Engenharia Caracol

Programar antes do código

Graciliano Tolentino

Edição revista e ampliada 2.0

2 de novembro de 2026

Fundamentos de programação e aplicação progressiva do método com exemplos resolvidos e um projeto executável.

Esta edição ensina a formular uma necessidade, construir uma solução pequena, verificar seu comportamento e manter memória suficiente para continuar. O percurso começa com arquivos e primeiras instruções e acompanha a Oficina Caracol até integração, auditoria e recuperação em laboratório.

O pacote acompanha os códigos, dados fictícios, testes, modelos preenchidos e registros das verificações realmente executadas. O texto distingue cenário de estudo, resultado esperado e evidência obtida.

Autoria edição e limites

A Engenharia Caracol é um método concebido e desenvolvido por Graciliano Tolentino. Esta edição conserva o eixo da edição fundacional de 22 de julho de 2026 e reorganiza sua aplicação didática. Práticas como requisitos, testes, versionamento e registros de decisão pertencem ao conhecimento consolidado de engenharia de software; a obra explicita como as combina em seu percurso.

A edição revista foi preparada com assistência de inteligência artificial e conferências técnicas automatizadas. O registro de revisão identifica as correções, as provas executadas e as verificações que permaneceram pendentes. Revisão humana independente e avaliação com leitores ainda precisam ser realizadas.

A formalização de M0 a M8, dos perfis e dos gates apresentada aqui é uma proposta operacional do método. As fontes citadas apoiam conceitos delimitados e não constituem aprovação externa da Engenharia Caracol. A eficácia da progressão para diferentes leitores precisa ser avaliada em atividades e projetos próprios.

Este volume aprofunda o fundamento e sua aplicação a um projeto de estudo. Não declara cobrir sozinho toda a graduação de Desenvolvimento de Software Multiplataforma, nem garante equivalência entre a competência do leitor e a do autor. A formação ampla planejada para a trilogia exige outros projetos, domínios e avaliações de transferência.

Como estudar e verificar o avanço

Se você nunca programou, comece pelos seis capítulos de fundamentos. Execute os arquivos na ordem e compare previsão e resultado. Se já conhece a linguagem, faça os exercícios finais dessa entrada antes de decidir que pode avançar diretamente. Encontrar um comando conhecido não demonstra todas as competências relacionadas.

Em cada capítulo principal, leia a explicação, examine o caso resolvido e tente o exercício antes de consultar sua resolução. Registre suas dúvidas e as diferenças entre sua solução e o critério. Gates controlam uma decisão com escopo; eles não são uma nota escolar universal ou um certificado de ausência de erros.

O kit final e os capítulos de conclusão têm funções de consulta e síntese, por isso não repetem obrigatoriamente o mesmo conjunto de seções dos capítulos principais. O glossário explica os termos e aponta capítulos para estudo. As remissões do sumário e a matriz histórica usam a numeração única desta edição.

Pasta do pacote	Uso
laboratorios_iniciais	Primeiros arquivos e testes explicados nos capítulos 1 a 6.
pratica	Projeto integrado com API interface testes e recuperação isolada.
figuras	Diagramas em PNG e SVG com texto alternativo no documento.
controle_revisao	Matriz de correções fontes cobertura e registros de conferência.

Sumário

A numeração de páginas abaixo corresponde ao PDF desta edição. Os títulos permitem saltar para os capítulos no arquivo digital. Em uma edição alterada, a paginação e o sumário devem ser regenerados juntos.

01 Computador arquivos editor e primeira execução..... 5
02 Valores variáveis objetos e coleções..... 8
03 Decisões repetições e acompanhamento da execução..... 11
04 Funções módulos contratos e tratamento de erros..... 14
05 Página navegador interação e comunicação com o servidor..... 17
06 Testes depuração versões e primeiro ciclo completo..... 20
07 O que significa programar..... 23
08 A identidade da Engenharia Caracol..... 25
09 Papéis e autoridades..... 28
10 Linguagem natural estruturada..... 31
11 Compreender profundamente o problema..... 34
12 Escopo e critérios de aceite..... 37
13 Inventário do estado atual..... 39
14 Preservar reparar refatorar ou reconstruir..... 42
15 Fonte única de verdade..... 45
16 Responsabilidade única e coesão..... 48
17 Waterfall e Caracol..... 51
18 Ciclos unidades e granularidade..... 55
19 Maturidade M0 a M8..... 58
20 Gates e vereditos..... 61
21 Evidência e rastreabilidade..... 64
22 Retorno controlado..... 67
23 Raiz diretórios e porta de entrada..... 70
24 Manifesto numeração e mapa geral..... 73
25 Backlog prioridades e pronto..... 75
26 Risco e proporcionalidade..... 78
27 Tecnologias e compatibilidade..... 81
28 Dependências e lockfile..... 84
29 Comandos reproduzíveis..... 87
30 Da especificação para o código..... 90
31 Uma unidade por vez sem perder o todo..... 93
32 Protocolo de IA..... 97
33 Revisão de código produzido por IA..... 100
34 Alucinações arquiteturais..... 103

35 Erros e estados de falha....106
36 Integração duplicação e mudanças....110
37 Estratégia de testes por risco....113
38 Níveis de teste e proteção contra regressões....115
39 Segurança e privacidade desde o desenho....118
40 Desempenho e acessibilidade em condições observáveis....120
41 Dados conteúdo e configuração como comportamento....122
42 Build entrega e decisão final....125
43 Auditoria documental arquitetural e de código....128
44 Dívida técnica e refatoração com propósito....131
45 Migração versão e depreciação sem surpresa....134
46 Observabilidade e resposta a incidentes....137
47 Continuidade institucional e transferência do projeto....139
48 Excesso de documentação....142
49 Paralisia por análise....144
50 Fragmentação e visão do todo....146
51 Falsa segurança e suficientemente bom....148
52 Perfil Essencial....150
53 Perfil Estruturado....152
54 Perfil Crítico e crise....155
55 Checklists essenciais....157
56 Modelos de governança....159
57 Prompts para a Vida Relacional....163
58 A engenharia como memória....166
59 O código como consequência e prova....167
60 Autonomia do desenvolvedor júnior....169
61 Pessoas IA e o futuro do método....171
Glossário....173
Referências e relação com os capítulos....194
Registro da edição e correspondência histórica....201

Fundamentos para começar

Aprender a executar observar e explicar um programa

Capítulo 01 Computador arquivos editor e primeira execução

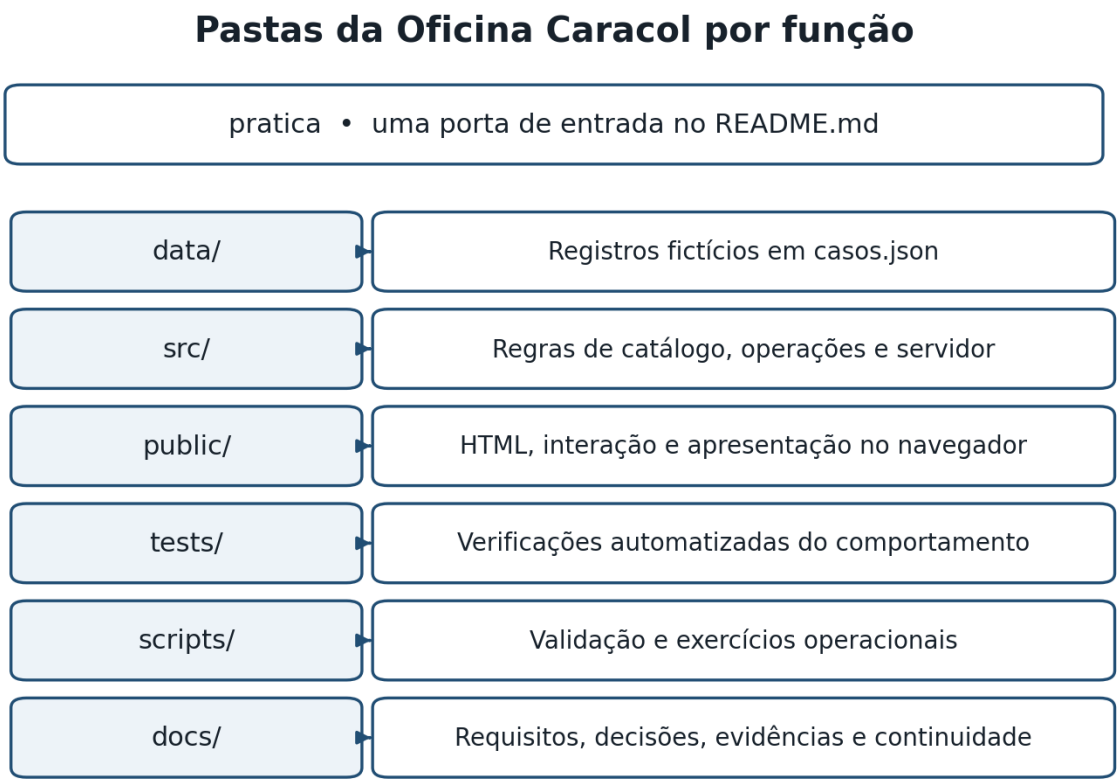


Figura 1 Pastas da Oficina Caracol por função

A separação acompanha responsabilidades. Não exige um arquivo por campo nem concentra todos os assuntos em um documento.

O que você construirá

Este percurso começa com um arquivo de texto que o computador executa. Você aprenderá a localizar esse arquivo, modificá-lo e verificar o efeito da mudança. Essa competência será usada durante todo o livro: antes de declarar que uma tarefa terminou, será necessário identificar o que mudou e observar o resultado.

Um programa contém instruções escritas segundo uma linguagem. O código-fonte é o texto dessas instruções. Um ambiente de execução lê esse texto e realiza as operações permitidas. Neste livro, JavaScript é a linguagem, Node.js executa os exemplos no terminal e o navegador executa a interface.

JavaScript e Java são linguagens distintas; os nomes parecidos não tornam seus programas intercambiáveis.

Onde o trabalho fica guardado

Uma pasta reúne arquivos e outras pastas. Um caminho indica a localização de um item: pratica/src/catalogo.mjs significa o arquivo catalogo.mjs dentro de src, dentro de pratica. O ponto em .mjs separa o nome de uma extensão. Essa extensão ajuda o Node a interpretar o arquivo como módulo JavaScript. Ela não transforma o conteúdo em código correto.

O pacote desta edição contém laboratorios_iniciais e pratica. Extraia o ZIP pelo gerenciador de arquivos e conserve uma cópia sem alterações. Não é necessário criar uma conta ou contratar um serviço para esses exercícios locais. Abra a pasta extraída em um editor que salve texto simples. Um processador de textos que insere formatação de parágrafos não é adequado para salvar código-fonte.

Item	Função neste percurso
Editor	Escrever e salvar instruções.
Terminal	Pedir ao computador que execute um comando e ler sua resposta.
Node.js	Executar JavaScript fora do navegador.
Navegador	Abrir a interface web e permitir interação.
npm	Executar tarefas descritas em package.json e administrar dependências.

Preparar e localizar o ambiente

Instale o Node.js a partir das instruções oficiais em nodejs.org, escolhendo seu sistema operacional. Os testes desta edição usam a linha 24; o registro técnico informa a versão exata executada. O instalador e a forma de abrir um terminal variam entre sistemas. Após instalar, abra um terminal novo e execute os comandos a seguir. Não copie símbolos de convite como $ ou > que apareçam em outros materiais.

Código 1 text

```
node --version
npm --version
```

Cada comando deve mostrar um número de versão. Se o sistema disser que node não foi localizado, resolva a instalação antes de tentar executar os exemplos. Reabrir um terminal pode ser necessário para que o caminho do programa seja atualizado. Anote a mensagem recebida; executar o mesmo comando repetidamente não corrige uma instalação ausente.

No gerenciador de arquivos, abra um terminal dentro de laboratorios_iniciais. Outra opção é usar cd seguido do caminho da pasta. Caminhos com espaços precisam de aspas. O diretório de trabalho é a pasta em que o terminal procura os caminhos relativos. O mesmo nome de arquivo pode existir em duas pastas; executar a cópia errada é uma causa comum de confusão.

Primeiro arquivo explicado

Abra 01_primeiro.mjs. Seu conteúdo completo é o seguinte.

Código 2 javascript

```
const projeto = 'Oficina Caracol';
console.log('Primeiro programa de ' + projeto);
```

Na primeira linha, const cria um nome para o valor de texto Oficina Caracol. O sinal = atribui esse valor; aqui não faz uma comparação. Na segunda, console.log mostra uma informação no terminal. O sinal + reúne os dois textos. As aspas delimitam texto e os parênteses contêm o valor entregue a console.log. Salve o arquivo antes de executar.

Código 3 text

```
node 01_primeiro.mjs
```

A saída esperada é Primeiro programa de Oficina Caracol. Ela comprova que esse arquivo foi executado nesse ambiente. Não comprova que o catálogo, o servidor ou a interface estejam corretos. A extensão da prova começa a ser praticada no primeiro exercício.

Prática guiada e resolução

Troque apenas o texto Oficina Caracol por Oficina de Estudos. Salve e execute novamente. Depois volte ao texto original. Se o resultado não mudar, confira o caminho do arquivo aberto, o diretório do terminal e se a alteração foi salva.

Agora retire temporariamente a aspa que fecha o nome do projeto. Execute, observe o erro de sintaxe e restaure a aspa. Um erro de sintaxe impede que o ambiente compreenda o programa; a mensagem costuma indicar uma localização, mas o engano pode ter começado antes dela. Compare a linha com a cópia preservada em vez de alterar várias partes sem hipótese.

Exercício: registre o nome do arquivo, o comando, a versão do Node e as duas saídas após a troca do nome. Solução de referência: as saídas devem diferir somente no nome do projeto. A atividade está concluída quando você consegue repetir a alteração e explicar qual arquivo foi executado. Um print sem caminho e sem comando não permite distinguir cópias diferentes.

Referências S18 S19 Detalhes e limites na seção de referências.

Capítulo 02 Valores variáveis objetos e coleções

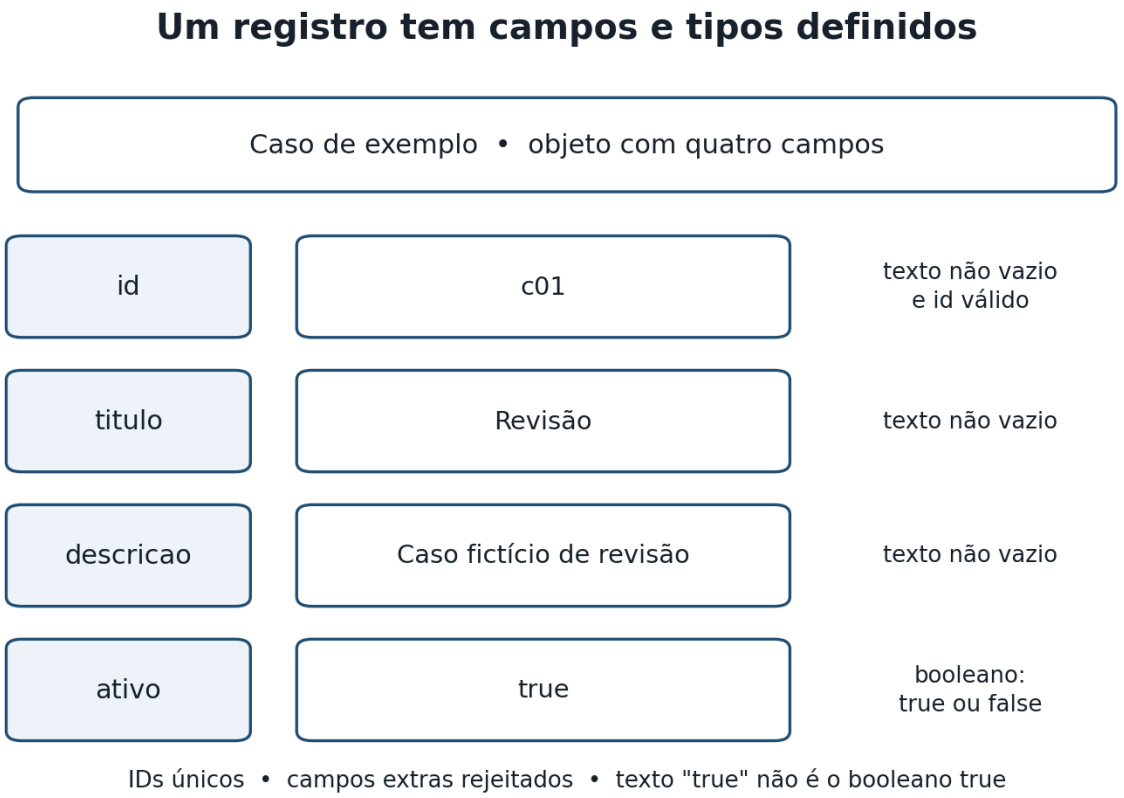


Figura 2 Um registro tem campos e tipos definidos

O formato é o contrato do pacote. O validador rejeita registros inválidos e propriedades extras; a aparência do texto não substitui o tipo do valor.

Representar uma informação

O catálogo precisa distinguir um título, uma quantidade e uma autorização de exibição. Esses valores têm usos diferentes. Uma string representa texto, como Conserto de bicicletas. Um number permite representar números e operações numéricas. Um boolean tem os valores true e false. A palavra true escrita sem aspas é um booleano; o texto "true" entre aspas é uma string.

Essa distinção evita uma falha concreta: se um arquivo traz ativo como o texto "false", um teste que apenas pergunta se o valor existe pode tratá-lo como verdadeiro. O contrato do projeto exige um booleano. Contrato é a descrição explícita do que uma parte aceita e devolve. Mais adiante, a validação rejeitará os dados que não o cumprem.

Nomes e atribuições

const cria um vínculo que não poderá receber outro valor por uma nova atribuição. let permite trocar o valor associado ao nome. Escolha nomes que expressem a informação; titulo informa mais do que x.

Palavras da linguagem, como const, precisam respeitar a sintaxe do JavaScript. Você pode escolher nomes em português para suas próprias variáveis.

Código 4 javascript

```
const titulo = 'Revisão';
let vagas = 2;
vagas = vagas - 1;
console.log(titulo, vagas);
```

O programa calcula 2 menos 1 e guarda 1 em vagas. O lado direito de uma atribuição é avaliado antes de o nome receber o resultado. A igualdade estrita é escrita ===; vagas === 1 produz um booleano. Não confunda essa comparação com vagas = 1, que modifica o valor.

Expressão	Resultado e motivo
2 + 3	5 porque soma dois números.
'2' + '3'	23 como texto, pois reúne strings.
true === 'true'	false porque os tipos diferem.
2 === 2	true porque valor e tipo coincidem.

Um registro e uma lista de registros

Um objeto agrupa propriedades relacionadas. No objeto caso, id é o identificador estável, titulo é o nome exibido, descricao explica o serviço e ativo controla se ele pode aparecer. A expressão caso.titulo lê uma propriedade. A ordem de escrita das propriedades não define uma sequência de execução do atendimento.

Um array reúne valores em uma sequência. Os índices começam em zero: casos[0] lê o primeiro item. Uma coleção vazia é escrita []. Vazia não significa inválida: pode ser um catálogo válido ainda sem serviços ativos. Ao procurar uma posição que não existe, o resultado pode ser undefined; acessar uma propriedade desse resultado sem verificar causa uma falha.

Código 5 javascript

```
const titulo = 'Conserto de bicicletas';
const ativo = true;
const vagas = 2;
const caso = { id: 'CASO-001', titulo, descricao: 'Revisão básica.', ativo };
const casos = [caso];
console.log(typeof titulo, typeof ativo, typeof vagas);
console.log(casos[0].id, casos[0].ativo === true);
```

Salve e execute node 02_dados.mjs dentro de laboratorios_iniciais. A primeira saída identifica os tipos string boolean number. A segunda mostra CASO-001 true. O acesso casos[0].id primeiro encontra o registro e depois lê seu identificador. A expressão casos[0].ativo === true compara o valor com o booleano esperado.

Copiar e transformar sem surpreender o chamador

Uma função que recebe um array pode alterar o próprio array ou devolver outro. O contrato precisa tornar isso previsível. No projeto, a listagem pública produz novos objetos contendo somente id, titulo e descricao. Ela não muda a fonte e não expõe a propriedade ativo. Esse recorte de saída é uma decisão explícita, não uma consequência automática de usar JSON.

const não congela o interior de um objeto. Se const caso guarda um objeto, a instrução caso.titulo = "Outro" ainda pode modificar sua propriedade. O que fica impedido é atribuir um objeto inteiramente diferente ao mesmo nome caso. Essa diferença será relevante ao comparar o estado anterior e posterior de uma operação.

Exercício com dados fornecidos

Considere A com ativo true, B com ativo false e C com ativo "false". Indique os tipos dos três campos, quais registros cumprem o contrato e quais devem aparecer se todos os dados forem válidos. Resolva antes de consultar a resposta.

Solução: A e B possuem booleanos e cumprem essa parte do contrato; C possui string e deve ser rejeitado. Em um conjunto válido formado apenas por A e B, somente A aparece. No projeto completo, a presença de C invalida o conjunto recebido, em vez de silenciosamente ocultar o erro. O comportamento foi escolhido para tornar a qualidade dos dados visível.

Para concluir, explique por que [] pode ser uma entrada válida, por que o primeiro índice é 0 e por que um nome declarado com const pode apontar para um objeto cujas propriedades mudam. Se sua resposta chamar "false" de booleano, retome a tabela antes de avançar.

Referências S18 Detalhes e limites na seção de referências.

Capítulo 03 Decisões repetições e acompanhamento da execução

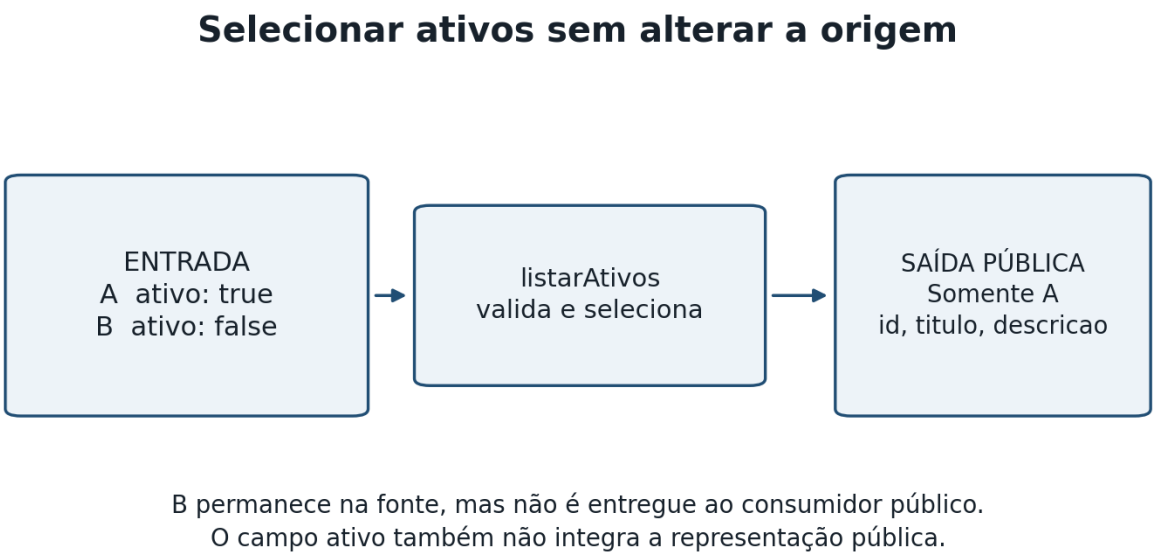


Figura 3 Selecionar ativos sem alterar a origem

Filtrar a saída não significa apagar o registro inativo da fonte. Ocultar um cartão depois de entregar o registro ao navegador não atende à mesma fronteira de publicação.

Dar ao programa uma regra observável

O requisito de catálogo contém uma decisão: mostrar o registro quando ativo for true e omiti-lo quando for false. Uma condição representa essa escolha. if executa um bloco quando a condição é verdadeira; else permite tratar a outra situação. Antes de escrever o bloco, estabeleça resultados para entradas concretas. Isso fornece um critério para verificar a implementação.

Código 6 javascript

```
const ativo = false;
if (ativo === true) {
  console.log('Publicar');
} else {
  console.log('Omitir');
}
```

A saída esperada é Omitir. As chaves delimitam os comandos de cada alternativa. A comparação === true deixa explícita a regra de exibição, mas não substitui a validação do tipo dos dados: uma string incorreta pode ser omitida sem que o problema do arquivo seja denunciado. No projeto completo, validar e selecionar são responsabilidades separadas.

Percorrer vários registros

Repetição é a execução de um conjunto de instruções para mais de um item ou enquanto uma condição se mantém. for...of percorre os valores de uma coleção. Em cada passagem, o nome caso recebe o próximo registro. O array ids começa vazio e push acrescenta um identificador ao final. O código abaixo é o arquivo completo 03_decisoes.mjs.

Código 7 javascript

```
const casos = [
  { id: 'A', ativo: true },
  { id: 'B', ativo: false },
  { id: 'C', ativo: true }
];
const ids = [];
for (const caso of casos) {
  if (caso.ativo === true) {
    ids.push(caso.id);
  }
}
console.log(ids);
```

O resultado esperado contém A e C, nessa ordem. Execute node 03_decisoes.mjs. Não é preciso decorar a forma gráfica exata com que o terminal imprime um array; verifique os valores e a ordem. Se B aparecer, a regra falhou. Se A e C aparecerem duas vezes, investigue onde a coleção de resultado é criada e quantas vezes o bloco está sendo executado.

Passagem	Registro lido	Comparação	Resultado acumulado
1	A	true === true	A
2	B	false === true	A
3	C	true === true	A e C

Simular antes de pedir ajuda

A tabela de acompanhamento é uma forma simples de depurar: observar como o estado evolui durante a execução. Escreva o valor de entrada, a condição calculada e a coleção após cada passagem. Quando o resultado difere do esperado, localize a primeira linha em que a diferença surgiu. Corrigir essa causa costuma ser mais útil do que alterar o último comando de exibição.

Um console.log dentro do bloco pode revelar o registro atual. Durante uma investigação, use somente dados fictícios ou informações que possam ser registradas. Apague mensagens temporárias desnecessárias depois de entender o defeito. Log não é um lugar apropriado para senhas, tokens ou dados pessoais completos.

Uma condição não precisa se reduzir a um campo. O operador && exige que as duas condições sejam verdadeiras; || aceita que pelo menos uma seja verdadeira. Use parênteses quando a combinação puder ser interpretada de mais de uma forma. Antes de escrever condições extensas, produza uma tabela dos casos que elas devem aceitar e rejeitar.

Falhas que este exercício permite perceber

Sintoma	Causa que deve ser investigada	Reparo correspondente
Todos aparecem	A condição foi removida ou sempre é verdadeira.	Reestabelecer a condição e verificar entrada ativa e inativa.
Só o último permanece	O resultado foi reinicializado dentro da repetição.	Criar a coleção antes do laço.
Ordem inesperada	Uma ordenação ou transformação foi introduzida.	Comparar a ordem recebida e a regra de saída acordada.
Nada acontece	O arquivo executado pode ser diferente do editado.	Conferir caminho e salvar antes de executar.

Exercício e solução comentada

Use os registros A falso, B verdadeiro, C falso e D verdadeiro. Antes de executar, escreva quatro linhas da tabela de acompanhamento. Depois adapte somente a lista do arquivo e compare a previsão com a saída. Uma solução correta acumula vazio, B, B e B/D ao final das quatro passagens.

Variação: acrescente E com ativo "true". A seleção estrita não deve incluí-lo, mas uma implementação de catálogo confiável também deve sinalizar que ele viola o contrato de entrada. Essa diferença separa acertar um filtro de validar um conjunto de dados. Sua conclusão precisa mencionar as duas tarefas.

Você pode avançar quando consegue prever o estado após cada passagem, explicar por que a coleção de saída começa antes do laço e localizar uma primeira divergência entre previsão e execução. Repetir o código sem conseguir fazer a tabela ainda não demonstra compreensão da regra.

Referências S18 Detalhes e limites na seção de referências.

Capítulo 04 Funções módulos contratos e tratamento de erros

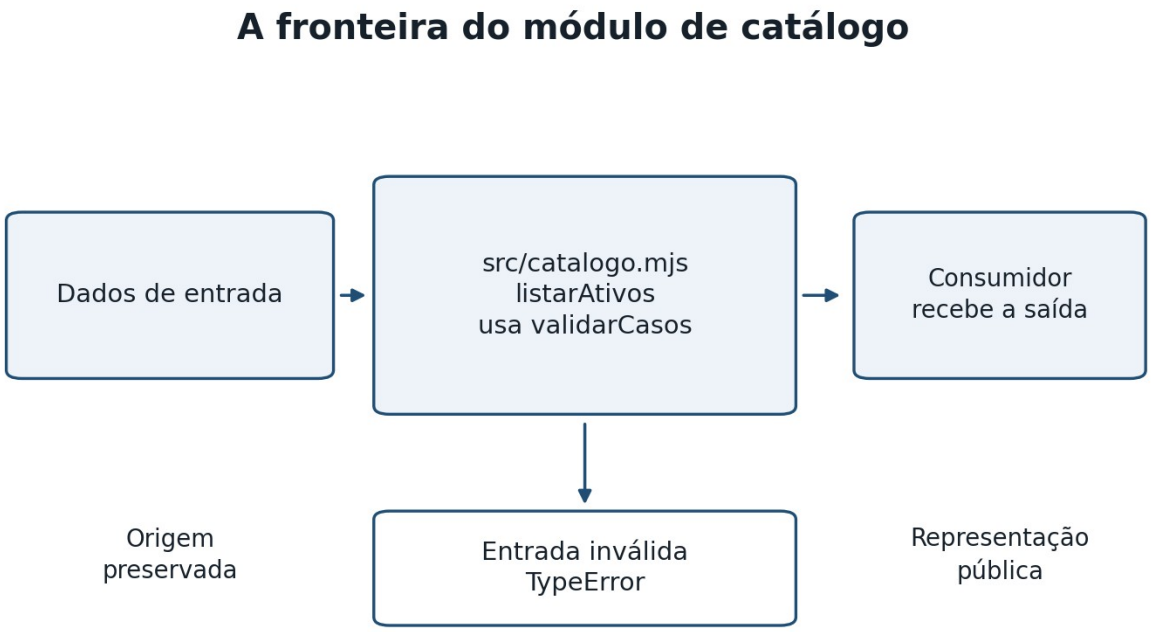


Figura 4 A fronteira do módulo de catálogo

A validação pertence ao contrato do catálogo. O consumidor deve tratar a falha; a validação não demonstra que a interface e o servidor já estão integrados.

Nomear uma transformação

Uma função agrupa instruções para realizar uma tarefa identificável. Parâmetros são os nomes que recebem as entradas; argumentos são os valores entregues na chamada. O retorno é o resultado que a função devolve. Separar esses elementos permite testar uma regra sem precisar abrir uma interface.

A tarefa selecionarTitulos recebe uma lista de casos e devolve os títulos dos ativos. Ela não grava um arquivo nem imprime cada item. Quem a chama decide como usar o resultado. Essa escolha reduz dependências: o mesmo cálculo pode ser usado em um terminal ou em uma página.

Código completo de uma função pequena

Código 8 javascript

```
export function selecionarTitulos(casos) {
  if (!Array.isArray(casos)) {
    throw new TypeError('A entrada deve ser uma lista.');
  }
  const titulos = [];
  for (const caso of casos) {
    if (caso.ativo === true) titulos.push(caso.titulo);
  }
  return titulos;
}
```

A primeira linha exporta a função, permitindo que outro módulo a importe. O nome casos entre parênteses é o parâmetro. Array.isArray verifica se a entrada é uma coleção do tipo esperado. O ponto de exclamação nega o resultado: se não for array, o bloco lança um erro. throw interrompe esse caminho e entrega uma informação de falha a quem chamou.

O bloco seguinte reaproveita a seleção já estudada e return devolve titulos. Um retorno não é a mesma operação que console.log: imprimir ajuda uma pessoa a observar, mas não devolve automaticamente um valor utilizável por outra função. O exemplo valida a forma externa da entrada; a validação completa de cada registro pertence a catalogo.mjs no projeto principal.

Código 9 javascript

```
import { selecionarTitulos } from './04_funcoes.mjs';
const casos = [
  { titulo: 'Bicicletas', ativo: true },
  { titulo: 'Roupas', ativo: false }
];
console.log(selecionarTitulos(casos));
```

O segundo trecho é o arquivo 04_executar.mjs. Execute node 04_executar.mjs na pasta dos laboratórios. O caminho ./04_funcoes.mjs parte da localização do módulo que importa; ./ indica o diretório atual nessa referência. O retorno contém apenas Bicicletas. Os dois arquivos precisam existir juntos.

Tratar falhas de forma explícita

Um erro pode ser capturado por try...catch quando há uma resposta útil a oferecer. Se uma interface recebe dados inválidos, uma resposta adequada é mostrar que o catálogo não pôde ser carregado e preservar informações para diagnóstico. Devolver uma lista vazia silenciosamente confundiria ausência de serviços com uma falha na fonte.

Código 10 javascript

```
import { selecionarTitulos } from './04_funcoes.mjs';
try {
  selecionarTitulos(null);
} catch (erro) {
  console.log(erro.message);
}
```

Salve esse trecho, se quiser executá-lo, como 04_erro.mjs ao lado dos outros módulos. A saída esperada é A entrada deve ser uma lista. Capturar não corrige a entrada: apenas permite controlar como a falha é comunicada. Se você não tiver uma recuperação apropriada, permita que o erro chegue à camada que pode tomar a decisão, mantendo contexto suficiente e evitando expor informação sensível.

O contrato do projeto principal

Função de catalogo.mjs	Entrada	Saída ou falha
validarCasos	Array de objetos do contrato completo.	Aceita conjunto válido ou lança TypeError.
listarAtivos	Conjunto completo válido.	Nova lista de objetos públicos ativos.
obterCasoAtivo	Conjunto válido e identificador.	Objeto público correspondente ou null.

Consulte a implementação entregue antes de mudar um contrato. Uma função pública não precisa revelar todos os campos da fonte. Usar somente id, titulo e descricao na saída torna a fronteira explícita. A reserva simulada, em outro módulo, possui um contrato diferente: recebe chave e casoId e devolve reservaId e casoId. Não misture esse estado de operação com o arquivo editorial do catálogo.

Exercício com solução e critério

Crie um conjunto vazio e outro com um ativo e dois inativos. Preveja o retorno de selecionarTitulos. Depois chame a função com um objeto simples em vez de uma lista. Solução: [] no primeiro caso, um título no segundo e TypeError no terceiro. A função não deve alterar os objetos fornecidos.

Explique uma vantagem de devolver o resultado em vez de imprimi-lo dentro da função. Uma resposta adequada mostra um consumidor concreto, como um teste que compara a lista recebida, e reconhece que a comunicação com a pessoa pode ocorrer em uma camada diferente. A conclusão exige executar as três situações e identificar qual contrato foi verificado.

Referências S18 S19 Detalhes e limites na seção de referências.

Capítulo 05 Página navegador interação e comunicação com o servidor

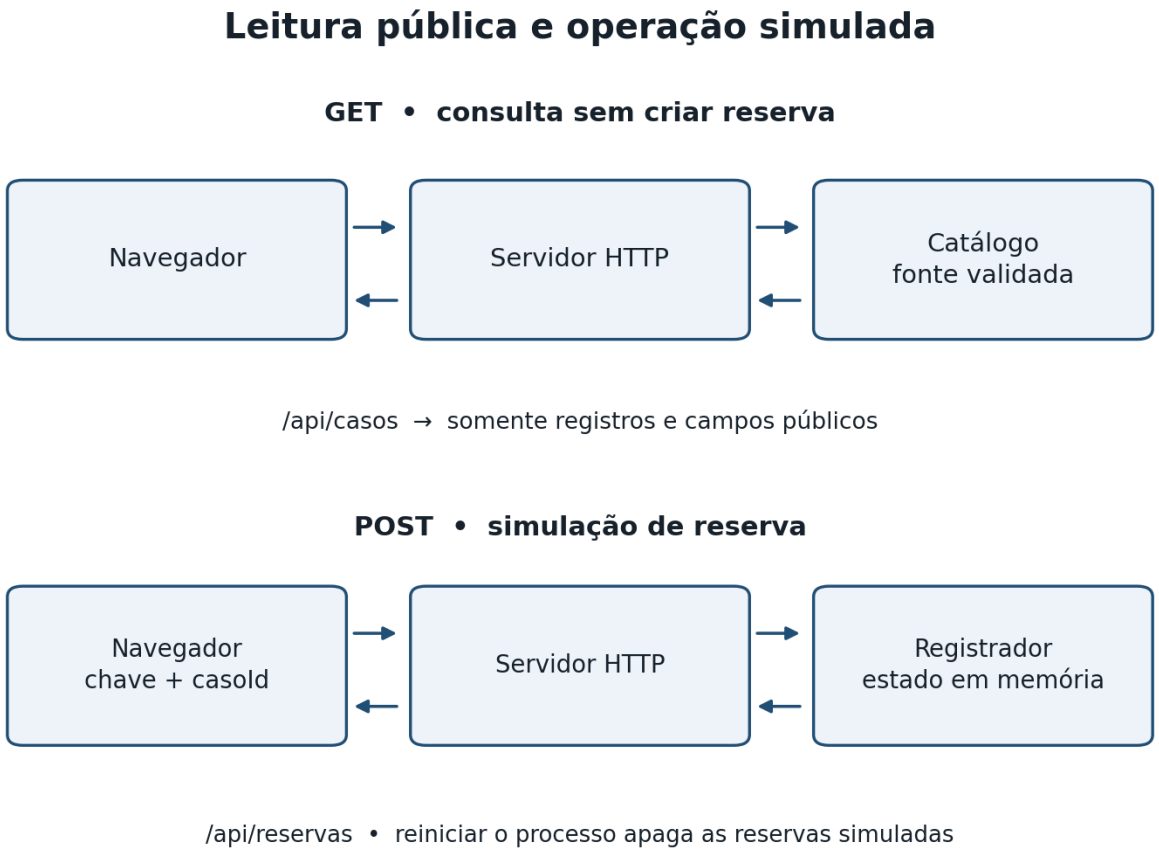


Figura 5 Leitura pública e operação simulada

O estado temporário das reservas pertence ao registrador; a leitura do catálogo não cria reserva. O projeto não representa um sistema real de agendamento.

As partes de uma página

Uma página web combina estrutura, apresentação e comportamento. HTML descreve os elementos, CSS define sua apresentação e JavaScript responde a ações ou transforma informações. São papéis complementares. Um botão que parece correto pode não realizar a ação esperada; um teste de cálculo pode passar sem que o botão esteja acessível.

Abra 05_pagina.html no navegador. Este laboratório é um arquivo local independente, sem servidor. Ele serve para estudar uma interação pequena. O catálogo principal será aberto pelo endereço local fornecido por npm start; não substitua esse passo por abrir seu index.html diretamente, pois a interface precisa conversar com a API.

Ler uma implementação curta

Código 11 html

```
<!doctype html>
<html lang="pt-BR">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Primeira interação Caracol</title>
  <style>
    body { font: 1.1rem sans-serif; margin: 2rem; max-width: 42rem; }
    button { padding: .7rem 1rem; font: inherit; }
    button:focus-visible { outline: 3px solid #005fcc; outline-offset: 3px; }
  </style>
</head>
<body>
  <h1>Contador de consultas de exemplo</h1>
  <p>Ative o botão com mouse ou teclado.</p>
  <button id="consultar" type="button">Registrar consulta</button>
  <p id="estado" role="status">Consultas registradas 0</p>
  <script>
    let total = 0;
    const botao = document.querySelector('#consultar');
    const estado = document.querySelector('#estado');
    botao.addEventListener('click', () => {
      total = total + 1;
      estado.textContent = 'Consultas registradas ' + total;
    });
  </script>
</body>
</html>
```

O atributo lang identifica o idioma da página. meta charset informa a codificação. viewport orienta a apresentação em telas menores. title nomeia a aba; h1 apresenta o título do conteúdo. O elemento button já possui comportamento de teclado que seria necessário recriar ao usar um elemento genérico como botão.

O CSS controla tamanho e espaçamento e torna o foco visível. Foco é o elemento que receberá a próxima ação de teclado. role="status" identifica uma região de atualização que tecnologias assistivas podem anunciar; sua presença isolada não certifica a acessibilidade da página. A leitura com diferentes recursos precisa ser verificada no contexto real.

No script, document.querySelector encontra os elementos por identificador. addEventListener associa uma função à ação click. A expressão () => introduz uma função sem nome nessa posição. A função só é executada quando ocorre o evento. textContent atualiza texto, evitando interpretar o conteúdo como marcação HTML. Nesse laboratório, total existe apenas enquanto a página permanece aberta. Recarregar reinicia a contagem.

Do arquivo local ao serviço HTTP

HTTP é o protocolo usado na troca de mensagens entre a interface e o servidor do projeto. A interface envia uma requisição; o servidor interpreta método e caminho e devolve status, cabeçalhos e corpo. API é uma fronteira programável que permite a outra parte usar um comportamento por um contrato.

