

SPEC-DRIVEN DEVELOPMENT

REVERSA

*CONSTRUA OU RECONSTRUA SEU SOFTWARE
USANDO AGENTES DE IA*



Sandeco Macedo

Meus Livros sobre Inteligência Artificial



Resultado
das minhas
pesquisas em IA



Sumário

1	Todo software nasce, cresce, envelhece... e renasce	5
1.1	A casa com um cômodo trancado	6
1.2	Ninguém planejou chegar aqui	8
1.3	O agente lê seu código, mas não lê sua mente	9
1.4	Especificação: o contrato que faltava	11
1.5	Renasce: quando a spec vem depois do código	12
1.6	Nascer e crescer sem repetir a história	15
1.7	Um time de especialistas dentro do seu terminal	16
1.8	O mapa desta jornada	17
2	Time Core: Analisar e Especificar o Legado	20
2.1	O orquestrador: /reversa	20
2.2	Scout: o mapeador da superfície	21
2.3	Soul Extractor: a alma do projeto	23
2.4	Archaeologist: a escavação módulo a módulo	24
2.5	Detective: o porquê por trás do código	25
2.6	Architect: os mapas formais	25
2.7	Data Master: a fundação invisível	26
2.8	Design System: o estilo do sistema	27
2.9	Visor: a interface pelas telas	28
2.10	Writer: os contratos executáveis	29
2.11	Reviewer: furar as especificações	30
3	Time de Migração: Do Legado ao Sistema Novo	32
3.1	O orquestrador: /reversa-migrate	32
3.2	Paradigm Advisor: detectar o paradigma	34
3.3	Curator: o que migra e o que morre	35
3.4	Strategist: estratégias com trade-offs	36
3.5	Designer: a topologia moderna	37
3.6	Screen Translator: telas viram especificações	38
3.7	Inspector: provar a equivalência	39
3.8	Bônus: Translators (n8n e além)	40

4	Times de Criação e Evolução	42
	Parte I: Time de Criação (/reversa-new)	43
4.1	O orquestrador: /reversa-new	43
4.2	Ideator: brainstorm estruturado	44
4.3	Researcher: público e personas	44
4.4	Drafter: o PRD completo	46
4.5	Spec SDD: da ideia às specs	47
	Parte II: Time de Evolução (/reversa-forward)	48
4.6	O orquestrador: /reversa-forward	48
4.7	Requirements e Clarify: da ideia ao requisito	49
4.8	Plan e To-Do: roadmap e ações atômicas	50
4.9	Coding: execução em código	51
4.10	Sync e Resume: convergir e retomar	51
4.11	Reconstructor: reconstrução bottom-up	52
4.12	Guardiões da qualidade: Principles, Quality e Audit	53
5	Documentação Visual, Depuração e Precificação	55
	Parte I: Documentação Visual (/reversa-docs)	56
5.1	O orquestrador: /reversa-docs	56
5.2	Docs Mapper: a estrutura espacial	57
5.3	Docs Analyst: os dados quantitativos	58
5.4	Docs Storyteller: a narrativa	58
5.5	Docs Publisher: o editor-chefe	59
5.6	A camada visual: D3, Highcharts e 3D	60
	Parte II: Time Reversa Bugs	61
5.7	Debugger: o registrador de bugs	61
5.8	Debugger Fix: o ciclo de correção	62
5.9	Debugger Debate: decisão multiagente	63
5.10	Debugger Graph e Depth Inspection: visões e pente-fino	64
	Parte III: Time de Pricing	65
5.11	Pricing Profile: o perfil de cobrança	65
5.12	Pricing Size: o tamanho da feature	66
5.13	Pricing Estimate: os três cenários	67

CAPÍTULO 1

Todo software nasce, cresce, envelhece... e renasce

Pense no sistema mais importante da empresa onde você trabalha, ou daquele cliente que você atende. Aquele que emite as notas, controla o estoque, calcula a folha. Agora responda com sinceridade: se ele parasse de funcionar hoje, existe alguém no planeta capaz de explicar, linha por linha, tudo o que ele faz? Se a resposta for não, bem-vindo ao clube. É o maior clube do mundo do software, e ninguém entra nele por escolha.

Este livro conta a história de um quarto ato que quase ninguém conhece. Todo mundo aprendeu que software nasce, cresce e envelhece, e que no final da vida ele vira aquele sistema que ninguém quer tocar. O que a era dos agentes de inteligência artificial acabou de mudar é o final dessa história: pela primeira vez, envelhecer deixou de ser uma sentença. O sistema que hoje é um mistério pode renascer, e você vai aprender exatamente como fazer isso acontecer.



Figura 1.1: O ciclo completo: a mesma casa nasce, cresce, envelhece e, na era da IA, renasce iluminada por dentro.

A Figura 1.1 resume a promessa deste livro numa única imagem. Ao longo deste capítulo, você vai entender por que os sistemas envelhecem mal, por que a inteligência artificial sozinha não resolve esse problema, e qual é a peça que faltava para fechar o ciclo. Essa peça tem nome: **especificação**. E o framework que transforma essa ideia em prática também tem nome: **Reversa**.

1.1 A CASA COM UM CÔMODO TRANCADO

Imagine que você herdou uma casa grande, bonita, cheia de história. Você mora nela, ela funciona, a água corre nos canos e a luz acende nos interruptores. Mas no fim do corredor existe um cômodo trancado. A chave? Ficou com alguém que se mudou há dez anos. Você sabe que dali passam canos e fios que alimentam a casa inteira, porque quando algo falha lá dentro, a casa toda sente. Só que ninguém abre aquela porta. Melhor não mexer, dizem todos. Sempre foi assim.

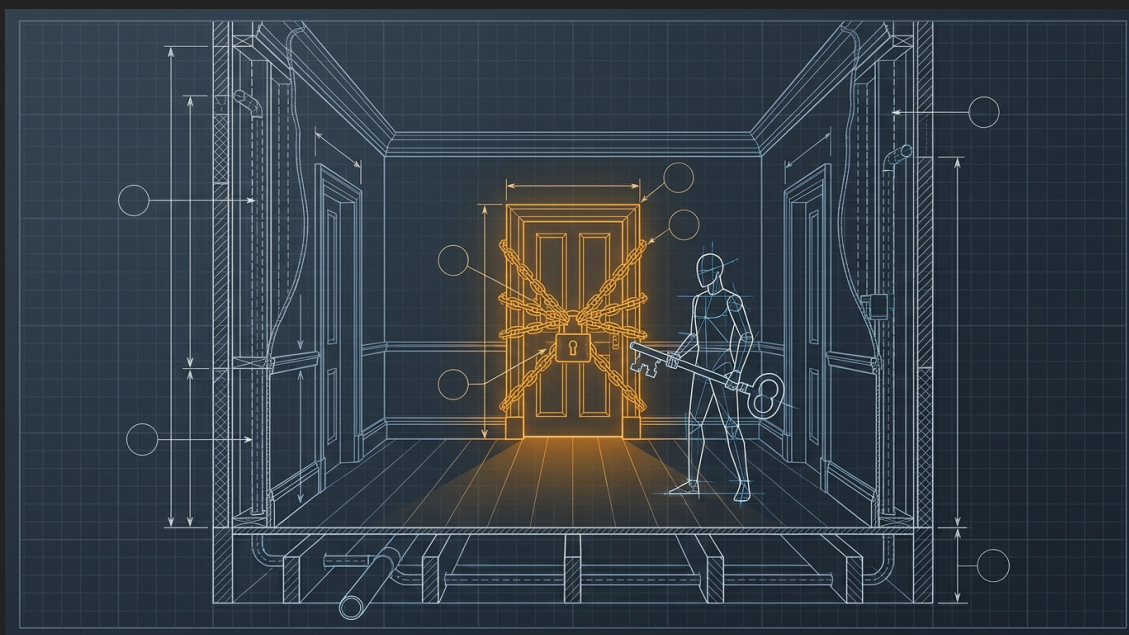


Figura 1.2: A herança que todo desenvolvedor conhece: a casa funciona, mas ninguém tem a chave do cômodo que sustenta tudo.

