint como "Satisfecha" y genera un vínculo permanente e inmutable en el log de

trazabilidad que conecta la intención original definida el lunes con el estado final del código que va a desplegarse en producción.

Esto no es burocracia. Es lo que permite, días o meses después, auditar exactamente qué se pidió, qué se construyó, qué se validó y por qué. Para clientes en sectores regulados (banca, salud, administración pública), este vínculo trazable es a menudo requisito legal, no opcional.

La firma del BO es electrónica y queda registrada con sello temporal. Es la firma que autoriza el merge a la rama principal y el despliegue a producción, que se ejecutan en la fase 7 por la tarde.

Excepciones: cuando el BO no está disponible

Hay veces en que el BO no puede firmar el viernes (viaje, indisponibilidad, baja, agenda imposible). La metodología prevé un mecanismo de delegación, pero con condiciones estrictas para que no se convierta en atajo habitual.

El Delivery Lead puede firmar de forma provisional si se cumplen estas condiciones:

- El 100% de las pruebas e2e de UAT pasadas el jueves han sido satisfactorias.
- No hay hallazgos pendientes de Error de Intención.
- El Health Score del jueves estaba por encima del umbral pactado.
- El BO ha sido notificado por canal trazable y no responde en el plazo acordado.

Esta firma provisional permite que el release no se bloquee, pero el BO tiene que validar formalmente en el siguiente sprint. Hasta esa validación, el desarrollo se considera entregado pero no aceptado.

Si la situación es delicada (cliente importante, módulo crítico, hallazgos limítrofes, o el Delivery Lead tiene dudas sobre si firmar), se escala al miembro del AIGB (AI Governance Board, el comité de gobernanza de IA del proyecto) que tiene autoridad de decisión última sobre estos casos. Ese miembro decide si autoriza el despliegue con firma provisional o si bloquea el release hasta que el BO esté disponible. Su decisión queda registrada también en el log de trazabilidad con su justificación.

Quiero ser honesto en un punto: en mis proyectos he ejercido a la vez de Intent Director y Delivery Lead, y los desacuerdos con el Business Owner los he resuelto siempre convenciéndolo, sin tener que llegar a escalar al AIGB. Pero el mecanismo de escalado tiene que existir igual, y por eso lo describo. Existe para los casos en que esos roles recaen en personas distintas que no se ponen de acuerdo, o para cuando el Intent Director no logra convencer al BO. Que a mí no me haya hecho falta usarlo no significa que se pueda prescindir de él; significa que, cuando los roles están bien ejercidos y la comunicación con el negocio es buena, el AIGB es la red de seguridad que casi nunca tienes que tocar.

Transición a la fase 7

Al final del viernes por la mañana, lo que tenemos es: una EIS satisfecha y firmada (por el BO o, en su caso, por delegación validada), todos los Errores de Intención corregidos en PRE, los Evolutivos autorizados aplicados y validados, y los hallazgos de scope nuevo derivados al backlog del próximo sprint. La firma habilita el cierre técnico del ciclo, que es lo que hacemos en la fase 7 por la tarde: merge a main, despliegue a producción y retrospectiva del sprint.

Fase 7: Viernes (PM) - Release, Merge y Retrospectiva del Sprint

El viernes por la tarde, una vez firmado el cierre del EIS por parte del BO en la fase 6, llega el momento de poner el software en producción, dejar registro inmutable de todo lo ocurrido durante el sprint, y permitir que el sistema aprenda de la semana para llegar al lunes siguiente con ventaja.

Es el equivalente a cerrar el ciclo, entregar el valor al cliente y dejarlo todo impecable para el siguiente arranque. No hay nuevas decisiones humanas que tomar (esas se han tomado a lo largo de la semana). Lo que queda es ejecución técnica supervisada y aprendizaje del sistema.

Merge, tagging y release a producción

El Agente Guardián ejecuta la integración de la rama del sprint a la rama principal del proyecto. Antes del merge, el sistema verifica con algoritmos asistidos que no haya colisiones con otros despliegues paralelos o con cambios recientes en la rama principal. Si las hay, se resuelven automáticamente cuando es posible, o se escala al Delivery Lead para decisión manual cuando la colisión requiere criterio humano.

Una vez completado el merge, el Guardián etiqueta la versión resultante (tagging) según el esquema de versionado del proyecto, que en la mayoría de los casos sigue la convención semver (semantic versioning), de forma que cualquier desarrollador o sistema que mire el repositorio en el futuro pueda identificar a primera vista qué tipo de cambio se introdujo en cada versión.

Después viene el despliegue a producción. Se hace de forma controlada, igual que el despliegue a PRE del jueves, con las mismas estrategias estándar de la industria:

- **Blue-Green Deployment:** dos entornos idénticos en paralelo, uno activo y otro preparado con la nueva versión. El cambio de tráfico de un entorno a otro es instantáneo, sin tiempo de inactividad. Si algo va mal, volver al entorno anterior es igual de rápido.
- **Canary Release:** se despliega primero a un pequeño porcentaje de usuarios (por ejemplo, un 5%), se monitoriza el comportamiento durante un tiempo definido, y si todo va bien se amplía progresivamente hasta el 100%. Si los indicadores se desvían, el sistema vuelve atrás antes de que el problema afecte a toda la base de usuarios.

