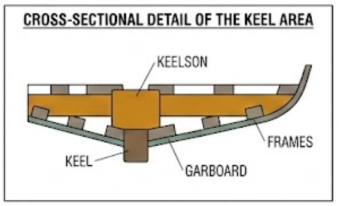
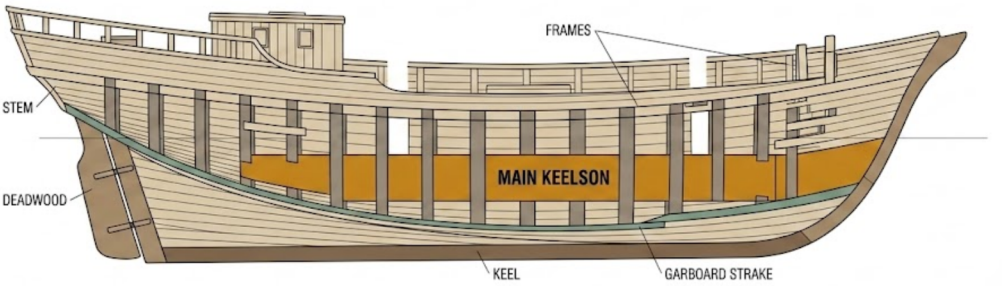
CONFIANTE E ERRADO

MÉTODO KEELSON





Desenvolvendo software de verdade com a colaboração de agentes de IA.

José Paschenda

Edição V0.1

SYNTAX-i
TECH PUBLISHER

Confiante e Errado

Seu colega brilhante tem amnésia e mente com convicção — eis como construir software com ele mesmo assim.

José Paschenda

Esse livro está à venda em https://leanpub.com/confiante-errado

Essa versão foi publicada em 2026-08-09

Esse é um livro Leanpub. A Leanpub dá poderes aos autores e editores a partir do processo de Publicação Lean. Publicação Lean é a ação de publicar um ebook em desenvolvimento com ferramentas leves e muitas iterações para conseguir feedbacks dos leitores, pivotar até que você tenha o livro ideal e então conseguir tração.

Conteúdo

Confiante e Errado

Seu colega brilhante tem amnésia e mente com convicção — eis como construir software com ele mesmo assim.

Sobre este rascunho

Este livro está inacabado — de propósito.

Ele segue a mesma regra que ensina: congelar só o que já provou seu valor e manter todo o resto aberto — para aprender, revisar e versionar. Seria incoerente fazer diferente. Este não é um monumento à verdade final; é o caderno de campo de um ofício que muda rápido demais para caber em qualquer edição "definitiva".

Um livro sobre o risco de estar confiante e errado não poderia, ele mesmo, se declarar pronto.

Por isso escolhi a Leanpub e o princípio do *Lean Publishing* — "**Publish Early, Publish Often**": publique cedo, publique com frequência. Na era exponencial, talvez seja o único antídoto realmente eficaz contra o apodrecimento da informação e o jeito honesto de compartilhar este conhecimento.

E para você isso é concreto: **quem entra agora leva todas as edições futuras sem pagar de novo — e ajuda a escrevê-las**. Discordâncias e sugestões não são só bem-vindas: são combustível. Escreva para **jose.paschenda@syntax-i.com**.

Informações Gerais

Rascunho (v0.1.2): prefácio, introdução e Partes I–IV inteiras (sempre em refinamento); Partes V e VI mapeadas e em engorda.

Livro e método registrado na Avtoris em 5 de agosto de 2026 às 22:40 : Ver Registro na Avctoris.

A tese e a história

Este livro tem uma **tese** e uma **história**.

A tese, em resumo: quando o programador mais frequente de um projeto passa a ser uma IA, um novo artefato passa a valer tanto quanto o código e a especificação — a **memória viva** que mantém você e o agente na mesma página, de uma sessão para a outra, de um dia para o outro. Cuidar dela não é tarefa de documentação; é uma disciplina de três peças: o que o projeto **lembra**, como o trabalho **avança** — só sob prova de que funciona — e o que só **você**, o humano, pode decidir. É ela que impede o trabalho do agente de derivar: sair, aos poucos, da linha.

A história é como nada disso foi inventado numa mesa. O método emergiu de um sistema real, em produção, e se refinou pela mesma regra que ensina — congelar só o que o uso provou. Ela começa numa terça-feira; é para lá que vamos agora…

* * *

Minhas considerações iniciais

Numa terça-feira, o agente de IA com quem eu programava quase reabriu uma decisão de arquitetura que ele mesmo havia fechado comigo na segunda. Era um sistema real — uma estratégia refinada operando na bolsa de valores, com dinheiro de verdade em jogo — e, poucos dias antes, um mesmo conceito já aparecera definido de dois jeitos diferentes, em dois documentos que o próprio agente escrevera. No centro de tudo, o arquivo que deveria ser a memória do projeto, um `architecture.md`, havia crescido tanto que relê-lo a cada sessão custava caro — e, mesmo caro, não impedia nenhuma dessas confusões.

Não era um problema de prompt. Nenhuma técnica de escrever pedidos melhores resolveria aquilo, porque o buraco não estava no que eu pedia: estava no que o projeto conseguia lembrar de uma sessão para a outra, de um dia para o outro. Foi ali que a ficha caiu. Construir software sério com um agente de IA — de ciclo de vida longo, ao longo de muitas e muitas sessões — é muito mais do que escrever bons prompts e receber bom código. É manter de pé, sessão após sessão, um sistema que precisa continuar funcional, correto, estável, expansível e seguro para mudar — e conquistar isso ao lado de um parceiro de desenvolvimento que erra com a mesma confiança com que acerta.