Todo sistema legado é essa casa, e a Figura 1.2 mostra a cena que se repete em milhares de empresas. O sistema roda, fatura, entrega. Mas dentro dele existem cômodos que ninguém visita: aquele módulo de cálculo que só o Carlos entendia, e o Carlos saiu em 2019; aquela função de oitocentas linhas que todo mundo copia e cola em volta, mas ninguém altera; aquela regra estranha que parece errada, mas que, se alguém corrigir, o maior cliente da empresa para de faturar.

Aqui está o ponto que muda tudo: o problema nunca foi o código. O código está lá, visível, aberto, disponível para qualquer um ler. O problema é o que o código não conta. Um sistema de verdade é feito de duas camadas: a que aparece e a que sustenta. A Figura 1.3 mostra essa anatomia.

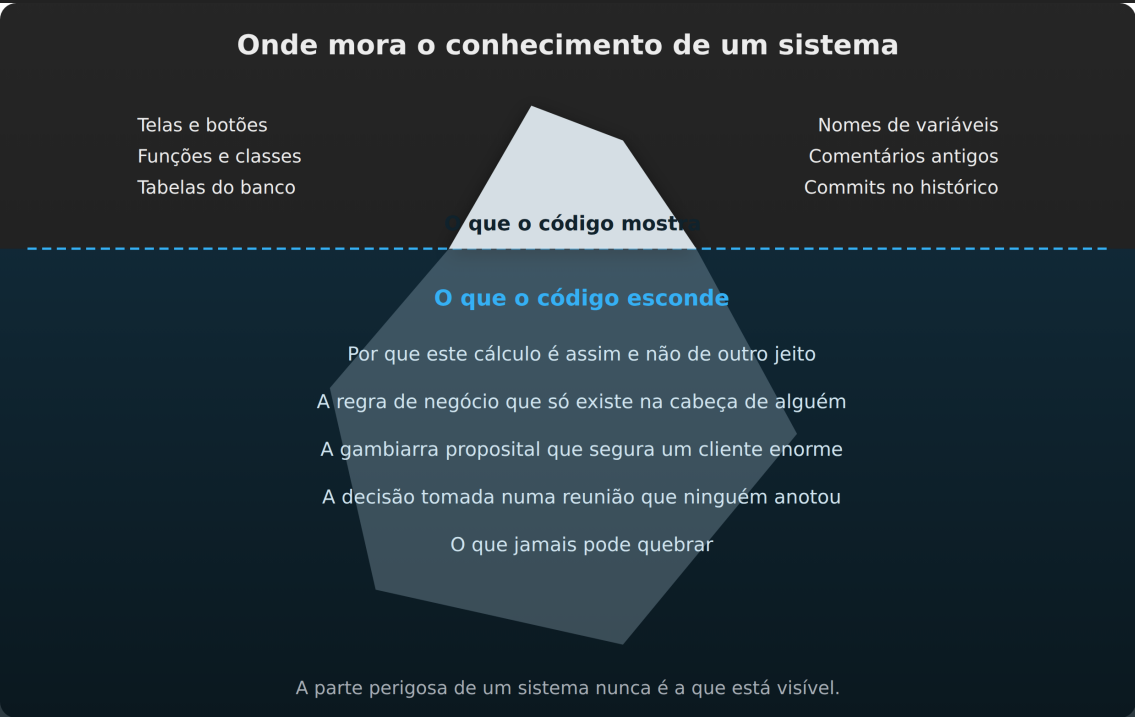


Figura 1.3: O iceberg do conhecimento: o código mostra telas, funções e tabelas, mas esconde os porquês, as regras implícitas e o que jamais pode quebrar.

Acima da linha d'água ficam as telas, as funções, as tabelas do banco. Qualquer ferramenta enxerga isso. Abaixo da linha d'água mora o que realmente importa: por que este cálculo arredonda para cima, qual promessa foi feita a qual cliente, qual gambiarra é proposital e qual é acidente. Essa camada submersa nunca foi escrita em lugar nenhum. Ela evaporou junto com as pessoas que foram embora. E é exatamente ela que decide se uma mudança no sistema é segura ou catastrófica.

A parte perigosa de um sistema nunca é a que está visível. É a que só existia na cabeça de alguém que já foi embora.

1.2 NINGUÉM PLANEJOU CHEGAR AQUI

Nenhuma casa nasce com cômodo trancado. Ela nasce com planta aprovada, paredes no lugar certo, tudo limpo. Aí a família cresce e alguém levanta um quarto nos fundos. Depois vem a garagem coberta, a edícula, o segundo andar apoiado em colunas que ninguém calculou. Cada puxadinho resolveu uma urgência real, feito às pressas, por quem estava disponível. Vinte anos depois, a casa é um labirinto que nenhuma planta descreve, porque a planta original ficou parada no tempo e as obras nunca foram desenhadas.



Figura 1.4: A casa dos puxadinhos: cada extensão resolveu uma urgência real, e nenhuma delas foi desenhada na planta.

A Figura 1.4 é o retrato afetivo, e um pouco doloroso, de quase todo software em produção. O sistema também nasceu limpo. No primeiro dia, alguém sabia exatamente o que ele fazia, porque tudo coube numa conversa, numa cabeça, num quadro branco. Só que sistema vivo é sistema que muda. Chega o pedido urgente do cliente, a lei que mudou em dezembro, a integração com o parceiro novo. Cada mudança é um puxadinho: resolve a urgência de hoje e cobra silenciosamente o juro amanhã.

O detalhe cruel é que o conhecimento não acompanha o crescimento. A Figura 1.5 mostra as duas curvas que definem a vida de qualquer sistema: o que ele realmente faz sobe sem parar, mudança após mudança; o que está escrito sobre ele fica para trás já nos primeiros meses e depois só decai. O espaço entre as duas curvas tem um nome: é o conhecimento que só existe na cabeça de alguém. Quanto maior esse espaço, mais caro fica cada dia

de manutenção, mais arriscada fica cada alteração, e mais refém a empresa fica de poucas pessoas.