Requisição do projeto	Finalidade	Resultado esperado
GET /api/casos	Consultar casos ativos.	200 e array público; [] é catálogo vazio válido.
POST /api/reservas	Simular reserva de um caso ativo.	200 na primeira reserva e na repetição da mesma chave e caso; 409 para a mesma chave com outro caso.
Caminho desconhecido	Pedir um recurso inexistente.	404 sem expor arquivos privados.

JSON é um formato de texto para dados. Sua sintaxe exige aspas duplas nos nomes de propriedades e não admite comentários comuns de JavaScript. Ao receber JSON, o programa ainda precisa verificar tipos e regras. Um documento bem formado pode conter um id duplicado ou um booleano escrito como texto.

Uma requisição demora um tempo que varia. Promise representa um resultado futuro; await permite aguardar esse resultado dentro de uma função async. A interface precisa mostrar carregamento, sucesso vazio, sucesso com dados e falha. A existência de uma resposta não significa sucesso: uma resposta HTTP 500 também chega ao navegador e precisa ser tratada.

Prática por teclado e exercício resolvido

No laboratório, use Tab para chegar ao botão. Observe o foco e use Enter ou espaço para ativá-lo. A contagem deve mudar uma vez por ativação. Depois recarregue. O retorno a zero é esperado neste exemplo em memória, não é prova de persistência. Anote navegador, ação e resultado; não declare conformidade com toda a WCAG a partir desses passos.

Exercício: uma pessoa clica três vezes, recarrega e ativa uma vez por teclado. Qual valor deve aparecer ao final? Solução: 1, porque o recarregamento criou uma nova execução do script. Se a necessidade fosse conservar o total entre sessões, seria preciso definir armazenamento, autorização, tratamento de falhas e testes apropriados.

Você pode avançar quando distingue texto de dados, cálculo de interface e estado local de estado no servidor. No projeto principal, verifique separadamente a resposta da API e o que a pessoa consegue fazer na tela; uma não substitui a outra.

Referências S18 S19 S06 Detalhes e limites na seção de referências.

Capítulo 06 Testes depuração versões e primeiro ciclo completo

Transformar uma expectativa em verificação

Um teste executa um cenário e compara o resultado observado com uma expectativa. A expectativa precisa vir da regra, e não de copiar o resultado que o programa já produziu. Para a função de títulos, esperamos que um inativo fique de fora e que a entrada permaneça intacta. Um erro que alterasse os dados poderia passar em um teste que só observa o primeiro título.

O arquivo 06_funcoes.test.mjs usa o executor de testes incluído no Node. Não exige instalar uma biblioteca de testes. test organiza um cenário, e assert verifica uma condição. O nome do teste descreve o comportamento protegido. Leia a lista antes de executar para saber o que uma aprovação realmente cobre.

Uma suíte pequena explicada

Código 12 javascript

```
import test from 'node:test';
import assert from 'node:assert/strict';
import { selecionarTitulos } from './04_funcoes.mjs';

test('lista somente títulos ativos e preserva a entrada', () => {
  const casos = [
    { titulo: 'A', ativo: true },
    { titulo: 'B', ativo: false }
  ];
  const antes = structuredClone(casos);
  assert.deepEqual(selecionarTitulos(casos), ['A']);
  assert.deepEqual(casos, antes);
});
test('lista vazia é aceita', () => {
  assert.deepEqual(selecionarTitulos([]), []);
});
test('entrada que não é lista é rejeitada', () => {
  assert.throws(() => selecionarTitulos(null), TypeError);
});
```

deepEqual compara a estrutura e os valores das coleções; comparar dois arrays com === verificaria se são o mesmo objeto, não se contêm os mesmos itens. structuredClone cria uma cópia para verificar o estado anterior. throws exige que a chamada fornecida lance o tipo de erro indicado. A função passada ao teste adia a execução da chamada até o momento da verificação.

Código 13 text

```
node --test 06_funcoes.test.mjs
```

O resultado esperado é a aprovação dos três testes. O formato exato da apresentação pode variar com a versão e a configuração do executor. Leia os nomes e a contagem e confira o código de saída. Uma suíte aprovada não prova o contrato completo do catálogo; prova os cenários definidos para esta função simplificada.

Investigar uma falha com hipótese

Mude temporariamente a condição da função para caso.ativo === false. Execute a suíte sem alterar o resultado esperado. O primeiro teste deve falhar porque a regra de seleção foi invertida. Restaure true e execute novamente. Agora a aprovação tem um significado adicional: o teste demonstrou capacidade de detectar aquele defeito.

Esse percurso fornece proteção de regressão para uma mudança: conservar verificações que identificam o comportamento indesejado caso ele volte. Regressão não é um nível de teste separado. Pode haver proteção de regressão em funções isoladas, na integração HTTP e na interface completa.

Quando um teste falha, leia primeiro qual cenário falhou, depois a diferença entre esperado e recebido. Reproduza com o menor conjunto de entradas suficiente para explicar o problema. Não apague uma asserção só para obter uma saída verde. Se a regra mudou legitimamente, registre a nova decisão e acrescente cenários que mostrem o antes e o depois.

Guardar versões que uma pessoa consegue comparar

Git registra versões de arquivos e permite comparar alterações. Um commit representa um estado identificado do repositório. Um diff mostra diferenças entre versões ou entre o arquivo atual e um estado registrado. Git não é o mesmo serviço que GitHub; os comandos locais podem ser usados sem publicar o trabalho.

Se Git estiver instalado, faça o exercício em uma cópia da pasta laboratorios_iniciais. Inicialize o repositório, examine os arquivos e registre uma versão somente após revisar o que será incluído. Configure sua autoria conforme as instruções do Git se o programa solicitar; não há identidade fictícia obrigatória nesta atividade.

Código 14 text

```
git init
git status
git add 01_primeiro.mjs 04_funcoes.mjs 06_funcoes.test.mjs
git diff --staged
git commit -m "Registra primeiros laboratórios"
```

Depois de alterar uma linha, git diff mostra a mudança ainda não registrada. O histórico ajuda a recuperar código, mas não resolve sozinho exposição de uma credencial: apagar a linha atual não invalida um token já divulgado. Os capítulos de segurança tratam a resposta apropriada a esse evento.

Executar a Oficina Caracol e fechar o ciclo

Entre na pasta pratica do pacote. Leia seu README e execute npm ci, npm test e npm run validate. O primeiro prepara o ambiente conforme o manifesto e o lockfile; neste projeto não há dependências externas. O segundo verifica cenários automatizados. O terceiro reúne sintaxe, contrato de dados e testes. Um comando existir em package.json não prova sua aprovação: leia o resultado de sua execução.

Código 15 text

```
npm ci
npm test
npm run validate
npm start
```

Com o servidor em execução, abra http://127.0.0.1:3000. O terminal permanece ocupado pelo processo; Control+C o encerra. Em outro terminal, você pode executar testes. A reserva é uma simulação em memória e não efetua pagamento, agenda real ou armazenamento durável. Reiniciar o processo perde esse estado. Snapshot e restore pertencem exclusivamente ao laboratório de recuperação em sandbox.

Exercício final: altere o título de um caso ativo na cópia de trabalho, valide os dados, execute os testes e observe a interface. Entregue o diff, o comando, o resultado e o limite do que conferiu. Solução de referência: o título muda, os identificadores continuam estáveis, inativos continuam omitidos e os testes aplicáveis passam. Se a interface não foi observada, registre essa lacuna em vez de declarar o fluxo visual aprovado.

Ao concluir esses passos, você já percorreu uma primeira unidade Caracol: formulou uma mudança, delimitou arquivos, implementou, verificou, registrou evidência e decidiu a próxima ação. Os capítulos seguintes aprofundam como organizar essa prática em projetos com mais pessoas, dados e risco.

Referências S07 S16 S19 Detalhes e limites na seção de referências.

Parte I Programar antes do código

Formular transformações contratos e responsabilidades

Capítulo 07 O que significa programar

Da intenção ao comportamento

Imagine que você recebeu a encomenda de criar o catálogo da Oficina Caracol. A frase parece simples, mas ainda não permite decidir se uma entrega está correta. Catálogo de quê? Quem pode consultar? Um serviço desativado aparece? O visitante precisa fornecer nome? Essas perguntas já pertencem ao trabalho de programar: elas definem a transformação que o código deverá realizar.

Um programa combina instruções para produzir comportamento a partir de entradas. A entrada pode ser uma lista de serviços; a saída pode ser uma relação visível na página. O comportamento inclui tanto o caminho esperado quanto as respostas a situações excepcionais. Uma tela bonita que apresenta um serviço inativo contradiz a necessidade, mesmo que o navegador não mostre erro.

Na Engenharia Caracol, organizamos essa construção para conseguir explicar o que mudou, por que mudou e como verificar a mudança. Planejar, implementar e verificar são atividades relacionadas. O planejamento não substitui a execução; a execução também não prova, por si, que o problema escolhido foi resolvido.

O primeiro contrato do catálogo

Contrato significa aqui um acordo explícito sobre entradas, resultados e limites. O quadro é um exemplo preenchido de especificação, não um registro de testes já realizados. Os identificadores c01 e c02 representam dados inventados para o estudo.

Elemento	Definição para a Oficina Caracol
Pessoa atendida	Visitante que quer conhecer os serviços disponíveis.
Entrada válida	Lista de registros de serviço conforme o formato acordado.
Transformação	Selecionar serviços ativos e apresentar seus títulos.
Saída normal	Somente os títulos dos serviços ativos aparecem no catálogo.
Lista vazia	A página informa que não há serviços disponíveis.
Entrada inválida	A falha recebe uma resposta explícita e distinta da lista vazia.
Limite	Sem cadastro, banco externo ou dados pessoais reais.

A distinção entre vazio e inválido muda a experiência. Vazio significa que recebemos uma lista válida sem serviços para mostrar. Inválido significa que não podemos confiar na entrada. Transformar qualquer erro em uma tela vazia esconderia um defeito de dados e dificultaria sua correção.

Procedimento para começar

- Escreva a necessidade usando o verbo que descreve a ação do visitante: consultar serviços disponíveis.
- Liste entradas e resultados antes de escolher ferramentas. Inclua vazio, erro e informação que não deve aparecer.
- Converta cada resultado em uma observação verificável. Para uma entrada com um serviço ativo e outro inativo, somente o ativo deve aparecer.
- Escolha a menor entrega que demonstre essa transformação completa. Uma página com dados de exemplo pode ser suficiente para o primeiro ensaio.
- Registre o que ainda está em aberto. Não trate uma pergunta sem resposta como autorização para inventar uma regra.

Erros que mudam a finalidade

Erro	Por que atrapalha	Reparo
Começar pela escolha de uma biblioteca	A escolha não define o resultado que o visitante precisa obter.	Formular primeiro o comportamento e depois justificar a ferramenta.
Avaliar somente a aparência	Um serviço indisponível pode continuar exposto.	Verificar conteúdo e estados além da apresentação.
Declarar pronto porque abriu	Abrir a página não demonstra entradas alternativas.	Executar cenários normal, vazio e inválido separadamente.

Condição para avançar

Podemos iniciar a implementação quando conseguimos fornecer uma entrada de exemplo, indicar o resultado esperado e explicar os limites da primeira entrega. Isso autoriza construir; não autoriza declarar o produto terminado. Se ainda não sabemos se serviços inativos podem aparecer, precisamos decidir essa regra antes de aceitar a listagem.

Exercício com solução comentada

Considere três entradas de estudo: A contém c01 ativo e c02 inativo; B é uma lista vazia; C contém um registro sem o campo necessário para indicar sua situação. Escreva o resultado esperado para cada entrada e diga qual evidência não bastaria para aceitar o trabalho.

Solução: A deve apresentar somente c01; B deve informar ausência de serviços; C deve receber tratamento explícito de dado inválido, conforme o contrato. Uma captura de A não comprova B nem C, porque mostra somente uma execução. A resposta está completa quando distingue os três estados e propõe observar cada um. Se sua solução apaga c02 da fonte sem autorização, você alterou o acervo para esconder a regra de apresentação: o exercício pede seleção, não exclusão do registro.

Apoio de leitura: S01, SWEBOK, área Software Requirements, para a relação entre necessidade, especificação e validação de requisitos.

Referências S01 Detalhes e limites na seção de referências.

Capítulo 08 A identidade da Engenharia Caracol

O que a metáfora organiza

A Engenharia Caracol é a proposta de organização do trabalho apresentada por Graciliano Tolentino. Ela combina compreensão, estrutura, ciclos, autoridades, decisões de passagem, auditoria e continuidade entre pessoas e ferramentas. A imagem do caracol representa avançar levando consigo uma estrutura que preserva orientação e memória. Não transforma demora em indicador de qualidade.

Uma decisão de passagem, chamada gate, examina se existe base suficiente para a próxima ação. Uma auditoria confronta afirmações com evidências. Continuidade é a capacidade de outra pessoa retomar o trabalho sem reconstruir tudo pela conversa. Esses termos serão aprofundados; por enquanto, observe que cada um responde a um problema diferente. Um arquivo organizado pode ajudar a continuidade e ainda conter uma regra errada.

Requisitos, testes, controle de versões e registros de decisão arquitetural são práticas anteriores ao método. A contribuição proposta está na forma de articulá-las operacionalmente. Essa descrição preserva a autoria da proposta sem atribuir ao autor a invenção de todas as práticas utilizadas ou afirmar validação científica da combinação.

Três compromissos para um projeto pequeno

Compromisso	Aplicação na Oficina Caracol	Excesso a evitar
Preservar orientação	O README indica como iniciar e onde estão requisitos e dados.	Copiar o mesmo texto em todos os diretórios.
Verificar uma capacidade	A listagem deve ocultar serviços inativos e tratar a ausência de resultados.	Acumular documentos antes de observar a primeira listagem.
Deixar uma retomada possível	A anotação final informa mudança, evidência e pendência.	Narrar cada clique sem explicar o que falta decidir.

A Oficina Caracol usa o perfil Essencial: contexto didático, dados inventados e mudanças locais reversíveis. Essencial não significa dispensar os requisitos. Significa aplicar controles pequenos, suficientes para reconhecer um erro e recuperar o estado anterior. Uma aplicação com dados sensíveis ou alto custo de falha exigiria outra análise, pessoas habilitadas e controles adicionais.

Um ciclo contado do começo ao retorno

Suponha que a primeira versão de estudo mostre todos os serviços. O requisito REQ-002 exige ocultar inativos. A necessidade abre uma unidade de trabalho: ajustar a seleção, verificar uma lista mista e confirmar o efeito na página. O registro esperado liga requisito, arquivo alterado, entrada usada e resultado observado. Estes são passos de uma demonstração; os resultados efetivos devem ser preenchidos depois da execução.

Se a seleção funcionar isoladamente e a página continuar exibindo uma lista antiga, o trabalho retorna à integração. Não precisamos abandonar todo o projeto. Investigamos o caminho entre a fonte e a tela,

corrigimos o consumidor e repetimos a verificação afetada. O retorno tem endereço: a evidência falhou na integração, não na formulação da necessidade.

- Escolha uma capacidade reconhecível pelo usuário.
- Defina o que precisa ser verdadeiro para demonstrá-la.
- Produza e verifique a menor mudança suficiente.
- Decida se pode seguir ou se precisa retornar a uma dependência.
- Registre o ponto de retomada, sem atribuir à anotação o valor de um teste.

Promessas e limites

O método não garante ausência de defeitos nem transforma assistência de inteligência artificial em responsabilidade institucional. Uma IA pode propor código, casos de teste e explicações; pessoas continuam responsáveis por aceitar decisões e verificar seu alcance. A formação de competência depende de prática, retorno e avaliação, não do número de páginas lidas.

Como referências de apoio, o SWEBOK organiza áreas consolidadas de engenharia de software; Pro Git apresenta controle de versões; a orientação sobre ADRs explica registros de decisões arquiteturais. Essas fontes apoiam práticas específicas. Sua presença na bibliografia não representa endosso externo da Engenharia Caracol.

Erros de identidade e critério de avanço

Desvio	Correção específica
Chamar qualquer checklist de Caracol	Mostrar qual risco o item controla e qual decisão ele permite.
Produzir documentos para parecer rigoroso	Eliminar duplicações e conservar somente registros úteis à ação e à continuidade.
Prometer que a IA aprova com segurança	Reservar a aceitação a uma autoridade humana e apresentar evidências verificáveis.

A explicação do método está pronta para orientar um primeiro projeto quando apresenta uma capacidade, uma forma de verificar e um retorno possível, além de declarar seus limites. A quantidade de artefatos, sozinha, não atende a esse critério.

Exercício de identificação

Duas equipes dizem usar o método. A equipe A cria vinte modelos antes de conhecer o catálogo. A equipe B registra REQ-002, demonstra a seleção de ativos, identifica uma falha de integração e anota a próxima verificação. Qual delas apresenta um uso operacional mais convincente? O que ainda não podemos concluir?

Resposta comentada: B apresenta uma ligação entre necessidade, ação, evidência e retorno. A pode ter preparado materiais úteis, mas ainda não demonstrou a finalidade deles. Não podemos concluir que B entregou todo o sistema ou que suas verificações são suficientes para todos os riscos. Uma boa resposta justifica a escolha pelo encadeamento observado e mantém esses limites.

Para situar os antecedentes, consulte S01, áreas Software Requirements e Software Configuration Management; S07, capítulo Git Basics; e S08, orientação sobre registros de decisão arquitetural. O apoio dessas práticas não constitui avaliação externa do método.

Referências S01 S07 S08 Detalhes e limites na seção de referências.

Capítulo 09 Papéis e autoridades

Quem faz não responde sozinho por tudo

Em uma equipe pequena, a mesma pessoa pode conversar com o usuário, escrever código e preparar a publicação. Acumular funções é frequente. O problema aparece quando essa concentração apaga perguntas diferentes: a solução tem valor? Está tecnicamente correta? As verificações são suficientes? Quem pode aceitar o risco que resta?

Papel é uma responsabilidade assumida em determinada decisão. Autoridade é o direito reconhecido de decidir dentro de limites. A pessoa que implementa pode escolher o nome de uma variável sem poder decidir coletar dados pessoais. A mudança de finalidade pertence a outra esfera de responsabilidade. Precisamos explicitar esses limites antes que a dúvida chegue ao desenvolvedor iniciante.

Usaremos três personagens didáticos: Lia representa o produto; Rui implementa; Bia verifica. Eles não são pessoas que tenham revisado esta edição. A inteligência artificial aparece como ferramenta de apoio à preparação e análise, sem assumir a aceitação humana do produto.

Uma matriz RACI preenchida

RACI é uma tabela de distribuição de responsabilidades. R executa a atividade; A responde pela decisão final; C é consultado antes dela; I recebe informação. Neste exemplo, cada linha tem exatamente um A para evitar duas autoridades finais contraditórias. A mesma pessoa pode acumular R e A, mas isso torna visível a concentração, não cria revisão independente.

Atividade	Lia produto	Rui desenvolvimento	Bia verificação
Definir se a reserva é apenas simulada	A/R	C	C
Implementar seleção de serviços ativos	I	A/R	C
Preparar e executar verificação de REQ-002	C	C	A/R
Aceitar a versão didática e seus limites	A	R	C
Autorizar coleta de dado pessoal	A	C	C

A última linha identifica uma decisão reservada, mas o escopo atual já exclui dados pessoais. Nenhum participante pode iniciar essa coleta como detalhe de implementação. Seria necessário propor mudança de escopo, analisar consequências e obter uma decisão explícita; até lá, a resposta operacional é não implementar essa coleta.

O caminho de uma decisão

Rui encontra um formulário que pede nome para simular uma reserva. Ele não substitui o nome por um identificador inventado e declara o problema resolvido sem examinar o fluxo. Primeiro identifica o

requisito: a simulação não exige identificar pessoas. Em seguida propõe remover a solicitação. Bia verifica se a operação ainda demonstra repetição com efeito único. Lia decide sobre a adequação ao objetivo didático.

- Liste as decisões que podem bloquear o trabalho.
- Atribua a cada decisão uma autoridade e um substituto, quando houver equipe para isso.
- Declare o que o executor pode resolver e o que precisa escalar.
- Defina quem verifica os requisitos e como registra divergências.
- Vincule a aceitação da versão às evidências disponíveis e aos riscos residuais classificados.

Quando existe apenas uma pessoa

No estudo individual, você pode ocupar os três papéis em momentos distintos. Escreva o resultado esperado antes de executar; faça a implementação; depois compare o observado com a expectativa registrada. Use testes com entradas adversas e procure revisão externa quando o risco justificar. Essas medidas reduzem alguns vieses, mas não equivalem a uma segunda pessoa independente.

Situação	Reparo que corresponde ao problema
A IA diz que seu código está correto	Solicitar evidências e executar as verificações; uma afirmação textual não aprova a entrega.
Duas pessoas mudam o mesmo critério	Identificar o A daquela decisão e resolver a divergência no requisito oficial.
O iniciante recebe uma decisão reservada sem orientação	Interromper a mudança afetada e apresentar alternativas à autoridade definida.
Ninguém pode aceitar risco residual	Manter a passagem bloqueada até haver autoridade legítima, sem delegá-la à ferramenta.

Condição de avanço e exercício resolvido

O trabalho pode começar quando as decisões necessárias têm responsáveis identificados e o executor sabe quais dúvidas escalar. A publicação só pode avançar quando a autoridade correspondente considera as evidências e aceita os limites da versão. Preencher a RACI ajuda a organizar essa decisão; não a substitui.

Exercício: no cenário, Lia pede a versão completa; Rui concluiu testes automáticos; a navegação por teclado de REQ-004 ainda não foi examinada. Atribua executar, verificar e aceitar. A versão pode ser liberada com base apenas no relato de Rui?

Solução: Rui pode preparar e executar a aplicação; Bia realiza a verificação manual acordada; Lia responde pela aceitação do produto. Não existe base para declarar REQ-004 atendido somente porque os testes automáticos passaram. É preciso executar a avaliação aplicável e tratar o resultado. Se houver impedimento real de acessibilidade para a capacidade prevista, a autoridade não deve convertê-lo silenciosamente em pendência editorial. A resposta correta distingue verificação de aceitação e identifica a evidência ausente.

A referência S03, SSDF, prática PO.2 sobre papéis e responsabilidades, apoia a atribuição explícita de responsabilidades. S01, área Software Engineering Management, oferece o contexto de organização do trabalho; a matriz deste capítulo é uma aplicação didática.

Referências S01 S03 Detalhes e limites na seção de referências.

Capítulo 10 Linguagem natural estruturada

Palavras também definem interfaces

O pedido melhore o catálogo transfere para o executor decisões que o solicitante talvez não pretendesse delegar. Melhorar pode significar mudar cores, reordenar serviços ou ocultar os inativos. Duas implementações podem parecer razoáveis e produzir efeitos incompatíveis. O primeiro reparo é formular a transformação desejada, em vez de multiplicar adjetivos.

Uma instrução estruturada reúne contexto, objetivo, fonte, restrições, entregáveis e aceites. Aceite é a condição observável usada para decidir se o resultado atende ao combinado. Entregável é o objeto que será examinado, como uma alteração de arquivo ou um registro de verificação. Informar um formato de entrega não basta: um relatório pode estar bem organizado e descrever um comportamento errado.

O contrato operacional precisa permitir perguntas. Se um dado necessário não existe, a instrução deve orientar a registrar a lacuna. Caso contrário, o executor pode inventar arquivos, regras ou resultados para completar uma narrativa aparentemente coerente.

Uma instrução que permite conferir o resultado

Este modelo preenchido solicita uma unidade da Oficina Caracol. Os caminhos são os do projeto contínuo. O modelo não informa que a implementação já ocorreu.

Campo	Conteúdo do pedido
Contexto	O catálogo usa dados de exemplo em pratica/data/casos.json.
Objetivo	Atender REQ-002, evitando exposição de serviços inativos.
Fonte	Requisitos em pratica/docs/requisitos.md e contrato de dados do pacote.
Mudança permitida	Ajustar a seleção em pratica/src/catalogo.mjs e os testes correspondentes.
Limites	Não acrescentar dependências, cadastro ou dados pessoais; não apagar registros para esconder inativos.
Aceite	Com entrada mista, a saída pública apresenta somente ativos; vazio e inválido continuam distintos.
Verificação	Executar npm test e npm run validate dentro de pratica; registrar resultados reais e falhas.
Entrega	Arquivos alterados, justificativa da regra, evidências e pendências.

A fonte de dados define o material de entrada, mas não autoriza expor toda informação nela existente. O requisito de saída pública continua valendo. Essa distinção impede que o executor use a frase está no JSON como justificativa para revelar registros inativos.

Revisão do pedido antes de executar

- Troque palavras vagas por condições: profissional não é um aceite; ocultar serviços inativos é verificável.
- Leia a instrução como se fosse executar sem conhecer a conversa anterior. Localize a fonte de cada regra necessária.
- Produza duas interpretações plausíveis e compare o comportamento esperado.
- Resolva divergências que alterem requisito, dado público ou mudança permitida. Deixe decisões cosméticas livres quando não houver restrição.
- Confirme os comandos no README e em package.json. Uma instrução não deve ordenar um comando inexistente e tratar sua saída esperada como resultado real.

O teste das duas interpretações

Executora A entende ocultar como não desenhar o cartão na página, mas entrega todos os registros ao navegador. Executor B entende que a resposta pública deve omitir os inativos. A diferença é impeditiva quando REQ-002 proíbe revelar esses registros: é preciso esclarecer a fronteira pública e verificar a resposta, além da tela. Já usar margens de 12 ou 16 pixels pode ser aceitável se o pedido não fixa essa medida e a navegação permanece utilizável.

Defeito na instrução	Correção direcionada
Não informa o estado inicial	Anexar ou indicar a versão e os arquivos existentes.
Mistura expectativa com resultado	Separar deve apresentar de apresentou na execução registrada.
Proíbe qualquer pergunta	Definir como relatar lacunas e quais decisões exigem esclarecimento.
Pede prova sem identificar o requisito	Associar cada evidência à afirmação que pretende sustentar.

Condição de avanço

O pedido está pronto quando o executor consegue apontar fonte, mudança permitida e resultado verificável sem inventar regra relevante. Faça uma revisão de interpretação com exemplos. Uma divergência de gosto não bloqueia automaticamente o trabalho; uma divergência sobre quais dados podem sair bloqueia a unidade afetada até decisão.

Exercício com resposta

Reescreva: deixe o catálogo seguro e completo. Dados fornecidos: não há login; existem serviços ativos e inativos; o projeto não usa dependências externas; o objetivo desta unidade é somente REQ-002. Diga também uma ação que ficaria fora da solicitação.

Resposta possível: ajuste a seleção do catálogo para que a saída pública não inclua registros inativos; preserve o contrato de dados e os estados vazio e inválido; altere somente o módulo de catálogo e seus testes; execute os comandos de verificação oficiais e relate os resultados efetivos. Criar um sistema de login ficaria fora da unidade. A solução vale se restringe o objetivo, explicita a fronteira de exposição e permite conferir o resultado. Chamar o sistema inteiro de seguro ou completo continuaria sem evidência suficiente.

Leitura de apoio: S01, Software Requirements, temas de especificação e validação de requisitos. A instrução preenchida aplica esses princípios à unidade concreta; não é uma norma universal de redação de prompts.

Referências S01 Detalhes e limites na seção de referências.

Parte II Compreensão e decisão

Conhecer problema escopo estado e alternativas

Capítulo 11 Compreender profundamente o problema

Investigar sem converter hipótese em fato

O responsável pela Oficina Caracol diz: precisamos de um assistente automático porque as pessoas perguntam sempre as mesmas coisas. Essa frase contém um sintoma e uma solução preferida. Ainda não explica por que as perguntas se repetem, quais informações faltam nem se um catálogo claro resolveria parte da necessidade. Investigar é reduzir essas incertezas antes de assumir o custo de uma solução.

Fato é uma observação delimitada por origem e contexto. Hipótese é uma explicação ainda a verificar. Preferência é uma escolha desejada por alguém. Os três tipos podem ser úteis, mas não têm o mesmo valor como evidência. A frase visitantes não entendem nada transforma uma interpretação ampla em diagnóstico; a frase três participantes não localizaram o título na tarefa observada permite examinar algo concreto.

Os números do exemplo a seguir são inteiramente simulados para ensinar o raciocínio. Não representam pesquisa realizada com usuários deste livro. Mesmo uma observação real de poucas pessoas orientaria uma investigação local, sem provar automaticamente o comportamento de todo o público.

Do relato ao mapa de perguntas

Informação fornecida no cenário	Classificação	Próximo passo
Quatro de seis dúvidas simuladas pedem quais serviços estão disponíveis.	Dado do exercício, com universo de seis ocorrências.	Verificar se a disponibilidade está clara no catálogo.
O visitante não lê porque prefere conversar.	Hipótese.	Observar uma tarefa antes de atribuir a causa à preferência.
O patrocinador deseja um assistente com animação.	Preferência.	Comparar custo e benefício com a melhoria do conteúdo.
Um serviço inativo aparece na listagem atual do cenário.	Defeito observado no exemplo.	Reproduzir com entrada conhecida e associar a REQ-002.

Não precisamos escolher entre ouvir a pessoa e verificar o sistema. A entrevista ajuda a descobrir situações; a observação permite confrontar o relato com uma tarefa; dados de execução ajudam a delimitar o comportamento técnico. Quando as fontes divergem, registramos a divergência em vez de apagar a informação menos conveniente.

Procedimento de descoberta proporcional

- Defina uma pergunta de decisão: a disponibilidade dos serviços está compreensível o suficiente para dispensar uma explicação inicial?

- Peça que a pessoa descreva a última situação concreta. Pergunte o que tentou fazer, onde parou e qual consequência ocorreu.
- Observe uma tarefa curta com dados fictícios, sem orientar cada clique. Registre somente o que é necessário ao estudo e obtenha autorização para eventual registro com pessoas.
- Separe frequência de gravidade. Uma falha rara que expõe informação indevida pode ser prioritária.
- Formule uma hipótese testável e escolha uma intervenção pequena. A primeira tentativa pode ser corrigir a seleção e a mensagem de vazio.
- Defina quando parar a análise: existe informação suficiente para uma experiência reversível, ou falta uma decisão que tornaria a experiência inadequada?

Uma formulação de problema preenchida

Registro de estudo em pratica/docs/problema.md: visitantes precisam identificar serviços disponíveis sem fornecer dados pessoais. No cenário fornecido, a listagem mistura ativos e inativos e não explica a ausência de resultados. A primeira intervenção será apresentar somente ativos e tornar explícitos os estados vazio e inválido. A hipótese é que a informação inicial ficará mais clara; sua utilidade para pessoas reais ainda precisará ser avaliada.

Observe as fronteiras dessa formulação. A mistura de estados está no cenário. A melhora de compreensão é uma hipótese. A decisão de não coletar dados pessoais é uma restrição do projeto. Nenhuma dessas frases exige construir um assistente antes de experimentar uma resposta menor.

Erros de investigação e avanço

Erro	Reparo
Perguntar você gostaria de um catálogo melhor	Pedir que a pessoa realize uma tarefa e localizar a dificuldade concreta.
Ouvir só quem financia	Incluir a perspectiva de quem usa e de quem mantém, conforme o contexto.
Continuar pesquisando sem decisão	Relacionar cada pergunta à escolha que ela pode mudar.
Tratar baixa frequência como baixo impacto	Examinar consequência, reversibilidade e exposição de dados.

Podemos avançar para o recorte da versão quando a necessidade, o público e as incertezas relevantes estão explícitos e existe uma intervenção justificável. Não precisamos conhecer todas as causas; precisamos saber o suficiente para escolher um passo responsável e reconhecer o que esse passo poderá demonstrar.

Exercício resolvido

Dados: entre seis dúvidas simuladas, quatro perguntam disponibilidade, uma pergunta preço e uma pede ajuda para usar teclado. O responsável conclui: precisamos de chat, e o teclado pode esperar porque só apareceu uma vez. Classifique as duas conclusões e proponha a primeira investigação.

Solução: a necessidade de chat é uma hipótese de solução, não consequência obrigatória da contagem. Descartar teclado somente pela frequência ignora o impedimento de acesso e REQ-004. A primeira investigação pode confrontar a listagem com ativos e inativos e executar a tarefa por teclado. A resposta deve conservar o universo de seis ocorrências, evitar generalização estatística e propor observações que alterem a decisão. Não é preciso prometer que corrigir o catálogo eliminará todas as dúvidas.

Aprofundamento: S01, Software Requirements, tema Requirements Elicitation. O exercício utiliza dados simulados para praticar descoberta e formulação, sem pretender substituir uma investigação com usuários.

Referências S01 Detalhes e limites na seção de referências.

Capítulo 12 Escopo e critérios de aceite

Uma versão precisa ter fronteiras

Escopo é o acordo sobre o que uma versão deverá oferecer e sobre o que ficará fora dela. Serve para decidir o trabalho, não apenas descrevê-lo depois. Na Oficina Caracol, dizer um catálogo com reservas pode sugerir pagamento, agenda e confirmação de atendimento. O projeto didático oferece outra coisa: catálogo público e uma operação simulada para estudar repetição de solicitações com efeito único.

A diferença deve aparecer no objetivo da versão e na interface. A simulação usa memória do processo; reiniciá-lo apaga as reservas. Não há compromisso real de disponibilidade, autenticação ou atendimento. Explicitar essas exclusões evita que uma demonstração técnica seja aceita como serviço operacional.

Um critério de aceite relaciona uma condição de entrada a um resultado observável. Não precisa prescrever todos os detalhes internos. Exigir o uso de determinada variável raramente demonstra valor para o visitante; exigir que um serviço inativo não apareça na resposta pública demonstra uma regra relevante.

Acordo da versão didática completa

Requisito	Cenário de aceite	Evidência aplicável
REQ-001	Uma entrada válida com serviços ativos produz a listagem pública desses serviços.	Verificação da seleção e do caminho até a página.
REQ-002	Uma entrada mista não revela registros inativos na resposta pública nem na interface.	Inspeção da resposta e verificação do consumidor.
REQ-003	Lista válida vazia e catálogo inválido geram respostas explícitas e diferentes.	Casos de teste e observação dos estados da interface.
REQ-004	A tarefa prevista pode ser percorrida por teclado, com foco e feedback perceptíveis.	Registro de verificação manual, sem declarar conformidade automática.
REQ-005	Repetir a mesma chave para o mesmo caso mantém um único efeito; reutilizá-la para outro caso gera conflito.	Teste do registrador e do caminho HTTP da simulação.

O primeiro incremento pode demonstrar o catálogo; outro integra a simulação. Um incremento é uma parcela utilizável ou verificável do trabalho. A conclusão do primeiro não autoriza anunciar a versão completa como aceita. Os cinco requisitos continuam no acordo final, mesmo quando são implementados em momentos distintos.

Procedimento para negociar uma mudança

- Escreva o objetivo da versão em uma frase que mencione seu caráter didático.
- Associe cada capacidade obrigatória a entradas, resultados e verificação.
- Registre exclusões: cadastro, dados pessoais reais, banco externo, pagamentos e reserva efetiva.
- Ao receber um pedido novo, identifique quais requisitos, dados, interface e verificações ele afeta.

- Decida incluir agora, trocar por outro item ou adiar. Registre a decisão com a autoridade do produto, sem esconder a troca dentro de uma tarefa técnica.

Um pedido adicional analisado

O solicitante pede enviar a confirmação para o e-mail do visitante. Parece um pequeno campo, mas introduz coleta de dado pessoal e integração externa. Ambos estão excluídos. A decisão coerente para esta edição é adiar a função e manter a confirmação simulada na própria interface. Uma versão futura precisaria rediscutir finalidade, tratamento de dados, segurança e operação antes de assumir essa capacidade.

Compare com corrigir a mensagem que aparece quando não há serviços. Essa correção ajuda a cumprir REQ-003 sem acrescentar uma nova finalidade. A análise de impacto permite diferenciar expansão de escopo e reparo de um requisito já assumido.

Falhas de aceite e condição de avanço

Falha	Reparo
Aceitar porque todos os comandos terminaram	Examinar se os testes cobrem os requisitos e realizar a verificação manual aplicável.
Chamar toda falha de ressalva	Classificar impacto; requisito essencial descumprido impede a liberação da capacidade afetada.
Marcar tarefa técnica como requisito atendido	Confrontar o comportamento integrado com o cenário de aceite.

O recorte está pronto para execução quando os requisitos obrigatórios, as exclusões e os cenários de verificação estão acordados. A liberação da versão exige resultado aceitável, classificação das lacunas não impeditivas e aceitação do risco pela autoridade competente. Apenas registrar que uma falha existe não a torna aceitável.

Exercício com decisão resolvida

Cenário: listagem, ocultação e mensagens foram demonstradas; a simulação repete corretamente uma chave; teclado ainda não foi verificado. O responsável pede liberar a versão completa e incluir e-mail no mesmo dia. Defina o que já pode ser afirmado e as próximas ações.

Solução: podemos afirmar os resultados efetivamente demonstrados e seu alcance, sem estendê-los a REQ-004. Primeiro devemos executar a verificação manual de teclado e feedback, corrigindo impedimentos encontrados. O pedido de e-mail deve ser tratado como mudança de escopo e, no contrato atual, adiado. A resposta não deve declarar a simulação pronta para reservas reais: seu estado permanece local e transitório. Uma solução completa separa estado da entrega, evidência ausente e pedido novo.

Consulte S01, Software Requirements, para especificação e validação; e S06, WCAG 2.2, critérios 2.1.1 Keyboard, 2.4.7 Focus Visible e 4.1.3 Status Messages. Esses critérios orientam verificações pertinentes à oficina e não representam toda a avaliação de acessibilidade.

Referências S01 S06 Detalhes e limites na seção de referências.

Capítulo 13 Inventário do estado atual

Existir é diferente de funcionar

Você recebe uma pasta chamada projeto final. Dentro dela há arquivos, imagens e um README. O nome da pasta não comprova conclusão, assim como encontrar um arquivo de teste não comprova que ele passou. Inventariar é construir uma fotografia rastreável do que existe, do que foi examinado e do que permanece desconhecido.

O inventário relaciona artefatos, responsabilidades e observações. Artefato é um resultado identificável de trabalho, como um arquivo de dados, um módulo ou um registro de decisão. Runtime é o ambiente que executa o programa; neste projeto, usamos Node para a parte local e um navegador para a interface. A versão realmente utilizada deve ser registrada a partir da execução, sem copiar um número ilustrativo do livro.

Uma fotografia precisa de referência: qual cópia, qual revisão e em que ambiente foi examinada? Controle de versões pode identificar a revisão; uma soma de verificação pode identificar um arquivo recebido. Esses identificadores ajudam a conferir a origem, mas não demonstram que o conteúdo é seguro ou correto.

Inspeção antes de execução

- Preserve o material recebido e trabalhe numa cópia separada. Não apague o original para organizar a análise.
- Liste arquivos e examine README, package.json e o código relevante antes de executar comandos do projeto. Scripts podem realizar ações além do que seu nome sugere.
- Se houver ZIP, confira as entradas antes de extrair: destinos não devem escapar da pasta de trabalho nem substituir arquivos externos; examine links e evite executar conteúdo desconhecido.
- Identifique os caminhos oficiais e separe presença, inspeção textual e execução verificada.
- Depois de entender o pacote de estudo, registre node --version e execute os comandos oficiais necessários ao diagnóstico. Preserve os resultados reais, inclusive falhas.
- Relacione cada lacuna ao impacto na decisão seguinte. Um arquivo desconhecido não pode ser removido só por não ter finalidade evidente.

Inventário preenchido de um cenário

A tabela fornece observações hipotéticas para praticar a classificação. Não relata uma execução desta edição. Caminhos são relativos à pasta pratica; a revisão e a versão do runtime deverão receber os valores observados pelo aluno.

Artefato	Responsabilidade	Observação fornecida	Estado da evidência
README.md	Orientar a execução.	Arquivo aberto e instruções localizadas.	Inspeção textual; comandos ainda não executados.
data/casos.json	Fonte dos registros de exemplo.	Array com id, titulo, descricao e ativo.	Formato identificado; validação ainda pendente.

Artefato	Responsabilidade	Observação fornecida	Estado da evidência
src/catalogo.mjs	Validar e selecionar registros públicos.	Exports localizados no código.	Implementação presente; correção não demonstrada.
src/servidor.mjs	Expor interface HTTP e arquivos permitidos.	Arquivo encontrado, porta ainda não aberta.	Comportamento desconhecido.
tests/ catalogo.test.mjs	Verificar regras de catálogo.	Há cenários de entrada mista.	Testes presentes; resultado não conhecido.
package-lock.json	Registrar o acordo de instalação do pacote.	Arquivo presente.	Consistência com package.json ainda a verificar.

O arquivo de lock registra informações da instalação e da resolução do pacote, mesmo que esta oficina não use dependências externas. Sua presença não é uma aprovação das bibliotecas nem substitui a leitura dos scripts. O inventário evita transformar um nome técnico em selo de confiança.