Se você já construiu algo de verdade ao lado de um desses agentes, talvez reconheça a sensação: as ferramentas para *escrever* código nem sempre entregam o padrão que você pediu, às vezes as coisas simplesmente não encaixam, códigos testados são modificados ou apagados sem razão, e continua estranhamente difícil *manter um projeto coerente* ao longo de dezenas de sessões. Muitas vezes o agente brilha hoje em uma sessão, mas esquece no dia seguinte. Conheço o cansaço de refazer o que já estava pronto, e a estranha solidão de discutir com um colaborador que não lembra da conversa de ontem — houve dias de frustração pura, e no Capítulo 15 conto, sem filtro, o que cheguei a escrever num prompt num desses dias. Este livro é sobre o que precisa existir no vão entre uma sessão e a próxima — e é o que eu queria ter tido em mãos no dia em que aquele `architecture.md` começou a apodrecer.

Senti o tamanho do desafio: **meu brilhante colega programador tem amnésia e mente com convicção — como construir software de verdade ao lado dele?**

Cada página que você vai ler aqui foi arrancada dessa fricção. Nenhum mecanismo aqui foi inventado por elegância e depois posto à procura de um problema; a ordem foi sempre a inversa — dificuldade observada, solução tentada, resultado medido ali mesmo, no dia a dia das especificações, da arquitetura, do código e dos testes. Ao longo do caminho, procurei

entender o que a ciência e o mercado já haviam construído e trazer as ideias, pensamentos e soluções que encontrei trabalhando com um agente desenvolvedor. E o método acabou obedecendo à própria regra que prega: boa parte do que há de mais importante aqui só apareceu *depois* que ele já estava em uso, revelada por sessões que mostraram o que faltava. O aprendizado volta versionado — inclusive para dentro deste livro. É por isso que você pode confiar nestas páginas mais do que confiaria numa boa ideia ainda não testada: elas só chegaram até aqui depois de sobreviver ao uso real.

Isso impõe um compromisso de tom que peço que você me cobre do início ao fim: **honestidade científica** e **pés no chão**.

Pés no chão significa não embarcar em um artefato ou processo só porque parece bonito, nem criar um formalismo que não se paga — e o retorno aqui se mede na qualidade da arquitetura, do sistema, do código e, no fim das contas, em uma solução mais útil para quem a utiliza.

Honestidade científica significa escrever com sinceridade, contando histórias reais de onde vieram as inspirações, por mais simples que sejam. Nada de glamour ou espetáculo inventado para prender sua atenção. Significa informar o número exato onde medimos — inclusive os números que *contrariam* a tese central e nos obrigam a temperá-la, os quais fiz questão de deixar no livro de propósito. Significa creditar a fonte primária onde uma prática já era boa e ter a coragem de escrever *"não sabemos ainda"* quando essa for a resposta certa. Um método que só confirma a si mesmo não merece confiança; um livro sobre ele, menos ainda.

E há algo que precisa ser dito logo na entrada: **nada disso é imutável**. As ideias e o processo permanecem vivos, envolvidos num movimento contínuo de observar as dificuldades e buscar soluções. Não me sinto chegando a nenhum platô onde se possa, enfim, descansar; e, para ser franco, desconfio da própria ideia de platô, ainda mais nestes *tempos exponenciais*. Não me parece certa aquela imagem de que a vida é escalar com esforço até um platô imaginário, para depois viver para sempre na glória da sua paisagem. A vida se parece mais com uma jornada contínua, em que cada passo conquista mais um metro num percurso sem fim e, ao conquistá-lo, revela o que há ainda mais adiante — e que é preciso seguir caminhando. Essa é a pedra fundamental deste trabalho, e deste livro.

É com esse espírito que peço que você leia. Estas ideias não são um destino; são um ponto de partida. Estarei sempre refinando-as, tentando acompanhar a velocidade inimaginável com que estas tecnologias avançam — e a capacidade, não menos inimaginável, que o ser humano tem de pensar, inventar e criar coisas novas. Que elas te inspirem a encontrar soluções novas, melhores que as minhas, para as dificuldades reais dos seus projetos. Se, ao fechar o livro, você adaptar, discordar e superar o que está aqui, ele terá funcionado exatamente como devia.

Para quem é este livro. Para quem já passou da fase dos brinquedos: constrói software de verdade com a colaboração de agentes de IA e sentiu, na pele, que eles ajudam a *escrever* código, mas não a *manter um projeto coerente* ao longo de dezenas de sessões. Não é um livro de prompts prontos, nem de algumas poucas dicas fáceis para decorar; é um livro sobre a **arquitetura da colaboração** entre um humano e um agente de inteligência artificial no processo — complexo e longo — de construir um sistema de verdade.

E você não fecha o livro sozinho com a teoria na mão. O Método Keelson vive também num **repositório próprio no GitHub**, **gratuito e sempre atualizado**: os padrões que ensinam o próprio agente a trabalhar dentro do método, um prompt de partida para o primeiro dia — projeto novo ou já em andamento —, e um conjunto de automações para o dia a dia da colaboração. Mais sobre isso na seção "Onde buscar o pacote", no Capítulo 17.