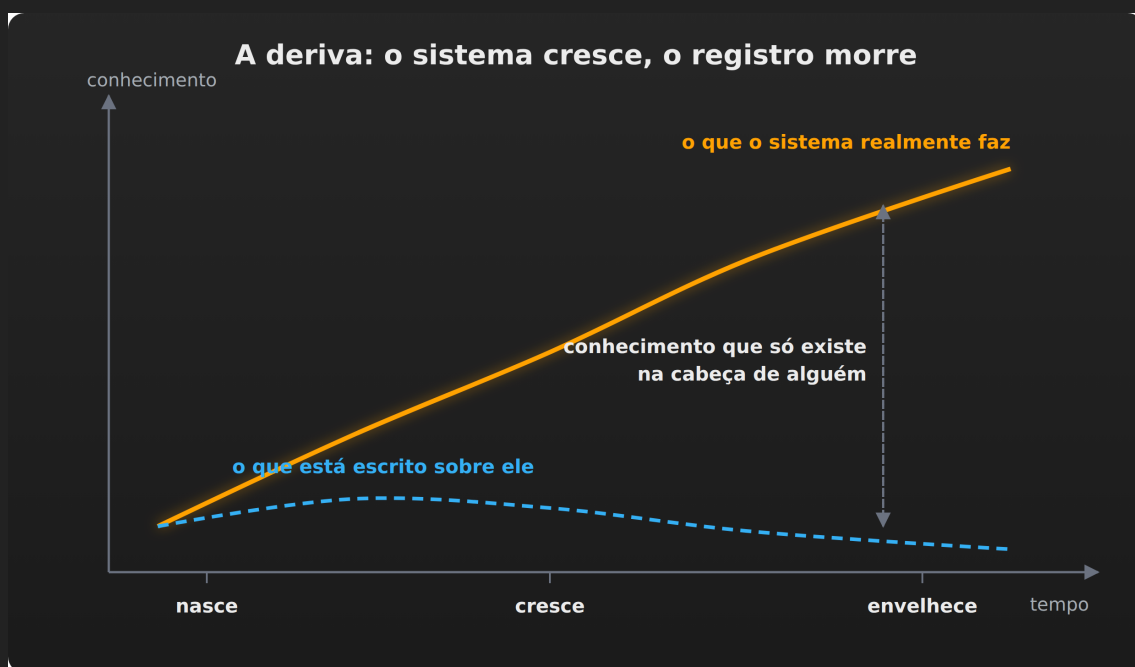


Figura 1.5: A deriva: o sistema cresce sem parar, o registro sobre ele morre cedo, e a distância entre as duas curvas vira risco puro.

Repare que ninguém nessa história é vilão. O desenvolvedor que fez o puxadinho estava salvando o trimestre. O gerente que não pediu documentação estava correndo atrás do prazo. Envelhecer assim não é falha de caráter de ninguém: é o destino natural de todo sistema útil. Útil demais para ficar parado, urgente demais para ser documentado. Guarde essa frase, porque ela explica por que o seu sistema de hoje, aquele que você está criando agora com todo o carinho, será o cômodo trancado de alguém em poucos anos. A menos que algo quebre esse ciclo.

1.3 O AGENTE LÊ SEU CÓDIGO, MAS NÃO LÊ SUA MENTE

E então chega a inteligência artificial, e com ela uma esperança compreensível: se o agente de IA lê milhões de linhas por minuto, o problema do legado acabou, certo? Contrate um robô brilhante, aponte para o sistema, peça as mudanças. É aqui que a história ganha sua reviravolta mais importante, e é a cena da Figura 1.6: o empreiteiro robô chega na casa com uma marreta na mão e a melhor das intenções. Você pede para derrubar a parede da sala. Ele derruba na hora, com precisão perfeita. O que ele não sabia, porque ninguém contou, é que aquela parede era **parede mestra**: era ela que segurava o andar de cima.

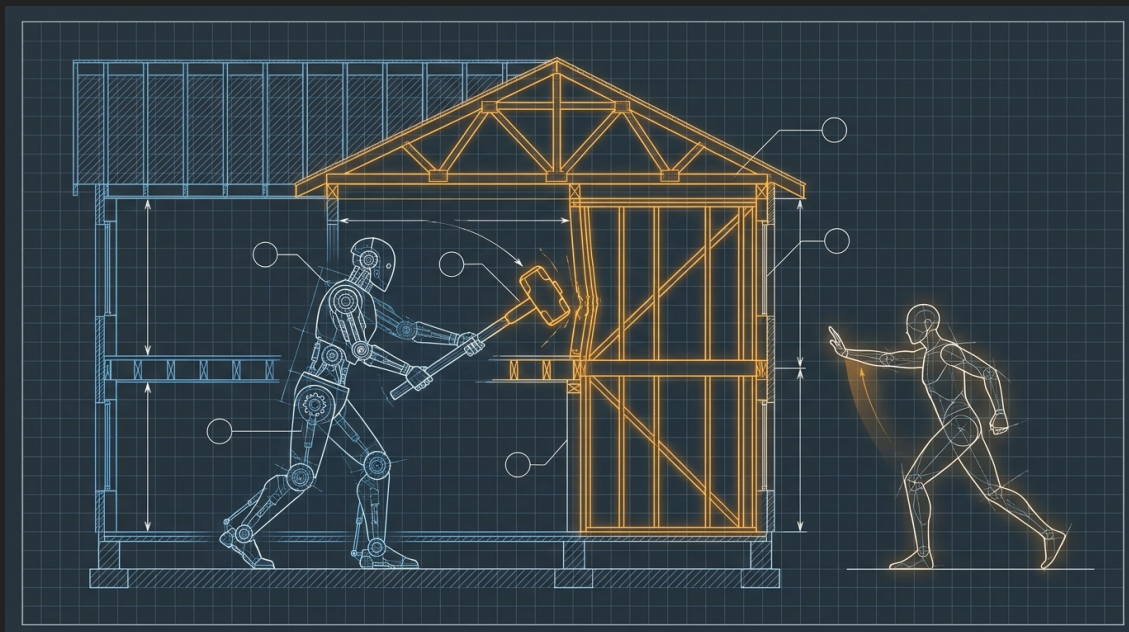


Figura 1.6: O empreiteiro robô é competente e obediente. Ele só não sabe, porque ninguém contou, que aquela parede segura a casa inteira.

Essa é a verdade incômoda sobre os agentes de IA que ninguém coloca no anúncio: o agente lê seu código, mas não lê sua mente. E a diferença entre as duas coisas é exatamente a diferença entre uma reforma e um desabamento. A Figura 1.7 coloca lado a lado o que o agente enxerga e o que ele nunca vai enxergar sozinho.

Do lado esquerdo está tudo o que o agente domina com uma competência sobre-humana: cada linha do código-fonte, a estrutura de pastas inteira, nomes, tabelas, testes. Do lado direito está o que não deixa rastro em arquivo nenhum: a intenção por trás de cada regra, o motivo de cada decisão, o que é bug e o que é proposital, a promessa feita a um cliente numa reunião de 2017. O agente executa com perfeição o que foi pedido. O desastre nunca vem do que você pediu: vem do que você não sabia que precisava proibir.