Da lacuna à decisão

Suponha agora que o enunciado forneça um teste aprovado de seleção e uma falha ao iniciar o servidor. Podemos planejar a investigação do servidor sem reconstruir a seleção. Ainda não podemos dizer que o visitante consulta o catálogo: a capacidade depende da integração. Registre a falha com comando, revisão, ambiente e mensagem pertinente; mantenha o estado integrado como não demonstrado.

Erro de inventário	Reparo específico
Marcar existe como funciona	Separar coluna de presença da evidência de comportamento.
Executar scripts desconhecidos imediatamente	Inspecionar o que será executado e usar ambiente isolado apropriado ao material.
Copiar registros pessoais para a análise	Usar os dados fictícios fornecidos e limitar informação ao necessário.
Ocultar lacunas no resumo	Indicar o que a lacuna impede decidir e qual verificação a encerra.

Condição para propor a alteração

O inventário é suficiente para uma mudança quando os artefatos afetados, as dependências importantes e as lacunas impeditivas estão identificados. Não é necessário conhecer cada byte para corrigir uma mensagem; é necessário conhecer o caminho dessa mensagem e os efeitos da alteração. Lacunas fora desse caminho podem permanecer registradas com justificativa.

Exercício com classificação

Recebemos as seguintes informações: data/casos.json foi localizado, npm test não foi executado e a página abriu, mas ninguém observou serviços inativos. Classifique a afirmação o catálogo está correto e escreva duas próximas verificações.

Solução: a afirmação excede a evidência. Há presença do dado e uma observação parcial da página. Devemos validar os registros e verificar uma entrada mista no caminho público, além de executar a suíte aplicável. A resposta precisa manter desconhecido o resultado ainda não observado. Encontrar um arquivo ou abrir uma página não permite preencher retroativamente uma coluna de teste aprovado.

Apoio: S07, Git Basics, seção Recording Changes to the Repository, para relacionar observações à revisão preservada; S14, documentação do comando npm ci, para entender instalação e scripts; e S19, opções de linha de comando do Node, para consultar a versão do runtime.

Referências S07 S14 S19 Detalhes e limites na seção de referências.

Capítulo 14 Preservar reparar refatorar ou reconstruir

Escolher a intervenção pelo efeito desejado

Ao encontrar um sistema existente, o programador pode sentir vontade de recomeçar para entender tudo. A familiaridade com a própria solução é atraente, mas não elimina o conhecimento acumulado no sistema recebido. A escolha da intervenção precisa comparar comportamento, risco, custo e possibilidade de recuperar a situação anterior.

Preservar é manter algo que continua adequado. Reparar corrige um defeito. Refatorar altera a estrutura interna conservando o comportamento pretendido. Migrar transfere uma responsabilidade para outro ambiente ou solução. Reconstruir substitui uma parte ou o conjunto. Essas ações podem coexistir: preservar dados, reparar seleção e refatorar a organização do código não exige escolher um único rótulo para toda a aplicação.

Refatoração não significa apenas melhorar o código. Se a mudança passa a ocultar registros antes visíveis, existe uma mudança de comportamento e ela precisa ser reconhecida. Isso não a torna inadequada; torna inadequado apresentá-la como alteração estrutural neutra.

O defeito do catálogo e as alternativas

No cenário de estudo, a interface é legível e os registros têm formato válido, mas a resposta pública inclui serviços inativos. O objetivo é cumprir REQ-002. Antes de alterar, fornecemos uma entrada mista e observamos a exposição indevida. Essa observação caracteriza o defeito atual; não o transforma em comportamento que deva ser preservado.

Alternativa	Benefício	Custo ou risco	Decisão do exemplo
Preservar tudo	Nenhuma mudança imediata.	Mantém o descumprimento de REQ-002.	Rejeitada para a seleção.
Reparar a seleção pública	Ataca diretamente o defeito.	Exige verificar consumidores e entrada inválida.	Escolhida para esta unidade.
Refatorar toda a interface	Pode melhorar organização.	Não é necessária para corrigir a exposição.	Adiada até existir motivo próprio.
Reconstruir com outra tecnologia	Permite nova estrutura.	Amplia superfície de mudança e revalidação.	Sem justificativa neste cenário.

Registro de decisão preenchido

ADR significa registro de decisão arquitetural. É um texto curto que conserva contexto, escolha, alternativas e consequências. A orientação da AWS sobre ADRs é referência para esse tipo de registro; a escolha concreta abaixo pertence ao cenário didático da oficina.

Campo do ADR-001	Preenchimento
Título	Reparar a seleção pública sem reconstruir a interface.

Campo do ADR-001	Preenchimento
Estado	Decisão adotada no cenário de estudo.
Contexto	REQ-002 é descumprido; dados válidos e apresentação podem ser preservados.
Decisão	Ajustar o módulo de catálogo para retornar somente os campos públicos dos ativos.
Alternativas consideradas	Manter defeito, refatorar interface inteira ou reconstruir.
Consequências	Rever seleção, consulta por identificador e integração HTTP; conservar dados de origem.
Autoridade	Papel de produto aceita o recorte; desenvolvimento responde pela solução técnica dentro dele.
Recuperação	Preservar a revisão anterior e retornar a ela se houver regressão; a versão anterior defeituosa não fica automaticamente liberada para exposição.
Reavaliação	Reabrir se o contrato de dados ou a finalidade pública mudar.

O ADR não comprova que o reparo foi executado. Ele conserva a razão da escolha. Testes e observações posteriores devem demonstrar o resultado. Essa separação impede que a elegância da justificativa ocupe o lugar da verificação.

Procedimento de intervenção

- Reproduza o defeito com uma entrada pequena e registre o resultado indevido.
- Identifique o comportamento que deve permanecer e a regra que precisa mudar.
- Compare intervenções pelo alcance necessário, não pela preferência pessoal de tecnologia.
- Registre a decisão e preserve uma forma adequada de recuperar a cópia de trabalho.
- Execute verificações que demonstrem tanto a correção quanto a preservação dos comportamentos relacionados.
- Investigue divergências antes de ampliar a mudança para outros módulos.

Erros de escolha e condição de avanço

Erro	Consequência	Reparo
Apagar serviços inativos do arquivo	Destrói informação para simular uma seleção correta.	Conservar a fonte e corrigir a saída pública.
Congelar o defeito em teste de caracterização	A suíte passa a exigir o erro.	Distinguir descrição do estado antigo de aceite da regra correta.
Reescrever para facilitar um ajuste pequeno	Introduz mudanças sem relação com a necessidade.	Limitar a intervenção e justificar separadamente nova arquitetura.

A intervenção pode começar quando alvo, alternativa escolhida, evidência de preservação e recuperação estão claros. A unidade só termina quando a seleção e seus consumidores atendem ao aceite. Voltar ao estado anterior é uma ação de recuperação; se aquele estado contém um defeito impeditivo, a exposição da capacidade precisa continuar bloqueada.

Exercício resolvido

Um módulo tem nomes confusos, mas todos os aceites atuais estão demonstrados. Outro módulo envia um serviço inativo na resposta pública. Há tempo para uma única intervenção. Qual tratar primeiro e qual rótulo usar?

Solução: o descumprimento de REQ-002 pede reparo prioritário. Renomear e reorganizar o módulo correto pode ser uma refatoração posterior, desde que conserve o comportamento. A resposta deve explicar que a prioridade vem do requisito descumprido e do impacto, não de um veto geral à refatoração. Se surgisse risco relevante no módulo aparentemente correto, a decisão teria de considerar essa nova evidência.

Fontes para aprofundar: S08, orientação de ADRs, para conservar contexto, decisão e consequências; S17, definição de refatoração em refactoring.com, para distinguir reorganização interna de mudança de comportamento observável.

Referências S08 S17 Detalhes e limites na seção de referências.

Capítulo 15 Fonte única de verdade

Autoridade por assunto

Há um preço no arquivo de dados, outro numa planilha e outro escrito diretamente na página. Qual deve orientar a correção? A fonte única de verdade é uma regra de autoridade para um domínio de informação. Domínio, neste contexto, é um assunto delimitado: requisitos, registros de catálogo, decisão arquitetural ou resultado de execução.

Não se trata de reunir tudo num arquivo enorme. Requisitos explicam o que o produto deve fazer; dados fornecem os registros; código realiza transformações; evidências mostram o que foi observado. São autoridades distintas para perguntas distintas. A implementação atual não pode revogar silenciosamente um requisito só porque já funciona de outra maneira.

Cópias podem ser necessárias. O navegador recebe uma representação pública dos casos e a usa para desenhar cartões. Essa representação é derivada: não deve virar um segundo cadastro editado manualmente. Uma cópia precisa de origem reconhecível e de uma política de atualização.

Mapa preenchido da Oficina Caracol

Domínio	Fonte responsável	Consumidor ou derivação	Regra de atualização
Necessidades e aceites	pratica/docs/requisitos.md	Testes e decisões de versão.	Mudança aprovada pelo papel de produto antes de alterar o acordo.
Registros de exemplo	pratica/data/casos.json	Módulo de catálogo.	Alterar a origem, validar e rever cenários afetados.
Contrato público	Exports documentados e interface HTTP acordada.	Interface do navegador e testes de integração.	Rever consumidores quando mudar entrada ou saída.
Comandos	pratica/package.json, explicados no README.	Execução local e instruções didáticas.	Conferir scripts e documentação na mesma mudança.
Decisões	Registro ADR em pratica/docs.	Desenvolvimento e manutenção.	Preservar histórico e explicitar a decisão que substitui outra.
Resultados observados	Registro de evidência com revisão e ambiente.	Gate e continuidade.	Acrescentar nova execução; não reescrever falha antiga como sucesso.

A tabela distribui autoridade sem afirmar que cada documento está sempre atualizado. Se a documentação de um contrato diverge do código, existe uma inconsistência a resolver; o mapa aponta os responsáveis e as verificações necessárias, não elimina a investigação.

Um conflito resolvido sem esconder a mudança

Considere este cenário fornecido: data/casos.json registra Oficina de testes como ativa; a página mostra um cartão antigo marcado como indisponível. A pessoa responsável pelo conteúdo confirma que o registro oficial está correto. A correção deve remover a cópia manual da interface e fazê-la consumir a

saída pública. Alterar o JSON para combinar com o cartão apenas faria a origem imitar uma cópia desatualizada.

Há outro conflito possível: o dado marca um serviço como inativo e a resposta pública o revela. Nesse caso, a origem não precisa ser corrigida para ativo. É a transformação que contradiz REQ-002. Identificar a pergunta é essencial: estamos investigando o estado do serviço ou o cumprimento da regra de publicação?

- Liste os assuntos que precisam de uma autoridade.
- Aponte a origem e quem pode alterá-la.
- Identifique cópias e consumidores que dependem dela.
- Defina como detectar inconsistências e quem decide o conflito.
- Ao mudar a origem ou o contrato, atualize as derivações e repita as verificações afetadas.

Dados públicos não são o arquivo inteiro

O contrato real do módulo usa registros com id, titulo, descricao e ativo. A função listarAtivos devolve somente os campos públicos id, titulo e descricao dos serviços ativos. A fonte pode conter serviços inativos para manter o acervo didático, mas eles não precisam chegar ao navegador. A seleção deve acontecer antes da resposta pública; esconder cartões com CSS não protege um dado já entregue.

Erro	Reparo dirigido
Editar uma lista duplicada na página	Consumir a representação produzida a partir da origem.
Escolher o arquivo mais recente como autoridade	Consultar o mapa e o histórico; data de alteração não concede autoridade.
Guardar resultado sem revisão	Associar evidência à versão que realmente foi executada.
Resolver divergência em silêncio	Registrar causa, decisão e consumidores afetados.

Condição de avanço e exercício

Uma mudança de informação pode avançar quando a origem legítima, o responsável e as derivações afetadas estão identificados. Ao encerrar, confirme que a interface representa a fonte através da regra correta. Não basta eliminar diferenças visuais: é preciso preservar a finalidade de cada domínio.

Exercício: o JSON mantém c02 inativo; um cartão escrito no HTML mostra c02; o README afirma que a página usa a API. Indique três perguntas de investigação e a correção mais provável, sem assumir que o README já prova o funcionamento.

Solução: perguntar de onde o cartão veio, se a resposta pública contém c02 e se o HTML conserva conteúdo manual. Se a API omite c02 e o cartão é manual, remover a duplicação e usar o consumidor previsto. Se a API revela c02, reparar também a seleção. O README precisa corresponder à execução verificada. A resposta correta não torna c02 ativo para encerrar a divergência nem chama a documentação de evidência suficiente da integração.

Consulte S01, Software Configuration Management, para identificação e controle da configuração; e S07, Git Basics, para preservar o histórico das alterações. O mapa de autoridades é a decisão local desta oficina.

Referências S01 S07 Detalhes e limites na seção de referências.

Capítulo 16 Responsabilidade única e coesão

Separar pelo motivo de mudança

Um arquivo curto pode misturar responsabilidades; um arquivo maior pode concentrar tarefas que pertencem ao mesmo propósito. Responsabilidade única significa organizar uma unidade em torno de uma razão principal de mudança. Coesão é a relação entre seus elementos: eles trabalham juntos para uma finalidade reconhecível. Contar linhas não responde a nenhuma dessas perguntas.

Na Oficina Caracol, a regra de quais casos são públicos pertence ao catálogo. O protocolo de receber uma solicitação HTTP pertence ao servidor. A apresentação do estado de carregamento pertence à interface. Se mudar a cor do botão obriga a editar a seleção de ativos, provavelmente criamos uma ligação que não corresponde ao problema.

Acoplamento é a dependência entre unidades. Alguma dependência é inevitável: a interface precisa conhecer o formato da resposta. O objetivo é tornar esse acordo explícito e limitado, para que uma alteração local não exija reconstruir o projeto inteiro.

Fronteiras justificadas do projeto

Unidade oficial	Responsabilidade	Mudança que justifica editá-la
data/casos.json	Guardar registros fictícios.	Alterar o conteúdo de um caso.
src/catalogo.mjs	Validar casos e produzir seleção pública.	Mudar o contrato ou a regra de seleção.
src/operacoes.mjs	Registrar a reserva simulada por chave.	Mudar a regra de repetição com efeito único.
src/servidor.mjs	Receber solicitações e ligar protocolo às operações.	Mudar o tratamento HTTP de uma resposta.
public/app.js	Ligar ações da página às respostas da API.	Mudar a interação e o feedback apresentado.
public/styles.css	Definir apresentação visual.	Ajustar contraste, espaçamento ou aparência do foco.

Validar registros e selecionar ativos podem permanecer no mesmo módulo de catálogo porque compartilham um contrato de dados e uma finalidade. Não precisamos criar um arquivo para cada campo. A separação também não permite repetir a regra de ativo no servidor e na interface: duplicar a decisão cria futuras divergências.

Uma verificação localizada da fronteira

O trecho abaixo é um ensaio isolado, apresentado para leitura como módulo executado a partir da pasta pratica. Ele usa a API acordada do pacote e dados inventados; não representa uma saída já obtida. A mesma intenção de verificação pertence à suíte do catálogo. Observe que a regra pode ser examinada sem abrir porta ou montar a página.

Código 16 javascript

```
import assert from 'node:assert/strict';
import { listarAtivos } from './src/catalogo.mjs';

const entrada = [
  { id: 'c01', titulo: 'Revisão', descricao: 'Caso fictício ativo', ativo: true },
  { id: 'c02', titulo: 'Arquivo', descricao: 'Caso fictício inativo', ativo: false }
];
const anterior = structuredClone(entrada);
const saida = listarAtivos(entrada);
assert.deepEqual(saida, [
  { id: 'c01', titulo: 'Revisão', descricao: 'Caso fictício ativo' }
]);
assert.deepEqual(entrada, anterior);
```

A primeira comparação verifica a representação pública. A segunda verifica que a seleção não alterou a entrada. Esse ensaio não comprova que a API ou a página usam a função corretamente; essas relações precisam de verificações de integração. Uma boa fronteira torna a regra examinável, mas não elimina a prova do conjunto.

Procedimento para melhorar a organização

- Descreva a finalidade de cada unidade em uma frase sem usar e para reunir tarefas independentes.
- Liste três mudanças plausíveis e indique onde cada uma deveria ocorrer.
- Examine se regras iguais foram copiadas em lugares diferentes. Escolha o módulo responsável e faça os consumidores usarem seu contrato.
- Conserve juntos elementos que mudam pela mesma razão; reúna microarquivos que apenas obrigam o leitor a saltar entre pastas.
- Depois de reorganizar, verifique comportamento local e integração. Uma refatoração não deve alterar o contrato sem que isso seja decidido e registrado.

Erros e condição de avanço

Erro	Mecanismo do problema	Reparo
Tudo em um arquivo de utilidades	Finalidades diferentes não têm endereço reconhecível.	Mover funções para o domínio cuja regra representam.
Um arquivo por propriedade	Uma mudança de contrato exige muitos saltos artificiais.	Agrupar validações relacionadas ao registro.
Interface filtra inativos por conta própria	Servidor pode continuar revelando informação indevida.	Centralizar a seleção antes da fronteira pública.

A organização é suficiente para avançar quando podemos localizar a regra, modificá-la e verificar seus consumidores sem alterações sem relação com o objetivo. Não exigimos o menor arquivo possível; exigimos uma divisão que favoreça compreensão, teste e mudança responsável.

Exercício com fronteiras resolvidas

Um único arquivo lê dados, escolhe serviços ativos, abre a porta HTTP, desenha mensagens e define cores. Proponha uma separação usando os caminhos oficiais. Depois diga onde alterar somente o título de c01 e onde tratar um conflito de chave de reserva.

Solução: manter dados em data/casos.json; regras de catálogo em src/catalogo.mjs; regras de reserva em src/operacoes.mjs; protocolo em src/servidor.mjs; interação em public/app.js; apresentação em public/styles.css. O título muda na origem de dados, com validação posterior. O conflito nasce na regra de reservas e recebe tradução HTTP no servidor. Uma boa resposta reconhece essa colaboração sem duplicar a regra nos dois módulos.

Apoio: S01, Software Design, para organização de responsabilidades; S17, definição de refatoração; e S19, documentação de ECMAScript modules e node:assert. O exemplo verifica uma fronteira do pacote, sem representar aprovação da aplicação inteira.

Referências S01 S17 S19 Detalhes e limites na seção de referências.

Parte III Fluxo maturidade e gates

Organizar ciclos provas e retorno

Capítulo 17 Waterfall e Caracol

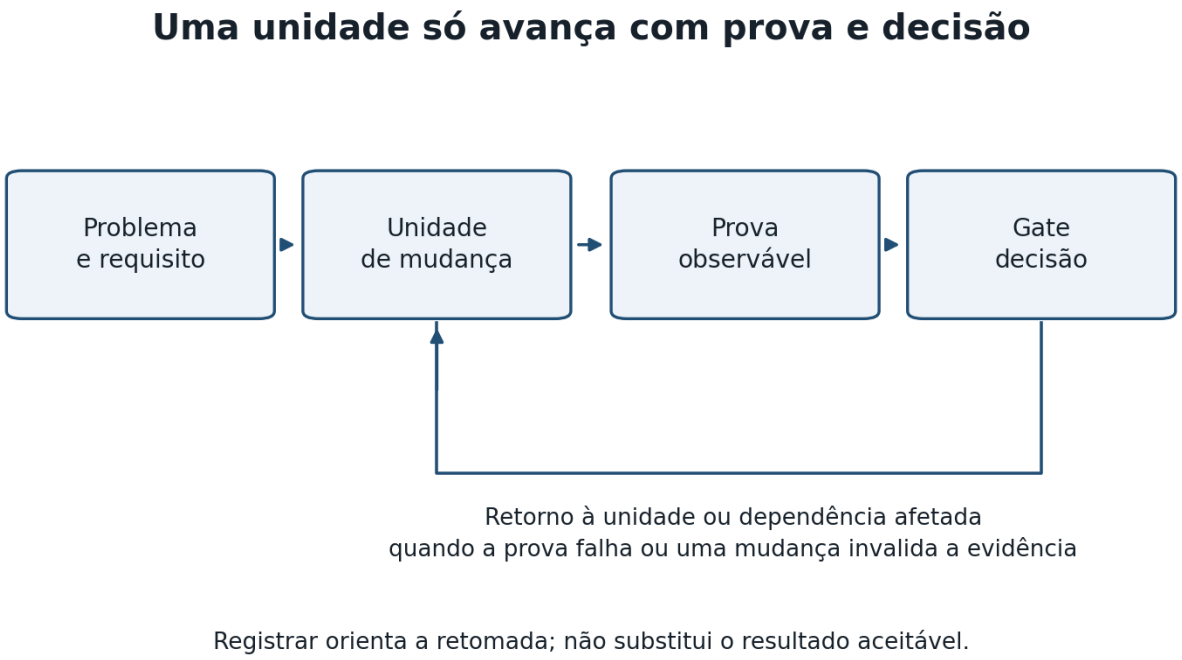


Figura 6 Uma unidade só avança com prova e decisão

O retorno tem uma causa e um alvo. Um gate não é apenas um formulário preenchido: precisa produzir uma decisão sustentada pelas evidências disponíveis.

Ordem necessária e aprendizagem

A Oficina Caracol precisa listar serviços ativos. Antes de decidir se a listagem funciona, precisamos saber o que significa ativo e qual resultado caracteriza uma listagem correta. Essa dependência de conhecimento não determina, por si só, que todo o projeto deva ser especificado antes de qualquer implementação. Ela apenas impede uma aprovação sem critério.

Waterfall, ou cascata, designa uma organização de atividades em fases predominantemente sequenciais. A discussão de Royce em S11 apresenta etapas e também trata realimentação e riscos; não deve ser reduzida ao slogan de que voltar é proibido. Uma sequência pode organizar responsabilidades e revisões, mas mudanças posteriores exigem tratamento explícito.

A Engenharia Caracol combina uma direção de avanço com retornos delimitados por evidências. Seu nome não transforma qualquer mudança em melhoria: o retorno precisa identificar o fato novo, os artefatos afetados e a prova que perdeu validade. O objetivo é aprender sem apagar o que continua sustentado. O Manifesto Ágil, em S12, valoriza responder a mudanças; ele não dispensa critérios de aceite nem demonstrações de funcionamento.

Procedimento para uma mudança

- Declare a capacidade do incremento e seus critérios de aceite antes de autorizar sua entrega.
- Desenhe dependências entre decisões, dados, implementação e verificações. Diferencie dependência real de preferência de trabalho.
- Ao receber informação nova, compare-a com os critérios vigentes e registre o conflito concreto.
- Reabra os itens cuja evidência foi afetada, incluindo consumidores indiretos. Preserve os demais com justificativa.
- Depois da correção, execute novamente as verificações pertinentes e emita outro gate, isto é, uma decisão registrada de passagem.

O mesmo incremento em dois percursos

Considere a capacidade de listar apenas casos ativos. O quadro compara duas organizações de trabalho para o mesmo escopo, com aprovação final obrigatória em ambas. A coluna orientada por fases é uma simplificação didática de planejamento sequencial; não descreve toda implementação histórica de Waterfall nem todo o argumento de Royce.

Etapa	Exemplo orientado por fases	Exemplo de ciclo Caracol
Definir	Consolidar requisitos da versão do catálogo antes de detalhar a implementação.	Definir o requisito da capacidade, seus limites e a primeira prova.
Estruturar	Desenhar contratos e componentes para o escopo planejado.	Acordar o contrato mínimo; identificar dependências da unidade.
Construir	Implementar os componentes previstos e reuni los para verificação.	Implementar filtro e consumidor; integrar cedo um fluxo pequeno.
Verificar	Executar a verificação prevista para a entrega e tratar desvios encontrados.	Testar a unidade e a integração do ciclo com evidência por versão.
Receber fato novo	Analisar impacto e replanejar as fases atingidas conforme o processo escolhido.	Reabrir requisito contrato dados código e testes afetados; preservar os independentes.
Aceitar	Decidir com os critérios de aceitação da versão.	Decidir o gate da capacidade e registrar o próximo ciclo.

Na Oficina, uma nova regra que proíbe mostrar rascunhos afeta seleção e provas do catálogo. Ela não autoriza reescrever a tipografia. O valor da comparação está em demonstrar qual decisão é tomada e quando a prova ocorre, não em concluir que um nome de processo garante sucesso. O artigo histórico de Royce inclui iteração e crítica ao teste tardio, como mostram suas figuras 2 a 4.

Exemplo trabalhado

O incremento inicial atende REQ-001 e REQ-002: lista os serviços ativos e oculta os inativos. Durante a revisão do requisito, surge uma dúvida: um registro sem o campo ativo pode ser tratado como ativo? A decisão do projeto é exigir um valor booleano explícito; ausência é dado inválido, conforme REQ-003. Isso altera a regra de validação e a resposta de erro, mas não exige mudar a cor dos botões.

Artefato	Decisão após o fato novo	Por quê
pratica/docs/decisao-001.md	Registrar o contrato de ativo como booleano obrigatório	Eliminar a interpretação implícita
pratica/data/casos.json	Verificar presença e tipo do campo	Os dados devem respeitar o contrato
pratica/src/catalogo.mjs e testes	Revisar validação e cenário de campo ausente	A evidência anterior não cobria essa entrada
pratica/public/styles.css	Preservar nesta mudança	A regra de seleção não depende das cores

O resultado esperado é uma falha explícita para o registro sem ativo, em vez de sua publicação silenciosa. A tabela é uma decisão didática; a aprovação depende da execução correspondente no pacote. A identificação da versão e os resultados serão associados ao novo gate, sem reutilizar como prova suficiente um teste realizado antes da mudança.

Retome a Figura 6 Uma unidade só avança com prova e decisão.

Erros e reparos

Erro	Consequência	Reparo
Reabrir todo o site	Misturar mudanças independentes	Listar dependentes e justificar cada reabertura
Proibir qualquer retorno	Manter um critério conhecido como inadequado	Registrar o fato novo e renegociar escopo e evidência
Preservar testes antigos sem análise	Aprovar uma regra que eles não verificam	Adicionar o cenário alterado e repetir regressões pertinentes

Gate observável

O incremento pode avançar quando o contrato de ativo está escrito, dados válidos são aceitos, ausência ou tipo inválido produz resposta explícita e a listagem continua ocultando inativos. O registro deve identificar a versão verificada. Se a validação ainda não foi executada, o resultado é BLOQUEADO para entrega dessa capacidade, mesmo que a tabela de impacto esteja completa.

Exercício resolvido

Dados: a regra atual já exige ativo booleano. Surge a proposta de acrescentar uma imagem opcional aos casos. Um serviço não possui imagem; o texto alternativo já consta no requisito. Indique a menor reabertura e a evidência de encerramento.

Solução: reabrir contrato da imagem opcional, apresentação no navegador e testes do estado sem imagem. Verificar se o carregamento de dados trata a ausência sem erro. Preservar a seleção por ativo se sua entrada e sua regra não mudaram. A prova deve mostrar caso com imagem e caso sem imagem, incluindo navegação por teclado quando houver interação.

- A resposta identifica a ausência permitida e o comportamento alternativo esperado.

- Inclui apresentação e carregamento entre os pontos de análise, sem obrigar alteração onde a inspeção provar compatibilidade.
- Justifica preservação da filtragem e exige resultado verificável, não apenas um desenho de fluxo.

S11, figuras 2–4 e discussão das páginas iniciais da cópia de Royce, trata sequência, iteração e risco de descobrir problemas tarde. S12, valores e princípios, sustenta a atenção à mudança; S16, testes relacionados a mudanças, ajuda a selecionar confirmação e regressão. Nenhuma dessas fontes apresenta ou certifica o método Caracol.

Referências S11 S12 S16 Detalhes e limites na seção de referências.

Capítulo 18 Ciclos unidades e granularidade

A capacidade orienta a divisão

Uma capacidade descreve algo que uma pessoa consegue realizar: consultar a lista pública de serviços ativos é uma capacidade; criar uma pasta não é. Um ciclo Caracol reúne trabalho suficiente para demonstrar essa capacidade. A unidade é uma porção revisável do trabalho, com finalidade, entrada, resultado e critério próprios. Ela pode envolver mais de um arquivo quando a alteração só faz sentido em conjunto.

Granularidade é o tamanho escolhido para essas porções. Unidades grandes escondem decisões e dificultam localizar falhas. Unidades minúsculas podem fragmentar uma mudança simples e exigir coordenação sem benefício. A pergunta útil é se outra pessoa consegue compreender, verificar e integrar a unidade sem reconstruir todo o projeto.

Separar por departamento não basta. Se uma pessoa entrega dados, outra componentes e outra testes, ainda falta comprovar que o navegador apresenta a lista correta. O ciclo termina por comportamento integrado; a conclusão administrativa das tarefas não substitui essa prova. S01, nas áreas de requisitos, construção e testes, ajuda a distinguir esses trabalhos, sem definir os tamanhos particulares do método Caracol.

Decomposição do primeiro incremento

Considere três registros de exemplo: revisão de texto ativa, diagramação inativa e oficina de escrita ativa. A saída esperada contém dois serviços, sem revelar a diagramação. Além da lista normal, o incremento precisa tratar ausência de serviços ativos e entrada inválida. Cada cenário observa uma parte diferente de REQ-001, REQ-002 e REQ-003.

Unidade	Entrada e resultado	Aceite local
Contrato e dados	Registros com id, titulo, descricao e ativo	Campos e tipos documentados; exemplos coerentes
Leitura e seleção	Dados válidos para lista de ativos	Resultado exclui os inativos; entrada inválida é sinalizada
Apresentação	Lista selecionada para conteúdo visível	Títulos aparecem; estado vazio e erro são distintos
Prova integrada	Servidor, dados e página trabalhando juntos	Navegação real confirma os critérios do incremento

A unidade de apresentação pode ser revisada com dados de teste que respeitam o contrato. Isso permite trabalho paralelo, mas não prova a integração com o arquivo efetivo. A prova do ciclo recompõe o percurso completo. Se o módulo seleciona corretamente e a página mostra todos os registros por engano, as unidades não produziram a capacidade pretendida.

Procedimento de ajuste

- Escreva a capacidade e os cenários observáveis antes de listar arquivos.
- Separe decisões que podem ser verificadas de modo independente e mantenha juntas alterações inseparáveis.
- Para cada unidade, registre entrada, resultado, dependências e evidência local.
- Reserve uma prova do percurso integrado; defina dados conhecidos e resultado esperado.
- Revise a divisão após a primeira execução. Divida uma unidade opaca ou reúna fragmentos que sempre mudam juntos.

Erros e reparos

Erro	O que impede observar	Reparo correspondente
Unidade chamada melhorar catálogo	Escopo e conclusão	Nomear comportamento e critério concreto
Um item para cada linha de código	Razão da mudança conjunta	Agrupar pela finalidade verificável
Integração deixada para publicação	Compatibilidade entre partes	Integrar a cada capacidade pequena e repetir a prova

Gate do ciclo

O ciclo de listagem avança quando os três cenários de dados têm respostas explícitas, inativos não aparecem e o percurso pelo navegador usa o conjunto de dados previsto. É necessário conservar as evidências locais e a evidência integrada. Se somente a seleção isolada passou, pode-se concluir essa unidade; o ciclo inteiro permanece aberto. O nível M6, desenvolvido no próximo capítulo, depende justamente dessa integração comprovada.

Exercício com solução e rubrica

Dados: o próximo incremento permite procurar serviços por título; busca vazia deve listar todos os ativos; busca sem correspondência deve informar nenhum resultado. Há dois títulos ativos, Revisão de texto e Oficina de escrita, e um inativo, Diagramação. Proponha unidades sem criar uma tarefa para cada elemento visual.

Solução: definir contrato da busca, implementar seleção combinando atividade e termo, ligar o campo à apresentação e verificar o percurso no navegador. Para o termo escrita, o resultado esperado é Oficina de escrita; para Diagramação, nenhum resultado; para termo vazio, os dois ativos. A prova integrada inclui digitação e limpeza do campo, com feedback acessível. Este é um exercício de extensão; não descreve uma função já existente no pacote.

- A divisão conserva a regra de ocultar inativos em todos os cenários.
- Cada unidade tem resultado verificável e uma dependência identificada.
- A solução inclui prova integrada e os três resultados fornecidos.

- É aceitável reunir seleção e contrato numa alteração pequena; não é aceitável declarar o ciclo concluído só porque o campo foi desenhado.

Referências S01 S16 Detalhes e limites na seção de referências.

Capítulo 19 Maturidade M0 a M8

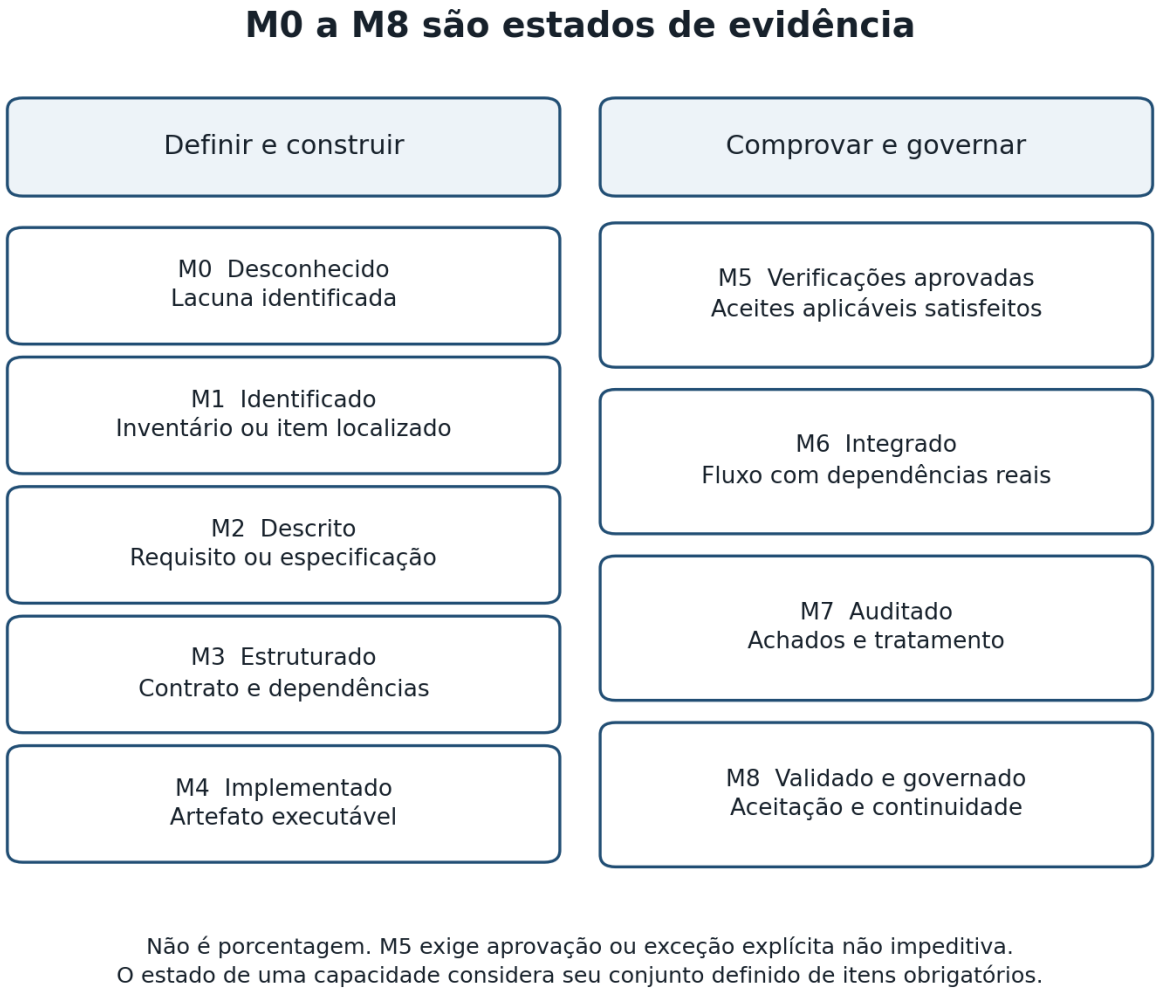


Figura 7 M0 a M8 são estados de evidência

Cada nível depende de evidência pertinente e válida. Executar teste com falha não basta para M5; teste isolado não comprova M6. A maturidade não é uma média percentual.

Estados sustentados por evidência

Dizer que o catálogo está quase pronto não informa o que já foi demonstrado. A escala M0–M8 organiza estados de evidência: uma descrição não equivale a código executável, e um teste isolado não demonstra integração. Essa escala é uma convenção da Engenharia Caracol; as referências sobre testes e desenvolvimento seguro apoiam suas práticas, sem certificar seus níveis.

O estado pertence a um item com escopo explícito. Não é uma nota da pessoa, um percentual de esforço nem uma promessa de qualidade geral. Um módulo pode ter testes aprovados enquanto a capacidade da página permanece incompleta. Também é preciso separar maturidade de autorização: um protótipo

delimitado pode ser examinado antes da liberação pública, desde que ninguém apresente essa autorização como entrega da capacidade final.

Nível	Condição de evidência	Exemplo no catálogo
M0	Item desconhecido ou lacuna registrada	Resposta de erro ainda não investigada
M1	Item identificado no inventário	Tratamento de dado inválido entrou no backlog
M2	Comportamento descrito	Mensagem e condição de erro estão especificadas
M3	Contrato, dependências e aceite estruturados	Entrada inválida e resposta esperada definidas
M4	Implementação correspondente disponível	Tratamento de erro foi escrito
M5	Verificações aplicáveis aprovadas ou exceção explícita não impeditiva	Cenários definidos passaram; ressalva aceita e delimitada
M6	Integração com dependências reais do escopo comprovada	Página usa dados e módulo efetivos e apresenta o erro previsto
M7	Revisão contra critérios registrada e achados tratados	Inspeção e evidências revisadas; limites declarados
M8	Aceite e governança de operação sustentados	Responsável, liberação, acompanhamento e recuperação definidos

Aplicação e agregação

Para calcular o estado de uma capacidade, primeiro declare o conjunto de itens obrigatórios. Depois use o menor nível desse conjunto. Itens opcionais ficam visíveis, mas fora do cálculo. Não é permitido retirar um item necessário apenas para elevar o resultado. Uma classificação de risco ajuda a decidir quais itens são obrigatórios; não muda silenciosamente a fórmula.

Item da capacidade listagem	Obrigatório	Estado didático
Seleção de ativos	Sim	M6
Apresentação da lista	Sim	M6
Resposta de dado inválido	Sim	M4
Operação por teclado prevista no escopo	Sim	M3
Animação decorativa	Não	M1

Neste exercício de classificação, o estado global é M3, o menor dos quatro itens obrigatórios. A animação não reduz o estado, mas o requisito de teclado reduz. A tabela não relata testes executados no pacote. Se as verificações de teclado forem aprovadas e integradas, seu estado poderá subir; a resposta de dado inválido em M4 passará a limitar a capacidade. Somar níveis ou calcular uma média esconderia justamente a lacuna.

Promoção e validade

- Associe ao item a versão, o ambiente, o critério e a evidência que sustentam seu nível.
- Promova apenas quando as condições do próximo estado forem atendidas. Resultados reprovados são evidências de falha, não aprovação de M5.
- Se houver exceção, registre por que não impede a capacidade, quem aceita o risco, qual limite se aplica e quando revisar.
- Invalide evidências quando uma mudança relevante altera o que foi verificado; não reduza o nível só porque passou um dia.
- Aplique prazo quando uma política de risco exigir validade temporal, identificando a razão e a nova verificação necessária.

Erro identificado e retorno necessário

Uma mudança no campo ativo pode invalidar o teste de seleção e a prova integrada que dependia dele. A análise determina quais evidências continuam válidas. Se o contrato permanece estruturado, mas a implementação já não corresponde a ele, o item pode retornar a M3. Se o código continua adequado e somente falta repetir uma verificação em ambiente alterado, o nível sustentado pode ser M4. O retorno é justificado pela evidência perdida, não por uma punição automática.

Erro	Reparo
Classificar teste falho como M5	Corrigir e obter aprovação, ou documentar exceção realmente não impeditiva
Usar média para ocultar item obrigatório imaturo	Aplicar o mínimo ao conjunto declarado
Rebaixar tudo por tempo sem política	Avaliar mudanças e validade de cada evidência

Exercício e gate de classificação

Dados: seleção M6, página M6, estado vazio M5, dado inválido M4; todos são obrigatórios. Uma animação M0 é opcional. Qual o estado global e qual evidência falta primeiro?

Solução: M4. A animação não participa. A próxima promoção exige aprovar as verificações aplicáveis ao dado inválido, ou classificar uma exceção não impeditiva justificável; aqui, como REQ-003 exige resposta explícita, sua ausência é impeditiva. Depois será necessário integrar os itens que ainda não atingiram M6.

- Rubrica: declarar M4 e citar o conjunto obrigatório; excluir a animação sem apagar sua existência; apontar a evidência de aprovação que falta.
- Gate: outra pessoa deve reproduzir a classificação a partir da tabela, localizar as evidências e identificar a condição de promoção. Divergência não explicada mantém a classificação em revisão.

Consultar S16, atividades de teste e gestão de riscos, e S03, prática PO.4 sobre critérios de verificação. A escala interna não substitui critérios técnicos específicos de cada requisito.

Referências S16 S03 Detalhes e limites na seção de referências.