La elección entre una u otra depende del tipo de producto y del perfil de riesgo del cliente. Blue-Green es más limpio para sistemas con tráfico predecible y necesidad de cambios atómicos. Canary es preferible cuando hay millones de usuarios y conviene detectar problemas con una muestra pequeña antes de comprometer al resto.

Durante el despliegue y en las horas posteriores, el sistema de monitorización en producción vigila métricas clave: tasa de errores, latencia, consumo de recursos, comportamiento anómalo de usuarios. Si detecta un pico que supere los umbrales pactados, ejecuta un auto-rollback al último estado estable conocido y notifica al equipo. Nadie tiene que estar mirando los dashboards cruzando los dedos.

El Delivery Lead supervisa la ejecución técnica del proceso, pero no firma nada nuevo: la firma del BO de la fase 6 es la única firma humana de cierre del sprint, y es la que ha autorizado este despliegue. Concentrar la responsabilidad humana en una sola firma evita ceremonia innecesaria.

Cierre del Traceability Log

Mientras el despliegue se ejecuta, el Agente Guardián consolida toda la evidencia generada durante el sprint en un único repositorio de trazabilidad. Esto incluye:

- La EIS original firmada el lunes y sus eventuales modificaciones autorizadas (evolutivos aprobados).
- Las decisiones arquitectónicas tomadas el martes, registradas como ADRs firmados por el Arquitecto.
- El código fuente del sprint con sus tests, su documentación inline y el log de decisiones técnicas del Agente Constructor.
- Los reportes de seguridad del Security Sentinel, incluyendo el Health Score del jueves y la lista de hallazgos resueltos.
- Los microajustes del miércoles, los hot-fixes del viernes y cada parche aplicado con su justificación.
- Las aprobaciones humanas del recorrido: firma del Intent Director para evolutivos, firma del Arquitecto para ADRs, firma del BO para cierre de EIS, escalados al AIGB si los hubo.
- Los resultados de UAT, tanto los del jueves (agénticos) como los del viernes (humanos del BO y QA).
- Los logs del despliegue, con hashes criptográficos que prueban la integridad de cada artefacto desde su construcción hasta su llegada a producción.

Es literalmente la 'caja negra' del sprint: registro completo, no manipulable, recuperable en cualquier momento futuro.

De hecho, creo conveniente que el WORM se vaya construyendo en paralelo a todas las fases del sprint, acompañando a cada artefacto generado (código, pruebas, ADRs, microajustes, despliegues). De esta forma, si en cualquier momento aparece un problema, se puede volver al punto exacto previo al fallo y reconstruir el contexto sin perder horas de trabajo.

Voy a contar una experiencia personal donde este registro inmutable me ayudó enormemente. Estaba trabajando con una plataforma agéntica que coordinaba dos agentes en paralelo: por un lado, Antigravity sobre VSCode con un agente Gemini; por otro, la extensión de Claude en el mismo entorno. Ambos agentes estaban ejecutando tareas a la vez, y uno de ellos había empezado a hacer commit y push de los resultados de un sprint.

En medio de ese proceso, Antigravity se actualizó automáticamente, se desconectó de VSCode y dejó el push a medias. La operación quedó en un estado inconsistente: muchos ficheros ya integrados en la rama principal, otros pendientes, y ningún registro claro de dónde estaba el corte.

Tuve que cambiar de plataforma para continuar (en este caso, a Claude Code), pero llegué sin contexto. Aparentemente quedaban setecientos ficheros sin procesar, aunque también había muchos ya pasados. ¿Cómo saber exactamente cuáles? ¿Cómo no duplicar trabajo o, peor, sobrescribir lo que ya estaba bien?

La respuesta vino del propio WORM. Le pedí al sistema que consultara el log inmutable, identificara los últimos artefactos confirmados como integrados, y a partir de ahí reconstruyera el trabajo pendiente. La recuperación de contexto fue cuestión de minutos, no de horas. El artefacto demostró su valía en una situación que no tenía nada que ver con las auditorías de compliance: era simplemente un fallo operacional de plataforma, y el log permitió resolverlo sin pérdida real.

Desde entonces, defiendo el WORM no solo como cumplimiento normativo, sino como seguro operacional. Cuando trabajas con agentes que generan miles de líneas por hora, el coste de no saber dónde se quedó cada cosa es enorme. El coste de mantener el registro, mínimo.

Retrospectiva del sprint: el sistema aprende solo

Una vez el software está en producción y todo el log está consolidado, se ejecuta lo que en Designio llamamos retrospectiva agéntica. A diferencia de las retrospectivas humanas de Scrum (reuniones de equipo con post-its, "qué ha ido bien, qué no, qué cambiar"), esta la hace el sistema sobre sí mismo, analizando los datos objetivos que ha recogido durante la semana.