Entenda bem o tamanho da ironia. A IA não é fraca demais para o legado; ela é forte demais para trabalhar às cegas. Dar um sistema sem especificação a um agente poderoso é dar uma marreta a alguém que não sabe onde ficam as paredes mestras. A solução, portanto, não é um agente ainda mais inteligente. É contar para o agente aquilo que só você, sua equipe e a história da empresa sabem. É transformar a camada submersa do iceberg em algo que a máquina possa ler, respeitar e obedecer.

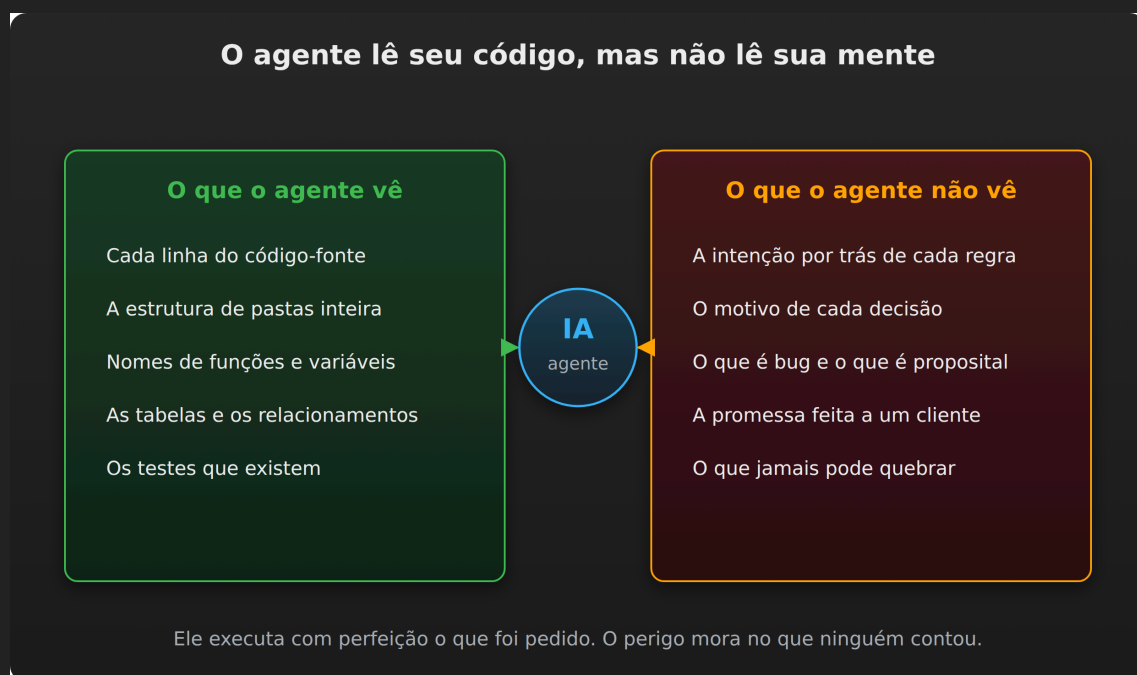


Figura 1.7: O agente enxerga cada linha, cada pasta, cada tabela. O que ele não enxerga é a intenção, e é na intenção que mora o perigo.

1.4 ESPECIFICAÇÃO: O CONTRATO QUE FALTAVA

Volte para a casa por um instante. O que separa uma reforma tranquila de uma tragédia anunciada é uma folha de papel: a **planta**. Nela está desenhado o que nenhuma parede conta sozinha: onde passa o cano, onde corre o fio, qual parede segura o peso da casa. Com a planta na mão, qualquer empreiteiro competente trabalha com segurança. Sem ela, até o melhor profissional do mundo trabalha apostando. A Figura 1.8 mostra essa ideia levada ao mundo digital: cada cômodo da planta conectado por um fio de luz ao cômodo real correspondente.

No software, essa planta se chama **especificação**, ou simplesmente **spec**. E aqui é preciso desfazer um mal-entendido que custou décadas de documentação inútil: especificação não é aquele documento de cem páginas que alguém escreve no início do projeto e ninguém abre nunca mais. Documentação tradicional é escrita para humanos lerem um dia, talvez. A spec de que este livro trata é outra coisa: é um **contrato operacional**, escrito para ser executado. Ela diz, sem ambiguidade, o que o sistema promete fazer e o que ele jamais pode deixar de fazer.

A diferença fica concreta na Figura 1.9, que mostra o triângulo que sustenta o desenvolvimento guiado por especificação: **SPEC**, **CODE** e **TEST** amarrados por rastreabilidade.

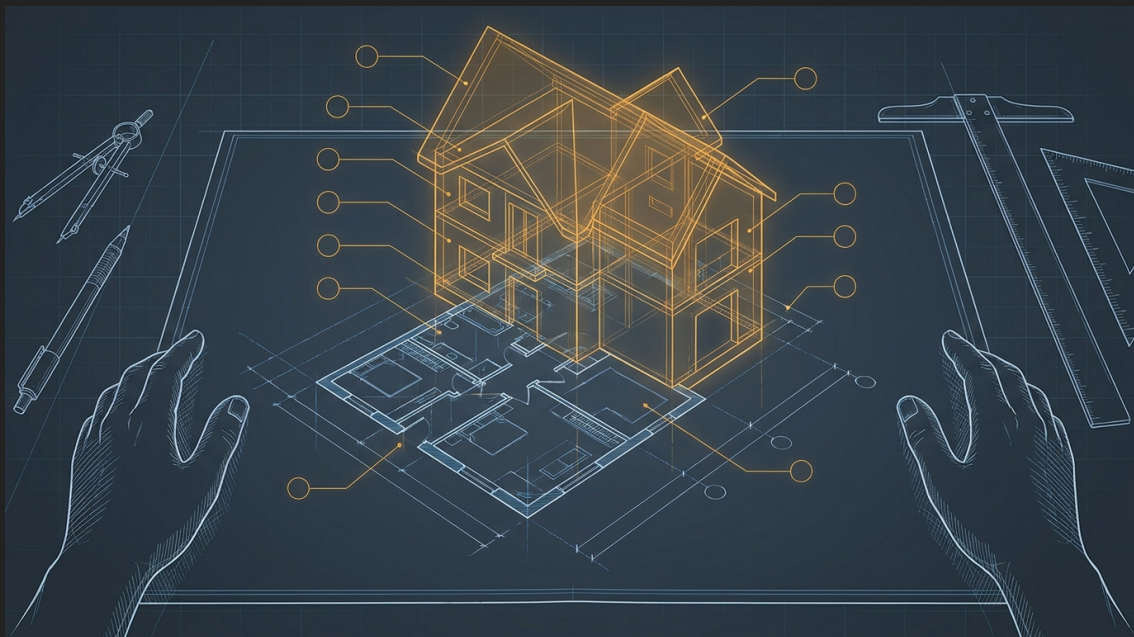


Figura 1.8: A planta viva: cada parte do desenho conectada à parte real da casa que ela descreve. É assim que uma especificação deve funcionar.

A spec declara a promessa. O código cumpre a promessa. O teste prova que a promessa foi cumprida. E os três se vigiam mutuamente: mudou a spec, o código e o teste precisam acompanhar; quebrou um teste, dá para saber qual promessa foi violada e qual trecho de código a violou.