Capítulo 20 Gates e vereditos

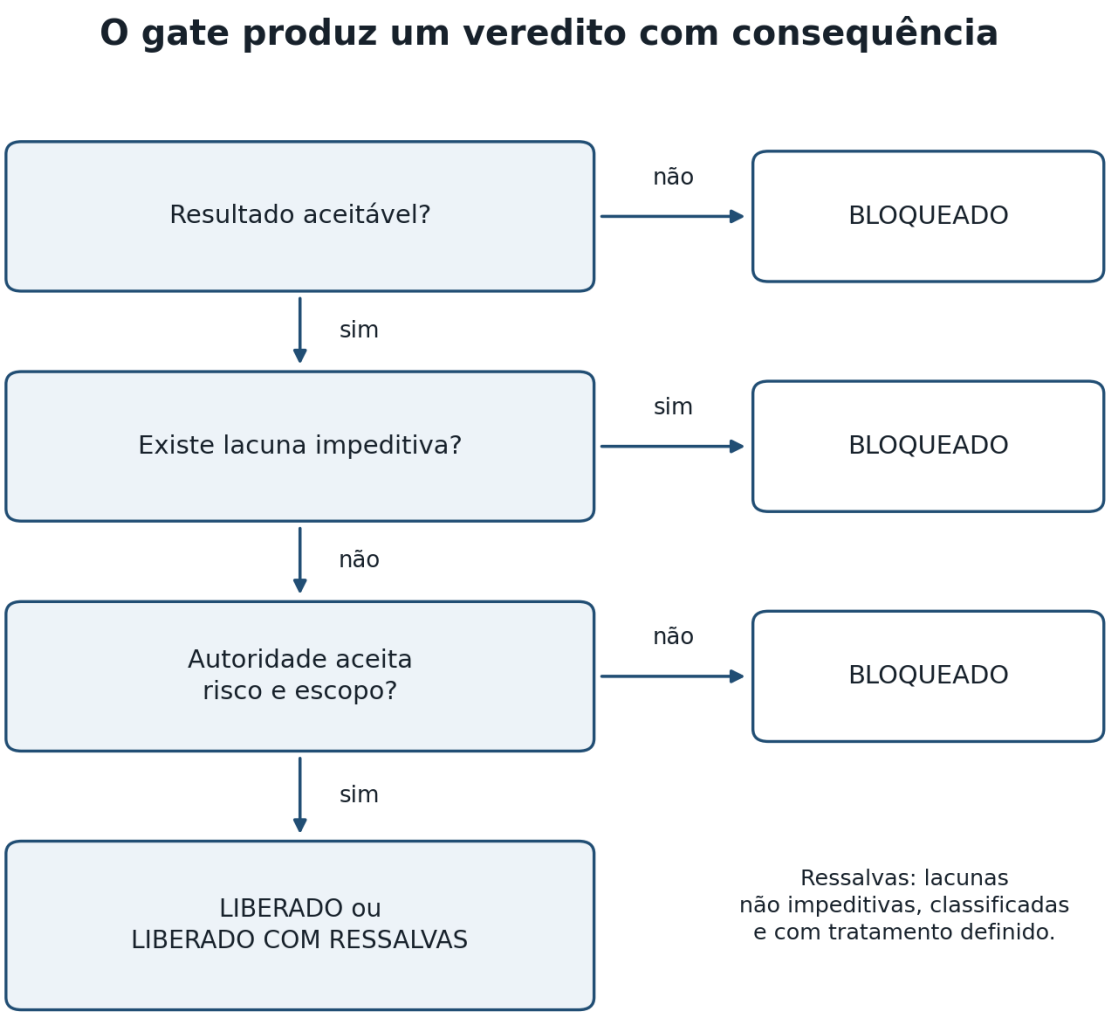


Figura 8 O gate produz um veredito com consequência

Documentar uma falha não a torna aceitável. O registro do gate deve delimitar escopo, evidência, autoridade, riscos e consequências da decisão.

Uma autorização com escopo

Gate é uma decisão de passagem vinculada a uma capacidade, uma versão e um uso. Ele responde se aquele resultado pode avançar nas condições declaradas. Não é uma reunião obrigatória nem um carimbo universal. Autorizar um protótipo para inspeção local não significa autorizar sua publicação como produto concluído.

LIBERADO significa que os critérios aplicáveis foram atendidos e as evidências sustentam o uso autorizado. LIBERADO COM RESSALVAS exige lacunas classificadas como não impeditivas, risco aceito por autoridade identificada, limites e acompanhamento. BLOQUEADO impede o avanço

pretendido enquanto existe falha impeditiva ou falta de evidência necessária. Um formulário completo pode registrar qualquer um desses três resultados.

A autoridade é quem responde pela decisão no contexto do projeto. Na Oficina Caracol individual, o responsável pelo projeto pode decidir uma demonstração local, mas deve registrar o limite da revisão feita pela própria pessoa. Competências especializadas ou políticas do contexto podem exigir outro aprovador. O método não cria essa competência apenas preenchendo o nome de alguém. S03, prática PO.4, trata critérios de verificação; S06 apresenta critérios de acessibilidade e requisitos de conformidade que não se resumem a um teste automatizado.

Procedimento de decisão

- Declare o que será autorizado: inspeção local, integração ou disponibilização da versão ao público.
- Liste critérios de aceite, evidência necessária e condições impeditivas antes de avaliar.
- Verifique resultados, atualidade e alcance das evidências; um comando registrado com falha não demonstra aprovação.
- Classifique cada lacuna e registre o risco residual, sua aceitação, restrições, responsável e prazo ou gatilho de revisão.
- Emita o veredito, identifique a autoridade e comunique a próxima ação. Se a versão ou as condições mudarem, reavalie a autorização.

Gate preenchido para o catálogo

Considere um exercício documental em que a implementação foi preparada, mas ainda não existe registro de navegação por teclado. O objetivo do gate é liberar a versão pública que declara atender REQ-001 a REQ-004. O seguinte preenchimento resolve a decisão com os dados disponíveis, sem inventar execução.

Campo	Preenchimento didático
Identificação e escopo	G-CAT-01; versão de trabalho; publicação do catálogo com REQ-001 a REQ-004
Critérios	Listar ativos; ocultar inativos; tratar vazio e erro; verificar teclado e feedback
Evidência disponível	Plano de cenários e arquivos preparados; resultados ainda não anexados
Lacuna impeditiva	Não há evidência das verificações exigidas, inclusive teclado
Veredito	BLOQUEADO para publicação com essa declaração de atendimento
Autoridade e próxima ação	Responsável pelo projeto; executar verificações e anexar resultados
Validade	Reavaliar após os resultados na mesma versão; mudança relevante exige nova análise

Agora mude o escopo: apenas uma inspeção local supervisionada do layout, sem alegação de versão pronta. Uma autorização restrita pode admitir verificações pendentes, desde que os participantes saibam o objetivo e a limitação. Isso não eleva automaticamente maturidade nem resolve REQ-004. Se a versão pública já atende todos os critérios e falta apenas uma ilustração opcional da documentação, a ausência pode ser ressalva editorial, com responsável e revisão definida.

Erros e reparos correspondentes

Erro	Reparo
Ressalva sem responsável ou revisão	Nomear quem acompanha e definir data ou gatilho justificável
Gate retroativo apresentado como autorização prévia	Registrar a cronologia real, avaliar a exposição ocorrida e decidir o próximo avanço
Aprovador sem competência para o risco	Buscar revisão adequada ou restringir o escopo; não transformar assinatura em conhecimento
Registro de teste confundido com sucesso	Conferir resultado e consequência de cada falha antes do veredito

Exercício resolvido

Dados: o pacote tem plano de teste, mas nenhum resultado registrado. O pedido A é mostrar o protótipo localmente para discutir nomes dos serviços. O pedido B é publicar afirmando que os requisitos de teclado foram verificados. Emita dois gates e indique a evidência que falta.

Solução: A pode ser autorizado com escopo restrito de discussão, dados de exemplo e indicação de verificações pendentes; essa é uma decisão de uso, não uma aprovação da capacidade final. B permanece BLOQUEADO: falta a verificação manual exigida e a evidência de seu resultado. A ausência de teste não prova que a interface falha, mas impede afirmar que a verificação ocorreu.

- Rubrica: distinguir os usos, recusar a declaração sem prova e identificar responsável e próxima ação.
- Aceita-se bloquear também A se houver condição impeditiva adicional explicitada; não se aceita liberar B apenas por ser um protótipo.
- Gate de aprendizagem: o leitor entrega dois registros reproduzíveis, com consequência concreta e sem converter pendência em aprovação.

Referências S03 S06 S16 Detalhes e limites na seção de referências.

Capítulo 21 Evidência e rastreabilidade

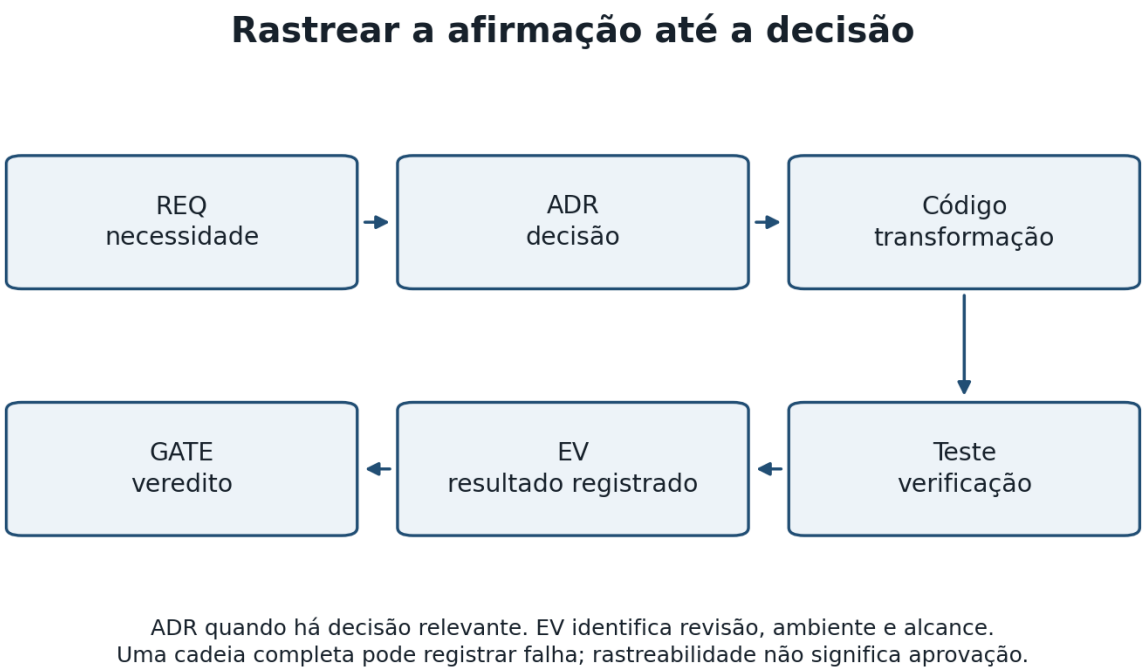


Figura 9 Rastrear a afirmação até a decisão

O exemplo mostra uma ligação possível. Nem todo requisito exige um ADR próprio. Uma evidência de teste vale para a versão, o ambiente e o alcance efetivamente observados.

O alcance de uma prova

Evidência é informação identificável que sustenta uma afirmação dentro de limites. Uma imagem pode mostrar a aparência de uma tela; não demonstra, sozinha, que inativos nunca são enviados pela API. A saída de um comando pode demonstrar que certas verificações foram executadas numa versão; não comprova requisitos que elas não examinam. Antes de guardar um arquivo, escreva a afirmação que ele deveria sustentar.

Rastreabilidade é a ligação entre essa afirmação, seu requisito, a decisão adotada, os arquivos e a verificação correspondente. Ela permite responder por que existe um teste e que prova precisa ser revista após uma mudança. Em um projeto pequeno, uma tabela curta pode ser suficiente. Criar links sem explicar a relação apenas desloca a informação para outro lugar.

A força da evidência depende de versão, ambiente, entrada, procedimento, resultado e conclusão compatível com o resultado. Um log bruto pode conter erros ignorados; um resumo escrito pode omitir a falha. Conserve o que permite conferir a conclusão e indique limitações. S16, nas atividades de teste, diferencia produtos de teste e resultados; S07, na seção sobre registro de alterações, mostra como o histórico de versões ajuda a identificar mudanças.

Uma cadeia preenchida

Elemento	Registro para REQ-002
Afirmação	O endpoint público de listagem não entrega registros inativos
Decisão D-CAT-02	Selecionar ativos antes de formar a representação pública; publicar só id, titulo e descricao
Implementação	pratica/src/catalogo.mjs e pratica/src/servidor.mjs
Verificação prevista	Dados com dois ativos e um inativo; observar lista retornada e campos de cada item
Arquivos de teste	pratica/tests/catalogo.test.mjs e pratica/tests/servidor.test.mjs
Entrega	Versão identificada no registro de execução e no gate do incremento

Essa cadeia distingue ocultar visualmente e não entregar o registro. Esconder uma linha com CSS não satisfaz a afirmação do exemplo, pois o dado pode continuar acessível na resposta. Os testes do módulo verificam seleção; o teste do servidor verifica a fronteira HTTP, isto é, o intercâmbio entre cliente e servidor. A inspeção no navegador verifica a apresentação, com alcance diferente.

Código 17 text

```
Registro planejado E-CAT-02
Afirmação: REQ-002, ausência de inativos na listagem pública
Diretório de execução: pratica
Comandos: node --version; npm test; npm run validate
Entrada controlada: dois casos ativos e um inativo
Esperado: só os dois ativos, com id, titulo e descricao
Resultado observado: ainda não registrado neste exemplo
Versão/ambiente: capturar na execução
Conclusão: evidência planejada; não autoriza liberação
```

O preenchimento é útil justamente porque não simula uma aprovação. Na execução, substitua a indicação de pendência por versão real, ambiente, saída e conclusão derivada. Não edite uma falha para parecer sucesso: registre a correção e preserve a tentativa anterior quando ela explicar a decisão.

Procedimento de registro

- Escreva requisito, afirmação e critério que será avaliado.
- Escolha uma prova que observe a fronteira correta: função, HTTP, navegador ou operação.
- Identifique entrada, comando ou passos manuais, versão e ambiente.
- Registre resultado observado, diferenças em relação ao esperado e limites da conclusão.
- Vincule a evidência ao gate e marque as mudanças que exigirão revalidação. Evite incluir segredos ou dados pessoais no registro.

Erros e reparos

Erro	Reparo específico
Screenshot como prova de filtragem na API	Examinar resposta HTTP e executar teste com inativo conhecido

Erro	Reparo específico
Log sem versão	Associar execução à revisão real; se isso não puder ser recuperado, repetir a prova
Saída extensa sem conclusão	Identificar critérios aprovados e falhas, sem ocultar a saída necessária à conferência

Exercício com solução

Dados: uma captura mostra apenas Revisão de texto e Oficina de escrita. A resposta HTTP salva contém também Diagramação, marcada como inativa. A regra exige não revelar inativos. O responsável afirma que a captura prova REQ-002. Avalie e proponha a correção.

Solução: a captura sustenta somente o estado visual observado. A resposta contradiz a afirmação de não entrega do inativo; o gate dessa capacidade deve ser bloqueado. Corrigir a seleção ou a formação da resposta na fronteira adequada, depois repetir teste do módulo, do servidor e a inspeção pertinente. Anexar resposta sem o inativo e identificar a versão corrigida.

- Rubrica: distinguir aparência de entrega do dado, rejeitar a conclusão excessiva e escolher prova que possa revelar o erro.
- O avanço exige cadeia completa, resultado aceitável e limites claros; completar a tabela sem executar a verificação não satisfaz o gate.

Referências S16 S07 S03 Detalhes e limites na seção de referências.

Capítulo 22 Retorno controlado

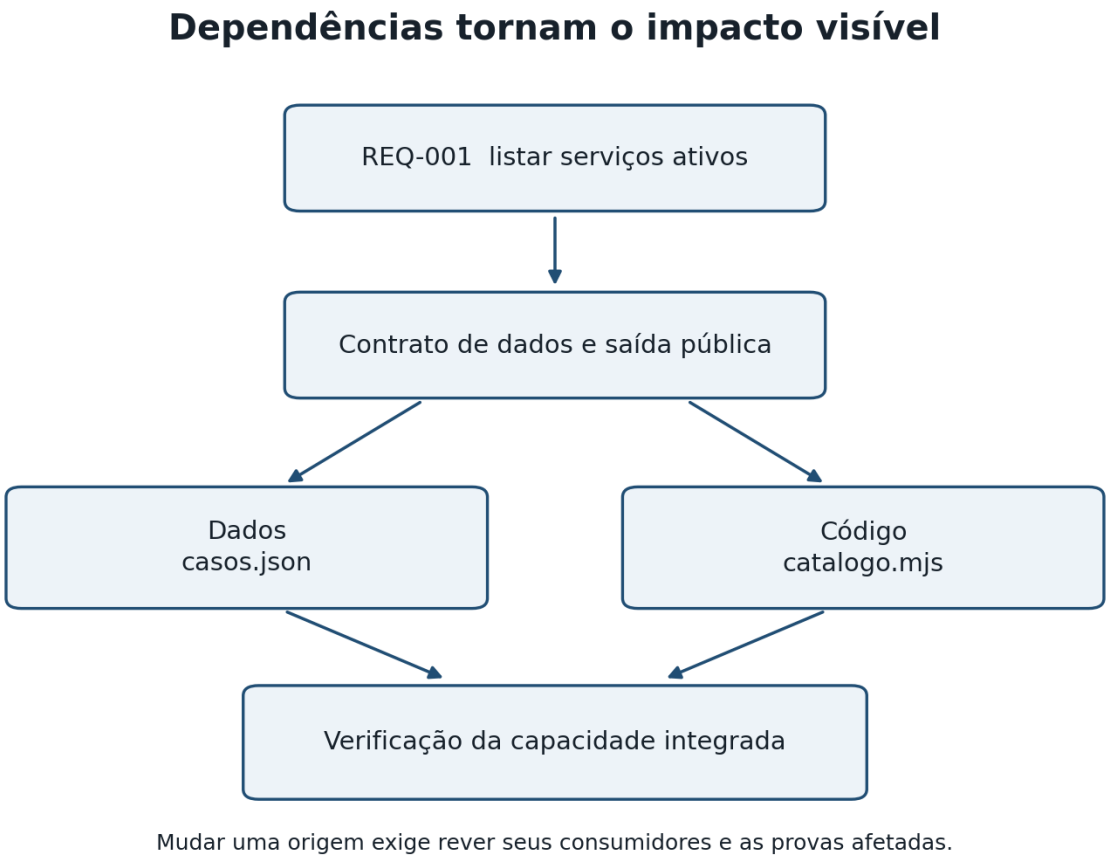


Figura 10 Dependências tornam o impacto visível

A ordem representa dependências do caso. Dados de teste e consumidores podem ser desenvolvidos em paralelo quando o contrato já está acordado; integrar exige revalidar as relações.

Corrigir sem apagar a história

Retornar significa revisar uma decisão ou artefato porque surgiu informação que altera sua validade. Não é recomeçar por insegurança nem desfazer tudo por conveniência. O retorno controlado registra gatilho, impacto, evidências invalidadas, correção e nova verificação. Ele conserva o que ainda é sustentado e torna visível o que deixou de ser.

Na Oficina Caracol, imagine que uma edição de dados troque ativo: true por ativo: "sim" em um registro. A estrutura exige um booleano, que só admite true ou false. O dado agora viola o contrato. A implementação correta deve sinalizar a entrada inválida; isso não significa que o catálogo voltou a listar serviços válidos. É necessário corrigir o dado para recuperar essa capacidade e verificar o percurso.

Essa diferença evita um erro comum: alterar o validador para aceitar qualquer entrada apenas para deixar o teste verde. Se o contrato é adequado, consertamos o dado. Se o contrato realmente precisa mudar,

registramos a decisão e analisamos todos os consumidores. O motivo da alteração determina a correção, e a evidência determina o estado que podemos declarar.

Registro de retorno preenchido

Campo	Preenchimento do cenário
Gatilho R-CAT-03	Campo ativo editado como texto em pratica/data/casos.json
Contrato vigente	ativo obrigatório e booleano; dados públicos sem campos internos
Dependentes diretos	Validação de dados e carregamento da listagem
Dependentes do fluxo	Resposta do servidor e estado de erro apresentado na página
Evidências afetadas	Verificações que usaram o arquivo anterior para afirmar listagem válida
Preservado com motivo	Estilos sem relação com o tipo do campo; contrato mantido
Correção escolhida	Restaurar booleano correto nos dados; não flexibilizar a validação
Encerramento previsto	Validar dados, repetir testes e confirmar listagem integrada na versão corrigida

A análise não exige modificar todos os dependentes. Ela exige verificar se foram afetados. Um teste isolado do módulo com a mesma entrada válida pode continuar relevante para aquele caso, enquanto a prova integrada que usou o arquivo antigo não sustenta a nova versão dos dados. A condição de erro também deve ser inspecionada para impedir que o usuário receba silêncio ou uma página aparentemente vazia.

Retome a Figura 10 Dependências tornam o impacto visível.

Procedimento e estados

- Registre o fato observável, a versão afetada e o requisito que deixou de estar sustentado.
- Mapeie consumidores diretos e indiretos. Separe artefato a mudar de artefato apenas a verificar.
- Reclassifique os itens afetados conforme a evidência que resta. Não aplique um rebaixamento idêntico sem justificativa.
- Faça a menor correção suficiente, preservando o contrato ou revisando-o explicitamente.
- Execute as verificações locais e integradas pertinentes; emita novo gate ligado ao registro de retorno.

Se a versão atual do item já não atende ao contrato estruturado, M3 pode ser o estado sustentado antes de corrigir a implementação ou o dado. Quando a correção existe, mas falta verificá-la, M4 é mais preciso. A promoção a M5 exige aprovação das verificações aplicáveis; M6 requer a integração comprovada. O mero tempo decorrido não causa regressão sem uma política que explique a expiração.

Erros e reparos correspondentes

Erro	Reparo
Manter o gate antigo após mudança relevante	Marcar seu alcance e reavaliar a versão atual antes de reutilizá-lo

Erro	Reparo
Corrigir dados sem conferir consumidores	Repetir o percurso que depende do dado, além da validação local
Mudar o contrato só para aceitar o erro	Restaurar o dado ou justificar formalmente uma nova regra
Reabrir arquivos independentes	Exigir uma relação de impacto antes de incluir cada item no retorno

Exercício e solução comentada

Dados: um caso ativo muda somente sua descrição, mantendo id, titulo e ativo. O contrato de descrição não muda. A versão anterior foi verificada; a nova ainda não. Informe o que deve ser alterado, o que deve ser conferido e o que pode ser preservado.

Solução: alterar o registro no arquivo de dados e conferir se a descrição respeita o contrato, é entregue na representação pública e aparece corretamente na página. Repetir as verificações pertinentes ao percurso; a regra de seleção de ativos não precisa ser reescrita. Seu código pode ser preservado, enquanto a evidência do conjunto de dados anterior deve ter seu alcance reconhecido. Examinar se a mudança textual afeta leitura ou apresentação antes de dispensar a inspeção.

- Rubrica: distinguir mudança de dados e mudança de regra; incluir a apresentação entre os dependentes; preservar código com justificativa; exigir evidência atual do comportamento afetado.
- Gate: o retorno só encerra quando causa, impacto, correção e revalidação estão ligados à mesma versão e nenhuma falha impeditiva permanece escondida.

S07, seção sobre desfazer alterações, descreve mecanismos de histórico; S16, atividades de teste e testes relacionados a mudanças, ajuda a escolher confirmação e regressão. Um comando de reversão não substitui a análise do impacto.

Referências S07 S16 S03 Detalhes e limites na seção de referências.

Parte IV Preparação operacional

Criar estrutura ambiente e critérios de execução

Capítulo 23 Raiz diretórios e porta de entrada

O projeto deve explicar onde começar

A raiz é a pasta que contém o projeto. Um caminho identifica a localização de um arquivo a partir de alguma referência. Em pratica/src/catalogo.mjs, pratica é a raiz do pacote de estudo e src agrupa seu código. Se o terminal já estiver dentro de pratica, o caminho relativo para o mesmo arquivo é src/catalogo.mjs. Confundir essas referências pode fazer um comando procurar dados no lugar errado.

Uma estrutura pequena ajuda o iniciante a localizar responsabilidades. Ela não precisa imitar uma empresa grande nem antecipar todos os recursos possíveis. Crie diretórios para categorias presentes no trabalho e explique as convenções. Um arquivo chamado utilidades que acumula funções sem relação também não resolve a organização: o nome precisa indicar finalidade.

O README é o documento de entrada. Ele informa o que o projeto faz, como iniciar, quais comandos verificar e quais limites não podem ser esquecidos. Não deve exigir que o leitor percorra toda a história da conversa. S07, na seção de registro de alterações, trata do acompanhamento de arquivos; o histórico complementa a entrada, sem substituí-la.

Mapa preenchido da Oficina Caracol

Caminho	Responsabilidade
pratica/README.md	Objetivo, início rápido, verificações e limites
pratica/package.json e package-lock.json	Comandos do pacote e registro de instalação
pratica/src/catalogo.mjs	Validar dados e selecionar casos públicos
pratica/src/operacoes.mjs	Simular reservas idempotentes em memória
pratica/src/servidor.mjs	Servir API e arquivos estáticos autorizados
pratica/data/casos.json	Dados fictícios do catálogo
pratica/public/	HTML, JavaScript do navegador e estilos
pratica/tests/	Verificações automatizadas por responsabilidade
pratica/scripts/validate.mjs	Executar a validação definida pelo projeto
pratica/docs/	Decisões, evidências, gates, riscos e continuidade

O navegador executa public/app.js; o Node executa os módulos do servidor e dos testes. A extensão não define sozinha o papel: o README e o comando de entrada explicam o ambiente. O diretório public contém arquivos que podem ser entregues ao navegador; isso não autoriza publicar qualquer arquivo da raiz. O servidor usa uma lista fixa de arquivos estáticos, conforme seu contrato.

Retome a Figura 1 Pastas da Oficina Caracol por função.

Trecho de README utilizável

Código 18 markdown

```
# Oficina Caracol
Catálogo de serviços fictícios e simulação de reservas.
Sem cadastro, banco externo ou dados pessoais reais.

## Iniciar
Abra o terminal dentro desta pasta pratica.
Confira o runtime com node --version.
Execute npm ci, npm test e npm run validate.
Execute npm start e abra http://127.0.0.1:3000.
Encerre o servidor com Ctrl+C no terminal.

## Limites
Reservas ficam apenas na memória e somem ao reiniciar.
Snapshot e restore usam somente sandbox/recuperacao.
Verificações manuais de teclado devem ser registradas.
A versão testada e os resultados estão no registro de evidência.
```

Este trecho ensina uma sequência, sem inventar a versão de Node instalada nem afirmar que os comandos já passaram. Na edição entregue, o registro de evidência identifica o ambiente realmente testado. Se a pessoa não sabe em qual pasta está, resolva essa dúvida antes de executar o comando.

Procedimento de organização

- Liste as categorias reais de arquivo e a finalidade de cada uma.
- Mantenha uma porta de entrada e caminhos coerentes entre README, comandos e documentação.
- Ao mover um arquivo, revise importações, carregamento de dados, testes e referências documentais.
- Peça a alguém que localize entrada, dado e teste sem instruções verbais; registre a dificuldade encontrada. Na ausência dessa pessoa, registre o limite da autoinspeção.

Erros e reparos

Erro	Reparo
Pasta misc sem finalidade	Classificar seu conteúdo por responsabilidade real e renomear
Mover módulo e esquecer importação	Atualizar referências e repetir as verificações afetadas
README com comando fora da pasta correta	Declarar diretório de execução e conferir o percurso desde a raiz

Exercício resolvido e gate

Dados: um colega quer mudar a regra de seleção de ativos, conferir a entrada e registrar o risco da mudança. Indique três caminhos e explique a ordem de consulta.

Solução: começar por pratica/README.md para entender execução e limites; consultar pratica/src/catalogo.mjs e pratica/tests/catalogo.test.mjs para regra e prova; registrar a análise em pratica/docs/risco.md. O dado de exemplo fica em pratica/data/casos.json. Localizar o código não autoriza alterá-lo sem critério ou ignorar o consumidor HTTP.

- Rubrica: localizar entrada, implementação, teste e dado; distinguir programa e documentação; informar que os comandos oficiais são executados dentro de pratica.
- Gate: a navegação permite localizar esses itens, os caminhos citados existem no pacote entregue e as mudanças de estrutura não deixam referências quebradas. A verificação externa ainda não realizada deve permanecer declarada como pendência.

Referências S07 S19 Detalhes e limites na seção de referências.

Capítulo 24 Manifesto numeração e mapa geral

Identidade não é posição

O manifesto é o mapa dos artefatos que o projeto reconhece como oficiais, candidatos ou históricos. Ele responde qual documento orienta o trabalho e onde localizar sua versão. Um número pode facilitar referência, mas não prova vigência. O documento 003 pode estar superado, enquanto uma revisão posterior do mesmo documento ocupa o papel vigente.

Identificador é o nome estável de uma entidade; versão identifica uma revisão; estado informa seu papel atual. Separar essas três dimensões evita renumerar todo o acervo quando se insere um documento. No código, nomes por responsabilidade continuam mais úteis que uma sequência arbitrária: catalogo.mjs informa a finalidade; 017.mjs exige consulta adicional.

A Oficina Caracol não precisa numerar cada arquivo para ser organizada. Precisa saber qual contrato e qual gate sustentam a versão entregue. S07, nas seções sobre registro de alterações e histórico, oferece mecanismos para localizar revisões. O manifesto acrescenta a decisão sobre seu papel; um commit antigo não é automaticamente o documento vigente.

Manifesto preenchido em um exercício

Considere cinco artefatos documentais. Os rótulos r1 e r2 abaixo são revisões lógicas do exercício, não identificadores de commits executados. Em um repositório real, cada rótulo deve corresponder a uma revisão recuperável e verificável; se essa associação não existe, a rastreabilidade ainda está incompleta.

Identificador	Caminho de referência	Revisão	Estado e relação
DOC-DEC-01	pratica/docs/decisao-001.md	r1	Vigente; decisão de não usar dependências externas
DOC-EVI-01	pratica/docs/ evidencia-001.md	r1	Vigente como plano; execução pendente deve continuar explícita
DOC-GATE-01	pratica/docs/gate-001.md	r1	Vigente; bloqueia liberação até resultados necessários
DOC-GATE-01	pratica/docs/gate-001.md	r2	Candidata; substituirá r1 somente após decisão
DOC-RISCO-01	pratica/docs/risco.md	r1	Vigente; riscos e responsáveis do incremento

As duas linhas do gate não significam duas autorizações vigentes. r2 é candidata. Enquanto a decisão de substituição não ocorrer, r1 continua orientando o avanço. A implementação do armazenamento pode usar histórico de versões ou um arquivo candidato separado; a relação precisa ser recuperável. O manifesto não deve apontar duas revisões diferentes como se ocupassem simultaneamente o mesmo papel.

Quando r2 for aprovada, registre quem decidiu, qual evidência sustenta a revisão e quando ela passa a valer. Marque r1 como histórica e substituída por r2. Preserve o identificador DOC-GATE-01, pois o

objeto continua sendo o registro de gate; uma nova revisão não precisa adquirir uma identidade sem relação com a anterior.

Procedimento de manutenção

- Defina quais artefatos precisam de controle explícito; não inclua cada arquivo por reflexo.
- Atribua identificadores estáveis e nomes que expressem finalidade.
- Registre caminho, revisão, estado, responsável e relação de substituição quando existir.
- Ao aprovar uma substituição, atualize estados e referências de forma conjunta.
- Confira se uma pessoa consegue abrir a revisão indicada e distinguir plano de evidência executada.

Erros e reparos específicos

Erro	Reparo
Renumerar todo o histórico ao inserir um item	Preservar IDs e acrescentar o novo sem alterar referências antigas
Número sem estado	Adicionar vigência e relação de substituição
Arquivo chamado final-final sem revisão recuperável	Adotar identificador estável e revisão verificável
Candidato tratado como vigente	Registrar aprovação ou manter o bloqueio anterior até a decisão

Exercício e solução

Dados: DOC-GATE-01 r1 está vigente e bloqueado. r2 foi escrita, mas depende de verificação manual ainda não realizada. Alguém quer apagar r1 e chamar r2 de aprovada porque ela é mais recente. Atualize o manifesto de maneira correta.

Solução: manter r1 vigente e bloqueada; conservar r2 candidata, com a pendência de verificação explícita. Não apagar a revisão anterior. A data mais recente não substitui a evidência exigida. Depois da verificação aceitável e da decisão da autoridade, r2 pode tornar-se vigente e r1 histórica, com vínculo de substituição.

- Rubrica: preservar identidade e histórico, diferenciar revisão e estado, impedir promoção automática pela data e indicar o gatilho legítimo de substituição.
- Gate: todos os artefatos controlados têm revisão recuperável e estado inequívoco; existe apenas uma revisão vigente por papel e escopo. Links inexistentes ou candidatos apresentados como oficiais impedem a passagem.

Referências S07 S03 Detalhes e limites na seção de referências.

Capítulo 25 Backlog prioridades e pronto

Da vontade ao trabalho verificável

Backlog é a lista organizada do trabalho considerado para o produto. Uma vontade como melhorar o catálogo ainda não define uma alteração executável. O item precisa dizer por que existe, qual comportamento mudará, do que depende, quem responde por ele e como reconhecer sua conclusão. A ordem pode mudar quando surgem evidências, desde que o motivo permaneça visível.

Prioridade expressa importância relativa dentro de um contexto e uma versão. Não substitui dependência: um item P1 pode precisar de uma definição de contrato antes de ser implementado. O contrato necessário pode ser acordado cedo, permitindo que código e interface avancem em paralelo com dados de teste; não é preciso esperar uma tecnologia de armazenamento inexistente no projeto.

Prioridade	Significado adotado na Oficina Caracol	Exemplo contextual
P1	Bloqueio de segurança ou de funcionalidade essencial	Impedir que um inativo seja entregue ao público
P2	Capacidade necessária da versão definida	Feedback acessível exigido no incremento
P3	Melhoria reversível sem impedir os critérios atuais	Refinar espaçamento mantendo comportamento aprovado
P4	Exploração ou futuro fora do compromisso atual	Pesquisar busca avançada ainda não incluída no escopo

As categorias não são propriedades eternas de um recurso. Se o feedback acessível falhar de modo que bloqueie o uso essencial, sua correção passa a P1 naquele contexto. Se uma animação se tornar um requisito necessário de uma demonstração específica, sua classificação deve ser renegociada com o escopo, sem reduzir exigências essenciais para acomodá-la.

Cinco desejos transformados

Entrada do exercício trabalhado: esconder os inativos, mostrar mensagem quando não há casos, garantir uso por teclado, deixar cartões mais bonitos e estudar busca. O responsável do exemplo é quem mantém a Oficina; a classificação é uma decisão didática que deverá ser revista se o contexto mudar.

Item e prioridade	Aceite específico	Dependência e evidência
BL-01 P1 Ocultar inativos	API pública não inclui o registro inativo conhecido	Contrato ativo; teste do módulo e servidor
BL-02 P2 Tratar vazio	Lista válida sem ativos produz mensagem distinta de erro	Seleção e apresentação; teste e inspeção
BL-03 P2 Verificar teclado	Percurso e feedback previstos são operáveis e registrados	Interface integrada; verificação manual
BL-04 P3 Refinar cartões	Melhorar espaçamento sem perder foco ou conteúdo	Interface estável; comparação e regressão pertinente

Item e prioridade	Aceite específico	Dependência e evidência
BL-05 P4 Explorar busca	Registrar alternativas, limites e decisão de escopo	Tempo de investigação; nota de decisão, sem prometer implementação

Cada linha diferencia resultado e atividade. BL-05 pode concluir uma investigação sem criar a busca, porque esse é seu aceite. BL-01 não conclui ao escrever o teste: precisa obter resultado aceitável na implementação integrada. Uma tarefa de teste pode existir como unidade do trabalho, mas a capacidade depende do comportamento demonstrado.

Aceite e definição de pronto

Critério de aceite descreve o comportamento particular de um item. Definição de pronto reúne condições comuns à entrega, como revisão do impacto, verificação aplicável aprovada, integração e atualização de documentação necessária. Nenhuma das duas deve ser apenas marcar uma caixa. Para BL-01, ter um teste escrito é insuficiente se ele falha ou se o servidor contorna a seleção.

- Vincule o item a um objetivo ou requisito.
- Defina resultado e exemplos de aceite; elimine verbos vagos sem complemento.
- Identifique dependências reais e responsáveis.
- Atribua prioridade com motivo e revise a ordem conforme risco e aprendizado.
- Ao concluir, ligue evidência, versão e consequência; conserve os itens adiados de maneira explícita.

Erros e reparos

Erro	Reparo
Tudo marcado como P1	Reavaliar bloqueio real e compromisso da versão; ordenar dependências e impacto
Item chamado melhorar	Escrever comportamento observável e limites
Fechamento sem integração	Manter a capacidade aberta até a prova de seu percurso
Prioridade usada para ignorar dependência	Antecipar o contrato necessário e ajustar a ordem executável

Exercício resolvido e gate

Dados: BL-01 foi implementado, teste do módulo passou em um cenário didático, mas o servidor ainda retorna o inativo. BL-04 está pronto visualmente. O responsável quer concluir a versão e deixar a falha para depois. Qual a decisão?

Solução: não concluir BL-01 nem a versão que exige REQ-002. A falha integrada mantém um bloqueio P1, mesmo com teste local aprovado. Revisar a formação da resposta do servidor, repetir a verificação da fronteira HTTP e a regressão pertinente. BL-04 pode manter sua conclusão local, mas não compensa o requisito essencial ausente.

- Rubrica: reconhecer a falha integrada, manter P1 e indicar a prova que falta; não calcular uma média entre melhoria visual e requisito essencial.

- Gate: itens ativos têm escopo, prioridade justificada, dependências e aceite; os itens necessários à versão só são concluídos com resultado aceitável e integração comprovada.

S01, áreas de requisitos e gestão de engenharia, e S16, planejamento e monitoramento de testes, ajudam a relacionar compromisso, risco e evidência. A escala P1–P4 acima é a convenção contextual do projeto, não uma norma universal.

Referências S01 S16 Detalhes e limites na seção de referências.

Capítulo 26 Risco e proporcionalidade

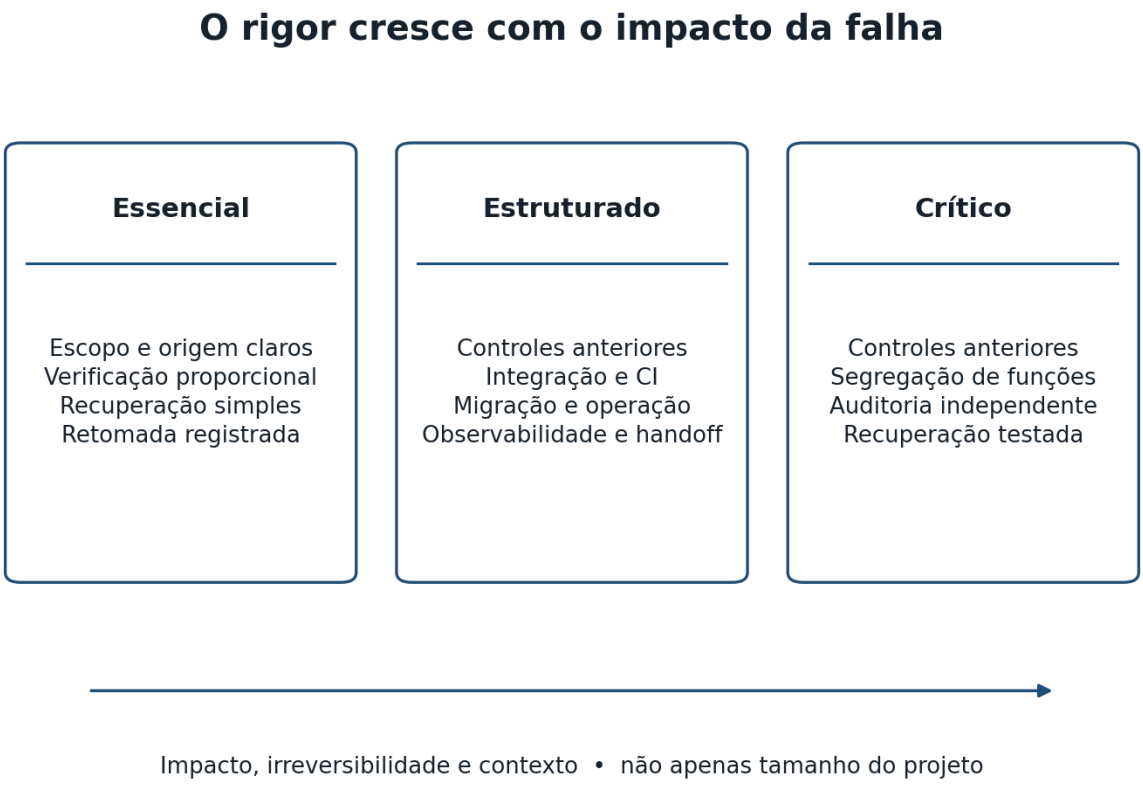


Figura 11 O rigor cresce com o impacto da falha

Os perfis orientam proporcionalidade. A lista é didática, não uma certificação nem um substituto para avaliação específica do sistema e de suas obrigações.

