Planificación de Implementación vs. Desarrollo: un modelo mental al que vuelvo a menudo

Una reflexión práctica sobre cómo planificar la implementación puede reducir esfuerzo de desarrollo - sin tratar el gráfico como un modelo científico.

es-ES - 11 de junio de 2026 - Desarrollo de Software / Planificación / Ingeniería

Esto no es un estudio. No es un benchmark. Y definitivamente no soy yo intentando demostrar que un gráfico puede explicar cada decisión de ingeniería.

Es solo un modelo mental al que vuelvo a menudo después de trabajar en funcionalidades, refactorizaciones, integraciones, migraciones y cambios más grandes en los que el verdadero reto no era solamente escribir el código, sino entender qué estaba a punto de tocar ese código.

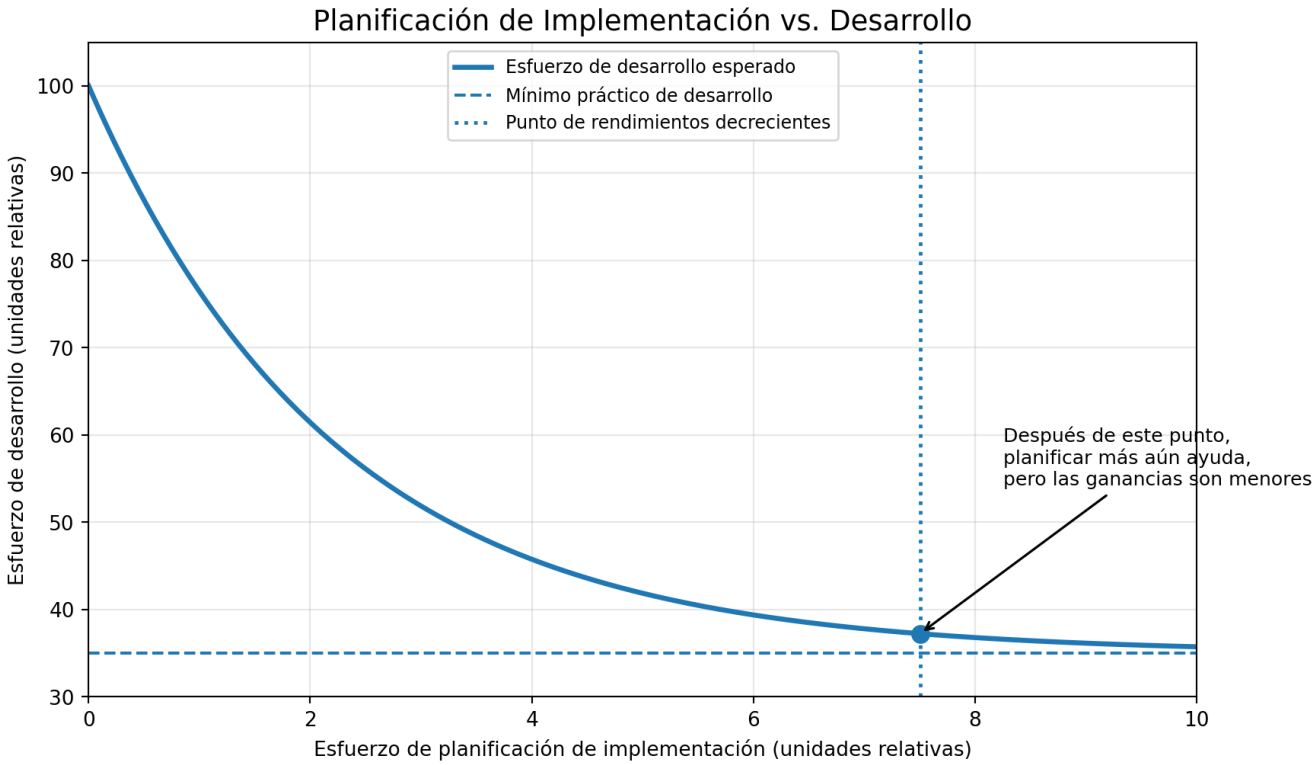
La idea se volvió más clara para mí después de un cambio en la forma en que nuestro equipo empezó a organizar parte del trabajo. En el proyecto en el que trabajo, empezamos a crear tareas específicas para **planes de implementación** antes de iniciar las tareas reales de desarrollo. En algunos casos, la tarea de desarrollo solo se creaba, se refinaba o se dividía correctamente después de terminar la tarea de planificación de implementación.

Al principio, esa separación me pareció un poco extraña. Una parte de mi cabeza quería ir directo al código, porque ahí es donde las cosas se sienten más concretas. Pero, después de ver este proceso algunas veces, empecé a entender el valor de tratar el plan como una pieza propia del trabajo. No como burocracia. No como ceremonia. Más bien como una pausa para entender el terreno antes de elegir la ruta.