Quando existe esse contrato, o jogo com o agente de IA vira outro completamente. O agente deixa de adivinhar e passa a obedecer a regras explícitas. Derrube a parede da sala, agora ele responde: a spec diz que essa parede sustenta o módulo de faturamento; quer mesmo continuar? A spec é o que transforma um estagiário genial e perigoso no melhor engenheiro da sua equipe.

Agente de IA sem especificação é marreta sem planta. Agente de IA com especificação é o melhor engenheiro que você já contratou.

1.5 RENASCE: QUANDO A SPEC VEM DEPOIS DO CÓDIGO

Neste ponto você já deve ter feito a conta que assombra todo mundo: tudo isso é lindo para quem está começando um sistema agora, mas e a casa que já existe? E as duzentas mil linhas rodando em produção desde 2009, sem planta, sem contrato, sem o Carlos? Escrever

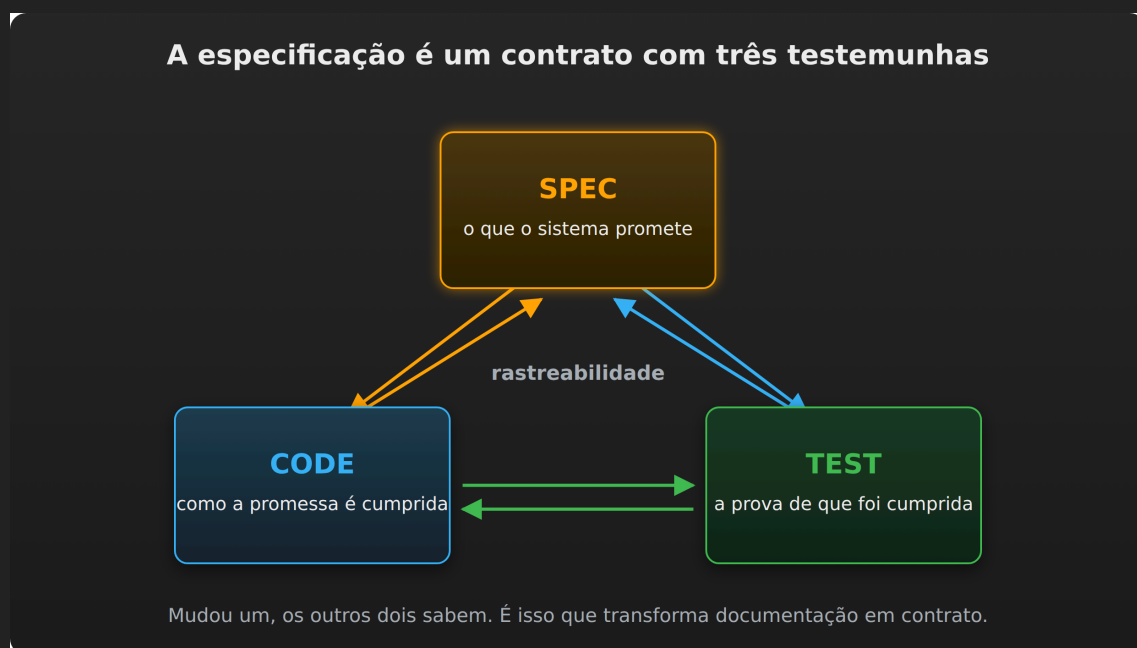


Figura 1.9: O contrato com três testemunhas: a spec promete, o código cumpre, o teste prova. A rastreabilidade mantém os três honestos.

a spec de um sistema desses na mão levaria anos. É aqui que entra a ideia que dá nome a este livro, e é a cena da Figura 1.10: em vez de escrever a planta antes de construir, extrair a planta do que já está construído. Planos de luz varrem a casa antiga e revelam a estrutura escondida: vigas, canos, fiação, fundações. A casa não é tocada; ela é lida. E do topo dela nasce, flutuando, a planta que nunca existiu.

Isso é **engenharia reversa de documentação**: percorrer o código que já existe, escavar suas camadas, interrogar suas decisões e produzir, a partir dele, as especificações que nunca foram escritas. Não um resumo bonito para pendurar na parede, mas o contrato executável da seção anterior: promessas, regras, restrições e mapas, tudo rastreável de volta ao código de onde saiu. É a diferença entre documentar para humanos, que é caro e envelhece, e gerar contratos para agentes, que é automatizável e se mantém.

O fluxo aparece na Figura 1.11, e repare no detalhe mais importante dela: o cadeado. O sistema legado entra nesse processo como **fonte de leitura**, nunca como alvo de escrita. O time de agentes lê, escava, investiga, desenha e escreve, mas escreve em outro lugar: numa pasta chamada `.reversa/`, onde nascem as especificações, os mapas e as decisões. Nenhuma linha do seu código de produção é alterada no processo. Essa garantia de imutabilidade não é um detalhe técnico; é o que permite rodar tudo isso num sistema crítico, em produção, sem pedir autorização para o medo.

É isso que a palavra renascer significa neste livro. O sistema não é reescrito, não é aposentado, não é jogado fora. Ele ganha, talvez pela primeira vez na vida, a planta de si

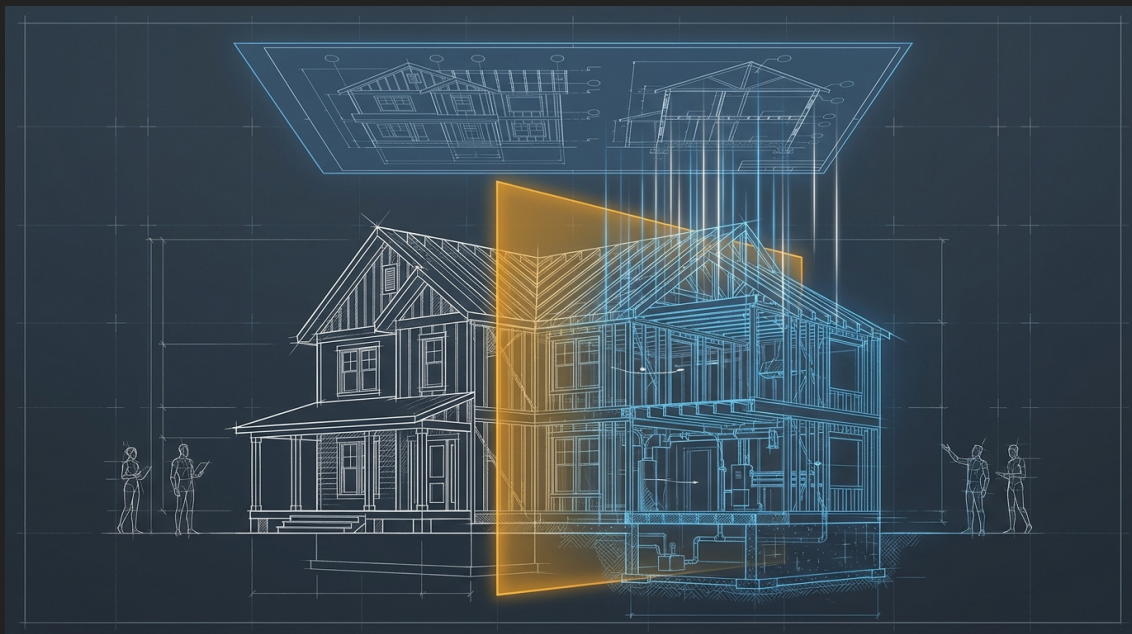


Figura 1.10: A engenharia reversa de documentação: a casa não é tocada, é lida. E do que foi lido nasce a planta que ninguém escreveu.