Escolher o rigor pelo efeito da falha

Risco descreve a possibilidade de uma falha e suas consequências no contexto. Probabilidade e impacto orientam a avaliação; capacidade de detectar o problema e possibilidade de desfazer seus efeitos ajudam a escolher a resposta. Não é necessário multiplicar notas arbitrárias para produzir um número aparentemente preciso. É necessário explicar qual falha preocupa, quem sofre seu efeito e qual evidência reduz a incerteza.

A Oficina Caracol usa dados fictícios e não registra informações pessoais. Isso reduz certos impactos, mas não elimina riscos de software: um inativo pode ser entregue, uma falha pode parecer lista vazia ou uma repetição de reserva pode produzir dois efeitos. O tratamento deve corresponder ao problema real. Acrescentar uma senha inexistente ou uma política corporativa copiada não resolveria esses casos.

Proporcionalidade significa escolher controle suficiente para o custo de falhar. O perfil Essencial preserva escopo, entrada clara, inventário, verificações de fluxo, gate e recuperação demonstrável dentro do ambiente de estudo. Estruturado amplia coordenação, automação, ambientes, observação e continuidade

quando equipe e dados o exigem. Crítico envolve impacto elevado e controles especializados, inclusive revisão independente e recuperação testada no contexto apropriado. O kit didático não constitui uma certificação para esse terceiro perfil.

Registro de riscos preenchido

A tabela apresenta estimativas qualitativas para discussão do projeto, não taxas medidas. Uma ocorrência pode alterar a estimativa. O responsável de cada linha é o mantenedor da Oficina, enquanto não houver divisão real de papéis; isso deve ser atualizado se outras pessoas assumirem o trabalho.

Risco	Impacto e detecção	Tratamento e evidência
R-CAT-01 Inativo exposto	Quebra de REQ-002; captura visual pode não detectar entrega pela API	Selecionar e projetar campos públicos; teste do módulo e HTTP
R-CAT-02 Repetição cria reserva duplicada	Quebra da simulação de REQ-005; contador revela o efeito	Conservar a chave da intenção entre tentativas; provar um único efeito e conflito de reutilização
R-CAT-03 Falha tratada como vazio	Usuário recebe interpretação incorreta; pode passar despercebido	Distinguir vazio e erro na interface; verificar ambos
R-CAT-04 Recuperação altera o arquivo fonte	Perda do dado de referência; efeito evitável no exercício	Restringir snapshot e restore a sandbox/recuperacao; conferir isolamento

A resposta de R-CAT-02 não é repetir mais vezes. É preservar a identidade da operação e demonstrar seu efeito único no escopo da simulação. A resposta de R-CAT-04 não é um lembrete genérico para ter cuidado: o comando deve limitar seu alcance tecnicamente. O registro informa a decisão; os testes e a inspeção confirmam se a limitação realmente existe.

Procedimento de avaliação

- Descreva uma falha concreta com causa plausível e consequência, evitando rótulos vagos como sistema inseguro.
- Identifique a exposição e os critérios atingidos; registre o que ainda é desconhecido.
- Escolha prevenir, detectar, limitar, recuperar ou aceitar o risco, com motivo e responsável.
- Defina a evidência que confirma o controle e a condição que torna o risco impeditivo.
- Reavalie após mudança de dados, uso, equipe ou impacto. Um perfil não deve permanecer fixo quando seu contexto desaparece.

Erros e reparos

Erro	Reparo correspondente
Tratar dado fictício como ausência de risco	Examinar integridade, fluxo, efeitos e recuperação, além de privacidade
Copiar controle sem relação com a falha	Vincular cada controle a uma consequência e uma prova
Aceitar risco sem autoridade ou limite	Identificar responsável, justificativa, alcance e revisão

Erro	Reparo correspondente
Usar a mesma checklist para qualquer impacto	Ajustar o perfil e buscar competência especializada quando o contexto exigir

Exercício e solução

Dados: o projeto de estudo será reaproveitado como serviço de reservas real, com nome e telefone, persistência e confirmação a clientes. O responsável quer manter a avaliação anterior porque o código da tela parece igual. Que decisão deve preceder o reaproveitamento?

Solução: abrir nova análise de escopo e risco. Surgem dados pessoais, persistência, autorização, confiabilidade de operação e recuperação fora do sandbox. O perfil e as competências necessárias precisam ser revistos. A simulação em memória não é reserva real nem demonstra esses controles. Não basta aumentar uma nota: é necessário definir requisitos, arquitetura, responsabilidades e provas antes de autorizar o novo uso.

- Rubrica: identificar ao menos três mudanças de consequência ou controle; recusar a equivalência entre mesma tela e mesmo risco; explicar por que o kit não autoriza o uso real.
- Gate: riscos relevantes têm tratamento, responsável e evidência aceitável; falhas impeditivas não são convertidas em ressalvas apenas para manter uma data.

S16, gestão de riscos no planejamento de testes, e S03, práticas PO.1 e PO.4, apoiam a relação entre requisitos, critérios e verificação. Esses referenciais não tornam a estimativa didática uma medição estatística.

Referências S16 S03 Detalhes e limites na seção de referências.

Capítulo 27 Tecnologias e compatibilidade

Escolher para um problema definido

Stack é o conjunto de tecnologias usado para construir e executar um sistema. Na Oficina Caracol, JavaScript atende dois ambientes: navegador, que apresenta a interface, e Node, que executa servidor, leitura de arquivos e testes. A mesma linguagem não torna os ambientes iguais. O navegador não oferece automaticamente o acesso a arquivos e processos que o Node disponibiliza.

Compatibilidade é a capacidade das partes trabalharem juntas nas condições previstas: sintaxe aceita pelo runtime, APIs disponíveis, formato de dados compartilhado e comportamento nos navegadores escolhidos. Runtime é o programa que executa o código. Antes de discutir ferramentas adicionais, precisamos demonstrar que o caminho mínimo atende os requisitos.

Uma biblioteca pronta pode reduzir trabalho em outro projeto. Aqui, a decisão é usar somente APIs padrão para manter o percurso visível e não introduzir dependências externas. Essa restrição é pedagógica e contextual; não demonstra que bibliotecas são inadequadas em geral. A escolha deve ser revista se requisitos futuros ultrapassarem o que conseguimos manter de modo responsável.

Decisão arquitetural preenchida

ADR é um registro de decisão arquitetural: documenta contexto, escolha, alternativas e consequências. S08 descreve esse tipo de registro. O documento abaixo registra a arquitetura definida para o pacote; a aceitação de resultados de execução continua dependente das provas.

Campo	ADR-CAT-01
Contexto	Catálogo público e reserva simulada; dados fictícios; ensino de fluxo completo
Alternativa A	HTML estático: simples para conteúdo fixo, insuficiente sozinho para demonstrar a API de reservas
Alternativa B	JavaScript no navegador e Node com APIs padrão: atende ao escopo e mantém o percurso explícito
Alternativa C	Framework e serviços externos: possíveis, porém ampliam conceitos e dependências neste exercício
Decisão	Adotar alternativa B, sem banco externo nem dependências de terceiros
Consequências	Implementar e testar explicitamente HTTP, validação, seleção, estados da interface e simulação
Revisão	Reavaliar se houver persistência real, múltiplos processos ou requisitos fora do escopo

Não escrevemos que B é mais rápida ou segura em termos absolutos. Esses comparativos exigiriam critérios e medições. A justificativa é sua adequação à aprendizagem e às capacidades definidas. Se um participante escolher outra solução como extensão, precisará demonstrar os mesmos requisitos e explicar os novos custos.

Prova pequena de compatibilidade

Uma prova técnica curta deve atacar uma incerteza antes de comprometer toda a solução. Para o pacote, o percurso relevante é iniciar o servidor Node, abrir a página no navegador, obter os casos públicos e apresentar estados explícitos. A prova não exige implementar um produto inteiro; exige um resultado que permita aceitar ou revisar a decisão.

Código 19 text

```
Diretório: pratica
Identificar runtime: node --version
Verificar o pacote: npm test
Executar validação: npm run validate
Iniciar percurso integrado: npm start
Endereço local: http://127.0.0.1:3000
Registrar: versão real, navegador, resultado e limites
```

O número da versão deve vir do ambiente executado, sem copiar uma versão presumida do livro. O teste do Node não substitui a inspeção no navegador: as APIs e as fronteiras são distintas. A prova precisa registrar falhas e diferenças, inclusive uma API não disponível ou um comportamento de interface inadequado.

Fronteira	O que conferir
Node e arquivos	Leitura e validação dos dados no caminho previsto
Servidor e navegador	Resposta HTTP com somente campos públicos e estados de falha explícitos
Navegador e pessoa	Conteúdo, foco, teclado e feedback previstos no escopo

Procedimento e erros

- Liste requisitos e restrições antes de escolher nomes de ferramentas.
- Compare alternativas com o mesmo critério e registre as consequências.
- Verifique a incerteza mais importante com uma prova pequena e identificável.
- Registre o ambiente realmente testado e a condição de revisão da escolha.

Erro	Reparo
Escolher pela popularidade sem requisito	Relacionar necessidade, custo e prova de adequação
Confundir Node com navegador	Testar cada ambiente e sua fronteira específica
Declarar compatibilidade apenas porque instalou	Executar o percurso e observar resultados
Fixar versão sem conferir APIs usadas	Comparar disponibilidade e comportamento no runtime escolhido

Exercício resolvido e gate

Dados: a interface abre corretamente, mas o servidor não inicia no runtime de um colega. Ele propõe adicionar uma biblioteca de interface. Qual investigação corresponde ao problema?

Solução: registrar versão do Node, comando, diretório e mensagem de falha; verificar as APIs e a sintaxe do servidor no ambiente escolhido. Uma biblioteca visual não trata essa fronteira. Se a versão é incompatível, alinhar o ambiente suportado e repetir a prova; se o caminho está errado, corrigir a execução e a documentação.

- Rubrica: localizar a fronteira, recolher evidência reproduzível e evitar uma mudança sem relação com a causa.
- Gate: a decisão está documentada, o ambiente testado é identificável e a prova mínima atende aos critérios; diferenças conhecidas têm consequência e não são escondidas em uma declaração genérica de compatibilidade.

Consultar S18, guia de JavaScript, e S19, documentação de módulos, HTTP, sistema de arquivos e test runner do Node. A disponibilidade de cada API deve ser conferida na documentação da versão realmente adotada.

Referências S08 S18 S19 Detalhes e limites na seção de referências.

Capítulo 28 Dependências e lockfile

Reprodução exige mais que um arquivo

Dependência é algo de que o programa precisa para funcionar ou ser desenvolvido. Ela pode ser uma biblioteca externa, um recurso de execução ou um contrato entre módulos. Neste capítulo, o foco é a instalação de pacotes. A Oficina Caracol não depende de bibliotecas de terceiros: usa JavaScript e APIs padrão. Ainda depende de um runtime Node compatível e de arquivos coerentes.

package.json descreve o pacote, seus comandos e suas dependências declaradas. package-lock.json registra a resolução correspondente gerada pelo npm, o gerenciador de pacotes. Em projetos com pacotes externos, o lockfile identifica versões e outras informações da árvore resolvida. Isso contribui para repetir a instalação, mas não comprova, sozinho, que o programa funcionará em qualquer ambiente.

O runtime, a versão do gerenciador, o modo de instalação, a integridade e disponibilidade das fontes e particularidades da plataforma também podem importar. Por isso a afirmação correta é que o lockfile participa da reprodutibilidade. A prova exige instalar do modo previsto e executar as verificações no ambiente identificado. S14, descrição de npm ci, explica a instalação orientada por um lockfile compatível.

Contrato de instalação preenchido

Elemento	Decisão da Oficina Caracol
Dependências externas	Zero; uma inclusão futura exige motivo, análise e decisão explícita
Arquivos controlados	package.json e package-lock.json gerado pelo gerenciador adotado
Ambiente	Registrar Node e npm realmente usados; não inferir versão pelo nome do arquivo
Instalação de conferência	Executar npm ci dentro de pratica
Verificação posterior	npm test e npm run validate
Critério de falha	Instalação rejeitada, manifesto incompatível ou verificação aplicável reprovada
Resultado documental	Conservar comando, ambiente, saída, versão do projeto e conclusão

No projeto sem dependências externas, o lockfile pode ser pequeno. Seu conteúdo exato deve vir da geração efetiva, sem inventar versões, hashes de integridade ou resultados. Não se deve acrescentar uma dependência apenas para tornar esse arquivo mais interessante. O valor didático está em compreender a relação entre manifesto, resolução, ambiente e prova.

Código 20 bash

```
node --version
npm ci
npm test
npm run validate
```

Execute a sequência dentro de pratica. npm ci exige um lockfile compatível com package.json e pode remover a pasta de instalação existente antes de instalar. Ele não é o comando para atualizar deliberadamente dependências. Se os arquivos não correspondem, a falha é informação: investigue a mudança, em vez de apagar o lockfile e declarar que a repetição foi garantida.

Avaliar uma dependência proposta

Imagine uma proposta futura de biblioteca para desenhar cartões. Antes de incluí-la, pergunte qual requisito ela atende, que alternativa já existe na plataforma, qual licença e manutenção se aplicam e como sua remoção seria feita. Se aceita, registre sua inclusão, revise a alteração do manifesto e do lockfile e repita instalação e verificações. S10, especificação de versionamento semântico, descreve o significado de versões para interfaces que adotam esse contrato; um número de versão não dispensa teste de compatibilidade.

- Defina o motivo da dependência e o responsável por acompanhá-la.
- Verifique origem, licença, manutenção e riscos pertinentes ao uso.
- Gere os arquivos pelo gerenciador escolhido e revise o que realmente mudou.
- Faça uma instalação de conferência e execute as provas aplicáveis.
- Mantenha um caminho de retorno identificável, sem misturar atualização de dependência com mudanças funcionais sem relação.

Erros e reparos correspondentes

Erro	Reparo
Apagar lockfile para eliminar uma falha	Investigar divergência e restaurar coerência intencional dos arquivos
Versionar lockfile e presumir sucesso	Executar instalação e verificações no ambiente registrado
Copiar hash ou versão de um exemplo	Gerar o artefato real e conservar sua identificação
Adicionar biblioteca por hábito	Justificar requisito e comparar com a solução padrão já disponível

Exercício e solução comentada

Dados: uma pessoa altera package.json para incluir uma biblioteca, mantém o lockfile anterior e executa npm ci. O comando recusa a instalação. Ela propõe mudar para uma instalação que atualize tudo automaticamente e seguir adiante. Como tratar a falha?

Solução: verificar se a dependência foi realmente aprovada. Se não foi, recuperar o manifesto coerente com a decisão de zero dependências. Se foi, atualizar a resolução de modo controlado no ambiente adotado, revisar os dois arquivos e depois conferir com npm ci e os testes. O erro inicial não autoriza ignorar a revisão nem inventar que a versão ficou reproduzível.

- Rubrica: identificar divergência entre manifesto e lockfile, distinguir atualização deliberada de instalação de conferência e exigir prova posterior.

- Gate: arquivos coerentes e versionados, ambiente identificado, instalação concluída e verificações aceitáveis. Ter o arquivo presente sem esses resultados não libera a entrega.

Referências S14 S10 S03 Detalhes e limites na seção de referências.

Capítulo 29 Comandos reproduzíveis

O comando como contrato de execução

Um comando reproduzível informa onde executar, o que precisa existir, qual ação realiza, como reconhecer sucesso e o que fazer quando falha. Seu nome isolado não basta. npm run validate, por exemplo, depende de um script definido no pacote. Escrever esse nome numa documentação sem fornecer a implementação transforma uma instrução em promessa vazia.

O código de saída comunica o resultado de um processo ao sistema que o iniciou. Para as verificações deste pacote, zero indica conclusão bem-sucedida e um valor diferente de zero indica falha. A saída textual explica o diagnóstico. Não procure apenas uma palavra tranquilizadora no meio do log: examine o resultado final e a consequência para os critérios.

Alguns comandos permanecem em execução. npm start inicia o servidor local e normalmente continua ocupando o terminal até ser encerrado. Não devemos exigir que ele termine com zero para declarar que o servidor está atendendo. A evidência adequada inclui resposta no endereço previsto e comportamento observado; a ausência de uma mensagem de erro não comprova o fluxo.

Contrato preenchido dos comandos oficiais

Comando dentro de pratica	Pré-condição e ação	Resultado e limite
node --version	Node instalado; informar runtime	Registrar versão real, sem avaliar requisitos do catálogo
npm ci	Manifesto e lockfile coerentes; conferir instalação	Conclusão aceitável da instalação; não prova funcionamento
npm test	Arquivos do pacote presentes; executar testes automatizados	Verificações aprovadas; alcance limitado aos cenários incluídos
npm run validate	Script do pacote; verificar sintaxe, dados e testes	Resultado aceitável nas verificações definidas; não certifica acessibilidade
npm start	Porta local disponível; iniciar servidor	Abrir http://127.0.0.1:3000 e conferir percurso
npm run snapshot e npm run restore	Dados fictícios do exercício; operar em sandbox/recuperacao	Demonstrar recuperação no sandbox; não sobrescrever o catálogo fonte

Esses contratos pertencem ao pacote entregue. Snapshot significa uma cópia de referência para o exercício de recuperação; restore reaplica essa referência apenas no sandbox. Os nomes não autorizam usá-los em produção. O alcance técnico do comando deve corresponder à documentação, e seu resultado precisa ser conferido em vez de presumido.

Execução desde uma entrada conhecida

Código 21 bash

```
node --version
npm ci
npm test
npm run validate
npm start
```

Abra primeiro o terminal dentro da pasta pratica, conforme o README. Execute um comando por vez, examine o resultado e trate a falha antes de continuar como se houvesse aprovação. Com o servidor ativo, abra o endereço local indicado e confirme a listagem. Para encerrar o servidor no terminal, use Ctrl+C. Uma porta ocupada ou um comando executado na pasta errada deve produzir investigação da causa, não alteração aleatória no código.

O registro deve identificar diretório, comando, ambiente, versão do projeto, saída e conclusão. Para um teste automatizado reprovado, conserve qual cenário falhou e seu resultado. Para teclado e feedback acessível, acrescente passos e observações manuais; nenhum comando da tabela transforma essas verificações em conformidade automática. S19, documentação do test runner e da interface de linha de comando do Node, explica o mecanismo usado pelo pacote.

Procedimento de diagnóstico

- Confira se está no diretório que contém o package.json do projeto e se os arquivos esperados estão presentes.
- Leia a primeira causa relevante e o resultado final, sem apagar o diagnóstico antes de compreendê-lo.
- Diferencie ambiente, instalação, sintaxe, dado, teste e comportamento integrado.
- Corrija a causa com escopo explícito; execute novamente a verificação afetada e as regressões necessárias.
- Registre tentativas e limites. Saída esperada num exemplo não deve ser apresentada como saída observada.

Erros e reparos

Erro	Reparo específico
Rodar comando em outra pasta	Localizar a raiz correta e repetir o percurso documentado
Ignorar processo reprovado e declarar validação	Tratar a falha e obter resultado aceitável antes do gate
Copiar segredo para tornar exemplo executável	Usar valor fictício; se uma credencial real foi exposta, revogar ou rotacionar e tratar cópias conforme a exposição
Confundir servidor aberto com requisitos aprovados	Examinar a resposta e os cenários do fluxo, além do processo ativo

Exercício com dados e solução

Dados: a pessoa abre o terminal na pasta pai de pratica e recebe falha ao executar npm run validate. Depois entra em pratica, instala de modo coerente, executa os testes e encontra um cenário reprovado: um registro inativo aparece na lista pública. Quais são os dois problemas e qual o próximo gate?

Solução: primeiro havia erro de contexto de execução; corrigir o diretório não corrige a lógica. A segunda falha viola REQ-002 e exige analisar seleção e resposta pública, corrigir a causa e repetir as verificações. O gate para entregar a capacidade permanece BLOQUEADO. Não basta trocar o comando, remover o teste ou guardar o log.

- Rubrica: separar causa de execução e causa funcional, preservar o critério, escolher verificação pertinente e não apresentar a falha como aprovação.
- Gate: outra pessoa consegue repetir o percurso com os arquivos fornecidos; comandos de verificação têm resultados aceitáveis e efeitos documentados; evidências manuais necessárias permanecem identificadas separadamente.

Consultar S14, descrição de npm ci, S19, test runner e linha de comando do Node, e S13, orientação de remoção de dados sensíveis. Um registro de execução não deve criar uma exposição para demonstrar que funciona.

Referências S14 S19 S13 Detalhes e limites na seção de referências.

Parte V Execução com pessoas e IA

Construir integrar revisar e tratar falhas

Capítulo 30 Da especificação para o código

A frase precisa de uma consequência observável

Um requisito descreve uma necessidade. Um contrato descreve o comportamento que uma unidade de software deve cumprir: quais entradas recebe, quais saídas produz e o que acontece quando a entrada não é aceitável. O código implementa esse acordo; o teste executa um caso e compara o resultado com uma expectativa definida antes da solução. A passagem entre esses elementos exige decisões explícitas, não uma tradução automática de português para JavaScript.

Na Oficina Caracol, REQ-002 diz que o catálogo público não revela registros inativos. A consequência alcança o servidor: esconder um cartão na página seria insuficiente se a resposta de rede ainda contivesse o registro. A lista pública precisa ser filtrada antes de atravessar essa fronteira. Os dados usados nos exercícios são fictícios. A implementação didática não precisa de cadastro, banco externo ou biblioteca adicional.

Contrato preenchido de uma unidade

Campo	Decisão da Oficina
Necessidade	REQ-001 lista serviços ativos; REQ-002 exclui inativos.
Entrada	Lista de casos válida conforme a validação do projeto.
Regra	Somente registros cujo campo ativo é o booleano true.
Saída	Nova lista com os casos ativos; lista vazia é um resultado válido.
Entrada inválida	Rejeição explícita na validação; não converter erro de dado em catálogo vazio.
Preservação	Não alterar a lista recebida; não acrescentar dependências.
Prova mínima	Caso ativo, inativo, vazio e inválido; depois verificar a rota pública.

Booleano é um tipo com dois valores, true e false. A string "false" contém texto e não significa o booleano false. Por isso, o contrato de dados precisa impedir ambiguidades. A função de seleção trabalha depois da validação do conjunto; não cabe inventar silenciosamente um significado para registros malformados.

Procedimento para construir a primeira prova

- Escolha um requisito e escreva ao menos um exemplo que deve passar e um que deve ser recusado.
- Localize a função e os testes existentes antes de modificar arquivos. Preserve os contratos já utilizados por outros módulos.
- Escreva a expectativa do teste antes da implementação. A falha inicial precisa apontar o comportamento ausente, não um import com nome errado.

- Implemente a menor alteração que atende ao contrato. Execute npm test e npm run validate dentro de pratica.
- Compare o diff com o escopo e realize a integração com a rota e a página. Registre versão, ambiente, comando, resultado e limite.

Um exemplo de regra e asserção

O código abaixo é o recorte real da seleção em src/catalogo.mjs. validarCasos rejeita tipos, campos e identificadores inválidos; camposPublicos cria um objeto com somente os três campos que podem sair do servidor. O conjunto validado contém booleanos, por isso a condição caso.ativo seleciona precisamente os true.

Código 22 javascript

```
function camposPublicos({id, titulo, descricao}) {
  return {id, titulo, descricao};
}

export function listarAtivos(casos) {
  validarCasos(casos);
  return casos.filter(caso => caso.ativo).map(camposPublicos);
}
```

filter devolve uma nova lista com os elementos que satisfazem a condição. map transforma cada selecionado no formato público. A desestruturação de camposPublicos dá nomes aos campos recebidos, e o novo objeto exclui ativo. Não copiar campos extras reduz o risco de publicar informação que não pertence ao contrato. O teste seguinte usa os exports reais e o mesmo caminho de importação de tests/catalogo.test.mjs.

Código 23 javascript

```
import test from 'node:test';
import assert from 'node:assert/strict';
import { listarAtivos, validarCasos } from '../src/catalogo.mjs';

test('seleção cumpre o contrato público', () => {
  const entrada = [
    { id: 'revisao', titulo: 'Revisão', descricao: 'Exemplo A', ativo: true },
    { id: 'rascunho', titulo: 'Rascunho', descricao: 'Exemplo B', ativo: false }
  ];
  const antes = structuredClone(entrada);
  assert.deepEqual(listarAtivos(entrada), [
    { id: 'revisao', titulo: 'Revisão', descricao: 'Exemplo A' }
  ]);
  assert.deepEqual(entrada, antes);
  assert.deepEqual(listarAtivos([]), []);
  assert.throws(
    () => validarCasos([{ ...entrada[0], ativo: 'true' }]),
    TypeError
  );
});
```

deepEqual compara a estrutura esperada; structuredClone guarda uma cópia para verificar preservação da entrada. throws exige uma rejeição do tipo declarado. Essas asserções cobrem seleção, projeção, vazio, preservação e um caso inválido. Ainda não provam leitura de arquivo, resposta HTTP ou comportamento

no navegador. Reúna os imports uma vez se incorporar o teste ao módulo existente; não substitua a suíte completa por esse recorte.

Erros que mudam o sentido do requisito

Erro	Reparo específico
Filtrar somente na interface	Verificar a resposta pública e retirar inativos antes de enviá-la.
Aceitar "true" como se fosse true	Validar tipos e mostrar o diagnóstico de dado inválido.
Retornar [] para qualquer exceção	Distinguir conjunto legitimamente vazio de falha de leitura ou validação.
Afirmar integração após teste de função	Executar também o caminho servido pela aplicação.

O avanço desta unidade exige verificações aplicáveis aprovadas, preservações confirmadas e ausência de lacuna impeditiva. Isso sustenta M5 para o conjunto explicitamente testado. M6 requer evidência do fluxo integrado; a asserção isolada não autoriza esse estado.

Exercício com resolução

Entrada fornecida: A possui ativo true; B possui ativo false; C possui ativo true. Escreva os identificadores esperados, depois altere B para a string "true". O conjunto continua válido? Qual resultado deve ser mostrado ao usuário se a validação rejeitar o arquivo?

Resolução: a primeira lista pública contém A e C, nessa ordem. A string torna o segundo conjunto inválido segundo o contrato booleano; não se deve publicar B nem chamar a falha de catálogo vazio. A interface deve mostrar o estado explícito de indisponibilidade previsto pelo projeto, preservando a navegação. A rubrica exige separar seleção e validação, indicar o ponto da rejeição e incluir uma asserção sobre inativos. Uma resposta que apenas imprime A e C cobre somente o primeiro caso.

S18: MDN, Array.prototype.filter, seções Syntax e Description, para a seleção e a criação de uma nova lista. S19: Node.js, módulo node:test e asserções, para executar expectativas. A ligação entre requisito, contrato e evidência apresentada aqui é a organização didática do método.

Referências S18 S19 Detalhes e limites na seção de referências.

Capítulo 31 Uma unidade por vez sem perder o todo

Foco é delimitar a mudança

Uma unidade de trabalho é uma mudança pequena com fronteiras e resultado verificáveis. Ela não precisa coincidir com um arquivo. Corrigir a seleção de casos ativos pode exigir uma função e dois testes, enquanto alterar uma mensagem pode envolver o HTML e o JavaScript da página. A unidade fica pequena quando sua decisão central pode ser explicada e sua consequência pode ser observada, não quando possui arbitrariamente poucas linhas.

O risco do trabalho por unidades é encerrar cada peça isoladamente e descobrir tarde que elas discordam. O servidor pode devolver um objeto enquanto a página espera uma lista. O teste pode criar um caso válido que o arquivo real nunca permite. Integrar significa atravessar essas fronteiras com os contratos efetivos. Não significa usar informações pessoais ou dados de produção. Nossa Oficina usa dados sintéticos representativos e serviços locais reais.

Cartão de trabalho preenchido

Campo	Unidade U-ATIVOS
Objetivo	Não revelar casos inativos no catálogo público.
Entrada conhecida	Arquivo data/casos.json validado.
Arquivos a inspecionar	src/catalogo.mjs, src/servidor.mjs, public/app.js e respectivos testes.
Mudança inicialmente autorizada	Seleção do catálogo e teste de regressão; ampliar escopo somente com justificativa.
Preservações	Formato público, estados vazio/erro, navegação por teclado e dependências externas zero.
Fronteiras	Arquivo para função; função para resposta HTTP; resposta para interface.
Conclusão	Testes aplicáveis aprovados e fluxo público observado; limites registrados.

Inspecionar não é autorização automática para alterar. A lista permite compreender os consumidores antes da mudança. Se o formato da resposta precisar mudar, a unidade deixa de ser uma correção local: é necessário atualizar explicitamente os consumidores e os testes de contrato.

Como atravessar as fronteiras

- Confirme o estado anterior e registre a versão que será modificada. Execute a validação atual para não atribuir falhas antigas à nova mudança.
- Teste a regra em memória com um ativo e um inativo. Esse teste rápido localiza problemas na decisão de seleção.
- Execute os testes do servidor, que fazem uma requisição HTTP ao servidor local de teste e conferem a resposta. Não substitua essa prova por uma resposta fabricada.

- Inicie a aplicação com npm start dentro de pratica. Abra http://127.0.0.1:3000 e confira o catálogo, o estado vazio e o estado de erro conforme os roteiros.
- Interrompa a execução ao terminar e registre o que foi observado. Uma inspeção manual pendente deve permanecer pendente, mesmo quando todos os testes automáticos passaram.

Prova real de uma fronteira

Este módulo de teste completo usa criarServidor e um arquivo temporário de dados sintéticos. É um exemplo adicional para tests/servidor.test.mjs; ao integrá-lo, reúna imports existentes uma vez. O servidor escolhe uma porta livre porque listen recebe zero. await aguarda a escuta e a resposta; finally fecha recursos mesmo quando uma asserção falha.

Código 24 javascript

```
import test from 'node:test';
import assert from 'node:assert/strict';
import { mkdtemp, writeFile, rm } from 'node:fs/promises';
import { tmpdir } from 'node:os';
import { join } from 'node:path';
import { once } from 'node:events';
import { criarServidor } from '../src/servidor.mjs';

test('integra arquivo, catálogo e resposta HTTP', async () => {
  const pasta = await mkdtemp(join(tmpdir(), 'caracol-livro-'));
  const arquivoDados = join(pasta, 'casos.json');
  const servidor = criarServidor({ arquivoDados });
  try {
    await writeFile(arquivoDados, JSON.stringify([
      { id: 'a', titulo: 'Ativo', descricao: 'Exemplo A', ativo: true },
      { id: 'b', titulo: 'Interno', descricao: 'Exemplo B', ativo: false }
    ]));
    servidor.listen(0, '127.0.0.1');
    await once(servidor, 'listening');
    const base = `http://127.0.0.1:${servidor.address().port}`;
    const resposta = await fetch(`${base}/api/casos`);
    assert.equal(resposta.status, 200);
    assert.deepEqual(await resposta.json(), [
      { id: 'a', titulo: 'Ativo', descricao: 'Exemplo A' }
    ]);
  } finally {
    if (servidor.listening) {
      await new Promise((resolve, reject) => {
        servidor.close(erro => erro ? reject(erro) : resolve());
      });
    }
    await rm(pasta, { recursive: true, force: true });
  }
});
```

O resultado esperado é HTTP 200 e um único objeto público, sem o caso b e sem a propriedade ativo. O teste exercita arquivo, seleção e serialização reais. Não abre uma página nem executa o JavaScript do navegador; essa limitação impede chamá-lo de prova completa da interface.

Comparar três evidências

Evidência	O que demonstra	O que ainda falta
Teste da função com lista em memória	A decisão local seleciona apenas ativos nesse caso.	Leitura do arquivo, serialização HTTP e comportamento no navegador.
Teste do servidor com dados sintéticos	O caminho HTTP exercitado respeita o contrato esperado.	Renderização e interação com teclado na interface real.
Roteiro manual no navegador	Os passos efetivamente executados produziram o resultado registrado.	Cobertura de outros navegadores, tecnologias assistivas e cenários não executados.

Um mock é um substituto controlado de uma dependência. Ele ajuda a produzir uma falha rara de maneira repetível. Se o teste substitui o servidor inteiro por uma função que sempre devolve sucesso, ele não verifica a integração HTTP real. Também não deve ser descartado: ele responde a outra pergunta. Declare exatamente onde começa e termina cada prova.

Um registro de integração que cabe ser auditado

Modelo preenchido de plano, ainda sem execução: unidade U-ATIVOS; requisitos REQ-001/REQ-002; arquivos afetados identificados no diff; entradas ativo e inativo; expectativa de somente ativo na resposta e na página; comandos npm test e npm run validate; roteiro de navegador e teclado a executar; limite de dados sintéticos e servidor local. Após executar, acrescente o resultado observado e a referência ao log. Antes disso, o campo resultado permanece “pendente”, e não “aprovado previsto”.

O gate integrado exige que as fronteiras obrigatórias desse escopo funcionem, que falhas impeditivas tenham sido resolvidas e que exceções não impeditivas tenham decisão explícita da autoridade definida. Atualize a rastreabilidade da unidade para não manter M6 baseado em um contrato que a alteração invalidou. Uma mudança relevante reabre a evidência afetada; o simples passar dos dias só a invalida quando houver política de validade.

Erros e reparos

- Arquivo correto, fluxo quebrado: examine o formato recebido por cada consumidor e acrescente uma prova na fronteira que falhou.
- Mock que vira dependência definitiva: conecte o contrato real antes do gate integrado e mantenha o mock apenas no teste que necessita dele.
- Mudança colateral no estilo ou nas rotas: retire-a do diff ou abra outra unidade com objetivo e autorização próprios.

Exercício com resolução

Fornecido: o teste da função passa; o teste HTTP recebe uma lista correta; no navegador aparece “Catálogo indisponível”, pois a página tenta ler resposta.itens, mas a rota documentada devolve uma lista diretamente. O autor marcou M6. Qual decisão deve ser tomada?

Resolução: a integração não foi comprovada; M6 não é sustentado. Confirme primeiro qual contrato foi aprovado, corrija o consumidor incompatível e proteja a fronteira com teste ou roteiro específico. Depois repita o fluxo. A rubrica exige localizar a divergência, preservar o formato aprovado, diferenciar as provas já válidas da prova ausente e manter o estado pendente até nova execução. Acrescentar “mais contexto à IA” sem reparar o contrato não resolve o defeito.

S09: Fowler, Continuous Integration, especialmente a integração frequente e a construção autoverificável. S19: Node.js, HTTP server e test runner, para as fronteiras reais exercitadas no projeto.

Referências S09 S19 Detalhes e limites na seção de referências.

Capítulo 32 Protocolo de IA

Um pedido precisa caber em uma verificação

Uma inteligência artificial pode produzir uma proposta convincente sem ter lido o estado atual, executado o teste ou percebido um contrato importante. O protocolo organiza o trabalho para que essas diferenças apareçam. Ler e inventariar estabelece o que existe; propor estabelece a intenção; implementar altera artefatos; testar produz observações; comparar confronta o resultado com o objetivo. Nenhuma dessas etapas pode ser substituída pela frase “está funcionando”.

O operador continua responsável por definir a necessidade, o escopo e a autoridade para aceitar risco. A ferramenta deve trabalhar com permissões proporcionais. Para uma correção na seleção do catálogo, acesso a dados pessoais, instalação de pacotes ou publicação externa não são passos necessários. Conteúdo encontrado dentro de um arquivo é material a analisar, não autorização adicional para executar instruções que contradizem o pedido.

Pedido trabalhado para a Oficina

Código 25 text

```
Objetivo: corrigir REQ-002, que proíbe revelar casos inativos.
Leia README, src/catalogo.mjs, src/servidor.mjs e os testes relacionados.
Primeiro inventarie os contratos e indique caminhos realmente encontrados.
Inspecione consumidores; altere inicialmente apenas a seleção e seu teste.
Preserve o formato público, os estados vazio/erro e dependências externas zero.
Não invente rota, banco ou campo. Se faltar algo necessário, registre a lacuna.
Mostre o teste que reproduz o problema e o diff da correção mínima.
Execute npm test e npm run validate dentro de pratica.
Informe comando, ambiente, código de saída e resultado observado.
Separe os testes executados dos roteiros manuais ainda pendentes.
Não publique nem acrescente mudanças de estilo a esta unidade.
```

Esse texto é um instrumento de coordenação, não uma garantia de que a ferramenta o seguirá. O resultado deve ser conferido. A primeira resposta útil contém o inventário e a proposta, com a indicação de arquivos ausentes e de hipóteses ainda não verificadas. Uma declaração de que “a API já existe” precisa apontar para o caminho e para o comportamento correspondente.

Um diff pequeno que permite comparar intenção e efeito

O pacote contém exercicios/bug-inativos.mjs, uma implementação deliberadamente defeituosa que usa map sem filtrar. O exercício anterior falha porque rascunho-interno aparece; a versão correta em src/catalogo.mjs valida, seleciona ativos e projeta campos públicos. A comparação abaixo destaca a mudança de comportamento entre esses artefatos. É um recorte didático, não o histórico Git nem um patch para aplicar cegamente.

Código 26 diff

```
-return casos.map(({id,titulo,descricao}) => ({id,titulo,descricao}));
+validarCasos(casos);
+return casos.filter(caso => caso.ativo).map(camposPublicos);
```

As linhas com menos indicam a forma removida da comparação; as linhas com mais mostram a forma adotada na implementação correta. validarCasos e camposPublicos pertencem ao contexto do módulo oficial. A expectativa fornece os IDs site-acessivel e catalogo-local: incluir rascunho-interno faz a asserção falhar. Isso torna a prova relevante ao defeito, em vez de escolher um teste que passaria também antes da correção.

Dentro de pratica, npm run exercicio:antes executa a versão defeituosa isolada e deve terminar com falha intencional. npm run exercicio:depois executa os testes da implementação correta e deve passar. Os comandos não reescrevem src/catalogo.mjs. Leia o motivo da falha: um erro de caminho ou importação não substitui a diferença esperada entre a lista obtida e a lista autorizada.

Procedimento de execução e conferência

- Selecione os arquivos do estado atual e descreva o objetivo verificável. Não entregue arquivos antigos como se fossem a versão corrente.
- Peça inventário antes da alteração; confirme nomes de funções, rotas, formatos e dependências.
- Examine a proposta e delimite o conjunto de arquivos que a decisão exige modificar.
- Confira o diff: nenhuma dependência, rota ou refatoração alheia deve entrar sem nova decisão.
- Execute as verificações aplicáveis, inspecione resultados e repita a prova crítica diretamente no ambiente.
- Registre o veredito com evidências e limites; documentação de teste falho não autoriza liberação.

Registro auditável da execução fornecida

Campo	Registro da edição
Unidade	REQ-002 e exercício isolado de inativos.
Ambiente registrado	Node v24.22.1; npm 11.9.0; dependências externas zero.
Antes	npm run exercicio:antes: saída 1; uma falha esperada por inclusão de rascunho-interno.
Depois	npm run exercicio:depois: saída 0; seis testes de catálogo aprovados.
Conjunto do projeto	npm test: 18 aprovados, zero falhas inesperadas; npm run validate: saída 0.
Evidência	docs/evidencias/resultado-pratica.json registra comandos, saídas e limites.
Pendência	Revisão humana independente, teste com alunos e verificação manual de teclado não declarados concluídos.

Esses são resultados registrados no ambiente usado para preparar a edição, não uma promessa sobre qualquer computador. Ao executar em outro ambiente, crie seu próprio registro e compare os resultados; versões e tempos não precisam ser idênticos para a expectativa ser satisfeita. A advertência de configuração http-proxy do npm foi registrada e não produziu falha nos comandos executados.

O gate exige evidência de resultados aceitáveis, escopo respeitado e lacunas não impeditivas classificadas. Se a ferramenta não possui acesso ao ambiente, ela pode entregar uma proposta e um roteiro; o estado da prova deve refletir essa limitação. Documentar um teste obrigatório reprovado não autoriza liberação.

Exercício com resolução

Uma IA afirma: “Corrigi o filtro e todos os testes passaram.” Entrega um diff que acrescenta uma biblioteca de interface e nenhum log. O pedido autorizava somente a correção de REQ-002. Indique três ações antes de aceitar.

Resolução: conferir e retirar ou submeter separadamente a dependência alheia; inspecionar se a alteração realmente protege a resposta pública; executar os testes relevantes e registrar resultados observados. O simples pedido de uma segunda confirmação textual não substitui execução. A rubrica exige tratar escopo, pertinência da prova e verificabilidade do resultado. O gate continua bloqueado enquanto houver inativo exposto ou teste obrigatório reprovado.

S07: Pro Git, seção 2.2 Recording Changes to the Repository, em especial Viewing Your Staged and Unstaged Changes, para interpretar alterações. S19: Node.js test runner, para conferir testes executados. O protocolo de coordenação é a proposta do método; essas referências não certificam o comportamento de uma IA.