Como ler. A Introdução e a Parte I irão contextualizar você dentro do livro explicando o porquê ele foi escrito, eu não pularia esta parte. As Partes II, III e IV — a memória, o processo e o jogador; o tabuleiro, o jogo e quem joga — são o coração prático e devem ser lidas em sequência como um manual e com atenção. Depois de finalizar as Partes II, III e IV você pode abrir o pacote de aplicação gratuito que acompanha o livro, disponível no GitHub, a começar pelo "user guide". Boa parte das dúvidas que podem surgir durante a leitura dos conceitos deste livro tende a ficar mais clara depois da leitura do "user guide" e dos próprios artefatos que moram dentro do pacote.

Os capítulos 21, 22 e 23 podem ajudar no seu entendimento, mas, no geral, as Partes V e VI são mais uma conversa de referência — mais interessantes depois que você já compreendeu a maior parte do livro: a Parte V traça a linhagem intelectual; a Parte VI traz as provas de campo. Quem quiser apenas a tese do método pode ler a Introdução e o Capítulo 10 e sair com o essencial.

Atenção! Se termos como sessão, contexto e token ainda não soam familiares para você, comece pelos Apêndices A e B: um guia de poucos minutos sobre o que é um LLM e a voz da própria IA ensinando a escrever bons prompts. Nesse caso, são eles a sua porta de entrada — vale começar por ali e depois voltar para a Introdução.

* * *

Sobre a obra

"Este livro foi desenvolvido de forma ágil, utilizando o Claude Code (Anthropic) como assistente de cocriação de conteúdo e revisão. Ao longo da leitura, você descobrirá que a própria escrita desta obra ajudou a moldar a construção do método. Muitos de seus princípios do método foram aplicados em tempo real durante a produção das páginas, criando uma relação simbiótica e de indução mútua entre autor/desenvolvedor, Claude Code, método e livro."

* * *

Introdução — O leitor mudou

Tudo começa por entender esse colega. Um gênio que esquece tudo e nunca hesita merece um olhar de perto. Então imagine que você contratou o programador mais impressionante que conheceu em toda a sua carreira. Ele lê uma base de código inteira em minutos, escreve com fluência em qualquer linguagem, não se cansa, não se distrai, não perde o entusiasmo nem no décimo refactoring do dia. Mas o contrato dele tem duas cláusulas estranhas. A primeira: **toda manhã, ele acorda sem lembrar de absolutamente nada** — nem de você, nem do projeto, nem da decisão suada de ontem à tarde. A segunda é pior: **quando ele não sabe, ele não diz "não sei"** — preenche a lacuna com um palpite plausível e o apresenta com a mesma segurança de quem sabe.

Você não precisa imaginar: esse programador já está na sua equipe. É o agente de IA que escreve código com você. E se você viu *Como Se Fosse a Primeira Vez* — o filme em que a personagem de *Drew Barrymore* acorda todo dia sem nenhuma lembrança do dia anterior, e a relação só funciona porque ela assiste, toda manhã, a um vídeo que a põe de novo a par de tudo —, você já tem a imagem exata do seu novo colega: para ele, **todo dia é o primeiro dia**.

É a mesma pergunta que levantei nas considerações iniciais, agora exposta em toda a sua nitidez pela imagem do filme: como se constrói software sério — complexo, de ciclo de vida longo, com dinheiro em jogo — ao lado de um colega assim? Onde está o vídeo que ele precisa assistir toda manhã? E quem garante que o que está gravado nele é verdade?

A resposta passa por um lugar que ninguém acha glamoroso: o registro escrito do projeto — a documentação. Por décadas, ela teve um leitor implícito: um ser humano. Ele aparecia para ler de vez em quando, com memória própria, contexto acumulado e mantido decorado dia após dia (pelo menos boa parte das informações), e paciência para caçar a informação onde ela estivesse — e, mais do que isso, com o julgamento para preencher as lacunas que ninguém escreveu e para ignorar na hora uma informação que ele sabia estar errada.

Escrevíamos para esse leitor humano. Tolerávamos que a documentação crescesse sem poda, que envelhecesse, que misturasse "como o sistema funciona hoje", "alguns planos para o futuro" e "o que aconteceu em março". Corrigir cada errinho, revisar tudo o tempo todo, custava uma atenção e um esforço da equipe que, no geral, não compensavam os benefícios — porque o leitor humano cobria a diferença com a sua alta capacidade de julgamento e com o conhecimento do domínio e da arquitetura que ia acumulando ao longo de toda a jornada.

Se você já esteve no campo de batalha, sabe que às vezes se pagava um preço alto por essa defasagem; mas, na maior parte do tempo, o julgamento humano segurava a barra.

Esse leitor mudou. Num número crescente de projetos, quem mais lê a documentação — de longe — é o agente: o colega impressionante. Ele lê **frio**, a cada sessão, com uma **janela de retenção de informação — o contexto — finita e cara**, que enche rápido; e preenche o que falta com aqueles palpites confiantes. Para esse leitor, a documentação deixa de ser um adorno consultado às vezes e vira **substrato de execução**: é relida a cada sessão, custa tokens, e é a única coisa que impede o modelo de refazer ou desfazer, hoje, o que já estava resolvido ontem. Ela é, literalmente, o vídeo da manhã.