Cuando digo **Planificación de Implementación**, me refiero al trabajo que ocurre antes de construir con las manos en el código: entender el impacto, revisar límites entre sistemas, pensar en la estructura, leer código existente, identificar patrones, listar casos de prueba y sacar a la superficie esos pasos manuales aburridos, pero importantes, que pueden sorprender al equipo más adelante.

Cuando digo **Desarrollo**, me refiero a la parte práctica: escribir código, ajustar pruebas, revisar, integrar, depurar, desplegar y lidiar con las cosas que solo aparecen cuando el cambio empieza a existir de verdad.

La forma en que imagino esta relación se parece más o menos a esto:



Modelo ilustrativo - los valores son conceptuales, no están calibrados con datos reales del proyecto.

Los números no son el punto principal. Son unidades relativas e ilustrativas. Lo útil es la forma de la curva.

La idea simple

Cuanto más útil sea la planificación de implementación, menos esfuerzo innecesario de desarrollo solemos crear después.

No menos esfuerzo en el sentido de que programar se vuelva mágicamente fácil. No lo hace.

Pero sí menos esfuerzo desperdiciado en confusión evitable.

Menos retrabajo causado por una regla de negocio escondida. Menos idas y vueltas porque la responsabilidad de un flujo no estaba clara. Menos descubrimientos tardíos de un detalle de integración. Menos "ah, esto también afecta a ese otro servicio". Menos adivinar cuando la implementación ya está en marcha.

Ahí es donde la planificación ayuda mucho.

Un buen plan de implementación no es un contrato con el futuro. Para mí, se parece más a un mapa. No elimina todas las sorpresas del camino, pero ayuda al equipo a no caminar en círculos.

Por qué una tarea separada de planificación puede ayudar

Algo que me gusta de tener una tarea dedicada a la planificación de implementación es que le da al equipo permiso para bajar la velocidad por la razón correcta.

Sin ese paso explícito, la planificación muchas veces ocurre en fragmentos: un comentario rápido aquí, una conversación en Slack allá, una pequeña investigación durante el desarrollo y algunos detalles importantes descubiertos solo después de que el trabajo ya empezó. A veces eso está bien. Pero, en cambios más grandes, esos descubrimientos dispersos pueden volverse costosos en

silencio.

Una tarea separada de planificación crea un pequeño espacio para lidiar con las incógnitas. Le da tiempo al ingeniero para leer el código, hacer preguntas, revisar supuestos y convertir una tarea de desarrollo vaga en algo que el equipo realmente puede razonar.

A veces el resultado no es un documento enorme. Puede ser un plan corto, una lista de áreas afectadas, una secuencia propuesta de implementación o algunos riesgos que necesitan alineación. El valor no está en producir un plan bonito. El valor está en hacer que la siguiente tarea sea menos ciega.

En ese sentido, la tarea de desarrollo que viene después del plan suele estar mejor formada. Tiene límites más claros, mejores expectativas de prueba, menos pasos manuales escondidos y una comprensión más honesta de lo que puede salir mal.

Qué intenta descubrir realmente la planificación

Para mí, la parte más valiosa de planificar no es crear un documento largo. Es forzar a que las preguntas correctas aparezcan antes de que cambiar de dirección sea demasiado caro.

Una feature o un gran cambio técnico normalmente trae más contexto del que muestra el ticket. Hay límites entre sistemas, reglas de negocio, decisiones heredadas, patrones de código, compromisos históricos, detalles de rollout y, a veces, algunas zonas del tipo "por favor, no toques esto si no sabes exactamente por qué".

La planificación de implementación es el momento en el que intento acercar esos detalles a la superficie.

Cosas como:

- ¿Qué sistemas, módulos, jobs, eventos o APIs se verán afectados?
- ¿Cuál es la estructura propuesta para la implementación?
- ¿Qué patrones existentes del código deberían guiar la solución?
- ¿Qué patrones probablemente deberían evitarse?
- ¿Qué reglas de negocio o comportamientos heredados pueden cambiar la dirección del trabajo?
- ¿Qué casos de prueba deben cubrirse?
- ¿Qué casos borde son fáciles de olvidar?
- ¿Qué pasos manuales están involucrados?
- ¿Hay scripts, migraciones, backfills, feature flags, cambios de configuración o pasos de rollout?
- ¿Qué necesita revisarse después del release?
- ¿Qué puede automatizarse y qué todavía necesita validación humana?

Cuanto más complejo es el sistema, más importan estos detalles.

En una base de código limpia y aislada, tal vez el plan pueda ser muy pequeño. Pero, en un producto real con historia, dependencias, lógica heredada, reglas específicas de negocio y decisiones técnicas

anteriores, la planificación de implementación deja de ser sobre escribir un plan y pasa a ser sobre reducir la cantidad de sorpresas costosas.

Por qué la primera parte de la curva cae rápido

Al principio, la planificación suele tener un retorno alto.

Incluso una conversación corta, una pequeña nota de diseño o una exploración rápida del código puede eliminar una cantidad sorprendente de incertidumbre.

A veces una hora de planificación ahorra muchas horas de desarrollo porque evita que el equipo empiece en la dirección equivocada.