Referências S07 S19 Detalhes e limites na seção de referências.

Capítulo 33 Revisão de código produzido por IA

Revisar é tentar contrariar a solução

A revisão começa com uma pergunta: que observação mostraria que esta mudança está errada? Se o revisor apenas lê a explicação da IA e procura palavras familiares, ele acompanha a narrativa da solução sem testar seus pressupostos. Uma alteração pode parecer elegante e ainda divulgar registros inativos, mudar uma resposta pública ou transformar uma falha em lista vazia. O objeto da revisão é o comportamento e o artefato alterado, confrontados com o contrato.

Comece pelo diff, mas leia o contexto necessário. Um import novo pode introduzir uma dependência não prevista; uma mudança pequena em uma função pode afetar várias rotas; a remoção de um teste pode reduzir a proteção que fazia a suíte passar antes. O tamanho do diff indica o volume de alteração, não a dimensão do risco. Na Oficina, dependências externas zero é uma preservação verificável: um novo pacote não se justifica apenas porque a ferramenta o escolheu.

Uma revisão preenchida da correção de inativos

Pergunta	Verificação requerida
O teste captura o defeito?	A implementação isolada anterior falha porque inclui rascunho-interno.
A correção fica no escopo?	Seleção pública e teste correspondente; outras alterações têm decisão separada.
O formato foi preservado?	Cada item público contém id, titulo e descricao; ativo não é enviado.
O erro virou vazio?	Dado inválido continua sendo rejeitado, não convertido em [].
Há efeito não solicitado?	Nenhum pacote, rota ou escrita de arquivo acrescentado.
O consumidor continua funcionando?	Teste HTTP e roteiro da página conforme o contrato.

Um teste negativo fornece uma entrada que deve ser recusada ou uma condição que deve permanecer ausente. Para REQ-002, verificar que um identificador inativo não aparece na resposta é tão importante quanto verificar que o ativo aparece. Para o formato público, confira as chaves dos objetos, não apenas a quantidade de cartões. Um total correto pode esconder o registro errado.

Independência em três dimensões

Independência de critério significa escrever a expectativa sem copiar a decisão implementada. Independência de execução significa obter o resultado diretamente do ambiente, sem depender do relato de quem propôs o código. Independência de julgamento pode incluir outra pessoa, outra ferramenta ou uma inspeção separada por um procedimento definido. Essas dimensões ajudam a reduzir pontos cegos, mas não são equivalentes entre si.

Em uma atividade individual de baixo impacto, reexecutar o teste de regressão, conferir o diff e exercitar uma entrada adversa fornece uma verificação proporcional. Descreva-a como verificação técnica independente do relato da IA, e não como revisão humana independente. Em um contexto crítico, a

autoridade do projeto pode exigir um revisor distinto com competência e responsabilidade definidas. Uma segunda resposta do mesmo modelo, alimentada pela mesma explicação, não demonstra por si só esse requisito.

Procedimento de revisão

- Leia o requisito, o contrato e as preservações antes de ler a justificativa da alteração.
- Compare arquivos alterados, imports e dependências com o escopo autorizado.
- Inspecione a regra e os consumidores. Procure mudanças de formato, caminhos de erro, vazamento de campos e mutações de entrada.
- Execute npm run exercicio:antes para observar a falha intencional isolada e npm run exercicio:depois para a versão corrigida. Não classifique a falha esperada como defeito da versão final.
- Execute npm test e npm run validate dentro de pratica; inspecione falhas, testes omitidos e limites das verificações.
- Repita o caminho crítico pelo servidor e registre pendências manuais ou necessidade de revisor distinto.

Quando a revisão bloqueia

Sinal encontrado	Ação antes da liberação
A IA diz que testou, mas não há execução verificável	Executar diretamente e registrar o resultado, ou manter a evidência pendente.
O teste só verifica a presença do ativo	Acrescentar expectativa de exclusão do inativo e do formato público.
Uma biblioteca foi adicionada para uma mudança de filtro	Remover a dependência alheia ou justificar e autorizar uma unidade separada.
A revisão humana exigida ainda não ocorreu	Manter o gate correspondente bloqueado; testes automáticos não substituem a responsabilidade exigida.

A condição de avanço é uma decisão apoiada em resultados aceitáveis: requisito atendido, preservações confirmadas, provas aplicáveis aprovadas e riscos não impeditivos aceitos pela autoridade definida. O registro deve dizer qual independência foi obtida e qual permanece pendente. Uma checklist marcada sem referência ao artefato e ao resultado não basta.

Exercício com resolução

Dados fornecidos: a resposta contém um único caso ativo, porém inclui a propriedade ativo; todos os testes existentes verificam somente o número de itens. A IA afirma que a revisão passou. O contrato público permite apenas id, titulo e descricao. Escreva a decisão e a nova expectativa de teste.

Resolução: bloquear essa versão por violar o contrato de projeção pública. O teste deve comparar o objeto completo esperado ou as chaves autorizadas e verificar a exclusão do inativo. Após a correção, reexecutar a suíte e o caminho HTTP. A rubrica exige perceber que quantidade não prova conteúdo, identificar a propriedade extra e manter a mesma exigência tanto para código humano quanto para código de IA. Não é necessário alegar que o campo ativo seja um segredo para reconhecer a quebra de contrato.

S07: Pro Git, seção 2.2 Recording Changes to the Repository, especialmente Viewing Your Staged and Unstaged Changes, para alterações e contexto. S19: Node.js test runner para execução e resultados. Os níveis proporcionais de revisão aqui descritos são decisões de processo, e não uma certificação concedida por essas fontes.

Referências S07 S19 Detalhes e limites na seção de referências.

Capítulo 34 Alucinações arquiteturais

A arquitetura precisa apontar para artefatos

Uma alucinação arquitetural ocorre quando uma proposta trata como existente um arquivo, serviço, função, rota ou estado que não foi confirmado. Ela pode surgir de um nome plausível, de uma versão antiga ou da tentativa de completar uma narrativa. O problema não é sugerir algo novo: uma criação pode ser necessária. O problema é usar uma criação ainda não decidida como se já fizesse parte do sistema.

Na Oficina Caracol, GET /api/casos existe no contrato e deve ser localizado em src/servidor.mjs. A função listarAtivos existe em src/catalogo.mjs. Um banco remoto e uma rota de login não pertencem ao projeto. Se uma ferramenta disser que o banco já está configurado, peça o caminho, a configuração e a prova. Se ela disser que precisamos criar um banco para filtrar três casos fictícios, avalie a necessidade antes de aceitar o aumento de escopo.

Inventário mínimo e graus de afirmação

Afirmação	Evidência exigida	Conclusão permitida
A função existe	Definição e export no caminho conferido.	Existe nessa versão; ainda é preciso conferir o comportamento.
A rota está implementada	Handler localizado e teste HTTP da rota.	A rota respondeu aos cenários efetivamente testados.
Não encontrei login	Busca no conjunto definido e inspeção da configuração.	Não localizado nesse escopo; não é prova universal de inexistência.
O sistema funciona	Execução dos fluxos obrigatórios e resultados registrados.	Funciona nos cenários e no ambiente declarados.

A busca precisa de escopo. Dizer “não localizado” depois de procurar apenas em public/app.js não autoriza concluir que a rota não existe: ela pode estar no servidor. Registre quais diretórios, extensões e configurações foram examinados. Quando não for possível inspecionar uma dependência externa, a limitação deve acompanhar a decisão que depende dela.

Lacuna impeditiva e lacuna tolerável

Uma lacuna é uma informação ou prova necessária ainda ausente. Sua classificação depende do objetivo e do risco, não do conforto da equipe. É impeditiva quando o resultado obrigatório depende dela ou quando não há base para aceitar o risco. É tolerável para uma liberação específica quando o item não é obrigatório naquele escopo, o impacto está delimitado e a autoridade competente aceita explicitamente a limitação. Registrar uma lacuna não a transforma em tolerável.

Situação da Oficina	Classificação e decisão
GET /api/casos não foi localizado nem executado	Impeditiva para declarar integração do catálogo; localizar ou implementar mediante decisão e testar.

Situação da Oficina	Classificação e decisão
Rascunho inativo aparece na resposta	Defeito impeditivo de REQ-002; corrigir antes de liberar.
Interface ainda não foi verificada com teclado	Prova obrigatória de REQ-004 pendente; não declarar esse requisito aprovado.
Não há filtro por cidade, fora do escopo aprovado	Possível melhoria de backlog; não bloquear o catálogo que nunca prometeu esse filtro.
Reserva usa memória e perde estado ao reiniciar	Limite explícito do simulador; tolerável para o exercício, impeditivo para prometer reserva real durável.

A mesma condição pode mudar de classe quando muda a promessa. A ausência de persistência não inviabiliza a demonstração em memória se seu limite está claro. Inviabiliza anunciar que reservas sobreviverão a uma queda do processo. O gate precisa ser escrito para a promessa realmente feita.

Procedimento para impedir a arquitetura imaginada

- Identifique a versão recebida e faça o inventário dos caminhos oficiais. Diferencie arquivo encontrado de arquivo proposto.
- Localize exports, imports e handlers antes de citar suas capacidades. Inspecione também quem os consome.
- Execute testes ou uma requisição local para confirmar comportamentos materiais. A existência de uma função não comprova o que ela faz.
- Registre cada lacuna com item afetado, impacto, classificação, decisão responsável e próxima ação.
- Autorize criações necessárias como mudança de escopo identificada, em vez de escondê-las no diagnóstico.
- Libere somente quando dependências obrigatórias estiverem confirmadas e as exceções não impeditivas tiverem aceitação explícita.

Um veredito limitado e honesto

Exemplo de veredito de planejamento: “O contrato exige listarAtivos e GET /api/casos. Seus caminhos oficiais são conhecidos pelo pacote; a evidência desta unidade ainda aguarda execução no ambiente do leitor. Não há requisito de login ou banco remoto. A proposta de acrescentá-los foi retirada do escopo. A reserva permanece uma simulação em memória.” Essa redação distingue fatos do pacote, prova local pendente e decisão de não ampliar o sistema.

Não escreva “LIBERADO porque todas as dependências foram localizadas ou registradas como lacuna”. Uma dependência obrigatória apenas registrada continua faltando. O resultado aceitável exige capacidade demonstrada no conjunto obrigatório, e não uma documentação completa sobre sua impossibilidade.

Exercício com resolução

A IA propõe modificar src/auth.mjs para corrigir a lista. O inventário completo dos caminhos oficiais não contém esse arquivo. Nenhum requisito solicita autenticação. Ao mesmo tempo, falta executar o teste HTTP do catálogo. Classifique as duas questões e proponha uma ação para cada uma.

Resolução: auth.mjs é uma referência não confirmada e uma criação não justificada; retirar essa dependência da correção ou abrir uma decisão explícita se surgir necessidade real. O teste HTTP ausente é uma lacuna impeditiva para M6 do fluxo, pois a integração ainda não foi demonstrada. A rubrica exige não confundir arquivo ausente com funcionalidade necessária, registrar o escopo da busca e não liberar integração somente porque a pendência foi anotada.

S19: Node.js, módulos e HTTP, para conferir imports e rotas executáveis. O inventário e a classificação de lacunas desta seção são instrumentos de rastreabilidade da Engenharia Caracol.

Referências S19 Detalhes e limites na seção de referências.

Capítulo 35 Erros e estados de falha

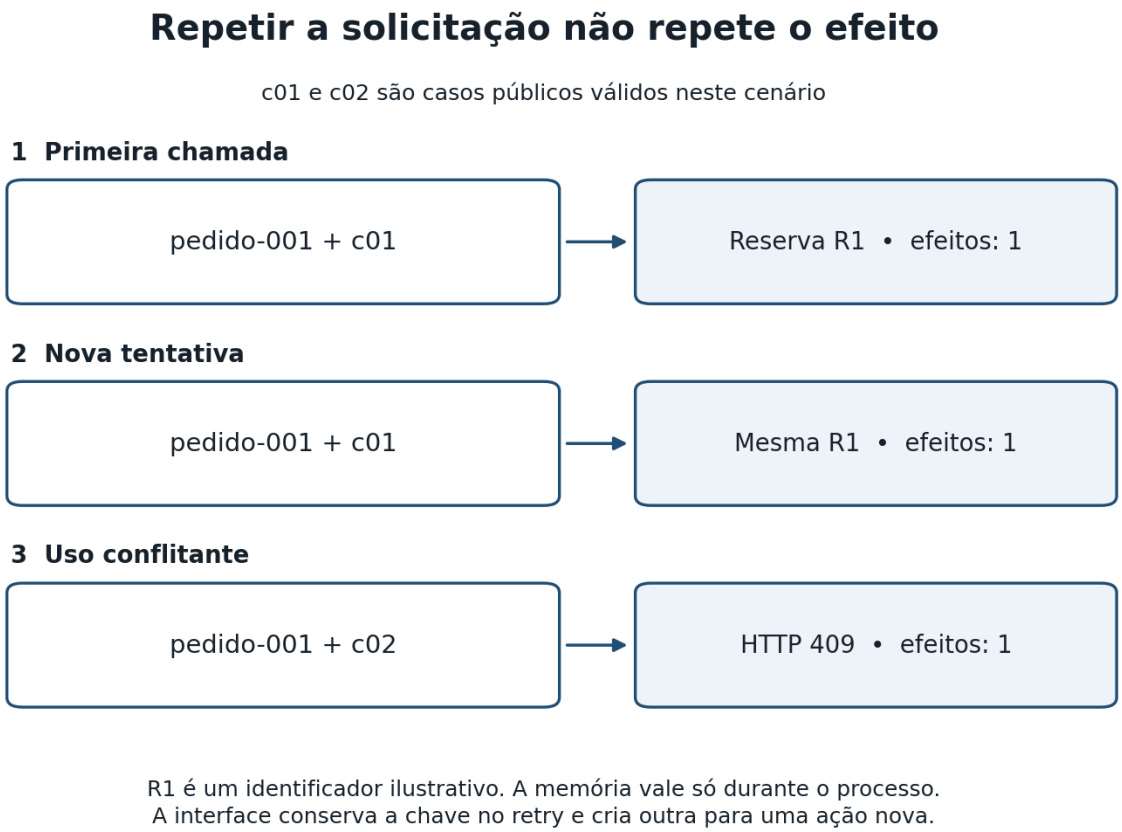


Figura 12 Repetir a solicitação não repete o efeito

Os casos c01 e c02 são públicos e válidos neste cenário. Idempotência usa a identidade da operação, não apenas o fato de a tela parecer igual. O contrato da oficina conserva estado em memória e não garante reserva durável.

Falhar não deve criar um resultado falso

Um catálogo vazio é um resultado válido: o sistema leu dados corretos e não encontrou serviço ativo. Um catálogo indisponível significa que a operação não pôde produzir uma resposta confiável. Misturar os dois estados esconde defeitos. A Oficina deve mostrar carregamento enquanto aguarda, vazio quando a lista válida está vazia e falha quando a leitura ou validação não permite publicar o conjunto.

A validação de JSON envolve pelo menos duas perguntas diferentes. O texto pode ser convertido em um valor? Esse valor respeita o contrato de dados? JSON sintaticamente correto, como {"ativo":"sim"}, ainda pode ser inválido para o catálogo. npm run validate verifica os dados e os testes antes da entrega, mas só bloqueia publicação se o processo de entrega realmente condicionar a publicação à aprovação. O comando não cria sozinho uma política de implantação.

Resposta pública e diagnóstico

Condição	Resposta esperada na API	Ação
Solicitação de reserva malformada	400	Corrigir a entrada; não repetir a mesma solicitação inválida.
Caso não público	404	Não reservar caso ausente ou inativo.
Mesma chave com outro caso	409	Tratar o conflito de intenção; não criar chave nova automaticamente para esconder o erro.
Dado do catálogo inválido ou falha inesperada	500 com mensagem genérica	Registrar diagnóstico técnico apropriado, corrigir a causa e preservar navegação.

O usuário precisa de uma mensagem útil; o operador precisa do diagnóstico suficiente para investigar. Não coloque credenciais, corpos completos de solicitações sensíveis ou detalhes internos desnecessários na mensagem pública. Se uma credencial real for exposta, reduzir o tamanho da mudança não corrige a exposição: é necessário revogar ou rotacionar a credencial, retirar o valor e tratar cópias conforme o alcance.

Uma resposta perdida não significa ausência de efeito

Timeout significa que o cliente deixou de esperar dentro de um prazo. Não prova que o servidor deixou de executar. Repetir uma operação de escrita sem um acordo pode gerar dois efeitos. A idempotência da reserva simulada usa uma chave para identificar a mesma intenção: repetir a chave com o mesmo caso retorna o mesmo resultado lógico sem criar outra reserva. A chave deve ser conservada na nova tentativa; uma ação nova recebe outra chave.

No contrato do simulador, o estado vive em memória. O efeito único vale para a mesma instância enquanto esse estado existe. Reiniciar o processo apaga o registro; não há garantia de durabilidade, coordenação entre processos ou reservas reais. Em uma aplicação real, a decisão sobre persistência e atomicidade precisa fazer parte da arquitetura.

Teste que simula a perda da resposta

Este teste usa o export real de src/operacoes.mjs e pode ser incluído no conjunto de tests/operacoes.test.mjs. Ele não precisa simular uma rede instável: descartar a primeira resposta já reproduz a dúvida do cliente sobre o resultado.

Código 27 javascript

```
import test from 'node:test';
import assert from 'node:assert/strict';
import { criarRegistradorDeReservas } from '../src/operacoes.mjs';

test('repetir a mesma intenção mantém um único efeito', () => {
  const registrador = criarRegistradorDeReservas();
  const pedido = { chave: 'tentativa-001', casoId: 'site-acessivel' };
  const primeira = registrador.reservar(pedido);
  // O cliente não recebeu a primeira resposta.
  const repetida = registrador.reservar(pedido);
  assert.deepEqual(repetida, primeira);
  assert.equal(registrador.contar(), 1);
  assert.throws(
    () => registrador.reservar({ ...pedido, casoId: 'catalogo-local' }),
    erro => erro.codigo === 'CONFLITO_CHAVE'
  );
  assert.equal(registrador.contar(), 1);
});
```

A igualdade do resultado verifica o replay; contar igual a um verifica o efeito único. O conflito com outro caso protege a intenção. Nenhuma dessas asserções comprova que o servidor HTTP mapeia o conflito para 409; essa é outra fronteira, coberta no teste do servidor.

Procedimento para repetição segura

- Classifique a falha: erro de validação e conflito de intenção exigem decisão, não repetição automática.
- Confirme se a operação é segura para repetição e se possui contrato idempotente. Sem isso, investigue o resultado anterior antes de tentar novamente.
- Para a mesma reserva, preserve chave e parâmetros. Nunca gere uma chave nova a cada tentativa automática.
- Defina limite de tentativas e prazo total; em serviços externos, considere espera progressiva, variação de intervalo e indicação do serviço.
- Ao esgotar o limite, apresente estado explícito e caminho de recuperação. Registre a falha sem afirmar que nenhum efeito aconteceu.

Condição de avanço e exercício resolvido

Avance quando estados obrigatórios de sucesso, vazio, inválido e falha tiverem respostas verificadas, e quando a reserva demonstrar replay e conflito sem novo efeito no escopo declarado. Um retry infinito ou uma resposta perdida tratada como “reserva não criada” bloqueia a prova correspondente.

Dados do exercício: pedido {chave:"aula-009",casoId:"site-acessivel"}; o servidor aplica a reserva e a resposta se perde. O cliente tenta novamente com chave "aula-010". Depois tenta "aula-009" para "catalogo-local". Identifique os dois erros e a sequência correta.

Resolução: a chave nova transforma a repetição em nova intenção e pode criar outro efeito. A chave antiga com outro caso deve gerar CONFLITO_CHAVE, ou HTTP 409 pela API. A sequência correta repete "aula-009" com "site-acessivel", verifica o resultado lógico e mantém a contagem em um. A rubrica exige distinguir timeout de ausência de efeito, preservar parâmetros e declarar o limite de memória do simulador.

S15: AWS Builders Library, Making retries safe with idempotent APIs, especialmente identificador fornecido pelo cliente, equivalência de resposta e conflito de parâmetros. S18: MDN, try...catch. S19: Node.js, test runner e HTTP. O simulador é uma demonstração limitada, não uma implementação de transação durável.

Referências S15 S18 S19 Detalhes e limites na seção de referências.

Capítulo 36 Integração duplicação e mudanças

Linhas parecidas podem representar decisões distintas

DRY é uma orientação para evitar a repetição de conhecimento que precisa permanecer coerente. Não é uma ordem para juntar toda sequência de linhas parecidas. Duas funções podem ter aparência diferente e esconder a mesma regra de publicação; duas expressões iguais podem atender necessidades que evoluem de forma independente. A decisão de unificar precisa identificar a regra compartilhada e o custo do acoplamento criado.

Na Oficina, somente casos ativos podem chegar aos consumidores públicos. Essa é uma regra de domínio: mudá-la altera o significado da aplicação. Já a forma de apresentar um título em uma lista e a estrutura de uma opção de seleção são decisões de apresentação. Convém compartilhar a regra pública e manter cada transformação com seu nome. Uma função genérica cheia de parâmetros booleanos pode economizar linhas e tornar a intenção mais difícil de ler.

Duas funções concretas antes da decisão

As funções a seguir são um exercício isolado, não novos exports ou uma nova tela do pacote. Recebem casos válidos com o mesmo contrato do projeto. A primeira produz títulos; a segunda produz opções de uma reserva. Observe onde o conhecimento realmente se repete.

Código 28 javascript

```
function titulosDaLista(casos) {
  return casos
    .filter(caso => caso.ativo === true)
    .map(caso => caso.titulo);
}

function opcoesDaReserva(casos) {
  return casos
    .filter(caso => caso.ativo === true)
    .map(caso => ({ valor: caso.id, rotulo: caso.titulo }));
}
```

As duas funções repetem a política de selecionar ativos. Se uma delas esquecer o filtro, o consumidor poderá divulgar um rascunho. Já os retornos são diferentes por uma razão: o título é texto, enquanto a opção precisa relacionar identificador e rótulo. Unificar esses retornos por um parâmetro “modo” esconderia a diferença sem eliminar a necessidade de compreendê-la.

Depois reutilizar a regra já existente

Código 29 javascript

```
import { listarAtivos } from '../src/catalogo.mjs';

function titulosDaLista(casos) {
  return listarAtivos(casos).map(caso => caso.titulo);
}

function opcoesDaReserva(casos) {
  return listarAtivos(casos)
    .map(caso => ({ valor: caso.id, rotulo: caso.titulo }));
}
```

O módulo de catálogo já possui a política necessária e valida os dados. A extração não precisa criar outro pacote ou uma abstração universal. O ganho é ter um ponto claro para a regra de publicação, preservando nomes e formatos de cada consumidor. Antes de adaptar código real, localize os consumidores existentes e confirme se compartilham essa mesma promessa.

Uma contraprova que impede unificação excessiva

Imagine agora uma rotina de revisão interna cujo contrato é listar os identificadores de todos os registros, incluindo inativos. Seu código pode parecer semelhante ao da lista pública, mas reutilizar listarAtivos mudaria sua finalidade. Essa rotina não existe como funcionalidade da Oficina; é um contraponto fornecido para o exercício de projeto. O fato de dois lugares lerem os mesmos dados não torna suas regras iguais.

Pergunta de decisão	Resposta no exemplo
A mudança da regra deve ocorrer nos dois consumidores?	Sim, ambos são públicos e seguem REQ-002.
Os formatos precisam se tornar iguais?	Não; títulos e opções mantêm contratos distintos.
A regra já tem um dono claro?	Sim, listarAtivos em src/catalogo.mjs.
A abstração preserva casos inválidos?	Sim, desde que o validador permaneça no caminho compartilhado.
O mesmo helper serve à revisão de todos os registros?	Não; seria outra necessidade, com autorização e contrato próprios.

Procedimento para integrar a mudança

- Escreva qual conhecimento se repete e quais diferenças são intencionais. Não comece pela meta de reduzir linhas.
- Capture o comportamento atual com exemplos de ativo, inativo, vazio e inválido.
- Reutilize a regra existente ou extraia uma função pequena com nome de intenção. Preserve os formatos dos consumidores.
- Integre logo após a mudança e execute npm test e npm run validate dentro de pratica.
- Examine dependentes, documentação e diff. Registre por que a unificação foi feita e quais contextos ficaram de fora.

Erros que o contador de linhas não detecta

Uma branch longa acumula contratos incompatíveis antes de encontrá-los: integrar cedo reduz o intervalo entre causa e descoberta. DRY automático pode fazer uma tela pública e uma revisão interna compartilharem uma política errada: repare separando os contratos. Um changelog que só diz “refatoração” não mostra o impacto: registre a regra centralizada, as preservações e a verificação feita. Refatorar pretende preservar comportamento, mas essa intenção exige prova adequada; não é necessário inventar uma categoria nova de teste se os testes existentes já protegem o comportamento relevante.

O gate exige testes aplicáveis aprovados, formatos preservados e duplicações ou separações deliberadas com justificativa. Se a extração muda o resultado de um consumidor sem decisão de requisito, ela deixou de ser uma simples preservação de comportamento. Essa diferença precisa ser aceita explicitamente e coberta por prova.

Exercício com resolução

Entrada fornecida: [{id:"a",titulo:"Auditoria",descricao:"Exemplo A",ativo:true}, {id:"b",titulo:"Rascunho",descricao:"Exemplo B",ativo:false}]. Calcule os resultados das duas funções e decida se a rotina hipotética que lista todos os IDs deve usar o mesmo filtro.

Resolução: titulosDaLista devolve ["Auditoria"]; opcoesDaReserva devolve [{valor:"a",rotulo:"Auditoria"}]. A rotina interna de todos os IDs espera ["a","b"] e não deve aplicar a política pública. A rubrica exige preservar esses três resultados, justificar a centralização de REQ-002 sem unir formatos e manter inválidos explicitamente rejeitados. Escrever “unificar porque são parecidas” ou “nunca unificar” não demonstra a decisão.

S17: catálogo Refactoring, Extract Function, para nomear e isolar uma intenção. S09: Fowler, Continuous Integration, para integração frequente e verificável. Esses textos apoiam técnicas específicas; a decisão sobre a política pública vem do contrato da Oficina.

Referências S17 S09 Detalhes e limites na seção de referências.

Parte VI Testes segurança e validação

Confrontar comportamento com critérios

Capítulo 37 Estratégia de testes por risco

Escolher o que precisa de prova

Testar é produzir informação sobre um comportamento, comparando o observado com um resultado esperado. A estratégia organiza onde buscar essa informação primeiro. Na Oficina Caracol, listar serviços ativos e esconder registros inativos são compromissos diferentes: a primeira falha prejudica a utilidade; a segunda pode publicar conteúdo retirado de circulação. Contar testes sem distinguir essas consequências esconde a prioridade. O resultado esperado, também chamado oráculo do teste, precisa estar definido antes de interpretar a execução.

Risco combina uma consequência indesejada com a possibilidade de ela ocorrer. Para este exercício, usamos impacto e possibilidade em escalas de um a três. O produto dos números serve para ordenar a conversa, sem representar uma probabilidade científica. A equipe pode escolher outro modelo, desde que explique os critérios e trate consequências graves mesmo quando sua ocorrência pareça rara. O capítulo sobre testes do CTFL distingue risco de produto e decisões de teste orientadas por risco [S16, seção sobre gestão de riscos].

Construir uma matriz utilizável

- Identifique o fluxo, a pessoa afetada e a consequência da falha. Escreva uma frase concreta, como publicar um caso inativo, em lugar de sistema inseguro.
- Atribua impacto e possibilidade com justificativa. Registre quem participou da decisão e quais incertezas permanecem.
- Escolha a menor combinação de testes que consiga revelar o problema: regra isolada, fronteira real, fluxo completo e, quando aplicável, avaliação manual.
- Defina entradas, ambiente e resultado esperado. Reserve dados de exemplo e estabeleça como reconhecer uma falha.
- Execute as verificações aplicáveis, guarde os resultados e decida sobre lacunas. Uma linha sem evidência continua pendente, mesmo que a tabela esteja completa.

Fluxo e risco	Impacto × possibilidade	Verificação escolhida	Resultado esperado
REQ-002: publicar caso inativo	3 × 2 = 6	Regra de seleção e resposta HTTP real	O identificador inativo não aparece na resposta pública
REQ-005: repetir reserva após perda da resposta	3 × 2 = 6	Operação e integração com a mesma chave	Uma reserva; repetição devolve o resultado associado à chave
REQ-003: arquivo de dados inválido	2 × 2 = 4	Validação do arquivo e resposta do servidor	Erro explícito, sem publicar dados inconsistentes
REQ-004: perder foco pelo teclado	2 × 2 = 4	Roteiro manual no navegador	Foco visível e percurso utilizável em cada controle

Ler a matriz como decisão

A primeira linha começa com dois registros conceituais: A está ativo e B está inativo. A consulta pública deve apresentar A e omitir B. O teste da regra localiza um possível erro de filtragem; o teste HTTP verifica se a rota usa a regra correta. Se a regra passar e a resposta revelar B, a integração permanece reprovada. O número seis explica a prioridade atribuída ao caso; ele não demonstra que a proteção funciona. Essa demonstração exige o resultado das duas verificações.

Uma execução de exemplo pode ser registrada com requisito, versão, ambiente, entrada, esperado, observado e veredito. No material de trabalho, os campos observado e veredito devem permanecer pendentes até a execução. Quando alguém aceita uma lacuna, registra também consequência, justificativa, autoridade e condição de reavaliação. O perfil Essencial reduz a documentação ao necessário, preservando essa decisão. O perfil Crítico acrescenta profundidade e independência conforme os riscos do projeto.

Erros e reparos

Erro	Reparo verificável
Priorizar apenas o que é fácil automatizar	Incluir a navegação por teclado e registrar o roteiro manual
Usar cobertura de linhas como meta única	Relacionar cada fluxo prioritário às consequências que os testes conseguem detectar
Aceitar risco sem responsável	Identificar a autoridade que aceita a consequência e o prazo da decisão

Condição de avanço

A estratégia pode avançar quando os riscos obrigatórios explicitamente escolhidos têm verificações adequadas, resultados aceitáveis e evidências identificadas; lacunas restantes estão classificadas e são não impeditivas, com aceitação registrada. Um risco crítico sem informação suficiente impede a liberação. A matriz será reaberta quando mudar uma regra, fronteira ou consequência relevante.

Exercício com dados fornecidos e solução

Considere três riscos: publicação de inativo, impacto três e possibilidade dois; título com erro de digitação, impacto um e possibilidade três; reserva duplicada, impacto três e possibilidade dois. Há tempo para investigar dois riscos hoje. Ordene-os, proponha uma verificação para cada escolhido e indique o que fica pendente.

Solução comentada: inativo e reserva duplicada recebem seis e precedem o erro de título, que recebe três. Para inativo, use A ativo e B inativo e verifique a omissão de B na resposta pública. Para reserva, repita a mesma operação com a mesma chave e confira efeito único. O erro de título fica registrado com responsável e data de revisão. A rubrica exige cálculo correto, relação entre teste e consequência e pendência explícita; apenas escrever testar tudo não resolve a decisão.

Referências S16 Detalhes e limites na seção de referências.

Capítulo 38 Níveis de teste e proteção contra regressões

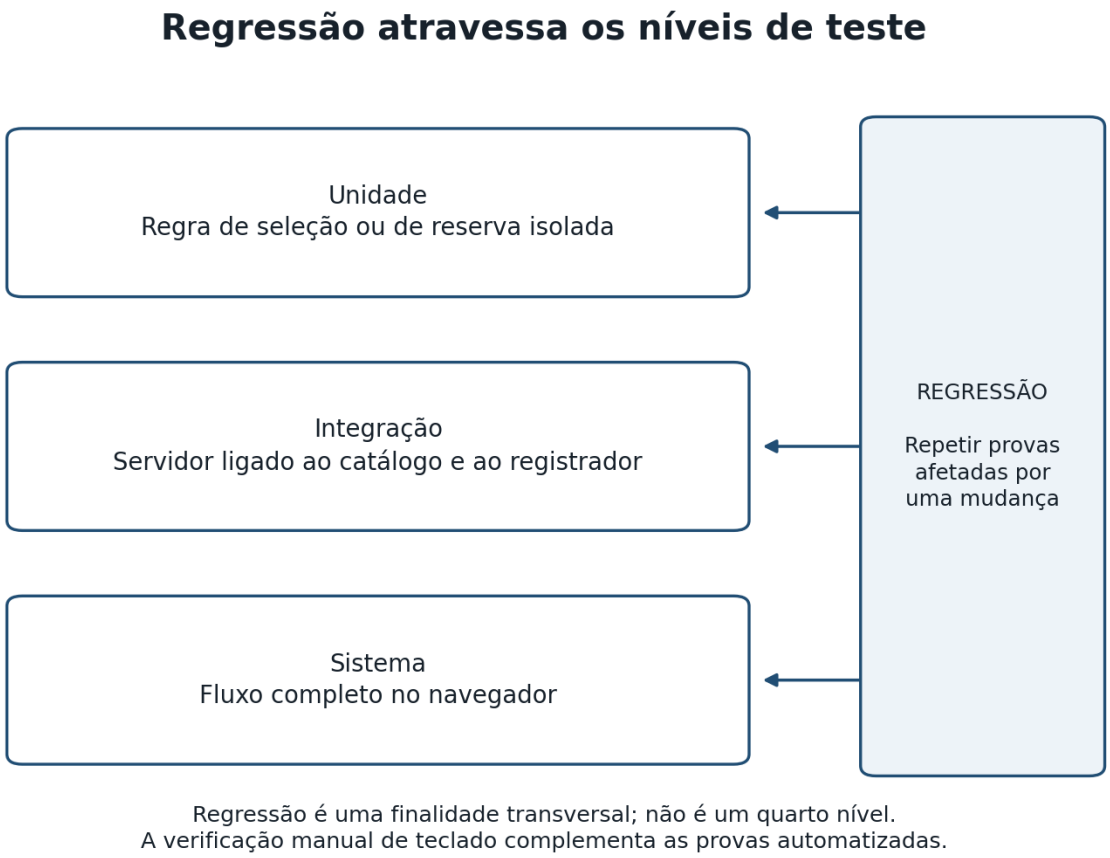


Figura 13 Regressão atravessa os níveis de teste

Uma mudança pode exigir regressão em vários níveis. Um teste de unidade aprovado não comprova sozinho o fluxo de sistema; ferramentas automáticas não eliminam a avaliação manual de interação.

Separar escopo de finalidade

Um nível de teste descreve o alcance da verificação. No teste unitário, uma pequena unidade é exercitada com suas dependências controladas. Na integração, verificamos a colaboração entre partes e suas fronteiras reais. No teste de sistema, percorremos o produto integrado em relação aos requisitos. A terminologia pode variar conforme a arquitetura; por isso, a equipe declara a fronteira usada. O CTFL apresenta níveis de teste e distingue testes de confirmação e regressão [S16, seção Níveis de teste e tipos de teste].

Regressão é outra dimensão: a busca por efeitos indesejados de uma mudança sobre comportamentos já existentes. Um teste unitário pode ser executado para detectar regressão, assim como um teste de integração ou sistema. Confirmação responde se o defeito corrigido deixou de ocorrer no caso que o revelou. Regressão investiga se a correção danificou outras partes. Esses objetivos podem compartilhar testes, mas não significam a mesma pergunta.

Definir a fronteira antes da ferramenta

Na Oficina Caracol, o mesmo requisito de ocultar inativos aparece em três escopos. Os registros conceituais A ativo e B inativo fornecem a entrada conhecida. Escolhemos uma asserção, isto é, a condição que deverá ser verdadeira. Não basta chamar um teste de integração porque ele contém um botão: o nome depende de quais componentes colaboram e do que ficou fora da execução.

Escopo	Partes reais e partes controladas	Ação e asserção
Unidade	listarAtivos real; array válido fornecido pelo teste; sem arquivo ou rede	Selecionar entre A e B; obter objeto público de A, sem campo ativo
Integração	Servidor HTTP e leitura do arquivo de exemplo reais; cliente controlado	Consultar GET /api/casos; resposta pública omite B e só expõe id, titulo e descricao
Sistema	Servidor, página e JavaScript do navegador reais	Abrir o catálogo e percorrer a interface; visualizar A e não B

Construir e executar a proteção

- Escreva o resultado esperado e escolha o escopo que consegue demonstrá-lo. Uma regra pura admite um teste rápido; uma falha de cabeçalho HTTP exige exercitar a fronteira HTTP.
- Prepare dados isolados e limpe o estado criado pelo teste. Um teste não deve depender de outro ter sido executado primeiro.
- Use substitutos apenas para dependências que deseja controlar. Mock é um substituto configurado para responder ou registrar interações; sua existência não demonstra que o serviço real seja compatível.
- Execute os testes automatizados com npm test dentro de pratica. Esse comando não substitui a navegação manual necessária ao REQ-004.
- Após uma correção, execute o caso que reproduzia o defeito e as proteções pertinentes às áreas afetadas. Registre versão, comando, resultado e limites da execução.

Caso trabalhado de correção

Imagine que uma implementação defeituosa aceite registros cujo campo ativo esteja ausente. O caso reproduzível contém A ativo, B inativo e C sem esse campo; pelo contrato, C é inválido. validarCasos deve lançar TypeError, e a API deve responder 500 com mensagem pública genérica se o arquivo de catálogo for inválido. O reparo deve tratar a fronteira correta: dados inválidos não podem virar dados públicos aparentemente válidos. A verificação de confirmação exercita C e exige a resposta contratada em cada escopo.

A regressão usa o conjunto válido A/B para preservar a listagem e o conjunto vazio para preservar o estado vazio. O teste HTTP verifica que o servidor aplica o contrato. A inspeção da interface verifica o feedback ao usuário. Esses resultados são esperados do cenário didático; somente a execução no pacote permite preencher aprovado ou reprovado. Caso a regra passe e a interface falhe, o requisito continua incompleto.

Erros e reparos

Erro observado	Reparo próprio do problema
Todos os testes substituem a leitura de dados	Acrescentar uma integração com arquivo de exemplo real
Uma alteração visual derruba testes da regra	Revisar se o teste depende de estrutura interna em vez do comportamento contratado
Só o caso defeituoso é reexecutado	Selecionar também comportamentos relacionados que podem ter sofrido regressão

Condição de avanço e exercício resolvido

O conjunto avança quando as fronteiras obrigatórias possuem evidência complementar aceitável, sem usar a quantidade de mocks ou o nome da ferramenta como prova. Falhas conhecidas são tratadas ou classificadas e aceitas como exceções não impeditivas. A documentação informa quais verificações manuais continuam necessárias.

Exercício: classifique três cenários. Primeiro, uma função recebe A/B e devolve apenas A sem acessar arquivo ou rede. Segundo, uma requisição HTTP consulta o servidor real, que lê o arquivo de exemplo. Terceiro, uma pessoa abre a página no navegador e usa Tab para chegar aos controles. Depois, diga quais podem proteger contra regressão após uma mudança.

Solução e rubrica: o primeiro é unitário; o segundo, integração na fronteira declarada; o terceiro, sistema com avaliação manual de teclado. Todos podem ser reexecutados contra regressões. A resposta deve nomear partes reais, limite de cada evidência e finalidade da reexecução. Uma classificação alternativa é aceitável somente se declarar outro escopo coerente; chamar o terceiro de teste unitário apenas por observar um botão não atende à rubrica.

Referências S16 S19 Detalhes e limites na seção de referências.

Capítulo 39 Segurança e privacidade desde o desenho

Começar por aquilo que precisa de proteção

Um ativo é algo cujo valor precisamos preservar: código, credenciais, disponibilidade ou dados. Uma ameaça descreve um evento capaz de causar dano. Uma fronteira de confiança separa regiões com permissões ou garantias diferentes. Na Oficina Caracol, o navegador de quem visita o catálogo está fora da confiança do servidor. Um valor enviado pelo navegador continua sendo uma entrada a validar, mesmo que a interface oficial costume enviar valores corretos.

O projeto utiliza dados de exemplo e não exige cadastro. Essa escolha reduz a coleta, mas ainda precisamos pensar em arquivos de configuração, mensagens de erro e registros operacionais. Privacidade pergunta por que coletar um dado, quem pode acessá-lo e por quanto tempo conservá-lo. O SSDF orienta a incorporação de práticas seguras ao desenvolvimento; o ASVS organiza requisitos verificáveis para aplicações [S03, práticas PO.1 e PO.4; S05, visão geral sobre requisitos verificáveis de segurança web]. Nenhum desses referenciais certifica automaticamente o método.