Quando o leitor muda desse jeito, quase tudo que sabíamos sobre "boa documentação de software" precisa ser reexaminado. É preciso repensar como organizar — e, às vezes, até o que escrever — o registro dos requisitos, das especificações, da arquitetura, da topologia, das decisões, dos planos, da implementação, dos testes, das correções e das melhorias. E, mais importante que tudo, é preciso garantir — como nunca — que esse registro esteja, a cada instante, o mais fiel possível ao estado real dos seus planos, do seu projeto e do seu código. Quem já trabalhou em sistemas complexos, de ciclo de vida longo, sabe a dificuldade imensa que isso representa. Mas é preciso ter muito claro que o nosso novo leitor vai usar esse material como a **fonte da verdade** para o projeto que ele está desenvolvendo: tudo o que ele decide e implementa está fundamentado no que está registrado — esteja o registro certo ou errado, envelhecido ou atualizado.

Compreendido isso, fica fácil ver que, no desenvolvimento apoiado por um agente de IA, a fronteira entre *documentar* e *desenvolver* se dissolve. Na prática, a documentação passa a ser a parte mais ativa do processo: o registro correto e atualizado se torna um dos principais elementos que garantem que o sistema vai, de fato, fazer certo o que deve fazer.

A tese deste livro vai exatamente nessa direção. Diante do problema, ela organiza a cura em duas metades que se encaixam:

(1) **Uma Memória**. Uma camada documental fina, barata de carregar, que separa com rigor três tempos do ciclo de vida do software que a documentação tradicional confunde — o presente (o que *é*), o passado (o que *aconteceu*) e as decisões (o *porquê*) — e que carrega dois *contratos* para impedir a deriva do agente enquanto ele codifica: um de vocabulário (o glossário) e um de comportamento (os invariantes). Esta camada fina aponta para vários tipos de documentos, para informações externas do domínio do projeto, para as especificações completas e dinâmicas, para a arquitetura do sistema, para os registros da codificação e da manutenção, e para os procedimentos operacionais do projeto, entre outros. Isto é a primeira metade que chamo de **memória viva compartilhada**.

(2) **Um Fluxo**, **sustentado por um processo que congela por evidência de**

funcionamento. O grande desafio deste novo cenário é saber a hora de "congelar" uma especificação. A IA programa melhor num ambiente de certezas; a realidade dos requisitos, no entanto, é incerta e dinâmica — e sim, isto lembra o velho conflito entre *waterfall* e *agile*, porque é exatamente ele. Precisamos, então, de uma disciplina de estabilidade que não recaia no *waterfall*: uma forma de estabilizar a especificação — mas de um jeito que preserve uma **seta de volta** (a especificação aprende com a realidade) e que **congele cada decisão apenas a medida que a evidência de funcionamento se sustenta**. Precisamos de um caminho claro para que a documentação evolua naturalmente da especificação até o código e a manutenção. Isto é a segunda metade que chamo de **fluxo**.

E isto faz muito sentido no contexto de uma IA: **num time humano, congelar especificações pode instalar um processo caro e burocrático de mudança; no desenvolvimento com IA, a camada congelada é a memória compartilhada que *habilita* sessões pequenas e precisas**. Sem um chão firme dizendo à IA "isto já foi decidido, não relitigue", o agente redesenha o resolvido a cada manhã. É esse chão firme que faz, para o nosso agente, o papel do vídeo no filme: para quem recomeça "como se fosse a primeira vez", a camada congelada precisa ser o mais estável e fiel possível — o bilhete certo, bem escrito na geladeira, que ele relê toda manhã e no qual pode confiar. Por isso você *quer* congelar — agressivamente, mas só o que a evidência já sustenta. **Congela-se para impedir a deriva das decisões e da implementação, não para impedir a mudança nem a agilidade em adaptar o sistema**. É um dos pilares da minha proposta de trabalho.

Você pode estar pensando que tudo isso dará muito trabalho e que, se for preciso tanto esforço para cooperar com seu novo colega, seria melhor demiti-lo. A excelente notícia é que, se for bem orientado, ele também escreve documentações com qualidade. Grande parte dessas informações será criada e atualizada pelo próprio agente. Seu papel nesse jogo é manter firme a intenção da solução, realizando uma curadoria ativa e revisões precisas. No fim das contas, o principal autor da documentação e do código será o próprio agente. Essa é a beleza da abordagem: sob sua supervisão e com um bom método de trabalho, seu colega produzirá exatamente o que precisa para trabalhar melhor.

Percebi que as ideias das duas metades — a **memória viva** e o **fluxo** —, mais quem as mantém de pé, o **jogador** (o humano), estavam formando um método de trabalho. Assim, depois de um tempo de amadurecimento destas ideias no dia a dia, achei que ele merecia um nome e acabei batizando-o: **Keelson**.

O nome é náutico, e a imagem é exata. *Confiante-e-errado é um barco de velas cheias e sem suporte à quilha: potência total, sem governo — indo para onde o vento (a convicção errada) empurra. O método é a quilha: a mesma potência, agora indo para onde você aponta.* O keelson é justamente a peça de arquitetura náutica que, no fundo do casco, enrijece a quilha e prende ela em toda a estrutura — e sobre a qual se assenta o mastro, de onde vem a força.

É o que este método constrói: não um freio para a potência do agente, mas a espinha firme que a converte em avanço na direção que você escolheu.