Esa es la parte empinada de la curva.

El equipo descubre que un flujo ya existe. O que un campo tiene un significado escondido. O que una dependencia se comporta distinto en producción. O que un cambio parecido ya causó problemas antes. O que la implementación puede ser mucho más simple si sigue un patrón existente.

Este es el tipo de planificación que se siente valiosa de inmediato. Convierte "empecemos a programar y veamos qué pasa" en "conocemos el camino principal, los riesgos principales y las primeras decisiones".

Por qué la curva termina aplanándose

Pero la planificación tiene límites.

En algún punto, planificar más todavía ayuda, pero la ganancia se vuelve menor. Esa es la parte en la que la curva empieza a aplanarse.

Hay algunas razones para eso.

Primero, el desarrollo nunca llega a cero. Incluso con un gran plan, alguien todavía tiene que escribir el código, probarlo, revisarlo, integrarlo, desplegarlo y corregir las cosas que solo aparecen cuando el sistema se ejercita.

Segundo, algunas incertidumbres no pueden eliminarse de antemano. Algunas preguntas solo se vuelven reales cuando el código se encuentra con el sistema existente. Se puede leer, inspeccionar y discutir mucho, pero a veces la respuesta honesta es: necesitamos construir una parte pequeña y validarla.

Tercero, demasiada planificación puede empezar a darle al equipo una falsa sensación de certeza. El documento crece, pero el riesgo no necesariamente disminuye al mismo ritmo.

Por eso no veo la planificación como algo que deba maximizarse.

La veo como algo que debe calibrarse.

Una pequeña ecuación para el modelo mental

Me gusta esta ecuación como una forma simple de representar la idea:

```
Desarrollo(P) = D_min + (D0 - D_min) * e^(-kP)
```

Donde:

- `P` es el esfuerzo de planificación de implementación
- `Desarrollo(P)` es el esfuerzo esperado de desarrollo
- `D0` es el esfuerzo de desarrollo cuando la planificación está cerca de cero
- `D_min` es el mínimo práctico de esfuerzo de desarrollo
- `k` representa cuánto la planificación reduce incertidumbre y retrabajo

Esta no es una fórmula que usaría para estimar una sprint real.

No pondría estos números en una hoja de cálculo fingiendo que son precisos.

La ecuación es útil solo porque captura la forma de la intuición: la planificación reduce el esfuerzo de desarrollo rápidamente al principio y, después, el retorno se vuelve menor.

Las dos trampas

La primera trampa es planificar de menos.

Esto ocurre cuando el equipo empieza a desarrollar antes de entender lo suficiente sobre impacto, dependencias, estructura, pruebas, rollout y restricciones específicas del sistema. La planificación que faltó no desaparece. Se mueve a la fase de desarrollo, donde aparece como interrupciones, retrabajo y descubrimientos de último minuto.

La segunda trampa es planificar de más.

Esto ocurre cuando el equipo intenta responder todas las preguntas posibles antes de construir cualquier cosa. El plan se vuelve más largo, pero el equipo no siempre está proporcionalmente más seguro. A veces el siguiente aprendizaje útil no es otra reunión ni otro diagrama. Es un pequeño spike de implementación, una prueba o validar un supuesto en el código.

Las dos trampas son comprensibles.

Planificar de menos muchas veces viene de la presión por avanzar rápido. Planificar de más muchas veces viene del deseo de evitar errores.

Yo he hecho ambas cosas. La mayoría de los equipos probablemente también.

Lo difícil es encontrar el punto medio.

Cómo se siente una planificación suficiente

Para mí, una planificación suficiente suele sentirse así:

El equipo entiende el camino principal. Las áreas de riesgo son visibles. La estructura de implementación no es un misterio. Los patrones de código son conocidos. La estrategia de pruebas está lo suficientemente clara. Los pasos manuales están listados. El rollout no depende de la esperanza. Las incógnitas que quedan son aceptables e intencionales.

Esa última parte importa: aceptables e intencionales.

Un buen plan no elimina todas las dudas. Hace que las dudas restantes sean lo suficientemente explícitas para que el equipo decida llevarlas al desarrollo.

Eso se siente muy distinto a descubrirlas por accidente más tarde.

Mi conclusión por ahora

La planificación de implementación es una forma de mover la incertidumbre hacia antes, cuando normalmente es más barato discutir, cuestionar y ajustar.

Ayuda a que el desarrollo sea más enfocado, pero tiene rendimientos decrecientes. El objetivo no es crear el plan más completo posible. El objetivo es crear claridad suficiente para construir con confianza.

Para mí, el punto ideal está en algún lugar entre caos y teatro.

No "simplemente programemos y descubramos en el camino".

No "planifiquemos hasta que nada se sienta incierto".

Más bien:

"Entendamos lo suficiente para tomar las próximas decisiones con responsabilidad, y luego dejemos que la implementación nos enseñe el resto."

Ese es el equilibrio en el que todavía estoy intentando mejorar.