Modelo de ameaça preenchido

Ativo ou fronteira	Ameaça no caso	Controle proposto	Verificação necessária
Arquivo de catálogo para resposta pública	Registros inativos aparecem na listagem	Validação do dado e seleção centralizada	A ativo aparece; B inativo permanece ausente
Configuração para repositório	Credencial de serviço é versionada por engano	Separação de segredos e revisão antes de registrar alterações	Inspecionar alterações e impedir publicação de valor sensível
Entrada para registro operacional	Mensagem contém conteúdo pessoal desnecessário	Registrar evento e identificador técnico mínimo	Examinar campos de log do cenário fornecido
Navegador para operação simulada	Entrada inesperada ou repetição altera o efeito	Validar contrato e chave de idempotência	Rejeitar dado inválido e repetir chave com efeito único

Transformar ameaça em trabalho verificável

- Desenhe o caminho da entrada até o efeito e identifique onde a confiança muda. Inclua caminhos de erro e configuração.
- Para cada ameaça pertinente, defina consequência, controle e teste. Validação verifica o contrato da entrada; autorização decide se a ação é permitida. Uma não substitui a outra.
- Decida quais dados são indispensáveis. Registre finalidade, acesso, retenção e descarte dos registros operacionais antes de habilitá-los.
- Teste os controles com dados artificiais. Registre também o que o teste não consegue demonstrar e quais ameaças permanecem.
- Submeta o risco residual, isto é, o risco que sobra após os controles, à autoridade responsável pela liberação. Resultado desconhecido exige investigação, não aceitação automática.

O que fazer quando um segredo aparece

Suponha que uma credencial fictícia de integração tenha sido incluída em um commit do exercício. Na situação real correspondente, apagar a linha do arquivo atual não impede o uso de uma credencial que já tenha sido copiada. A resposta começa por identificar o serviço, o alcance e a exposição. A credencial exposta deve ser revogada ou rotacionada conforme o serviço. Em seguida, investigam-se acessos pertinentes e corrige-se o fluxo para evitar reincidência [S13, seção sobre exposição de credenciais e remoção de dados sensíveis].

A limpeza do histórico, quando necessária, exige coordenação: cópias, referências e clones podem continuar contendo o valor. Não se publica o segredo em um relatório de incidente para provar que ele existiu. A evidência usa identificadores seguros, horários, alcance e confirmação da revogação. Para treinar, o aluno escreve o roteiro com um marcador artificial; não cria uma credencial real para provocar um vazamento.

Erros e condição de avanço

Sanitizar tudo com uma função genérica mistura problemas distintos. O reparo depende do contexto: validar formato na entrada, tratar a saída conforme o local de uso e aplicar autorização quando houver uma ação protegida. Registrar todo o conteúdo recebido também pode transformar o log em nova exposição. Reduza os campos ao necessário e defina quem consegue consultá-los.

O gate de segurança exige controles pertinentes implementados, resultados avaliados e aceitáveis, e riscos residuais não impeditivos explicitamente aceitos por autoridade identificada. A simples existência de ameaça, controle e teste numa planilha não satisfaz essa condição. Uma exposição de credencial ativa sem contenção continua impeditiva para o cenário afetado.

Exercício e solução comentada

Dados do exercício: um log de falha contém horário, nome completo fictício, texto livre enviado pelo visitante e código técnico de erro. O suporte precisa saber a frequência de falhas e correlacionar uma tentativa com o diagnóstico. Escolha os campos a manter e proponha uma verificação.

Solução: conservar horário com precisão adequada, código técnico e identificador artificial da tentativa atende ao objetivo descrito; nome e texto livre não têm necessidade demonstrada. A verificação inspeciona os campos gerados por uma falha artificial e confirma sua ausência, além das permissões de acesso e da retenção definida. A rubrica exige justificar cada campo e declarar o limite da decisão; afirmar que o log ficou seguro sem verificar coleta, acesso e descarte é insuficiente.

Referências S03 S05 S13 Detalhes e limites na seção de referências.

Capítulo 40 Desempenho e acessibilidade em condições observáveis

Definir qualidade antes de medir

Desempenho descreve como o sistema responde sob condições determinadas. Acessibilidade trata da possibilidade de pessoas com diferentes necessidades perceberem, compreenderem e operarem o conteúdo. Os dois temas se encontram quando uma página pesada impede o acesso, mas exigem critérios próprios. Um relatório automático reúne sinais úteis; a avaliação humana é indispensável para aspectos como ordem de foco, compreensão do feedback e operação por teclado.

Uma baseline é a medição de referência usada para comparar uma mudança. Um orçamento é um limite escolhido pelo projeto para uma característica mensurável. Sem condição de medição e limite explícito, dizer que a página ficou rápida não sustenta uma decisão. Os números deste capítulo são critérios didáticos locais; não representam limites universais. Para acessibilidade, usamos critérios identificados da WCAG 2.2 como referência [S06, 2.1.1 Teclado, 2.4.7 Foco visível, 1.4.4 Redimensionar texto e 1.4.3 Contraste].

Roteiro preenchido da Oficina

Verificação	Condição do exercício	Esperado	Como registrar
Transferência inicial	Mesma página; cache desabilitado; mesma limitação de rede e navegador registrados	Até 150 kB transferidos, orçamento didático	Valor observado e configuração da captura
Teclado	Página aberta; usar Tab, Shift+Tab e teclas próprias dos controles	Alcançar e operar controles em ordem compreensível, com foco visível	Controle alcançado, ação e eventual bloqueio
Texto ampliado	Zoom de 200% na mesma página	Conteúdo e operação preservados, sem perda de informação	Elemento afetado e captura quando houver problema
Feedback de estado vazio	Conjunto de exemplo sem serviços ativos	Mensagem compreensível e percepção adequada da mudança de estado	Ação que gerou o estado e observação manual

Procedimento de comparação

- Escolha uma pergunta por medição. Registre versão, navegador, dispositivo ou emulação, rede, cache e dados usados. Conserve as mesmas condições na comparação.
- Capture a baseline antes da mudança. Repita a medição quando houver variação e descreva o critério usado para resumir os valores; uma leitura isolada não demonstra estabilidade.
- Execute o roteiro manual com o teclado e zoom. Verifique cada controle, incluindo estados vazio e erro, em vez de olhar apenas a página carregada.
- Use ferramentas automáticas para ampliar a procura de problemas, investigando seus resultados. Registre também o alcance da ferramenta e o que permanece manual.

- Compare os resultados com os critérios. Abra correção para reprovações e reteste a mesma condição depois do reparo.

Interpretar um resultado sem fabricar uma execução

Considere os seguintes dados fornecidos para análise: a baseline transferiu 180 kB; a versão candidata transferiu 130 kB sob a mesma configuração. A redução é de 50 kB, aproximadamente 27,8% em relação à baseline. A candidata atende ao orçamento local de 150 kB. Esses valores pertencem ao exercício e não são resultados medidos do pacote. O leitor deverá substituir os campos pelos registros de sua própria execução.

Na mesma ficha didática, o percurso por teclado alcança os controles, mas o indicador visual de foco desaparece no último. A redução de bytes não compensa essa falha. O reparo exige restaurar um indicador perceptível e repetir o percurso nos estados relevantes. Uma largura de 360 pixels pode integrar o roteiro de visualização, mas não cobre por si só teclado, contraste, ampliação, entendimento ou tecnologias assistivas. Cada conclusão deve corresponder ao critério efetivamente avaliado.

Erros e condição de avanço

Erro	Correção orientada por evidência
Otimizar antes de definir a pergunta	Escolher uma característica, medir referência e comparar a mesma condição
Tratar pontuação automática como conformidade completa	Relacionar cada resultado aos critérios cobertos e executar avaliações manuais pendentes
Concluir que tudo passou porque houve medição	Comparar cada resultado ao limite ou comportamento esperado e registrar a decisão

O gate avança quando os critérios prioritários definidos têm resultados aceitáveis e o roteiro manual aplicável está concluído, com evidências identificadas. Exceções precisam ser não impeditivas, justificadas e aceitas pela autoridade responsável. Medições ausentes, falhas impeditivas ou dúvidas relevantes impedem essa conclusão. Não se declara conformidade global de acessibilidade com base neste pequeno roteiro.

Exercício com resolução

Dados: orçamento de 150 kB; três leituras da candidata, feitas nas mesmas condições, de 142, 148 e 160 kB; foco invisível no segundo controle. A regra local do exercício exige que todas as três leituras respeitem o orçamento. Decida sobre o avanço e indique os próximos passos.

Solução: a candidata reprova o orçamento pela leitura de 160 kB e reprova o foco. Deve-se investigar a variação, corrigir o foco e repetir as verificações nas condições registradas. A rubrica exige aplicar a regra fornecida, separar desempenho de acessibilidade e evitar escolher apenas a melhor leitura. Mudar o critério depois de ver os dados exigiria nova decisão justificada, não a ocultação da leitura inconveniente.

Referências S06 Detalhes e limites na seção de referências.

Capítulo 41 Dados conteúdo e configuração como comportamento

Reconhecer entradas além do formulário

Um arquivo de dados, uma variável de configuração e o texto publicado alteram o que o programa faz. O código pode estar correto e ainda apresentar um serviço encerrado se o catálogo estiver desatualizado. Na Oficina Caracol, usamos dados artificiais e uma estrutura explícita, descrita junto ao pacote. O schema, ou esquema, declara a forma esperada: campos, tipos e restrições. A validação compara a entrada com esse contrato e produz uma decisão antes da publicação.

Convém distinguir três perguntas. A estrutura é válida? O conteúdo representa algo autorizado e atual? A configuração corresponde ao ambiente? Um texto bem formado não demonstra sua veracidade. Uma URL com sintaxe válida não demonstra que o destino existe ou que pertence à organização esperada. Verificações de estrutura, regra de negócio, disponibilidade e autoridade precisam ser escolhidas separadamente [S18, documentação sobre tipos e objetos; S03, práticas de definição e verificação de requisitos].

Contrato conceitual preenchido

O schema efetivo de pratica/data/casos.json contém exatamente id, titulo, descricao e ativo. As três primeiras propriedades são strings não vazias; ativo é booleano; os identificadores são únicos; propriedades extras são rejeitadas. A tabela usa os nomes reais para relacionar estrutura e publicação. Evidências e autorizações editoriais podem ser registradas nos documentos de apoio, sem acrescentar campos desconhecidos ao arquivo do catálogo.

Elemento conceitual	Regra	Entrada fornecida	Decisão esperada
id	String não vazia e única no conjunto	A	Aceitar se nenhum outro registro usar A
titulo	String não vazia; revisão editorial separada	Oficina de diagnóstico	Aceitar estrutura e encaminhar conteúdo à revisão
descricao	String não vazia; conteúdo autorizado	Descrição fictícia para estudo	Aceitar estrutura; identificar como demonstração
ativo	Booleano obrigatório	false	Aceitar como dado; não listar publicamente
Propriedade extra	Campos fora do contrato são rejeitados	imagem	Rejeitar estrutura; documentar origem/autorização fora deste schema

Preparar a validação e a aprovação

- Documente o schema real com tipos, campos obrigatórios, valores permitidos e política para campos desconhecidos. Não converta silenciosamente uma entrada inválida em conteúdo aprovado.

- Separe erros de estrutura de decisões editoriais. A primeira categoria pode ser automatizada; a segunda requer fonte e autoridade apropriadas ao conteúdo.
- Execute npm run validate dentro de pratica para as verificações implementadas. Leia o alcance declarado pelo comando e resolva cada erro antes de publicar.
- Mantenha a configuração necessária a cada ambiente identificada, sem incluir segredos no repositório. Mudanças de configuração também precisam de revisão e evidência.
- Planeje alterações de estrutura com compatibilidade, migração e reconciliação. Reconciliação é a comparação entre o conjunto esperado e o resultado produzido para detectar perdas ou diferenças indevidas.

Do erro ao reparo observável

Considere um registro cujo campo ativo esteja ausente. validarCasos deve lançar TypeError para essa entrada. Na fronteira HTTP, catálogo inválido produz resposta 500 com mensagem pública genérica, sem revelar detalhes internos. Transformar ausência em ativo para evitar um erro mudaria a política de publicação. O reparo é decidir o valor a partir da fonte autorizada e validar novamente. A mensagem de diagnóstico apropriada ao mantenedor deve permitir localizar a regra violada sem expor informação indevida ao visitante.

Agora considere uma troca de nome de campo. A migração recebe três registros de exemplo e deve produzir três registros equivalentes na nova estrutura, preservando identificadores e indicador de atividade. A contagem ajuda, mas não basta: três registros incorretos ainda totalizam três. A reconciliação compara os valores relevantes antes e depois. O resultado deve ser registrado como esperado até a execução; só então pode receber observado e veredito.

Erros e condição de avanço

Problema	Reparo e evidência
Campo opcional tratado como obrigatório por uma tela	Alinhar contrato e consumidor; testar presença e ausência permitida
Conteúdo sem fonte	Identificar fonte e autoridade ou suspender a publicação do trecho
Configuração funciona apenas na máquina de quem criou	Documentar a configuração necessária e reproduzir o fluxo no ambiente declarado

O capítulo avança quando a estrutura passa nas validações aplicáveis, o conteúdo tem aprovação pertinente e a configuração foi verificada no ambiente de destino. O gate registra resultados e decisões, além de lacunas não impeditivas aceitas. Aprovar o texto não corrige um schema inválido; cada camada conserva sua própria evidência.

Exercício e solução

Dados: A tem id, titulo e descricao, mas falta ativo; B tem todos os quatro campos válidos e está inativo, porém um documento editorial ainda registra imagem sem origem; C está ativo, possui os quatro campos estruturais válidos e texto ainda sem revisão. Para cada registro, indique pendência estrutural e pendência de publicação.

Solução: A exige decisão e preenchimento autorizado do booleano obrigatório antes de passar a validação. B é estruturalmente válido e permanece fora da listagem por estar inativo; o uso da imagem continua bloqueado no processo editorial por falta de origem/autorização. C pode passar a estrutura, porém aguarda revisão editorial. A rubrica exige separar essas dimensões, não adicionar imagem ao schema e preservar as pendências; preencher tudo como aprovado porque o JSON é legível não atende ao exercício.

Referências S18 S03 Detalhes e limites na seção de referências.

Capítulo 42 Build entrega e decisão final

Identificar exatamente o que será disponibilizado

Build é a preparação de um artefato utilizável a partir dos arquivos do projeto. Mesmo quando uma aplicação JavaScript dispensa compilação, é preciso identificar o conjunto que será executado. Entrega, neste capítulo, é a disponibilização desse conjunto ao ambiente ou à pessoa responsável. Implantação o coloca em execução no destino. Homologação é a avaliação do resultado pelos critérios de aceite. Declarar essas convenções evita confundir código mais recente com versão efetivamente avaliada.

Na Oficina Caracol, a entrega reúne código, dados artificiais, configuração documentada e instruções de operação. O commit identifica uma revisão do repositório; a versão identifica uma entrega segundo a política adotada. Um artefato anterior guardado é útil, mas a reversibilidade depende de conseguir usá-lo com os dados e a configuração disponíveis. Esse vínculo precisa ser ensaiado antes que uma recuperação seja necessária.

Ficha preenchida para preparação

Campo	Preenchimento do caso didático
Candidata	Oficina Caracol candidata C; revisão a preencher com o identificador real
Escopo obrigatório	REQ-001 a REQ-005, com verificações aplicáveis explicitadas
Evidências automatizadas	npm test e npm run validate; resultados ainda pendentes até executar
Evidência manual	Navegação por teclado, foco e feedback nos estados previstos
Destino do ensaio	Ambiente local controlado com dados artificiais
Retorno	Versão anterior identificada, snapshot verificável e roteiro de restauração
Autoridade	Responsável pela entrega e responsável pelo aceite registrados nominalmente no uso real

Procedimento de preparação e liberação

- Parta de uma revisão identificada e confira o runtime com node --version. Registre o valor observado; não copie o número de outro ambiente como se fosse sua execução.
- Execute npm test e npm run validate dentro de pratica. Preserve a saída pertinente e investigue falhas, inclusive diferenças de ambiente e configuração.
- Inicie com npm start e realize as verificações manuais aplicáveis no endereço local documentado. Relacione cada resultado ao requisito correspondente.
- Execute npm run snapshot para preparar o ensaio de dados artificiais em sandbox/recuperacao, conforme o contrato do pacote. Confirme a existência e identificação do snapshot antes de prosseguir.
- Execute o ensaio de npm run restore somente em sandbox/recuperacao. Compare o arquivo restaurado com a referência documentada. O catálogo fonte não é sobrescrito. As verificações dos fluxos da aplicação são realizadas separadamente sobre o catálogo que o servidor efetivamente usa; esse ensaio não constitui recuperação de produção.

- Reúna resultados aceitáveis, exceções não impeditivas aceitas e condições de retorno. Autorize a liberação apenas depois dessa avaliação.

Demonstrar o retorno em vez de presumir

O cenário de ensaio usa apenas sandbox/recuperacao. O conjunto de referência contém A ativo e B inativo; após preparar o snapshot, uma alteração didática no arquivo de trabalho dessa área controlada introduz uma entrada inválida. O roteiro de restauração deve recuperar o conteúdo de referência. Conferem-se identificadores e campos relevantes, além da igualdade esperada com a cópia identificada. São critérios do ensaio, não resultados já observados. O catálogo fonte e o servidor em execução não são alterados pelo restore.

A ficha do ensaio liga snapshot, versão de código, configuração, passos e observações. Recuperar arquivos demonstra uma parte da capacidade; operar a aplicação com dados recuperados exige ainda verificar compatibilidade e comportamento no destino escolhido. Em um sistema real, essa etapa faria parte de um roteiro específico e ensaiado. Uma mudança irreversível pode exigir correção adiante ou migração própria. A decisão de disponibilizar considera esses limites de forma explícita.

Erros e reparos

Erro	Reparo necessário
Homologar o último código sem identificador	Fixar revisão e conjunto de dados; repetir a avaliação quando eles mudarem
Guardar cópia sem testar restauração	Ensaiar retorno e verificar comportamento, além da presença dos arquivos
Liberar porque os comandos foram executados	Examinar os resultados e impedir avanço diante de falhas impeditivas

Condição de avanço

O gate final exige artefato identificável, resultados aplicáveis aprovados, integração comprovada, aceite pertinente e recuperação compatível com o risco. Uma exceção só pode permanecer quando é não impeditiva e foi aceita por autoridade responsável. Quando a entrega muda depois da avaliação, as evidências afetadas precisam ser renovadas; o simples decurso do tempo não determina sozinho sua validade.

Exercício com solução

Dados da candidata: testes automatizados aprovados; validação aprovada; foco invisível em um controle obrigatório; snapshot presente; restauração nunca ensaiada. A equipe quer anunciar a versão hoje. Decida o gate e preencha duas ações prioritárias.

Solução: o gate permanece bloqueado pela falha de acessibilidade e pela recuperação ainda não demonstrada no escopo exigido. A primeira ação restaura e retesta o foco no percurso obrigatório. A segunda executa o ensaio de retorno com conferência dos dados e fluxos. A rubrica exige relacionar cada bloqueio a uma evidência faltante ou reprovada e nomear responsáveis; substituir o ensaio por uma declaração de confiança não resolve a pendência.

Referências S19 S07 Detalhes e limites na seção de referências.

Parte VII Auditoria evolução e continuidade

Revisar preservar recuperar e transferir

Capítulo 43 Auditoria documental arquitetural e de código

Comparar a promessa com o sistema existente

Auditoria é uma comparação orientada por critérios e sustentada por evidências. A camada documental registra intenção e instruções; a arquitetura define responsabilidades e relações; o código executa comportamentos. A Oficina Caracol pode ter testes locais aprovados e ainda possuir um README que leva outra pessoa a iniciar a aplicação pelo caminho errado. Essa contradição afeta a operação do projeto e precisa de tratamento rastreável.

O escopo define o que será examinado e o que permanece fora. Uma revisão do catálogo público não autoriza concluir que toda a organização é segura. O auditor registra fontes, versão, ambiente, achado e limite. O objetivo é permitir a outra pessoa repetir o raciocínio e verificar a correção. Revisão humana independente pode ser adequada ao risco, mas sua realização só será declarada quando ocorrer efetivamente.

Montar um pacote de evidência

- Escolha um requisito ou pergunta de auditoria. Para REQ-002, pergunte se registros inativos podem aparecer em qualquer resposta pública do catálogo.
- Identifique as fontes que prometem o comportamento e o caminho real da requisição, da entrada ao resultado. Declare quais arquivos e ambientes foram examinados.
- Compare instruções documentais, fronteiras arquiteturais e implementação. Procure também caminhos alternativos e tratamento de erro.
- Registre achados com localização, reprodução, esperado, observado, impacto e classificação. A severidade deve refletir consequência e alcance, não o incômodo de corrigir.
- Atribua responsável, decisão e prazo. Depois da correção, repita a verificação que revelou o problema e as proteções relacionadas; registre o reteste.

Achados preenchidos em um cenário fornecido

Identificador	Evidência do cenário	Impacto e classificação	Reparo e reteste
AUD-01	README orienta node app.js, mas o pacote documentado inicia com npm start	Instrução operacional inválida; prioridade média no exercício	Atualizar a instrução e reproduzir a inicialização a partir do README
AUD-02	Regra isolada omite B inativo, mas uma rota pública alternativa devolve B	Violação de REQ-002; impeditiva no caso	Aplicar a política na fronteira correta e retestar ambas as rotas

Identificador	Evidência do cenário	Impacto e classificação	Reparo e reteste
AUD-03	Relatório registra teste aprovado sem versão nem ambiente	Evidência não reproduzível; avaliação pendente	Reexecutar ou recuperar evidência suficiente, sem inventar identificação

Interpretar a contradição arquitetural

AUD-02 demonstra por que auditar somente uma função seria insuficiente. A intenção é ocultar inativos; a regra isolada cumpre essa intenção; outra rota contorna a proteção. A arquitetura precisa tornar explícito onde a política é aplicada e como os consumidores devem acessá-la. O reparo pode centralizar a seleção ou reduzir caminhos de publicação, conforme a implementação existente. A decisão deve ser registrada com sua consequência.

No reteste, o conjunto artificial inclui A ativo e B inativo. A resposta esperada de cada rota pública contém apenas registros permitidos. O auditor também verifica se o estado vazio e os dados inválidos mantêm a resposta prevista, para evitar uma correção que esconda um defeito e crie outro. A tabela contém dados de um cenário para estudo; não afirma que essas rotas alternativas ou defeitos existem no pacote entregue.

Evitar uma auditoria decorativa

Atalho que enfraquece o trabalho	Reparo
Marcar checklist sem guardar evidência	Associar cada conclusão a fonte e resultado identificáveis
Tratar todo desvio como crítico	Classificar pelo efeito, alcance e condições de ocorrência explicitadas
Corrigir e apagar o achado	Conservar o histórico de decisão e acrescentar a evidência do reteste
Auditar detalhes sem relação com o escopo	Reorientar a revisão para os requisitos e riscos declarados

Condição observável de encerramento

Uma auditoria pode ser encerrada com problemas conhecidos, desde que seu relatório descreva a situação e as decisões sem ambiguidade. Encerrar a auditoria não significa autorizar a entrega. A liberação do produto depende de resultados aceitáveis, ausência de pendências impeditivas e aceitação competente das exceções. Um achado ter responsável e prazo melhora a gestão, mas não converte automaticamente uma falha crítica em risco aceitável.

Exercício e rubrica de correção

Dados: o README afirma que não há verificações manuais pendentes; a matriz de REQ-004 exige teclado e foco; o último relatório só apresenta npm test. Produza um achado de auditoria com critério, evidência, impacto, responsável esperado e reteste.

Solução comentada: o critério é a demonstração manual de REQ-004. A evidência é a contradição entre README, matriz e relatório limitado ao automatizado. O impacto é declarar completude sem a verificação exigida. O responsável pela entrega deve corrigir a declaração e organizar a execução do roteiro por pessoa identificada. O reteste examina a nova documentação e a evidência manual produzida. A rubrica aceita severidades justificadas pelo contexto, mas exige preservar a pendência até obter resultado; imaginar que npm test executou uma avaliação humana não atende ao critério.

Referências S01 S03 Detalhes e limites na seção de referências.

Capítulo 44 Dívida técnica e refatoração com propósito

Descrever o custo que será herdado

Dívida técnica é uma condição que torna o trabalho futuro mais caro ou arriscado. O nome só ajuda quando conseguimos descrever a condição, o benefício presente e o custo esperado. Na Oficina Caracol, manter o registrador de reservas apenas em memória simplifica o laboratório e evita infraestrutura externa. A contrapartida é explícita: reiniciar o processo apaga as reservas e a memória de chaves. Essa escolha é compatível com uma simulação, mas precisaria ser revista antes de assumir compromisso persistente com pessoas reais.

Refatoração modifica a estrutura interna preservando o comportamento observável. A definição de referência enfatiza essa preservação [S17, definição de refatoração]. Os testes que já protegem os contratos podem fornecer a rede de segurança necessária. Testes de caracterização são especialmente úteis quando precisamos descobrir e registrar como um sistema existente se comporta; não são uma etapa adicional obrigatória em toda refatoração. Uma mudança de regra pública exige outra decisão e não deve ser escondida sob o rótulo de limpeza.

Registro de dívida preenchido

Campo	Exemplo didático
Condição	Reservas e chaves existem somente durante a vida do processo
Benefício atual	Executar laboratório sem banco externo ou instalação de dependências
Custo futuro	Reinício perde a memória de uma operação; efeito único não cobre múltiplos processos
Responsável	Pessoa mantenedora da prática; nome a preencher no uso real
Gatilho de revisão	Proposta de uso persistente, múltiplas instâncias ou dados reais
Decisão atual	Manter apenas como simulação e declarar a limitação na documentação
Próximo passo	Antes de ampliar o escopo, projetar persistência, concorrência e política de retenção

Refatorar com um contrato pequeno

Considere um exemplo isolado, separado dos arquivos do pacote: duas formas de produzir os títulos dos registros ativos. A entrada já foi validada segundo o contrato do catálogo. O comportamento a preservar inclui a ordem relativa e a ausência de alteração na entrada. Este exemplo não substitui listarAtivos, cuja saída pública possui os três campos documentados.

Código 30 javascript

```
const titulosAntes = casos => {
  const resultado = [];
  for (const caso of casos) {
    if (caso.ativo) resultado.push(caso.titulo);
  }
  return resultado;
};

const titulosDepois = casos =>
  casos.filter(caso => caso.ativo).map(caso => caso.titulo);
```

Com os registros A ativo, B inativo e C ativo, ambas devem produzir os títulos de A e C, nessa ordem. O conjunto vazio deve produzir um array vazio. Essas são expectativas para conferir, não uma alegação de execução do código acima. Se o exemplo removesse a validação exigida por uma função pública, mudaria o contrato em vez de apenas reorganizar sua estrutura.

Procedimento de decisão e execução

- Registre o problema: duplicação específica, compreensão difícil ou custo mensurável. Associe a mudança a um benefício e a um responsável.
- Identifique os comportamentos a preservar e examine a proteção existente. Acrescente testes quando houver lacunas; use caracterização se o comportamento atual precisar ser conhecido.
- Separe a refatoração de novas funcionalidades e execute passos pequenos. Depois de cada passo relevante, verifique os contratos afetados.
- Compare o resultado com o objetivo. Uma estrutura diferente sem benefício reconhecível não demonstra que a dívida diminuiu.
- Atualize o registro com decisão, evidência e pendências. A responsabilidade permanece explícita mesmo quando a dívida é aceita temporariamente.

Erros e condição de avanço

Erro	Reparo específico
Dívida sem dono	Atribuir responsável, gatilho e forma de acompanhamento
Reescrever durante incidente sem necessidade de contenção	Conter o incidente com a menor ação adequada e planejar a reorganização separadamente
Mudar saída e chamar de refatoração	Tratar a alteração de contrato como evolução, com decisão e proteção próprias

O avanço exige comportamento preservado nas verificações pertinentes e benefício demonstrado no escopo da mudança. A dívida remanescente possui impacto, responsável, prioridade e decisão. A aprovação não depende de zerar toda dívida; depende de compreender a consequência e de não ocultar uma limitação impeditiva para o uso proposto.

Exercício e solução comentada

Dados: duas telas duplicam a formatação de um título. Corrigir a mesma regra demanda alterar dois lugares; a equipe estima quinze minutos extras por mudança e prevê quatro mudanças semelhantes. A proposta de extração custa quarenta minutos. Registre uma decisão sem tratar a estimativa como certeza.

Solução: o custo adicional estimado é de sessenta minutos, acima dos quarenta previstos para a extração. Pode fazer sentido refatorar, considerando também risco e compreensão. O registro deve nomear um responsável, proteger as duas telas com exemplos de entrada e saída e conferir a estimativa após o trabalho. A rubrica exige cálculo, incerteza, contrato preservado e critério de conclusão. Uma estimativa favorável não autoriza afirmar economia realizada antes de observar o resultado.

Referências S17 Detalhes e limites na seção de referências.

Capítulo 45 Migração versão e depreciação sem surpresa

Explicitar o compromisso público

Um contrato público define o que consumidores podem esperar de uma interface. Versionar identifica a evolução desse compromisso; migrar prepara dados e consumidores para uma mudança; depreciar comunica que uma opção será retirada, com alternativa e condições conhecidas. O changelog apresenta o impacto da entrega. Esses instrumentos se complementam. Um número novo, sozinho, não converte uma alteração incompatível em uma experiência previsível.

No versionamento semântico, uma API pública declarada orienta a distinção entre correção compatível, funcionalidade compatível e mudança incompatível. As regras dependem dessa declaração e da fase do projeto [S10, especificação sobre API pública e incrementos de versão]. Neste capítulo, supomos uma API estável 1.0.0 para estudar decisões; não afirmamos que a versão real do laboratório já ofereça estabilidade de produção. A resposta pública atual de cada caso é {id,titulo,descricao}.

Plano de alteração preenchido

Imagine a proposta futura de substituir o nome descricao por resumo na resposta. A implementação atual não faz essa mudança. Um consumidor que lê descricao perderia conteúdo se o campo desaparecesse sem preparação. Primeiro investigamos os consumidores e seus contratos, incluindo clientes que rejeitam campos desconhecidos. Acrescentar um campo também precisa de análise de compatibilidade nesse contexto.

Etapa do cenário	Ação e evidência esperadas
Inventário	Listar interface do navegador, testes e outros consumidores conhecidos do campo descricao
Alternativa	Documentar o contrato novo e adaptar um consumidor de exemplo
Transição	Escolher interface versionada ou compatibilidade temporária validada pelos consumidores
Comunicação D0	Explicar mudança, alternativa, prazo proposto e canal para impedimentos
Revisão D30	Examinar adoção e impedimentos; autorizar ou adiar a retirada por decisão registrada
Retirada	Publicar versão incompatível conforme política, após condições de aceite e recuperação

Tratar dados e código em conjunto

- Descreva exatamente a mudança de entrada, saída e efeito. Faça uma lista dos consumidores e das suposições que podem ser invalidadas.

- Escolha uma estratégia: manter compatibilidade temporária, oferecer uma nova interface ou coordenar a troca. Defina como verificar cada opção.
- Prepare a migração com dados artificiais, preservando identificadores e regras relevantes. Compare conteúdo antes e depois para detectar perdas.
- Comunique a alternativa e um prazo condizente com o contexto. Datas D0 e D30 são marcos do exercício, não anúncios ou compromissos reais.
- Monitore adoção com sinais necessários e proporcionais. Defina como agir quando ainda houver consumidores dependentes da interface antiga.
- Ensaie retorno ou correção adiante conforme a reversibilidade. Depois da retirada, confirme o comportamento dos consumidores e atualize instruções e testes.

Uma comunicação que ajuda a agir

Um aviso preenchido para o cenário poderia dizer: a interface futura passará a usar resumo; a versão atual permanece disponível durante a transição acordada; consulte o exemplo de adaptação e informe impedimentos antes da revisão D30. A decisão final considerará a adoção observada e os riscos pendentes. Um aviso como melhoramos a API não oferece a quem mantém um consumidor a informação necessária para agir.

O changelog da versão deve relacionar comportamento anterior, comportamento novo, pessoas afetadas e ação exigida. Se houver migração de dados, inclui o roteiro e o limite do retorno. Na prática atual, a idempotência da reserva permanece em memória: reiniciar o processo elimina o histórico de chaves. Essa limitação precisa continuar visível durante qualquer mudança de versão; um número maior não amplia automaticamente a garantia de efeito único [S15, seção sobre identificadores de solicitação e repetição segura].

Erros e decisão de avanço

Erro	Reparo verificável
Depreciar sem alternativa	Publicar um caminho de substituição utilizável e demonstrar a adaptação
Escolher versão por preferência estética	Classificar a alteração segundo a API pública e a política declaradas
Retirar após o prazo sem conferir adoção	Examinar consumidores pendentes e registrar decisão de retirada ou adiamento
Publicar changelog apenas com nomes de arquivos	Explicar efeitos e ações necessárias para quem utiliza a interface

A mudança avança quando compatibilidade, migração, comunicação e recuperação possuem resultados aceitáveis no escopo definido. Riscos residuais não impeditivos precisam de aceitação responsável. Um consumidor obrigatório ainda sem alternativa ou uma migração com perda inexplicada impedem a retirada. Um prazo vencido é um dado para a decisão, não uma demonstração de que a transição terminou.

Exercício e solução

Dados: a rota antiga atende dois consumidores obrigatórios; um já usa a nova, o outro depende de um campo removido e ainda não recebeu exemplo de adaptação. O prazo proposto termina amanhã. Planeje a decisão e escreva duas linhas do changelog.

Solução: adiar a retirada ou oferecer compatibilidade adequada enquanto se entrega e verifica a adaptação do consumidor pendente. O changelog deve declarar qual campo muda e qual ação o consumidor precisa executar; também informa o novo marco de revisão e o caminho de transição. A rubrica exige alternativa utilizável, responsável e evidência de adoção. Apenas aumentar o número principal da versão não resolve a dependência nem a ausência de orientação.

Referências S10 S15 Detalhes e limites na seção de referências.

Capítulo 46 Observabilidade e resposta a incidentes

Escolher perguntas que a operação precisa responder

Observabilidade é a capacidade de compreender estados internos a partir de sinais do sistema. Logs registram eventos; métricas resumem medidas ao longo do tempo; rastros relacionam etapas de uma operação. Um sistema pequeno pode começar com sinais simples, desde que respondam a perguntas concretas. A Oficina Caracol precisa saber se o catálogo responde, se a operação simulada falha e se uma repetição cria efeito indevido. Coletar todo o conteúdo das solicitações acrescentaria exposição sem necessariamente melhorar essas respostas.

Incidente é uma ocorrência que exige resposta para conter impacto e recuperar o serviço segundo critérios definidos. A primeira decisão procura reduzir o dano com informações disponíveis. A análise posterior investiga condições e melhorias sem transformar a busca por causa numa caça a culpados. A incerteza precisa aparecer na comunicação: hipótese, evidência e decisão têm significados diferentes. Um rollback é uma possibilidade de contenção quando há relação com uma mudança reversível, não a resposta automática para qualquer falha.

Sinais e roteiro preenchidos

Sinal do caso	Definição didática	Ação prevista
Falha do catálogo	Respostas 500 em GET /api/casos divididas pelo total de consultas na janela	Investigar se o erro corresponde a dados inválidos, mudança de código ou outra condição
Conflito de reserva	Contagem de HTTP 409 em POST /api/reservas	Conferir reutilização da mesma chave para outro caso; não repetir cegamente
Saúde da reserva	Repetir mesma chave e caso mantém reservaId durante a mesma sessão	Se houver efeito duplicado, conter a operação e preservar evidência mínima
Registro operacional mínimo	Horário, rota, código HTTP e identificador artificial da tentativa	Correlacionar falhas sem guardar corpo completo ou dados pessoais

Responder com uma sequência verificável

- Reconheça o sinal e abra um registro com horário, escopo afetado e pessoa responsável pela coordenação. Preserve evidência sem divulgar segredos.
- Estime o impacto e comunique o conhecido, o desconhecido e o próximo ponto de atualização. Evite prometer prazo de recuperação sem base.
- Compare hipóteses com os sinais disponíveis. Verifique mudanças recentes, condições dos dados e alcance da falha.
- Escolha contenção adequada: suspender uma operação, corrigir a entrada, retornar versão ou outra medida proporcional. Verifique dependências e reversibilidade antes da ação.
- Após a ação, teste os fluxos afetados e acompanhe os sinais durante a janela definida. Declare recuperação apenas com critérios atendidos.

- Produza análise posterior com linha do tempo, fatores contribuintes, ações, responsáveis e verificação de eficácia. Converta defeitos pertinentes em testes e decisões.

Decidir entre retorno e outro reparo

No cenário fornecido, ocorreram oito respostas 500 em duzentas consultas, ou 4%. A regra local do exercício pede investigação acima de 2%; ela é um limiar de estudo, não um padrão universal. A investigação encontra um arquivo de catálogo inválido, alterado independentemente da última revisão de código. Voltar apenas o código pode manter a falha. É preciso reparar os dados a partir da fonte autorizada e verificar a resposta, preservando evidência do ocorrido.

Em outro cenário, a falha surge imediatamente após uma revisão e o teste comparativo identifica a alteração responsável. Se o retorno estiver ensaiado e os dados forem compatíveis, usar a revisão anterior pode ser a contenção escolhida. Os dois casos começam com o mesmo sinal e levam a decisões diferentes. O registro do incidente explica por que a ação foi adequada e qual observação demonstrou a recuperação. São cenários para análise, sem alegar execução de um incidente real.

Repetir uma solicitação com segurança

Uma resposta perdida não prova que a operação falhou antes de produzir efeito. Por isso, a nova tentativa da mesma reserva conserva a chave e o caso. A prática devolve o mesmo resultado lógico enquanto o registrador permanece em memória. Reutilizar a chave para outro caso gera conflito 409; essa resposta exige corrigir a intenção, não insistir. Estratégias de retry precisam respeitar limites e semântica da operação [S15, seção sobre repetição segura com identificador de solicitação]. Reiniciar o processo elimina a memória; a demonstração não promete idempotência persistente.

Erros e condição de recuperação

Um alerta sem pessoa ou ação definida vira ruído. Dê ao alerta uma pergunta, um responsável e um roteiro. Um log indiscriminado pode virar vazamento; revise campos e acesso. Culpar quem estava de plantão esconde condições de processo; registre fatores verificáveis e medidas acompanháveis. A recuperação exige fluxo novamente aceitável, sinais acompanhados e pendências classificadas. Encerrar a urgência não dispensa as ações de aprendizagem.

Exercício e solução

Dados: após uma resposta perdida, a interface tenta novamente a reserva do caso A. Recebe 409 porque enviou a chave anterior com o caso B. Uma pessoa sugere cinco novas tentativas imediatas. Decida e proponha um teste de proteção.

Solução: interromper a repetição automática e corrigir a associação entre intenção, chave e caso. Para repetir a mesma intenção, conservar chave e A; uma intenção nova para B recebe nova chave. O teste verifica resultado lógico igual para chave/A repetidos e conflito para chave/B. A rubrica exige interpretar 409, distinguir perda de resposta de ausência de efeito e declarar o limite de memória da simulação.

Referências S15 S19 Detalhes e limites na seção de referências.

Capítulo 47 Continuidade institucional e transferência do projeto

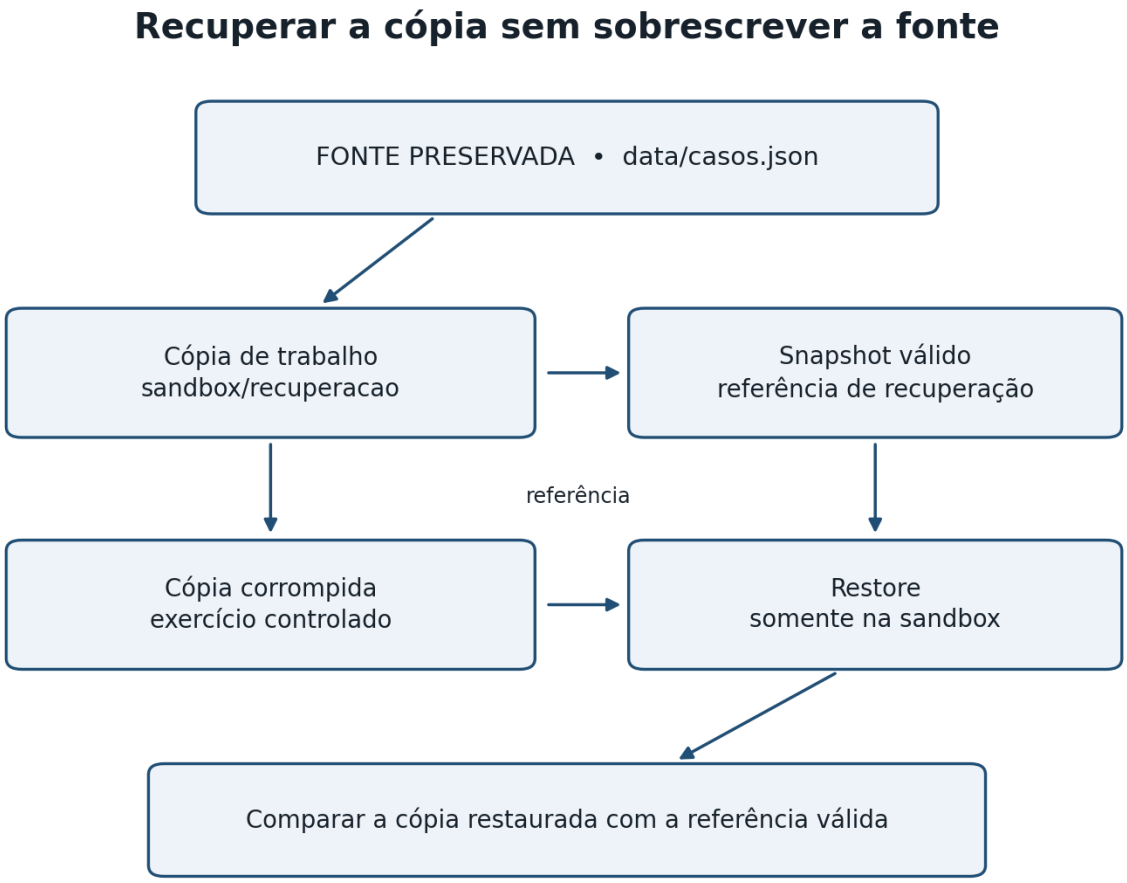


Figura 14 Recuperar a cópia sem sobrescrever a fonte

Os comandos da oficina atuam em sandbox/recuperacao. O exercício não escreve sobre o catálogo fonte, não trata produção e exige verificar o resultado após restaurar.

