Planejamento de Implementação vs. Desenvolvimento: um modelo mental ao qual eu sempre volto

Uma reflexão prática sobre como planejar a implementação pode reduzir esforço de desenvolvimento - sem tratar o gráfico como um modelo científico.

pt-BR - 11 de junho de 2026 - Desenvolvimento de Software / Planejamento / Engenharia

Isto não é um estudo. Não é um benchmark. E com certeza não sou eu tentando provar que um gráfico consegue explicar toda decisão de engenharia.

É só um modelo mental ao qual eu tenho voltado depois de trabalhar em features, refatorações, integrações, migrações e mudanças maiores em que o desafio real não era apenas escrever o código, mas entender tudo que aquele código estava prestes a tocar.

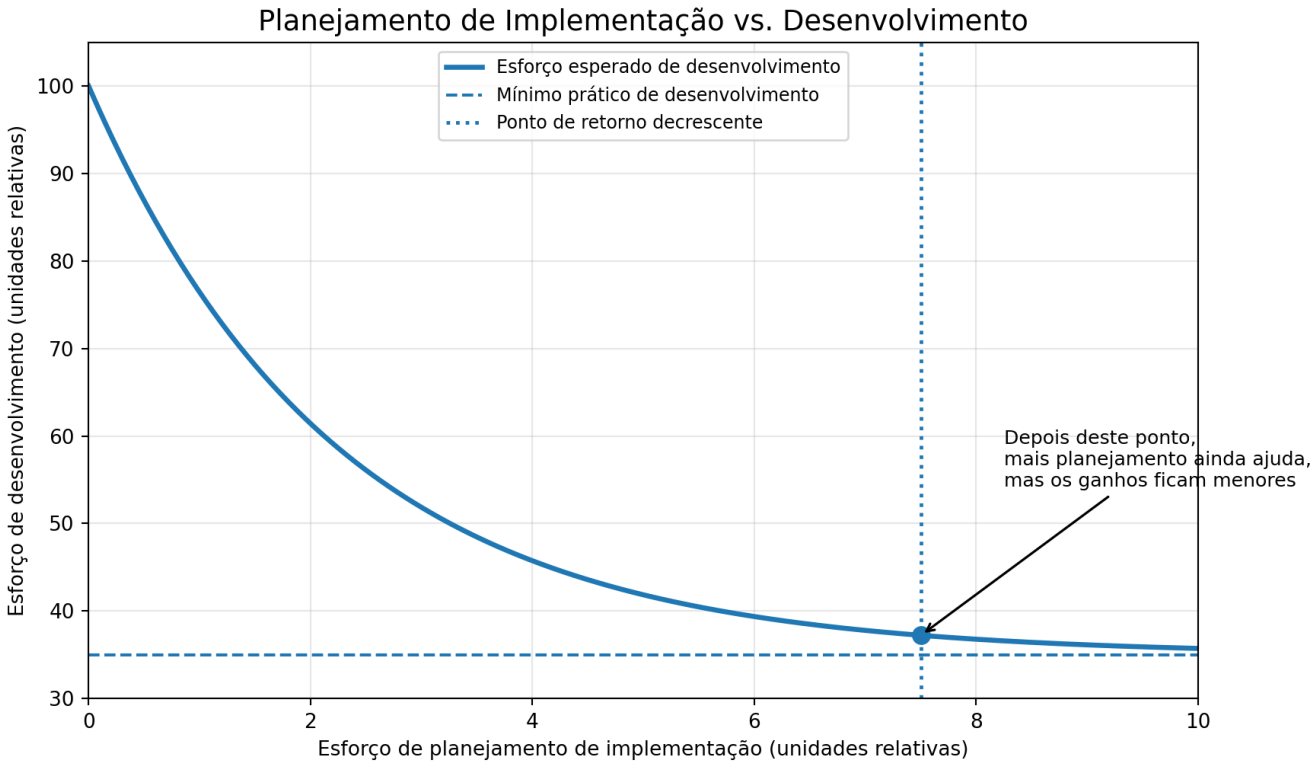
A ideia ficou mais clara para mim depois de uma mudança na forma como nosso time começou a organizar alguns trabalhos. No projeto em que trabalho, começamos a criar tarefas específicas para **planos de implementação** antes de iniciar as tarefas reais de desenvolvimento. Em alguns casos, a tarefa de desenvolvimento só era criada, refinada ou quebrada corretamente depois que a tarefa de planejamento de implementação era finalizada.

No começo, essa separação me pareceu um pouco estranha. Uma parte do meu cérebro queria ir direto para o código, porque é ali que as coisas parecem mais concretas. Mas, depois de ver esse processo acontecer algumas vezes, comecei a entender o valor de tratar o plano como uma parte própria do trabalho. Não como burocracia. Não como cerimônia. Mais como uma pausa para entender o terreno antes de escolher a rota.