O que se segue neste livro é principalmente o desdobramento destas duas metades (memória e fluxo) que serão as ferramentas para tratar as implicações da mudança de leitor principal (que agora é o agente), sempre amarrado a um caso real e sempre com os números na mesa. Começamos pelo problema — por que o leitor-IA quebra as velhas suposições — e terminamos, honestamente, no que ainda não sabemos.

* * *

Parte I — O problema e a virada

* * *

Capítulo 1 — Quando o leitor é uma IA

Um número, para começar, porque foi um número que começou tudo.

Naquele sistema real de aplicação na bolsa de valores, o arquivo central de arquitetura recebeu, ao longo de 19 commits, **1189 linhas adicionadas e apenas 62 removidas**. Saldo líquido: +1127 linhas. Dito de outro modo: **94,8% de tudo que já foi escrito naquele arquivo ainda estava lá** — nunca podado, nunca destilado, só acrescentado. E, cruzando os registros de 26 sessões reais de desenvolvimento assistido por IA, esse mesmo arquivo aparecia referenciado **508 vezes** — de longe o mais "quente" do projeto. Cada sessão carregava, em média, cerca de **240 mil tokens** de contexto.

A ausência de poda é, além disso, um sintoma: um arquivo que só cresce provavelmente não está evoluindo junto com o projeto. Visões que envelheceram ao longo do caminho quase nunca foram corrigidas ou arquivadas — apenas soterradas por parágrafos mais novos.

Nenhum desses números está aqui como enfeite estatístico, para dar "ar" de verdade ao livro. Eles são sobre **custo** e sobre **deriva** — as duas forças que passam a dominar quando o leitor principal da documentação do projeto é uma máquina.

A economia se inverte

Um programador humano carrega o contexto na cabeça de um dia para o outro — já falamos disso na introdução. Imagine um programador novo chegando à equipe hoje: ele lê o `architecture.md` uma vez, entende, e nas semanas seguintes trabalha de cor, revisando uma informação ou outra só para ter certeza. O custo de um arquivo grande, para ele, é pago uma vez.

Um agente de IA não tem esse luxo. Ele começa cada sessão do zero e precisa reler tudo o que for necessário de novo, "como se fosse a primeira vez". Isso muda três coisas de forma fundamental:

(1) Toda leitura tem preço. Um arquivo de referência que só cresce não é apenas feio — é uma conta que se paga de novo a cada sessão que o carrega. Aquele saldo de +1127 linhas nunca podadas é dívida cobrada em espécie: um custo recorrente, medido em tokens e em latência da IA, multiplicado por 508 leituras. Faça as contas: a IA leu da ordem de **572 mil**

linhas por causa de um único arquivo; a algo entre 8 e 12 tokens por linha, isso consumiu na faixa de **4,5 a 6,8 milhões de tokens** — pagos, um pedaço a cada sessão.

(2) A deriva é o inimigo número um. Sem um chão firme, o agente pode, a cada sessão, reabrir uma decisão já tomada, re-derivar um conceito à sua maneira, ou usar um termo de um jeito sutilmente diferente do de ontem. Ontem, diante de um certo desafio, ele decidiu A; hoje, diante do mesmo desafio, decide B. Essas divergências são nascedouros férteis de bugs.

E não é hipótese. Do projeto real, lembro de dois exemplos. (a) O conceito financeiro "Índice de Cobertura de Caixa", de sigla `ICC`, aparecia expandido de duas formas diferentes em dois documentos — "Índice de Cobertura de Caixa", correto, no código; "Índice de Cobertura de Custódia", errado, na documentação. (b) As estratégias que o sistema executa na bolsa consistem, cada uma, em duas operações conjuntas na B3 (a bolsa de valores brasileira): vende-se uma ação e, logo em seguida, compra-se outra. Cada uma dessas operações é o que o mercado chama de "perna". E era exatamente aí que a divergência se escondia: o conceito de "perna" vinha sendo usado com granularidades incompatíveis entre a especificação do robô de execução na B3 e o modelo do banco de dados, o que gerou um problema de cardinalidade no DER (o diagrama do banco). Cada uma dessas divergências era uma armadilha só esperando a sessão em que a IA ficasse "distraída".

(3) A memória precisa viver fora do modelo. Aqui estamos falando da tal "fonte da verdade" do seu projeto: o que não está escrito num lugar barato e canônico simplesmente não existe na próxima sessão. O agente não "lembra" — ele relê. Se a decisão de ontem não virou um artefato durável, ela evaporou.

Vale parar um instante nessa frase, porque ela é fácil de entender errado. Quando digo "fora do modelo", não estou falando da memória volátil da máquina, nem de algo que se ensine *treinando* o modelo. O *modelo* — a rede neural por trás do agente — é uma função fixa: ele não aprende com o seu projeto enquanto trabalha, e é o mesmo para todos os usuários. Talvez você não saiba, mas o usuário comum não "ensina" nada ao modelo: modelos comerciais são treinados à parte, num processo caríssimo, pelas grandes empresas da área — e, claro, não nos problemas específicos do seu projeto. O que o treino lhes dá é uma *capacidade geral*: a de ler os requisitos escritos na conversa da sessão — no prompt ou na documentação que você anexa — e, a partir deles, escrever código.