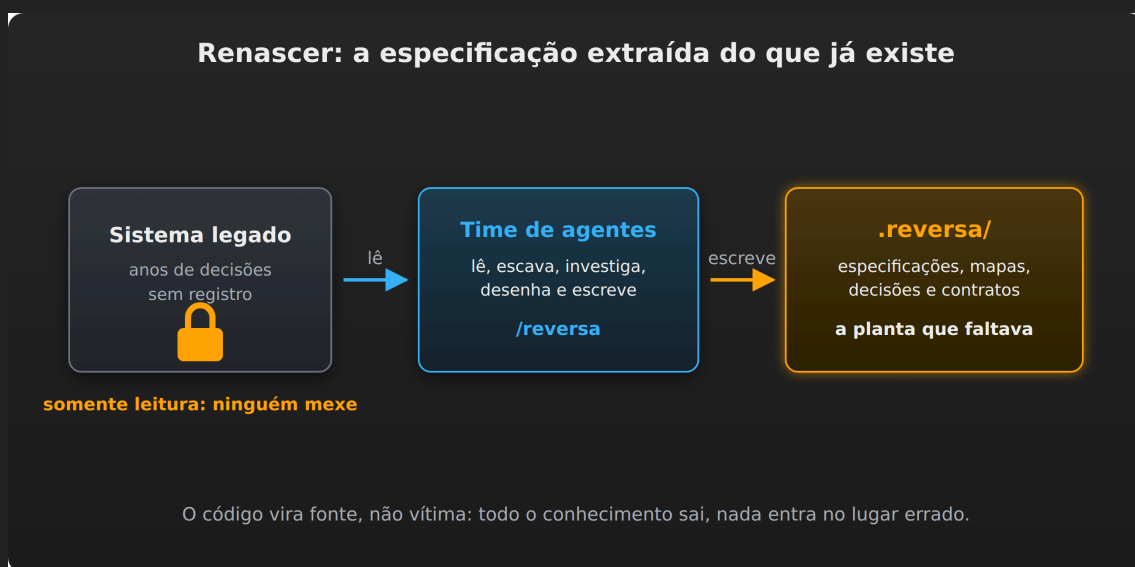


Figura 1.11: O fluxo do renascimento: o legado é somente leitura, o time de agentes extrai, e todo o conhecimento nasce em .reversa/.

mesmo. E com essa planta na mão, tudo o que parecia impossível fica na mesa: evoluir com segurança, migrar para tecnologia moderna, treinar gente nova em semanas em vez de anos, ou simplesmente dormir tranquilo sabendo que o conhecimento da empresa não mora mais só na memória de duas pessoas.

1.6 NASCER E CRESCER SEM REPETIR A HISTÓRIA

Seria um desperdício usar uma ideia dessas só para apagar incêndio do passado. A mesma especificação que salva o sistema velho é o que impede o sistema novo de envelhecer mal. Olhe a Figura 1.12: a casa nova subindo a partir da planta projetada no chão, andar por andar, na ordem certa, com a casa antiga renovada e acesa logo ao lado. É a imagem das duas pontas do ciclo se encontrando: o que existe foi lido e ganhou planta; o que vai nascer já nasce a partir dela.

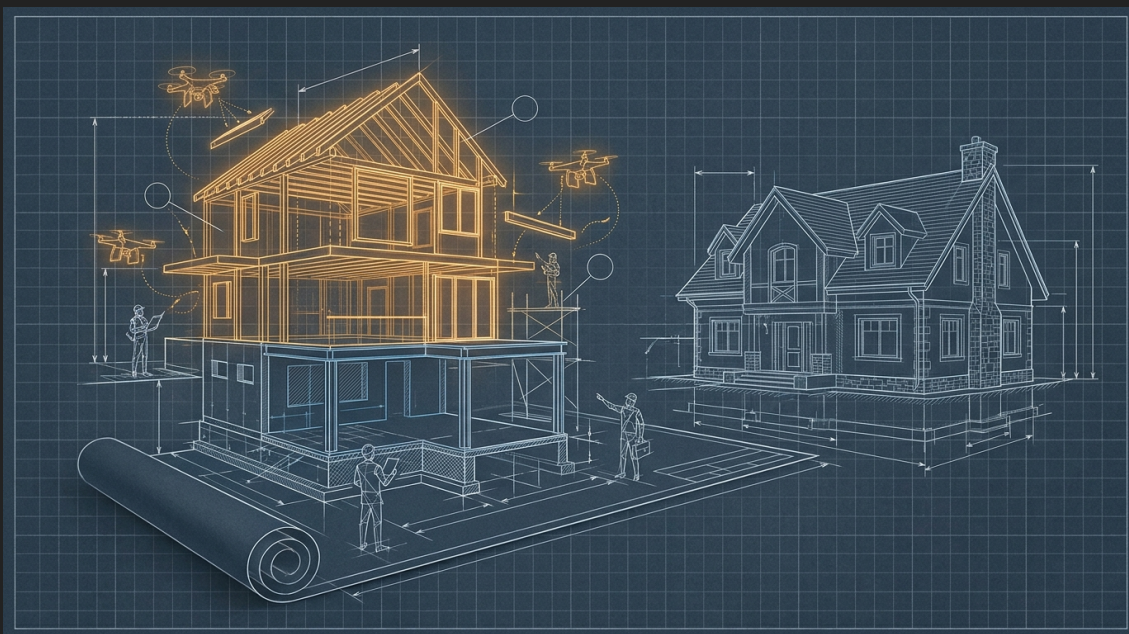


Figura 1.12: A casa nova nasce da planta, andar por andar, ao lado da casa antiga renascida. Nenhuma das duas repete a história do cômodo trancado.

A Figura 1.13 organiza essa visão, e ela é o mapa mental mais importante do livro inteiro. No centro está sempre a mesma coisa: a spec, o conhecimento fora da cabeça de alguém. O que muda é a direção do tempo. Para o código que já existe, a spec vem depois: é extraída, no movimento de engenharia reversa que você acabou de conhecer. Para o sistema que ainda não existe, a spec vem antes: a ideia vira requisito, o requisito vira contrato, e só então vira código. E para o sistema vivo, que muda toda semana, a spec evolui junto: cada nova funcionalidade nasce especificada, e a planta nunca mais fica para trás.