Quando eu digo **Planejamento de Implementação**, estou falando do trabalho que acontece antes de colocar a mão na massa: entender o impacto, checar fronteiras entre sistemas, pensar na estrutura, ler código existente, identificar padrões, listar casos de teste e expor aqueles passos manuais meio chatos, mas importantes, que podem surpreender o time depois.

Quando eu digo **Desenvolvimento**, estou falando da parte prática: escrever código, ajustar testes, revisar, integrar, debugar, fazer deploy e lidar com as coisas que só aparecem quando a mudança começa a existir de verdade.

A forma como eu imagino essa relação se parece mais ou menos com isto:



Modelo ilustrativo - os valores são conceituais, não calibrados com dados reais de projeto.

Os números não são o ponto principal. Eles são unidades relativas e ilustrativas. O que importa é o formato da curva.

A ideia simples

Quanto mais útil for o planejamento de implementação, menos esforço desnecessário de desenvolvimento a gente costuma criar depois.

Não menos esforço no sentido de que programar vira algo magicamente fácil. Não vira.

Mas menos esforço desperdiçado com confusão evitável.

Menos retrabalho causado por uma regra de negócio escondida. Menos vai e volta porque a responsabilidade de um fluxo não estava clara. Menos descoberta tardia de um detalhe de integração. Menos "ah, isso também afeta aquele outro serviço". Menos tentativa e erro quando a implementação já está em andamento.

É aí que o planejamento ajuda bastante.

Um bom plano de implementação não é um contrato com o futuro. Para mim, ele se parece mais com um mapa. Ele não remove todas as surpresas do caminho, mas ajuda o time a não ficar andando em círculos.

Por que uma tarefa separada de planejamento pode ajudar

Uma coisa que eu gosto em ter uma tarefa dedicada ao planejamento de implementação é que ela dá ao time permissão para desacelerar pelo motivo certo.

Sem essa etapa explícita, o planejamento muitas vezes acontece em pedaços: um comentário rápido aqui, uma conversa no Slack ali, uma pequena investigação durante o desenvolvimento e alguns detalhes importantes descobertos só depois que o trabalho já começou. Às vezes isso

funciona. Mas, em mudanças maiores, essas descobertas espalhadas podem ficar caras silenciosamente.

Uma tarefa separada de planejamento cria um pequeno espaço para lidar com as incertezas. Ela dá tempo para o engenheiro ler o código, fazer perguntas, checar hipóteses e transformar uma tarefa de desenvolvimento vaga em algo que o time consegue realmente discutir e entender.

Às vezes o resultado não é um documento enorme. Pode ser um plano curto, uma lista de áreas afetadas, uma sequência sugerida de implementação ou alguns riscos que precisam de alinhamento. O valor não está em produzir um plano bonito. O valor está em tornar a próxima tarefa menos cega.

Nesse sentido, a tarefa de desenvolvimento que vem depois do plano costuma nascer melhor formada. Ela tem limites mais claros, expectativas de teste melhores, menos passos manuais escondidos e uma visão mais honesta do que pode dar errado.

O que o planejamento está tentando descobrir

Para mim, a parte mais valiosa do planejamento não é criar um documento longo. É forçar as perguntas certas a aparecerem antes que mudar de direção fique caro demais.

Uma feature ou uma grande mudança técnica geralmente carrega muito mais contexto do que o ticket mostra. Existem fronteiras entre sistemas, regras de negócio, decisões legadas, padrões de código, compromissos históricos, detalhes de rollout e, às vezes, algumas áreas do tipo "por favor, não mexa aqui sem saber exatamente o porquê".

O planejamento de implementação é o momento em que eu tento trazer esses detalhes mais perto da superfície.

Coisas como:

- Quais sistemas, módulos, jobs, eventos ou APIs serão afetados?
- Qual é a estrutura proposta para a implementação?
- Quais padrões existentes do código devem guiar a solução?
- Quais padrões provavelmente devem ser evitados?
- Quais regras de negócio ou comportamentos legados podem mudar a direção do trabalho?
- Quais casos de teste precisam ser cobertos?
- Quais casos de borda são fáceis de esquecer?
- Quais passos manuais estão envolvidos?
- Existem scripts, migrações, backfills, feature flags, mudanças de configuração ou etapas de rollout?
- O que precisa ser verificado depois do release?
- O que pode ser automatizado e o que ainda precisa de validação humana?

Quanto mais complexo o sistema, mais esses detalhes importam.

Em uma base de código limpa e isolada, talvez o plano possa ser bem pequeno. Mas, em um produto real com história, dependências, lógica legada, regras específicas de negócio e decisões

técnicas anteriores, o planejamento de implementação deixa de ser sobre escrever um plano e passa a ser sobre reduzir a quantidade de surpresas caras.

Por que a primeira parte da curva cai rápido

No começo, o planejamento costuma ter um retorno alto.

Mesmo uma conversa curta, uma pequena nota de design ou uma exploração rápida do código pode remover uma quantidade surpreendente de incerteza.