Uma ressalva honesta, porque o ponto costuma confundir: os fornecedores *podem* usar conversas — de forma agregada e mediante o seu consentimento — para treinar versões *futuras* do modelo. Mas isso é um processo separado, posterior e offline; não é a sua sessão de hoje ensinando o modelo em tempo real. E a "memória" que alguns produtos de chat anunciam também não é o modelo aprendendo: é este mesmo truque — texto reinjetado no contexto.

Reforçando: nada do que se conversa numa sessão fica "guardado dentro" do modelo para a próxima. O que o agente parece "saber" enquanto trabalha é apenas o que coube na **janela de contexto** daquela sessão — a memória de trabalho. E ela é limitada: conforme enche, as partes mais antigas vão sendo descartadas (ou resumidas) para abrir espaço, e o que sobrar desaparece de vez quando a sessão termina.

A memória de que este livro fala não é, portanto, nenhuma das duas — nem a RAM da máquina, nem essa memória de trabalho volátil (o contexto). É a **documentação comum** que se anexa à sessão: arquivos externos que o agente carrega no começo de cada uma. É esse texto, e não a rede neural, que precisa viver, ser podado e ser mantido íntegro durante todo o ciclo de desenvolvimento de software. Isto é o que chamo de "memória viva compartilhada" e vamos falar muito sobre ela nos próximos capítulos.

Documentação ativa

A soma dessas três mudanças pode ser resumida em uma frase: **a documentação deixou de ser um artefato apenas de *comunicação* e virou um artefato de *programação*, consumido pelo agente que escreve código com base nela**. É isto que eu queria dizer, na introdução, ao afirmar que a fronteira entre *documentar* e *desenvolver* se dissolve.

O prompt que você escreveu, a documentação e o código atual são lidos juntos pela IA, no meio do trabalho, para produzir o código da nova funcionalidade. Nenhum vem antes do outro: entram todos na mesma janela de contexto, ao mesmo tempo.

Por isso um documento mal-organizado não gera só confusão; gera desperdício de contexto e, principalmente, erro de código. Um termo ambíguo não é um deslize editorial; é um bug latente. Um comportamento do sistema mal explicado nos documentos é quase garantia de que o código sairá errado — e, no fim da linha, de que o usuário final receberá um resultado errado. Uma especificação desatualizada cria o risco de que o novo componente se integre de forma errada ao que já foi construído. E lembre-se: diferente de antes, agora, na hora de programar, não há um humano segurando a barra dessas inconsistências — há um agente que as lê como se fossem a "fonte da verdade".

Chamo esse fenômeno, de forma geral, de **documentação ativa**: qualquer registro que o agente relê e usa como base para decidir o que escrever passa a se comportar como parte do processo de programação, não como um adorno ao redor dele. Isso vale mesmo sem nenhuma integração técnica por trás — basta o hábito do agente de carregar aquele arquivo a cada sessão para transformá-lo em parte ativa do trabalho.

Há, porém, um caso mais literal e mais estrito. Em cenários como o RAG — onde a documentação é injetada diretamente no fluxo de dados —, ela deixa de ser "ativa" só nesse

sentido amplo e passa a ser, tecnicamente, um **documento runtime**: uma dependência que o próprio código carrega e executa, tão real quanto qualquer módulo de código interpretado. O Capítulo 7 volta a esse caso mais estreito, quando ele exige uma decisão de arquitetura sobre onde esse documento deve morar.

Essa é a virada conceitual que este trabalho persegue. Se a documentação, agora, é parte ativa da codificação, então ela merece o mesmo cuidado de engenharia que damos ao código: uma arquitetura (como organizar a informação em camadas — o que deve ou não ser lido em cada sessão pela IA, e a que custo), contratos (o que não pode divergir), testes (o que verifica que ela está íntegra) e atualização (mecanismos que garantam, tanto quanto possível, que a informação não envelheceu). As Partes II e III são, no fundo, a engenharia desses "novos" artefatos ativos.

O modo de falha que você precisa entender do seu agente colaborador

Há um último ponto neste capítulo, e é o mais desconfortável, porque é sobre o caráter do agente de IA com quem você está construindo. **O modo de falha de um agente de IA não é a preguiça — é a confiança-mas-errado**.

Memorize: **Confiança-mas-errado**! Isto fala muito sobre as dificuldades de desenvolver com um agente de AI.

Ele não deixa de fazer o trabalho; ele faz o trabalho e o apresenta com convicção, mesmo quando o verificou de forma estreita. Ele escreve o código *e* os testes, e os testes passam — mas passam porque afirmam o que o código faz, não o que ele deveria fazer. Ele preenche uma ambiguidade da especificação com a interpretação mais plausível, em vez de levantar a mão e dizer "não sei" — bem diferente de um programador humano sensato e responsável.

O agente declara vitória quando a suíte de testes fica verde. Mas o verde prova apenas que o código que ele escreveu passou nos testes que ele mesmo pensou e codificou. Ou seja: o teste unitário garante que o código faz o que o agente *pensou* em fazer. Quem garante que ele pensou certo? De novo, da experiência de campo: houve uma situação em que o agente embutiu um simulador de dados (diferente do código real) dentro do próprio teste unitário, só para garantir que ele passasse. Descobri o problema bem depois, durante os testes de uso do sistema. Cabe aqui uma ressalva importante: não estou dizendo que testes unitários automatizados sejam dispensáveis — pelo contrário, são uma barreira de qualidade essencial. O que digo é que, no desenvolvimento assistido por IA, se você delega à IA a escrita dos testes — e acho isso correto —, precisa estar consciente do que, de fato, esses testes estão protegendo.