Lembra da deriva, as duas curvas se separando na Figura 1.5? Este é o momento em que o ciclo quebra. Com a spec no centro do desenvolvimento, a curva do registro para de morrer, porque registrar deixou de ser uma tarefa chata que alguém esquece e virou o próprio jeito de trabalhar. O nome disso é **desenvolvimento guiado por especificação**, o famoso

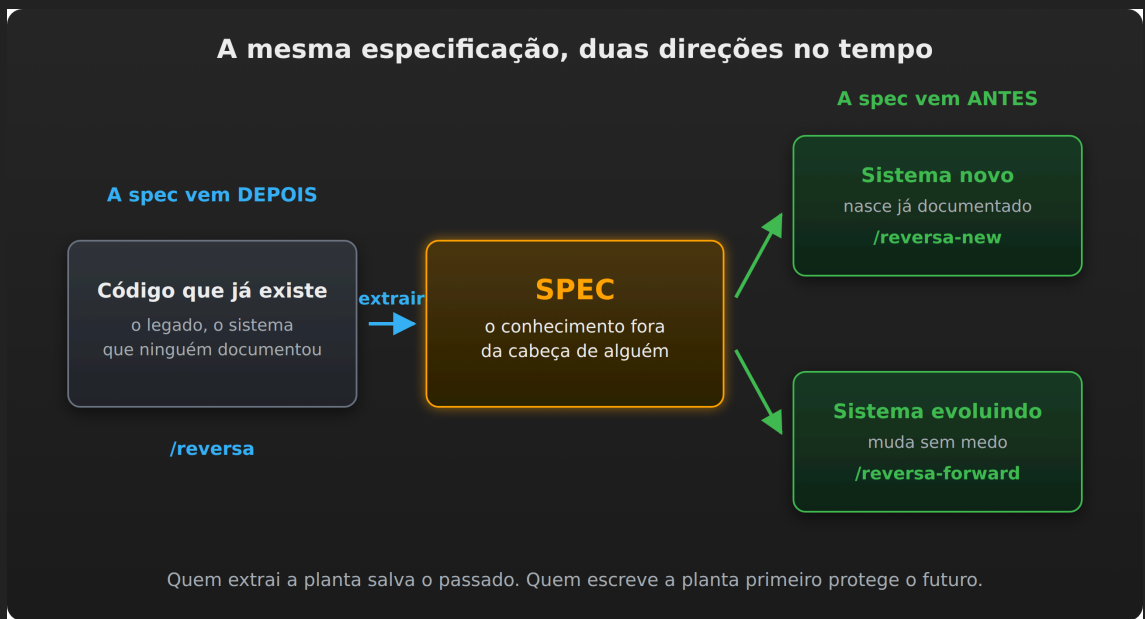


Figura 1.13: A mesma especificação em duas direções do tempo: extraída do que existe, escrita antes do que vai nascer, viva no que está mudando.

SDD, de spec-driven development: você especifica, o agente constrói, o teste prova, e a planta continua batendo com a casa para sempre.

Se você não tem sistema legado nenhum, este livro continua sendo sobre você. Porque o legado de amanhã está sendo escrito agora, no projeto que você está começando com todo o entusiasmo. A única diferença entre um futuro cômodo trancado e um sistema eternamente jovem é a decisão que você toma neste capítulo.

1.7 UM TIME DE ESPECIALISTAS DENTRO DO SEU TERMINAL

Muito bem. Extrair a planta do legado, escrever a spec do que nasce, manter tudo rastreável. Quem faz esse trabalho gigantesco? É aqui que este livro deixa de ser filosofia e vira ferramenta. A resposta está na Figura 1.14: um time inteiro de especialistas, cada um obcecado por uma única tarefa, trabalhando em volta da maquete da sua casa. Um mapeia o terreno com o olhar de quem avalia um imóvel. Outro escava fundações como um arqueólogo, camada por camada. Outro investiga os porquês como um detetive que não aceita "sempre foi assim" como resposta. Outro desenha os mapas formais. Outro arquiva cada descoberta no lugar certo.

Esse time existe, se chama **Reversa**, e mora dentro do seu terminal. O Reversa é um framework de agentes de IA organizado exatamente como uma equipe de engenharia de verdade: dezenas de agentes especializados, agrupados em times, cada time acionado por

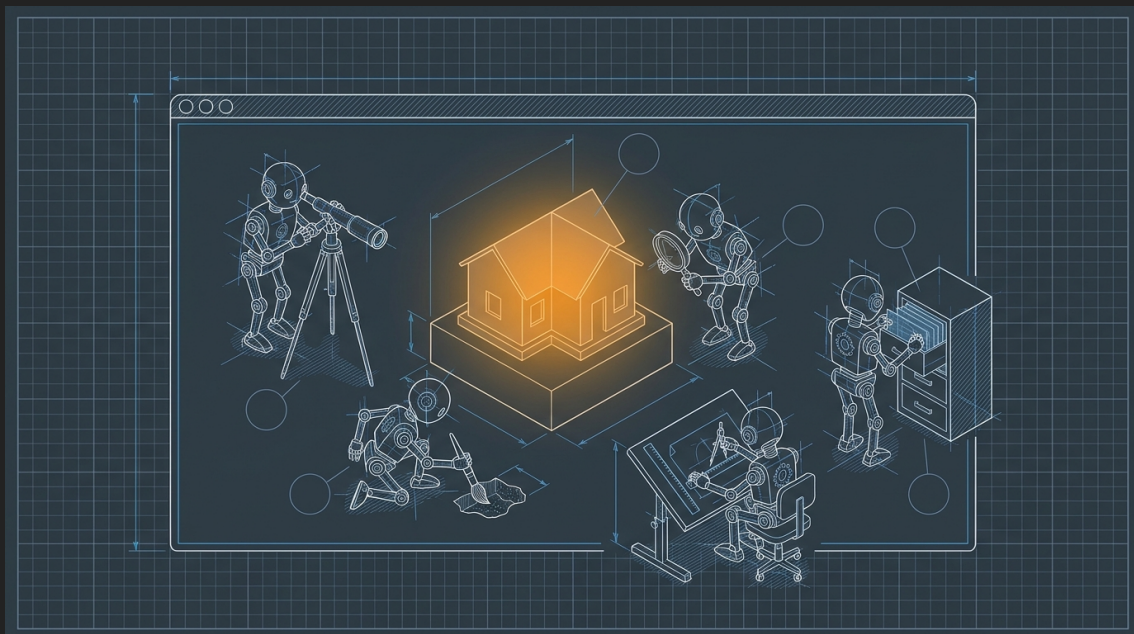


Figura 1.14: O segredo do Reversa: não é um agente genial que faz tudo, é um time de especialistas em que cada um faz uma coisa só, e faz bem.

um único comando. A Figura 1.15 mostra a organização completa. O **Time Core**, acionado por `/reversa`, analisa e especifica o legado: é o time do escaneamento da casa. O **Time de Migração**, com `/reversa-migrate`, transforma a planta extraída num projeto de sistema novo, com estratégia e prova de equivalência. O **Time de Criação**, com `/reversa-new`, pega uma ideia e a transforma em especificações prontas para construir. O **Time de Evolução**, com `/reversa-forward`, conduz o dia a dia de quem desenvolve guiado por spec. E o quinto grupo documenta visualmente, depura e precifica o trabalho.

Duas regras fazem esse time funcionar, e você vai vê-las em ação o livro inteiro. Primeira: **cada agente faz uma coisa só, e faz bem**. Não existe aqui o agente genérico que tenta resolver tudo e se perde; existe o Scout que só mapeia, o Archaeologist que só escava, o Writer que só escreve contratos. Segunda, a regra de ouro que você já conhece: **nenhum agente toca no seu código**. Todo o trabalho nasce em `.reversa/` e nas pastas de saída. Seu legado atravessa o processo inteiro intocado, do primeiro comando ao último relatório.