Às vezes uma hora de planejamento economiza muitas horas de desenvolvimento porque evita que o time comece na direção errada.

Essa é a parte íngreme da curva.

O time descobre que um fluxo já existe. Ou que um campo tem um significado escondido. Ou que uma dependência se comporta diferente em produção. Ou que uma mudança parecida já causou problemas antes. Ou que a implementação pode ser muito mais simples se seguir um padrão existente.

Esse é o tipo de planejamento que parece valioso imediatamente. Ele transforma "vamos começar a codar e ver o que acontece" em "sabemos o caminho principal, os principais riscos e as primeiras decisões".

Por que a curva eventualmente fica mais plana

Mas o planejamento tem limites.

Em algum ponto, planejar mais ainda ajuda, mas o ganho passa a ser menor. Essa é a parte em que a curva começa a ficar mais plana.

Existem alguns motivos para isso.

Primeiro, o desenvolvimento nunca chega a zero. Mesmo com um ótimo plano, alguém ainda precisa escrever o código, testar, revisar, integrar, fazer deploy e corrigir coisas que só aparecem quando o sistema é exercitado.

Segundo, algumas incertezas não podem ser removidas antecipadamente. Algumas perguntas só se tornam reais quando o código encontra o sistema existente. Dá para ler, investigar e discutir bastante, mas às vezes a resposta honesta é: precisamos construir uma parte pequena e validar.

Terceiro, planejamento demais pode começar a dar ao time uma falsa sensação de certeza. O documento fica maior, mas o risco não necessariamente diminui na mesma proporção.

Por isso eu não vejo planejamento como algo a ser maximizado.

Eu vejo como algo a ser calibrado.

Uma pequena equação para o modelo mental

Eu gosto desta equação como uma forma simples de representar a ideia:

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

Onde:

- P é o esforço de planejamento de implementação

- `Desenvolvimento(P)` é o esforço esperado de desenvolvimento
- `D0` é o esforço de desenvolvimento quando o planejamento é quase zero
- `D_min` é o mínimo prático de esforço de desenvolvimento
- `k` representa o quanto o planejamento reduz incerteza e retrabalho

Esta não é uma fórmula que eu usaria para estimar uma sprint real.

Eu não colocaria esses números em uma planilha fingindo que eles são precisos.

A equação é útil apenas porque captura o formato da intuição: o planejamento reduz o esforço de desenvolvimento rapidamente no começo e, depois, o retorno vai ficando menor.

As duas armadilhas

A primeira armadilha é planejar de menos.

Isso acontece quando o time começa a desenvolver antes de entender o suficiente sobre impacto, dependências, estrutura, testes, rollout e restrições específicas do sistema. O planejamento que faltou não desaparece. Ele se muda para a fase de desenvolvimento, onde aparece como interrupções, retrabalho e descobertas de última hora.

A segunda armadilha é planejar demais.

Isso acontece quando o time tenta responder todas as perguntas possíveis antes de construir qualquer coisa. O plano fica maior, mas o time nem sempre fica proporcionalmente mais seguro. Às vezes o próximo aprendizado útil não é outra reunião ou outro diagrama. É um pequeno spike, um teste ou a validação de uma hipótese no código.

As duas armadilhas são compreensíveis.

Planejar de menos muitas vezes vem da pressão para andar rápido. Planejar demais muitas vezes vem da vontade de evitar erros.

Eu já fiz os dois. A maioria dos times provavelmente também.

A parte difícil é encontrar o meio.

Como se parece planejamento suficiente

Para mim, planejamento suficiente costuma ter esta sensação:

O time entende o caminho principal. As áreas de risco estão visíveis. A estrutura de implementação não é um mistério. Os padrões de código são conhecidos. A estratégia de testes está clara o bastante. Os passos manuais estão listados. O rollout não depende de esperança. As incertezas que sobraram são aceitáveis e intencionais.

Essa última parte importa: aceitáveis e intencionais.

Um bom plano não elimina todas as dúvidas. Ele torna as dúvidas restantes explícitas o suficiente para que o time decida carregá-las para o desenvolvimento.

Isso é bem diferente de descobri-las por acidente mais tarde.

Minha conclusão por enquanto

Planejamento de implementação é uma forma de mover a incerteza para mais cedo, quando normalmente é mais barato discutir, questionar e ajustar.

Ele ajuda o desenvolvimento a ficar mais focado, mas tem retornos decrescentes. O objetivo não é criar o plano mais completo possível. O objetivo é criar clareza suficiente para construir com confiança.

Para mim, o ponto ideal fica em algum lugar entre caos e teatro.

Não "vamos só codar e descobrir no caminho".

Não "vamos planejar até nada parecer incerto".

Mais como:

"Vamos entender o suficiente para tomar as próximas decisões com responsabilidade e deixar que a implementação nos ensine o restante."

Esse é o equilíbrio que eu ainda estou tentando melhorar.