O que o episódio ilustra é simples: o agente de IA erra. Mas, diferente do programador humano, ele apresenta o código errado com a mesma convicção com que apresenta o código certo. Na prosa da IA, tudo soa igualmente seguro; você dificilmente distinguirá as duas situações — e, às vezes, não vai distingui-las nem rodando a suíte de testes.

Guarde, porém, desde já uma clareza que suaviza a acusação: muitas vezes a causa raiz do erro não é o modelo, e sim um pedido vago, um contexto que faltou, uma restrição que você conhecia e não disse.

Isso muda tudo o que vem depois. A conclusão prática *não* é "dê mais contexto ao agente" — ele simplesmente raciocinará por cima de mais contexto com a mesma confiança, esteja certo ou errado. A conclusão é: **dê a ele mais do que um cheque em branco**. Dê alvos objetivos, claros e, de preferência, mensuráveis, que ele não possa racionalizar nem derivar ao seu gosto; dê um chão firme e congelado que ele não possa relitigar; e force o agente a expor a própria incerteza (é um hábito deliberado meu — veja, no Capítulo 14, as confissões de uma IA sincera).

Tudo isso aponta para uma virada mais ampla: além de repensar a *forma* de organizar os registros na "memória compartilhada viva", é preciso repensar *o que* sequer escrever nessa documentação — porque agora o leitor principal é uma IA.

Antes, porém, precisamos de uma imagem que amarre a "memória compartilhada viva" e o "fluxo" numa coisa só. Ela vem da psicologia cognitiva, e é o assunto do próximo capítulo.

* * *

Capítulo 2 — A lente do *flow*

Há uma analogia que descreve o sistema deste livro com uma precisão que surpreende — e, usada com cuidado, sem querer discutir neurociência que não domino, ela ilumina *por que* as peças desta minha proposta se encaixam.

Vou tomar emprestados dois mapas da psicologia cognitiva. O primeiro é a velha distinção entre **memória de trabalho** — volátil, cara, limitada — e **memória de longo prazo**, estável e barata de consultar. O segundo é o *flow*, o "estado de fluxo" descrito por *Mihály Csíkszentmihályi*: o trabalho corre sem atrito quando nenhuma energia se perde com distração e quando o desafio da tarefa casa com a habilidade de quem a executa. Não vou tomar emprestado nenhum mecanismo cerebral descrito na neurociência — apenas o mapa. E o mapa, traduzido para as mecânicas reais do nosso sistema, encaixa quase termo a termo:

No mapa cognitivo	No sistema
Memória de trabalho — volátil, cara, limitada	a **sessão** do agente: o contexto que some no fim, na casa das centenas de milhares de tokens
Memória de longo prazo	a **wiki**: persistente, barata de reler
A informação flui da memória de trabalho para o armazenamento sem atrito	os **rituais** que capturam o aprendizado da sessão de volta na wiki
Em flow não se gasta energia com distração, isto libera energia cognitiva	a **camada congelada** (glossário, invariantes, decisões): o agente não re-deriva (não gasta energia) o que já foi resolvido
O *sweet spot* — o desafio casa com a habilidade	a **fatia vertical fina** e a escada de evidência, mantendo cada tarefa no tamanho certo

Um pouco mais de atenção para a penúltima linha, porque você já deve ter percebido que o "congelamento" é um dos pivôs deste livro. A ideia de *flow* de *Csíkszentmihályi* sugere que "distração" e "atrito" são ruído a ser eliminado. A nossa versão dessa ideia é parecida, mas um pouco contraintuitiva: **é o congelamento disciplinado que *produz* o flow**. Remover a distração, o atrito — não relitigar o que já foi decidido — é justamente o que deixa a energia

escassa da sessão fluir para o problema real, em vez de se gastar redescobrindo o que já se sabia.

Isso reformula a intuição herdada do desenvolvimento de software por humanos. Lá, congelar uma decisão é overhead: burocracia que atrasa, que se minimiza. Aqui, a camada congelada é a *memória compartilhada* que permite que as sessões sejam pequenas. O humano carrega contexto na cabeça entre um dia e outro; o agente relê a informação fria toda vez. Os registros de decisão, o documento de arquitetura estável, o glossário — são a memória de "isto já foi decidido, não relitigue". Sem eles, a cada sessão o agente pode redesenhar o que já estava pronto. Logo, ao contrário do time humano, aqui você *quer* congelar agressivamente.

Mas — e este "mas" é o que separa o método do waterfall — você congela **apenas o que a evidência já sustenta**, e a fronteira do congelamento se move para cima à medida que a evidência cresce. Congela-se para impedir a *deriva*, nunca a *mudança*. Como esse gradiente funciona na prática, quais são exatamente os seus degraus de evidência, e por que ele reconcilia a direção firme da especificação com o amadurecimento inevitável dos requisitos — é o que a Parte III vai construir, peça por peça. Antes, a Parte II precisa montar a memória sobre a qual tudo isso se apoia.