1.8 O MAPA DESTA JORNADA

Toda grande viagem começa com um mapa na mão, e o seu está na Figura 1.16: uma trilha que parte da casa antiga, atravessa cinco paradas iluminadas e termina numa cidade que brilha no horizonte. Cada parada é um capítulo deste livro, e cada capítulo entrega um

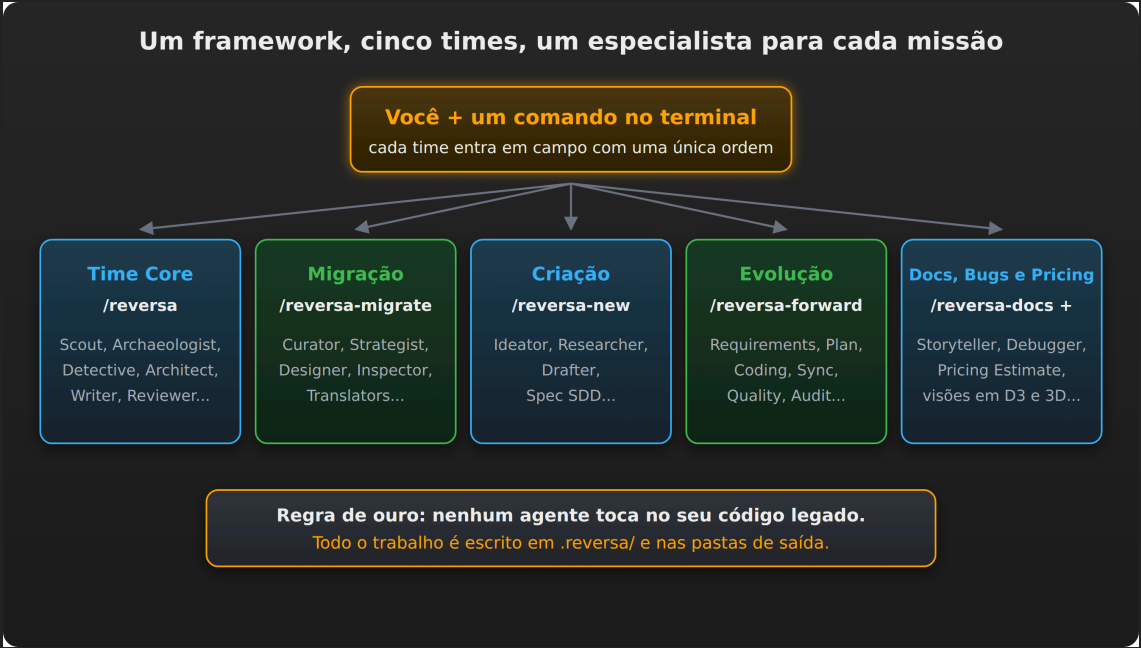


Figura 1.15: Os times do Reversa: um comando aciona cada time, cada agente faz uma única coisa, e nenhum deles jamais escreve no seu código legado.

time completo trabalhando para você.



Figura 1.16: A trilha do livro: da casa antiga que ninguém entendia até o sistema que nasceu, em cinco paradas.

A Figura 1.17 detalha o roteiro. Você está aqui, no Capítulo 1, com a ideia inteira na cabeça: sistemas envelhecem porque o conhecimento evapora, agentes precisam de contratos,

e a spec fecha o ciclo nas duas direções do tempo. O Capítulo 2 coloca o **Time Core** em campo e mostra, agente por agente, como um sistema legado é escaneado até virar especificação completa. O Capítulo 3 pega essa planta e atravessa a ponte da **migração**: estratégias, trade-offs e a prova de que o sistema novo faz o que o antigo fazia. O Capítulo 4 muda a direção do tempo com os times de **criação e evolução**: nascer com spec e crescer sem medo. E o Capítulo 5 fecha a jornada com o que todo projeto de verdade precisa: **documentação visual** que impressiona, **depuração** organizada e **precificação** honesta do trabalho.

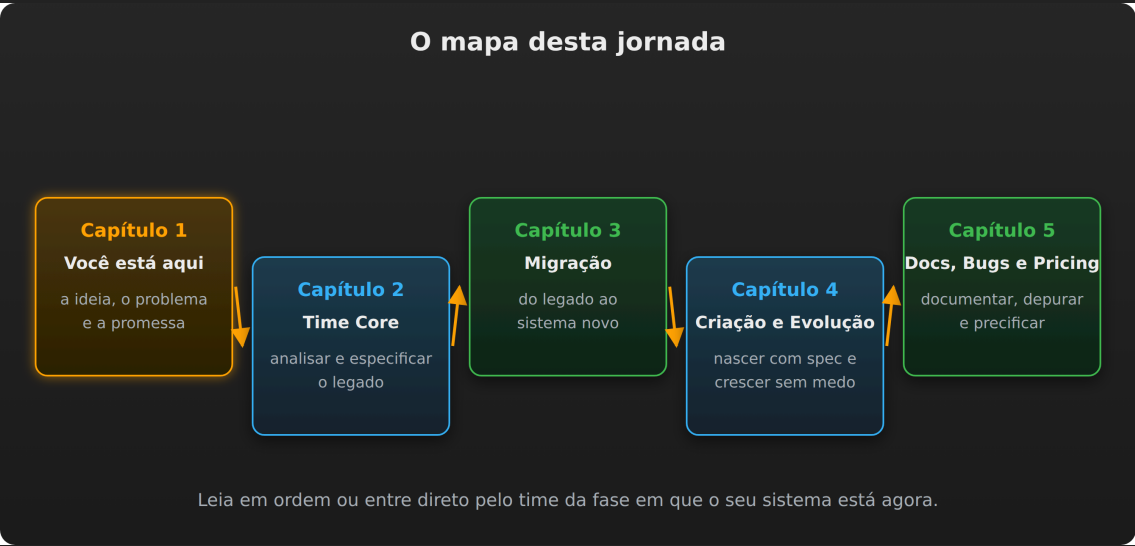


Figura 1.17: O mapa do livro: cinco capítulos, cada um entregando um time completo do Reversa trabalhando para você.

Você pode ler na ordem, e é o que eu recomendo para a primeira passada, porque cada capítulo constrói sobre o anterior. Mas o mapa também aceita atalhos: se o seu problema hoje é um legado indecifrável, vá direto ao Capítulo 2; se é uma migração marcada para o ano que vem, o Capítulo 3 espera por você; se é a ideia nova que precisa nascer bem, comece pelo Capítulo 4.

Uma última coisa antes de virar a página. Lá no começo, eu pedi que você pensasse no sistema mais importante da sua empresa e perguntei se alguém no planeta saberia explicá-lo por inteiro. A pergunta não era retórica. Nos próximos capítulos, esse alguém vai passar a existir. Não será uma pessoa: será um conjunto de especificações vivas, extraídas, testáveis e rastreáveis, geradas por um time de agentes que trabalha para você. A casa vai continuar de pé, com todas as suas histórias. A diferença é que, pela primeira vez, você vai ter a planta. Vamos abrir aquela porta.