El sistema revisa varias dimensiones:

- Calidad de la ingesta del lunes: cuántas repreguntas necesitó el agente interrogador para construir EIS con cero ambigüedad. Si el número fue alto, se ajustan las plantillas de captura de requisitos para el próximo sprint, afinando la capacidad de comprensión del agente.
- Ajuste de prompts del Constructor y del Shadow Architect (si intervino): si durante la semana se detectaron errores recurrentes en el código generado o en las propuestas arquitectónicas, se re-calibran las instrucciones base de esos agentes. El sistema aprende las preferencias estilísticas, las restricciones técnicas y los estándares específicos del cliente.
- Cuellos de botella del sprint: el sistema identifica dónde hubo más intervención del Delivery Lead, qué agentes necesitaron más re-prompts, qué partes del EIS generaron más dudas. Si hay patrones que se repiten sprint tras sprint, se ajustan parámetros más profundos: temperatura de inferencia, reglas de razonamiento, límites de validación, flujos de interrogación.
- Cierre financiero de tokens: se compara el gasto real frente al presupuesto proyectado el lunes. Si hay desviación significativa, se analiza dónde se produjo y se ajustan las cuotas y los modelos para el próximo sprint. A veces la conclusión es que se puede usar un modelo más ligero para tareas rutinarias; otras veces, que conviene invertir más en tareas creativas o críticas. El Financial Guard del paso 3 alimenta esta dimensión con los datos del consumo real.

La diferencia con una retrospectiva tradicional es que aquí no hay reuniones ni post-its ni subjetividad. El sistema mira lo que ha pasado, identifica los patrones, y aplica los ajustes automáticamente para el lunes siguiente. Esto es lo verdaderamente diferencial de Designio: el equipo no mejora solo porque sus miembros se acostumbren con el tiempo; mejora porque el propio sistema agéntico se va afinando sprint a sprint.

Retrospectiva humana: solo cuando hace falta

Conviene aclarar algo, porque puede parecer que Designio elimina al equipo humano del aprendizaje, y no es así.

En proyectos donde el sprint ha transcurrido sin fricción significativa, la retrospectiva agéntica es suficiente. No tiene sentido convocar al equipo a una reunión semanal de retrospectiva si la semana ha ido bien y los ajustes que hay que hacer son técnicos del sistema. Eso es ceremonia vacía, y Designio prefiere ahorrar ese tiempo.

Pero en sprints donde sí ha habido fricción importante (un BO descontento, un evolutivo controvertido, un Error de Intención que ha costado varias iteraciones, un escalado al AIGB), conviene convocar una conversación breve del equipo. Treinta minutos como máximo, con las personas implicadas: Intent Director, Delivery Lead, Arquitecto si participó, y cualquier perfil relevante. Tres preguntas:

- ¿Qué ha funcionado bien en la colaboración humano-agente esta semana?
- ¿Dónde hemos tenido fricción que se pueda evitar la próxima vez?
- ¿Qué cambiamos para el próximo sprint?

Esta retrospectiva humana no es obligatoria cada viernes. Se convoca por necesidad, no por calendario. Y cuando se convoca, sus conclusiones se registran en el log de trazabilidad como entrada cualitativa que complementa los ajustes técnicos del sistema. De esta forma, el aprendizaje humano y el aprendizaje del sistema avanzan en paralelo cuando hace falta, sin redundar el uno con el otro cuando no hace falta.

Cierre del viernes

Al final del viernes, lo que tenemos es: software en producción, una EIS marcada como satisfecha y firmada, una rama de sprint cerrada e integrada en main con su tag de versión, un Traceability Log completo y sellado, un sistema agéntico ajustado para el próximo sprint con los aprendizajes técnicos de esta semana, y, si fue necesario, una retrospectiva humana breve con acuerdos del equipo.

El lunes siguiente arranca el siguiente sprint con un sistema un poco más afinado, un equipo un poco más calibrado, y un cliente que ya tiene una versión más del producto funcionando. Esa es la promesa de Designio: no solo construir software con IA, sino que cada ciclo deje el sistema y al equipo mejor preparados para el siguiente. Sprint a sprint, lo que al principio cuesta esfuerzo se va volviendo natural, y lo que al principio requería intervención humana constante se va volviendo automático.

Por eso conviene insistir en algo que ya hemos mencionado en otros puntos del libro: los primeros sprints son los más exigentes. No por falta de capacidad del sistema, sino porque todavía no ha aprendido el contexto concreto del proyecto. A partir del tercer o cuarto sprint, cuando el sistema ya ha calibrado plantillas, prompts, presupuestos y patrones, la productividad del equipo da un salto cualitativo que en metodologías clásicas simplemente no es alcanzable. Ese es el momento en que las cifras de autonomía del 95% que mencionamos en la fase 4 dejan de ser aspiración y empiezan a ser realidad medible.

Y con esto, el sprint queda cerrado.

¿Te ha resultado útil el corazón operativo del método?

Este es solo uno de los ocho bloques del libro: la Fase 0 de cimientos, el ecosistema técnico, la jerarquía de modelos con fallback y las cuentas del dividendo de la intención te esperan en el resto.

Comprar «Soberanía de la Intención» — 19,95 € en Amazon

https://www.amazon.es/dp/B0H4X2QQCK