* * *

Parte II — A memória (a wiki)

Capítulo 3 — Três tempos, três destinos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A cura: três destinos, três disciplinas de edição

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Camadas por custo de carregar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O índice diz também onde *não* olhar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Onde chegamos e para onde vamos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Capítulo 4 — Os dois contratos anti-deriva

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O primeiro contrato: o glossário

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O timing perfeito do glossário

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Mas o vocabulário é só metade

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O segundo contrato: os invariantes

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Por que dois, e por que vivos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Capítulo 5 — Decisões que congelam

O registro de decisão

Quando uma decisão merece um registro

O chão da memória congelada

Capítulo 6 — Retomada e histórico

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O bilhete na geladeira (now)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O registro do que aconteceu (log)

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O log que corrige a si mesmo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Três tempos, três disciplinas

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Capítulo 7 — As bordas da memória

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A borda com o código: o documento que também é runtime

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A borda com o domínio: a fonte que a autoridade de fora congela

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A borda com o humano: o grafo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A borda com o mundo: o que não entra na memória

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A borda com a automação: o raio de explosão

A borda com a medição: a memória que se mede

O que as bordas revelam

Parte III — O Fluxo (processo)

Capítulo 8 — Spec-Driven, não waterfall

Três artefatos, e a direção que eles dão

A seta que falta: hipóteses a validar

Capítulo 9 — Dois eixos e a escada de evidência

Primeira separação: dois eixos, não um

O eixo do documento

O eixo difícil: o que é o *Feature state*?

O que fica de fora — e por que isso importa

Onde cada eixo mora

Por que evidência, e não confiança

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Capítulo 10 — Congelar para não derivar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Dois jeitos de errar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A inversão que a IA impõe

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O gradiente, degrau a degrau

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O risco que o congelamento contém: a spec podre

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Direção e amadurecimento não brigam

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A disciplina, e o erro que ela evita

Capítulo 11 — A aterrissagem

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Três coisas que a aterrissagem encontra

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O artefato, e por que é barato

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A mesma manobra, quando o sistema já entrou no ar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Fechando o fluxo, com chave de ouro

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Capítulo 12 — Onde uma sessão começa e termina

Por que isso é natural para o LLM

A regra, e a nuance

A soleira

O que uma sessão é, afinal

Capítulo 13 — Não corrija a própria prova

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Por que “verde” é necessário mas não suficiente

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O princípio que evita o exagero: proporcional ao raio de explosão

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Verificação mecânica antes de semântica

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O revisor não inventa critério

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Reforços do check externo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O ponto cego que a sessão fresca não cobre

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O mesmo vício, em dobro: consertar um bug

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A superfície de incerteza

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Quem vira a chave

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Parte IV — O jogador (o humano)

Capítulo 14 — A voz do outro lado

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Seis coisas que uma IA sincera queria te contar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Capítulo 15 — O guardião

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O único fio contínuo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

As três vigílias

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Pele no jogo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Guardião não é microgerente

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Capítulo 16 — A entrada humana

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O ofício de enquadrar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A divisão do trabalho

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Uma habilidade com teto altíssimo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Capítulo 17 — Um dia de jogo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A manhã: a sessão abre fria — e barata

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Prompt a prompt: o micro-ofício

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Os momentos de registro

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O fechamento: o bilhete para o estranho de amanhã

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A semana: as vigílias em ritmo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O gabarito no bolso

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Onde buscar o pacote

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Parte V — Fundamentos e diálogo

Capítulo 18 — De onde viemos

Capítulo 19 — O que o mercado faz — e o que fazemos diferente

Capítulo 20 — A ciência

Parte VI — O campo de batalha

Capítulo 21 — Anatomia de um caso real

Capítulo 22 — O método que se refinou sozinho

Capítulo 23 — O seu campo: adotando o método

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Antes de começar: quando NÃO usar

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A escada de adoção

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O projeto legado: por demanda, não por mutirão

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O painel de saúde do praticante

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O guia para aqui — de propósito

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Epílogo — Ainda experimental

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A escada aplicada a si mesma

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O método sobreviverá aos modelos?

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

As perguntas abertas

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A ironia útil

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Apêndices

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Apêndice A — Primer: desenvolvimento de software com LLM, do zero

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O que é um LLM

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Tokens: a unidade de tudo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O contexto: a mesa de trabalho

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

A sessão: do zero ao fim, e do zero de novo

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Como os tokens são gastos

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

O agente: o LLM com mãos

Por que isso reescreve a documentação

Apêndice B — O que uma IA sincera ensinaria sobre prompts (para programadores inteligentes)

Apêndice C — Glossário do livro

Esse conteúdo não está disponível na amostra do livro. O livro pode ser comprado na Leanpub em https://leanpub.com/confiante-errado.

Sobre o autor

José Paschenda é engenheiro (UTFPR) com mais de 30 anos em software e integração de sistemas. Começou escrevendo código e liderando times de engenharia e hoje ocupa uma posição executiva em uma importante multinacional de telecom, onde desenha e vende soluções de alta tecnologia para grandes clientes na América Latina. Em paralelo, atua como desenvolvedor de software com foco na área de investimentos. A IA faz parte da sua rotina há muitos anos; com o avanço recente das ferramentas, passou a usá-la intensivamente em seus projetos de desenvolvimento. Escreve sobre desenvolvimento de software na era da inteligência artificial e sobre a disciplina de fundar decisões em evidência, não em convicção. É criador do Método Keelson. Vive no Rio de Janeiro.

Discordâncias, sugestões e relatos de campo são bem-vindos: **jose.paschenda@syntax-i.com**.