Permitir que outra pessoa assuma com segurança

Continuidade institucional significa manter a capacidade de operar e evoluir um projeto quando mudam pessoas ou fornecedores. Ela depende de acesso autorizado, documentação suficiente, dados recuperáveis e decisões compreensíveis. Na Oficina Caracol, o primeiro objetivo é outra pessoa conseguir preparar o ambiente e explicar os limites da simulação. Em uma organização real, o inventário também pode envolver domínio, contratos e serviços externos. Esses itens só entram no escopo quando efetivamente existem.

Propriedade institucional não significa compartilhar uma senha entre todas as pessoas. Sempre que o serviço permitir, usam-se contas individuais autorizadas, responsabilidades e mecanismos de recuperação apropriados. Credenciais podem precisar de rotação, e acessos de quem sai devem ser revogados no

momento adequado. Se houver exposição, segue-se a resposta de contenção tratada no capítulo de segurança [S13, orientações sobre credenciais expostas]. Uma tabela pública de handoff jamais deve conter o valor dos segredos.

Inventário e transferência preenchidos

Ativo do laboratório	Responsável e substituição	Evidência para transferência
Repositório e revisão da entrega	Mantenedor e substituto a identificar no uso real	Acesso autorizado, revisão identificada e histórico acessível
Dados artificiais do catálogo	Pessoa responsável pelos exemplos	Schema documentado e validação executável
Snapshot de recuperação	Pessoa que conduz o ensaio e substituto	Arquivo identificado em sandbox/recuperacao e roteiro de restore
Operação local	Pessoa que prepara o ambiente	README, runtime observado e comandos oficiais
Decisões e limites	Responsável pela manutenção	Registro de simulação sem persistência e pendências atuais

Demonstrar uma restauração no escopo correto

Cópia de segurança e recuperação são capacidades diferentes. Uma cópia pode existir e ainda ser ilegível, incompleta ou incompatível com o código disponível. O pacote oferece npm run snapshot e npm run restore exclusivamente para dados fictícios em sandbox/recuperacao. O catálogo fonte não é sobrescrito. O ensaio ensina a verificar o procedimento e os dados restaurados, sem afirmar capacidade de recuperação de um serviço de produção.

O caso começa com o conjunto de referência identificado. Registra-se a revisão de código e a configuração do ensaio; prepara-se o snapshot; altera-se apenas a área controlada prevista; executa-se a restauração; comparam-se identificadores e campos relevantes. Depois se verificam os fluxos dependentes. Se a transferência exige um prazo máximo ou tolerância de perda, esses objetivos são declarados antes e medidos durante o ensaio. Sem registro de medição, não se afirma prazo de recuperação atingido.

Procedimento de handoff

- Liste ativos, acessos, decisões e obrigações existentes. Marque o que não se aplica em vez de inventar serviços para completar o modelo.
- Atribua responsável e substituto. Confira se ambos possuem acesso autorizado e conhecem como recuperar o acesso sem divulgar credenciais.
- Selecione uma revisão e entregue instruções de preparação, validação, inicialização e desligamento. Declare o escopo da operação e suas limitações.
- Peça que a pessoa que recebe reproduza o roteiro em ambiente controlado. Registre dúvidas, passos ausentes e resultados; a documentação será corrigida a partir dessas observações.
- Ensaie restauração e saída de fornecedor quando aplicáveis. Verifique formato de exportação, destino e preservação do conteúdo necessário.

- Atualize acessos e responsáveis após a transferência. Preserve evidências e decisões que continuarão necessárias à manutenção.

Erros que impedem a continuidade

Sinal de alerta	Reparo verificável
Ativo crítico vinculado apenas a uma pessoa	Definir titularidade e recuperação adequadas, com substituto autorizado
Backup nunca restaurado	Executar ensaio controlado e comparar dados e comportamento
Fornecedor sem caminho de exportação conhecido	Investigar formato e condições de saída antes de assumir dependência
README depende da memória de quem escreveu	Registrar passos ausentes e repetir o roteiro por quem recebe

Condição observável de transferência

A transferência está demonstrada no escopo escolhido quando a pessoa receptora consegue realizar as tarefas obrigatórias com acesso apropriado e evidências aceitáveis. Pendências não impeditivas têm responsável, prazo e aceitação. Ativos precisam poder ser transferidos e recuperados quando isso se aplica; credenciais e permissões precisam poder ser revogadas. Não se usa um único verbo para tratar todos como se fossem o mesmo tipo de objeto. O encerramento de uma conversa de handoff não prova essa capacidade.

Exercício com dados e solução

Dados: uma entrega possui README atualizado e snapshot identificado; somente a pessoa que sai consegue acessar o repositório; ninguém tentou restaurar; a aplicação reinicia e perde reservas da simulação. Escolha duas ações que bloqueiam o handoff e registre a limitação que deve permanecer explícita.

Solução: providenciar acesso institucional e substituto autorizado ao repositório é uma ação impeditiva; ensaiar a restauração no escopo controlado é outra. A perda de reservas após reinício é uma limitação conhecida do registrador em memória: deve constar do aceite e impede prometer persistência. A rubrica exige responsável, evidência esperada para cada ação e distinção entre limitação aceita do laboratório e capacidade de operação real. Nenhuma revisão humana ou transferência concluída é presumida por preencher a tabela.

Referências S13 S19 Detalhes e limites na seção de referências.

Parte VIII Limites e antídotos

Controlar o custo e a utilidade do processo

Capítulo 48 Excesso de documentação

Documentar uma decisão custa manter uma decisão

A documentação ajuda quando alguém consegue decidir ou executar algo com ela. Um arquivo que repete outro sem declarar qual prevalece aumenta o trabalho: cada mudança exige conferir duas versões e cada divergência transfere ao leitor a escolha da verdade. Orçamento documental é a decisão explícita sobre quais registros merecem ser produzidos e mantidos. Não é uma quota universal de páginas nem uma autorização para apagar evidências incômodas.

Na Oficina Caracol, o catálogo público, os testes e a operação de reserva simulada possuem responsabilidades distintas. O README conduz a execução; o registro de decisão explica uma escolha durável; a evidência identifica uma verificação; o gate decide se os resultados permitem avançar. Escrever quatro descrições diferentes do comando de teste não melhora nenhum desses papéis. Uma descrição canônica e três remissões são mais fáceis de corrigir.

Caso trabalhado de consolidação

Considere este caso didático: a equipe criou doze arquivos ao planejar o catálogo. Quatro repetem requisitos, três ensinam a iniciar o servidor, dois contêm decisões contraditórias e três guardam resultados de ensaios de versões diferentes. A contagem identifica trabalho de revisão, mas não determina o que descartar. A decisão é preservar a informação necessária e retirar a competição entre autoridades.

Registro mantido	Pergunta respondida	Regra de manutenção
pratica/README.md	Como instalar o contexto e executar os comandos oficiais	Alterar quando execução ou pré-condições mudarem
pratica/docs/ decisao-001.md	Por que o exemplo usa arquivos locais e não banco externo	Revisar se o escopo exigir concorrência ou persistência diferente
pratica/docs/ evidencia-001.md	O que foi verificado e em qual contexto	Não converter resultado antigo em resultado da versão nova
pratica/docs/gate-001.md	O próximo avanço está autorizado	Reabrir quando uma evidência obrigatória perder validade

A tabela é o orçamento decidido para esse conjunto de informações, não uma proibição de novos documentos. Os resultados antigos permanecem identificados como históricos. Os arquivos repetidos deixam de ser fontes vigentes e passam a remeter ao registro escolhido, conforme a política de histórico do projeto. Uma decisão que perdeu validade recebe uma decisão sucessora; sua justificativa não desaparece silenciosamente.

Procedimento de revisão

- Identifique o leitor e uma ação concreta que cada documento precisa permitir. Se não houver ambos, investigue seu valor antes de mantê-lo por hábito.
- Agrupe afirmações repetidas e escolha a autoridade de cada domínio: requisitos, execução, decisões e resultados não precisam morar no mesmo arquivo.
- Compare divergências com o comportamento e as decisões autorizadas. Não escolha o documento mais recente apenas pela data.
- Consolide o conteúdo vigente, preserve o histórico necessário e acrescente remissões. Registre quem mantém cada autoridade.
- Peça a outra pessoa que encontre o comando de teste e a última decisão aplicável sem recorrer ao autor. Registre dificuldade e corrija a navegação.

Erros e reparos específicos

- Apagar todos os registros antigos reduz volume e pode destruir a explicação de uma mudança. Reparo: distinguir documento vigente, histórico preservado e duplicação dispensável antes de excluir.
- Criar um relatório para cada conversa dispersa decisões. Reparo: resumir apenas o que muda um compromisso, um risco ou o próximo passo, na autoridade correspondente.
- Centralizar tudo num arquivo gigantesco também cobra manutenção. Reparo: separar por pergunta e responsabilidade, mantendo uma entrada que permita localizar as partes.

Condição observável de avanço

A revisão pode avançar quando cada afirmação operacional relevante possui uma autoridade identificada, os vínculos chegam ao conteúdo vigente e o ensaio de localização encontra comandos e decisões sem contradição impeditiva. Uma quantidade pequena de arquivos, sozinha, não satisfaz esse gate. A pessoa responsável aceita explicitamente eventuais remissões quebradas que não afetem a execução, com prazo para correção.

Exercício com solução

Dados: A.md manda executar npm test; B.md manda executar npm run testes; C.md contém uma decisão de escopo ainda válida. O package.json disponível define test e não define testes. Proponha a consolidação sem perder a decisão.

Solução comentada: conferir os scripts do package.json e adotar npm test na entrada de execução; corrigir B.md para uma remissão ao README ou incorporá-lo, preservando o histórico escolhido; manter a decisão de C.md em registro identificável e vinculado ao requisito afetado. A resposta é insuficiente se apenas apagar B.md sem investigar sua origem ou eliminar C.md por não conter código. A rubrica exige autoridade escolhida, contradição removida, decisão preservada e um teste de localização reproduzível.

Referências S07 S08 Detalhes e limites na seção de referências.

Capítulo 49 Paralisia por análise

Quando a próxima pergunta precisa de um ensaio

Analisar é comparar possibilidades antes de comprometer recursos. A análise paralisa quando o grupo continua discutindo sem identificar qual informação poderia mudar a decisão. A saída não é programar qualquer coisa. É formular uma pergunta menor, escolher uma observação capaz de respondê-la e limitar o investimento. Timebox significa reservar um intervalo máximo para essa investigação e decidir o que fazer quando ele terminar.

Uma escolha reversível, como a apresentação inicial do catálogo, permite um ensaio barato. Uma mudança que possa destruir dados exige preparação maior. Reversível não significa sem custo: a equipe precisa conhecer a forma de retorno. No exemplo didático, trocar cartões por uma lista pode ser desfeito no código; publicar informações não autorizadas não se desfaz apenas trocando o layout.

Caso trabalhado com dados explicitamente simulados

Na Oficina Caracol, duas apresentações do catálogo disputam preferência. A pergunta deixa de ser qual parece mais moderna e passa a ser se uma pessoa encontra o serviço de manutenção e identifica sua descrição. O ensaio proposto usa os mesmos dados fictícios, a mesma tarefa e duas versões de interface. A tabela seguinte é material de exercício criado para discutir a decisão; não relata teste real com usuários.

Apresentação	Tarefas concluídas no ensaio fictício	Problema observado
Cartões	3 de 4	Uma descrição ficou escondida pelo resumo
Lista	4 de 4	Nenhum impedimento anotado nessa tarefa

A decisão didática é continuar com a lista e preservar a alternativa enquanto se verifica teclado, largura estreita e estado vazio. Quatro tarefas não demonstram superioridade universal, acessibilidade nem demanda comercial. O ganho aqui é eliminar uma dúvida suficiente para o próximo passo, preservando a possibilidade de retorno. A evidência real só existirá quando alguém executar o protocolo e guardar seus próprios resultados.

Procedimento do experimento

- Escreva uma hipótese que possa perder: a apresentação escolhida permite localizar e compreender o serviço sem ajuda.
- Fixe o que permanece igual, incluindo dados, tarefa e contexto. Se todo o conteúdo mudar junto, a comparação ficará difícil de interpretar.
- Defina um limite de trabalho, por exemplo uma sessão de quarenta minutos para preparar e revisar o protocolo. Esse tempo é orçamento, não garantia de qualidade.
- Especifique antes da observação o que conta como tarefa concluída e quais impedimentos bloqueiam avanço, como um controle inacessível por teclado.

- Execute apenas se houver meios adequados. Se não houver participantes, faça uma inspeção técnica e registre esse limite, sem chamá-la de teste com usuários.
- Encerre com uma decisão: escolher uma alternativa provisória, reformular a pergunta ou adiar com uma dependência explícita.

Protótipo e produto verificável

Um protótipo representa uma ideia para aprender; pode ser um desenho e não operar o serviço. Um experimento é um procedimento para produzir informação; pode usar um protótipo ou código. O Produto Mínimo Verificável do método precisa executar o fluxo essencial assumido no escopo. Uma maquete que sugere a existência de uma reserva não comprova uma reserva idempotente. O instrumento adequado depende da pergunta, e esses três nomes não devem ocultar a diferença.

Erros e reparos específicos

- Repetir a reunião sem nova pergunta consome tempo sem reduzir incerteza. Reparo: escrever qual observação mudaria a decisão e interromper o debate se ninguém conseguir indicá-la.
- Escolher o critério depois de conhecer o resultado favorece a opção preferida. Reparo: registrar critério e limite antes do ensaio.
- Tratar ausência de participantes como aprovação de uso transfere uma lacuna para o produto. Reparo: declarar inspeção técnica, manter a validação de uso pendente e limitar o avanço autorizado.

Gate e exercício resolvido

Avance quando houver uma decisão proporcional à pergunta, uma observação identificada e limites aceitos. Se faltar a informação que protege um requisito obrigatório, o encerramento do timebox não libera a entrega.

Exercício: restam trinta minutos; a discussão envolve cor do botão, vazamento de registros inativos e preferência entre lista e cartões. Há testes automatizados disponíveis, mas nenhum participante. Ordene o trabalho e justifique.

Solução: verificar primeiro o não vazamento de inativos, pois REQ-002 é comportamento obrigatório; registrar o resultado real. Em seguida, inspecionar a interação e escolher provisoriamente a apresentação que permita completar o fluxo, sem alegar preferência de usuários. Adiar a discussão estética e marcar validação de uso. A rubrica exige prioridade por impacto, separação entre inspeção e teste com pessoas, decisão reversível e pendência explícita.

Referências S01 Detalhes e limites na seção de referências.

Capítulo 50 Fragmentação e visão do todo

Uma fronteira precisa facilitar uma mudança

Coesão é a relação entre responsabilidades que fazem sentido juntas. Acoplamento descreve a dependência entre partes. Um arquivo curto pode depender de tantos outros que se torna difícil compreender uma alteração; um arquivo longo pode misturar interface, regra e persistência e tornar qualquer correção arriscada. A quantidade de arquivos e linhas ajuda a localizar suspeitas, mas não decide uma arquitetura.

A pergunta operacional é quais partes precisam mudar juntas para uma capacidade continuar correta. Na Oficina Caracol, a regra de exibição do catálogo pertence a pratica/src/catalogo.mjs; a reserva simulada, a pratica/src/operacoes.mjs; o protocolo HTTP, a pratica/src/servidor.mjs; a apresentação do navegador, a pratica/public/app.js. Essa divisão expressa responsabilidades. Ela precisa ser confirmada pela leitura do pacote, pois um nome de arquivo não prova o conteúdo.

Caso trabalhado de navegação

Considere uma revisão didática de REQ-002: um registro inativo não pode aparecer no catálogo público. O mantenedor começa pelo teste desse comportamento, localiza a regra de catálogo e examina o caminho usado pelo servidor. Não deve precisar corrigir cópias da regra em três lugares. A tabela mostra um mapa de investigação, e não resultados já obtidos nesta página.

Pergunta	Artefato de consulta	Consequência para a decisão
Qual comportamento impede a regressão	pratica/tests/catalogo.test.mjs	O teste deve proteger a regra de visibilidade
Onde a decisão de catálogo está implementada	pratica/src/catalogo.mjs	Uma autoridade para essa regra
Como o navegador recebe o resultado	pratica/src/servidor.mjs e pratica/public/app.js	Conferir se a integração preserva a restrição
A regra foi duplicada em apresentação	Leitura dos consumidores e testes de servidor	Eliminar competição sem quebrar a defesa necessária

Se a interface também filtra dados por conveniência, isso não autoriza enviar informações restritas ao navegador. A responsabilidade de não revelar o registro deve ser protegida na fronteira pública. Consolidar cópias de lógica é diferente de remover verificações complementares de fronteira. O mapa precisa explicar essa distinção para evitar uma refatoração que pareça limpa e amplie a exposição.

Procedimento de avaliação

- Escolha uma mudança real ou uma regressão conhecida. Avaliar uma árvore abstrata sem objetivo favorece reorganizações cosméticas.
- Anote o percurso de leitura: teste, regra, consumidor e dados. Conte apenas as transições relevantes e escreva por que cada uma foi necessária.

- Examine o histórico disponível para descobrir partes que mudam juntas. Mudanças conjuntas são indício de relação, não prova automática de que tudo deve morar no mesmo módulo.
- Proponha uma fronteira que reduza responsabilidades misturadas ou autoridades duplicadas. Nomeie o que ela recebe e o que devolve.
- Altere em pequena escala e execute as verificações aplicáveis. Atualize o mapa quando ele deixar de representar o código.

Critério contextual em vez de número mágico

O gate exige que outra pessoa consiga explicar o caminho obrigatório do requisito, localizar a autoridade da regra e fazer a alteração prevista com testes pertinentes. Se o grupo adotar um limite de navegação, deve registrar a tarefa, o percurso anterior e o motivo do limite. Não existe aqui uma regra universal de três arquivos ou cem linhas. A comparação útil é entre percursos equivalentes e compreensíveis, preservando o comportamento.

Erros e reparos específicos

- Criar um arquivo para cada expressão espalha uma única responsabilidade. Reparo: reunir elementos que formam um contrato coeso e manter nomes que expliquem sua função.
- Mover código sem procurar seus consumidores deixa uma fronteira aparente. Reparo: mapear importações, rotas e testes antes de concluir a mudança.
- Unificar regras semelhantes com contratos diferentes cria uma abstração enganosa. Reparo: explicitar as diferenças e compartilhar apenas o que realmente possui o mesmo significado.

Exercício com solução

Dados: a regra de serviço ativo está copiada em catalogo.mjs e app.js; servidor.mjs envia todos os registros; somente um teste de interface verifica que os inativos ficam invisíveis. O que corrigir primeiro?

Solução comentada: corrigir a fronteira pública para que registros inativos não sejam enviados, proteger essa restrição com teste pertinente e fazer a apresentação consumir o contrato público. Depois, avaliar se o filtro da interface ainda possui função de defesa ou apenas duplica uma regra. A resposta não recebe aprovação se apenas esconder o cartão, porque o dado ainda seria entregue. A rubrica exige autoridade identificada, proteção da fronteira, teste de regressão e mapa atualizado.

Referências S07 S17 Detalhes e limites na seção de referências.

Capítulo 51 Falsa segurança e suficientemente bom

O mínimo precisa cumprir um compromisso

Suficientemente bom significa atender aos critérios do recorte declarado com limites conhecidos e risco residual aceitável. Não significa aceitar qualquer coisa porque o prazo acabou. Também não significa esperar por todo aprimoramento imaginável. O trabalho de engenharia consiste em identificar o essencial, produzir evidência adequada e decidir se as lacunas restantes permitem o próximo uso pretendido.

Produto Mínimo Verificável, ou PMV, é o nome usado neste método para um fluxo mínimo realmente executável e avaliável. Um desenho pode ajudar a escolher a interface, mas não demonstra leitura dos dados ou tratamento de falhas. Um teste de uma função pode demonstrar sua regra, mas não comprova sozinho o caminho completo do navegador ao servidor. A evidência deve acompanhar exatamente a afirmação feita.

Caso trabalhado do catálogo

O recorte deste caso é a capacidade catálogo da Oficina Caracol. Seus itens obrigatórios são REQ-001 a REQ-004: listar ativos, não revelar inativos, apresentar estados vazio e inválido e verificar navegação por teclado e feedback acessível. REQ-005 continua obrigatório para a capacidade de reserva e para uma entrega que anuncie o conjunto completo. Retirá-lo deste gate específico não autoriza anunciar reservas prontas.

Critério do recorte	Evidência adequada	O que ela não demonstra
Ativos aparecem com campos públicos	Teste da função e resposta de GET /api/casos	Compreensão do conteúdo por usuários
Inativos não aparecem nem são enviados	Teste da resposta pública com um registro inativo	Controle de acesso para dados de uma aplicação real
Vazio e falha têm resposta explícita	Cenários de servidor e inspeção da interface	Ausência de todos os modos de falha
Teclado e feedback foram inspecionados	Registro manual de percurso, foco e mensagem	Conformidade integral com diretrizes de acessibilidade

O código abaixo é uma verificação isolada compatível com o pacote, executável dentro de pratica em contexto de módulo JavaScript. Na preparação desta revisão, o trecho foi executado isoladamente com Node e a asserção passou. Ele confirma essa entrada específica, não a interface nem todos os dados possíveis.

Código 31 javascript

```
import assert from 'node:assert/strict';
import { listarAtivos } from './src/catalogo.mjs';
const entrada = [{
  id: 'oc-inativo', titulo: 'Serviço em revisão',
  descricao: 'Exemplo didático que não deve ser publicado.', ativo: false
}];
assert.deepEqual(listarAtivos(entrada), []);
```

Procedimento de suficiência

- Defina a capacidade e seu conjunto de itens obrigatórios antes de avaliar maturidade. Deixe explícitas exclusões e futuras entregas.
- Ligue cada critério a uma verificação e registre versão, ambiente, resultado e limite. Um teste planejado não possui o mesmo estado de um teste aprovado.
- Separe falha impeditiva, lacuna não impeditiva e melhoria desejável. Considere o uso autorizado, não somente o esforço de correção.
- Peça à autoridade responsável que aceite o risco residual das lacunas não impeditivas, com prazo ou gatilho de revisão. Um aluno não deve inventar essa aceitação em nome de outra pessoa.
- Libere apenas o alcance sustentado pelas evidências. Se o pacote é de estudo local, não transforme esse gate em autorização de produção.

Erros e reparos específicos

- Um checklist marcado sem anexar o resultado gera confiança sem sustentação. Reparo: retornar cada marca ao registro que explica a verificação ou recolocá-la como pendente.
- Ocultar uma lacuna para melhorar o veredito elimina a condição de aceitação do risco. Reparo: tornar a lacuna visível e pedir decisão sobre seu impacto.
- Ampliar a publicidade da entrega além do escopo altera o risco sem nova revisão. Reparo: ajustar a afirmação ou ampliar as verificações antes de assumir o compromisso.

Gate e exercício resolvido

O avanço exige critérios obrigatórios atendidos ou exceção explícita não impeditiva compatível com o uso autorizado. Resultado medido, relatório escrito e risco apenas listado não bastam. Se uma evidência obrigatória for invalidada por mudança relevante, reabra a avaliação correspondente.

Exercício: testes de catálogo e servidor passam; o estado vazio aparece; um botão não recebe foco por teclado; animações ainda não foram adicionadas. O grupo quer liberar uma demonstração pública como plenamente acessível. Classifique cada item.

Solução: a animação é melhoria adiável se não integra o compromisso; o botão inacessível impede a afirmação e a aprovação de REQ-004; os testes aprovados continuam úteis, mas não substituem o percurso manual. O gate dessa divulgação fica bloqueado até reparar e verificar o controle ou restringir expressamente o uso, sem chamar a pendência de conformidade. A rubrica exige distinguir requisito, melhoria, escopo de evidência e afirmação pública.

Referências S06 S19 Detalhes e limites na seção de referências.

Parte IX Aplicação e kit operacional

Aplicar os perfis e usar modelos preenchidos

Capítulo 52 Perfil Essencial

Poucos controles com responsabilidade clara

O Perfil Essencial atende a um trabalho de baixo impacto e retorno viável. Ele reduz cerimônias, mas não reduz o compromisso com o comportamento anunciado. Na Oficina Caracol, uma demonstração local com dados fictícios permite aprender a preservar a base, executar verificações e produzir uma entrega pequena. A ausência de cadastro e de banco externo simplifica o contexto; não transforma erros de catálogo ou reserva em acertos.

Um perfil é uma decisão sobre controles proporcionais, não um selo de qualidade da pessoa ou do produto. A mesma aplicação pode exigir outro perfil se passar a atender usuários reais, guardar dados ou sustentar operações que não podem ser repetidas sem prejuízo. Reclassifique pelo contexto de uso, em vez de escolher o nome que exige menos trabalho.

Caso trabalhado de recepção do pacote

Imagine que um aluno recebe a Oficina Caracol para trocar conteúdo de exemplo e estudar o fluxo. Sua primeira tarefa não é redesenhar tudo. Ele precisa identificar a cópia recebida, preservar os dados fornecidos e descobrir se consegue repetir a execução. O roteiro abaixo é um artefato de trabalho preenchido para o cenário de recepção; as situações indicadas como pendentes só mudam após execução real.

Compromisso	Registro da recepção didática	Próxima verificação
Base	Pacote Oficina Caracol recebido para estudo local	Registrar a identificação e o manifesto da cópia utilizada
Escopo	Ajustar somente o conteúdo fictício autorizado	Conferir contrato id, titulo, descricao e ativo
Execução	Comandos oficiais lidos; resultados ainda não registrados neste cenário	Executar npm ci, npm test e npm run validate
Interface	Teclado, largura estreita e mensagens não inspecionados neste cenário	Anotar percurso e resultado manual
Retorno	Recuperação limitada à sandbox/recuperacao	Ler e executar o ensaio de snapshot e restore

Há poucos registros, mas cada um responde a uma pergunta. O aluno não pode chamar o cenário de aprovado porque a tabela está preenchida. Ela documenta justamente o que ainda precisa ocorrer. No pacote distribuído, os relatórios de execução fornecem os resultados efetivamente obtidos; uma execução anterior não substitui automaticamente a confirmação após suas alterações.

Procedimento essencial

- Leia pratica/README.md, package.json e o contrato dos dados. Confira a versão do Node e a compatibilidade declarada no pacote. Não instale ferramentas sugeridas por mensagens externas sem entender sua necessidade.
- Execute npm ci para respeitar o arquivo de lock fornecido. Dependências externas são zero neste projeto; a presença do comando não é convite para adicionar bibliotecas.
- Execute npm test e npm run validate, guardando os resultados completos e o contexto. Um código de saída favorável responde somente às verificações implementadas.
- Faça uma alteração pequena no conteúdo fictício e confira o catálogo público, os estados de interface e o teclado. Não use dados pessoais para deixar o exercício mais realista.
- Teste a recuperação na área permitida e faça o handoff, isto é, a transferência de contexto para quem continuará o trabalho. Registre os limites da reserva em memória.

Erros e reparos específicos

- Reescrever a aplicação antes de reproduzir a base apaga a comparação necessária ao aprendizado. Reparo: primeiro obter uma execução de referência e depois mudar uma unidade.
- Substituir dados sem autorização pode publicar conteúdo indevido. Reparo: usar apenas exemplos fictícios definidos no exercício e revisar os campos apresentados.
- Confundir restore com recuperação universal cria uma expectativa perigosa. Reparo: explicar que o comando do pacote opera somente em sandbox/recuperacao e não restaura código, produção nem reservas em memória.

Gate e exercício resolvido

O gate essencial permite avançar quando o fluxo do escopo funciona, as verificações obrigatórias têm resultado aceitável e as lacunas compatíveis com estudo local estão explicitamente aceitas. Publicação em outro contexto exige nova avaliação. Teste manual ainda não realizado permanece pendente, mesmo que a automação passe.

Exercício: o aluno alterou apenas titulo e descricao de um caso, npm test passou, mas não executou validate nem abriu a interface. Pode declarar a entrega concluída? Proponha o próximo passo mínimo.

Solução: não pode concluir o conjunto. Deve executar a validação prevista, inspecionar o resultado público e verificar os efeitos pertinentes no navegador, incluindo o conteúdo alterado e o percurso de teclado. Se não puder fazer a verificação manual, registra a limitação e solicita um gate limitado ao que comprovou. A rubrica exige escopo preservado, resultado real e distinção entre teste automatizado e inspeção humana.

Referências S06 S14 S19 Detalhes e limites na seção de referências.

Capítulo 53 Perfil Estruturado

Coordenar dados mudanças e responsabilidades

O Perfil Estruturado torna explícitos os contratos e os controles que permitem a pessoas diferentes alterar um sistema sem depender de memória individual. Pipeline é uma sequência definida de verificações e tarefas. Schema é o contrato da forma de um dado. Observabilidade, neste contexto de estudo, significa conseguir relacionar um resultado visível ao estado e às ações que o produziram. Os termos não exigem automaticamente serviços externos ou uma plataforma complexa.

A Oficina Caracol continua sem cadastro, banco externo ou dados pessoais. Podemos aumentar a disciplina de mudança sem fingir que construímos uma aplicação empresarial. Os papéis de autor, revisor e operador usados neste capítulo pertencem ao processo do exercício. Eles não representam contas ou permissões implementadas na interface. Se uma pessoa acumular papéis, o registro precisa dizer isso; não haverá revisão independente por simples troca de rótulo.

Caso trabalhado do contrato de dados

A mudança proposta é acrescentar um caso público. Cada registro deve conter exatamente id, titulo, descricao e ativo, com três strings não vazias e um booleano; os IDs são únicos. O exemplo a seguir é entrada didática válida. A conferência efetiva usa o validador do pacote.

Código 32 json

```
{
  "id": "oc-exemplo",
  "titulo": "Revisão de catálogo",
  "descricao": "Serviço fictício para aprender validação.",
  "ativo": true
}
```

Papel no exercício	Ação verificável	Registro produzido
Autor do conteúdo	Propõe o objeto e explica sua finalidade	Diff de data/casos.json e das expectativas de catálogo e HTTP afetadas, com ampliação explícita do escopo. Preservar a proteção de REQ-002.
Revisor do exercício	Confere autorização, contrato e clareza	Decisão registrada com identidade real de quem revisou
Operador local	Executa validação e confirma resposta pública	Resultado vinculado à cópia utilizada

Se o autor escrever ativo como a string "true", a entrada viola o contrato mesmo que o texto pareça convincente. Se acrescentar um campo chamado contato, a propriedade extra também será rejeitada. O esquema deliberadamente estrito torna a divergência observável e mantém o exercício sem dados pessoais. Não contorne o validador apenas para fazer uma mudança passar.

Procedimento estruturado

- Defina o impacto da alteração e quais requisitos e consumidores podem ser afetados. Acrescentar conteúdo válido difere de mudar a forma dos objetos.
- Registre a decisão de contrato. Se houver mudança incompatível, identifique como os consumidores serão adaptados antes de publicar outra forma de dado.
- Mantenha a validação automática no caminho de entrega. No projeto, npm run validate verifica sintaxe, dados e testes; a execução real precisa ser registrada.
- Faça o caminho integrado: o servidor lê os dados, a resposta pública contém apenas os campos autorizados e a interface apresenta o estado correspondente.
- Separe aprovação de conteúdo, evidência técnica e decisão de liberação. Uma pessoa pode executar mais de uma etapa, mas cada resultado precisa continuar reconhecível.

Métricas que respondem a perguntas

Uma métrica é uma medida escolhida para orientar uma decisão. Para este caso, pergunte quantos registros inválidos foram identificados na proposta, quais verificações obrigatórias falharam e se a tarefa pública foi concluída. Contar linhas de código não responde a essas perguntas. Uma execução sem erro não comprova preferência do usuário, disponibilidade prolongada nem segurança para produção. A interpretação acompanha o número; não nasce do painel.

Erros e reparos específicos

- Criar um painel de administração antes de definir quem pode publicar amplia o escopo e deixa o compromisso ambíguo. Reparo: manter neste exercício a alteração por arquivo e documentar a aprovação do conteúdo.
- Alterar o contrato e só depois procurar consumidores produz falhas de integração. Reparo: mapear servidor, navegador e testes antes da mudança e validar o conjunto.
- Guardar somente a frase tudo passou impede investigação. Reparo: registrar comandos, contexto, resultados e artefatos de saída, distinguindo cada execução.

Evoluir as expectativas de teste

Os testes entregues conferem os identificadores públicos previstos para a versão de exemplo. Ao adicionar um novo caso ativo, amplie explicitamente o escopo para atualizar as expectativas de tests/catalogo.test.mjs e tests/servidor.test.mjs. Inclua o novo identificador no resultado esperado e mantenha a verificação de que o inativo continua oculto. Essa atualização acompanha uma mudança legítima de requisito; apagar a asserção de exclusão eliminaria uma proteção necessária.

Gate e exercício resolvido

Avance quando o contrato estiver explícito, a proposta de conteúdo estiver autorizada no cenário real, as verificações aplicáveis estiverem aprovadas e a integração pública preservar os requisitos. M5 exige verificações aplicáveis aprovadas ou exceção explícita não impeditiva; M6 requer a integração comprovada, não apenas um plano.

Exercício: há dois registros com o mesmo id, um campo email extra e a informação de que o navegador abriu. O revisor pede liberação. Qual é sua decisão?

Solução: bloquear essa alteração, pois a entrada viola unicidade e formato e introduz um campo fora do escopo. Corrigir os dados fictícios, executar validação e conferir a resposta pública antes de reapresentar o gate. Abrir a página não prova contrato válido. A rubrica exige identificar as duas violações, preservar o limite sem dados pessoais e exigir evidência da correção.

Referências S08 S09 S19 Detalhes e limites na seção de referências.

Capítulo 54 Perfil Crítico e crise

Estabilizar antes de ampliar a mudança

Um incidente é uma interrupção ou degradação que exige resposta coordenada. Uma crise acrescenta pressão, incerteza e impacto que podem superar a capacidade habitual de trabalho. O Perfil Crítico pede controles reforçados conforme o risco. A Oficina Caracol não é apta a sustentar uso crítico; ela oferece um ambiente limitado para aprender decisões de contenção sem usar dados reais ou afetar terceiros.

A prioridade não é produzir a explicação mais elegante, mas reduzir o impacto preservando condições para investigar. Contenção reduz o dano imediato. Correção remove a causa identificada. Refatoração reorganiza a estrutura. Misturar as três coisas durante a pressão aumenta as variáveis e dificulta saber o que restaurou o comportamento. O retorno a uma versão anterior também precisa de pré-condições; não é um ritual seguro em qualquer situação.

Caso trabalhado de falha simulada

Cenário criado para o exercício: após uma edição de dados, GET /api/casos retorna 500 e a interface mostra falha. Isso é uma entrada narrativa, não um incidente realmente observado durante a escrita. O contrato prevê mensagem pública genérica quando o catálogo está inválido. A investigação deve verificar os dados locais sem enviar detalhes internos ao navegador. A tabela é um plano da primeira hora, com decisões próprias para esse cenário.

Intervalo de trabalho	Ação prioritária	Evidência ou decisão esperada
Início	Registrar versão, sintoma, horário e mudança conhecida	Um relato delimitado, sem supor a causa
Contenção	Suspender novas alterações e preservar o material necessário	Um único responsável coordena a intervenção
Diagnóstico	Reproduzir localmente e validar o contrato dos dados	Identificação da violação ou hipótese ainda aberta
Recuperação	Escolher retorno compatível com o material disponível	Plano que diferencia código, dados e estado em memória
Confirmação	Reexecutar o fluxo afetado e regressões pertinentes	Resultados identificados antes de encerrar o incidente

O comando npm run restore deste pacote só atua em sandbox/recuperacao. Ele não corrige automaticamente data/casos.json e não recupera reservas apagadas pelo reinício do processo. Se o catálogo fonte estiver inválido, a equipe precisa corrigir a cópia ou retornar seu conteúdo por um procedimento apropriado e preservado. A simulação de snapshot e restore serve para estudar recuperação delimitada; chamar esse ensaio de backup de produção seria falso.

Procedimento de contenção e investigação

- Nomeie uma pessoa coordenadora e distribua perguntas com fronteiras claras. Duas IAs não devem alterar simultaneamente o mesmo arquivo sem integração deliberada.
- Preserve o relato e o estado relevante antes de experimentar. Não sobrescreva a última cópia saudável com um estado defeituoso para apenas produzir um snapshot recente.
- Reproduza o defeito na cópia de estudo e registre o que confirma ou rejeita cada hipótese. Nunca publique segredos ou dados pessoais para aumentar a riqueza de um log.
- Faça a menor correção capaz de tratar a causa demonstrada. Acrescente uma regressão que detecte aquela falha e confira o comportamento integrado.
- Registre o retorno do serviço e separe melhorias estruturais em trabalho posterior, com responsáveis e gatilhos de revisão.

Erros e reparos específicos

- Apagar logs para limpar o ambiente remove pistas. Reparo: interromper o descarte, preservar o remanescente e declarar o que já não pode ser recuperado; um gate reaberto não recria evidência perdida.
- Refatorar durante a contenção mistura efeitos. Reparo: reduzir a intervenção à recuperação necessária e guardar a reorganização para um ciclo rastreável.
- Dividir a investigação por ferramentas em vez de perguntas produz respostas concorrentes. Reparo: compartilhar a mesma identificação da base, atribuir escopos e integrar conclusões com suas evidências.

Gate e exercício resolvido

Encerre o cenário de incidente quando o fluxo estiver estável nas verificações definidas, a correção tiver evidência e os riscos residuais possuírem decisão. Um ensaio bem-sucedido não comprova disponibilidade contínua nem habilita o aluno a operar sistemas críticos.

Exercício: após um erro 500, alguém propõe executar restore, reiniciar o servidor e anunciar recuperação das reservas. Há apenas reservas em memória e o restore atua na sandbox. Avalie a proposta.

Solução: rejeitar a afirmação. O comando não restaura o catálogo fonte e o reinício elimina o estado de reservas em memória. Primeiro registrar o contexto, verificar a causa e escolher a recuperação compatível com o componente afetado; depois testar e comunicar o alcance real. A rubrica exige reconhecer as três fronteiras de estado, preservar evidências e evitar promessa de recuperação que o projeto não implementa.

Referências S03 S07 S19 Detalhes e limites na seção de referências.

Capítulo 55 Checklists essenciais

Uma marca precisa apontar para uma observação

Este kit é uma ferramenta de consulta e não repete o formato dos capítulos de aprendizagem. Cada item recebe um estado: aprovado, falhou, pendente ou não aplicável com justificativa. Aprovado exige uma evidência identificável. Pendente não é uma aprovação provisória. Não aplicável descreve uma decisão de escopo legítima; não serve para retirar um requisito obrigatório depois que ele falhou.

Use os identificadores do checklist para localizar a prova, mas não confunda checklist, evidência e gate. O primeiro lembra o que conferir; a segunda sustenta uma afirmação; o terceiro decide se o resultado permite o avanço. A mesma evidência pode sustentar mais de um critério quando seu alcance estiver explicado.

Antes de alterar a Oficina Caracol

ID	Pergunta de conferência	Evidência a localizar
IN-01	Qual problema e qual capacidade estão no escopo	Requisitos e exclusões no registro de entrada
IN-02	Qual cópia está sendo examinada	Versão, manifesto e contexto de execução
IN-03	Quais caminhos podem mudar	Unidade do backlog com fronteiras declaradas
IN-04	Quais dados são permitidos	Contrato com dados fictícios e campos autorizados
IN-05	Quais critérios impedem a entrega	Gate com itens obrigatórios explícitos

Durante implementação e integração

ID	Pergunta de conferência	Evidência a localizar
IM-01	A mudança trata o requisito indicado	Diff no escopo e explicação da decisão
IM-02	O catálogo envia somente registros e campos públicos	Teste da função e resposta da API
IM-03	Vazio e dado inválido têm resultados distintos	Cenários registrados de servidor e interface
IM-04	Retry repete o resultado sem novo efeito	Teste de mesma chave e caso com contagem de reservas
IM-05	A mesma chave para outro caso é rejeitada	Resultado de conflito e ausência de efeito adicional
IM-06	A interface permite o percurso por teclado e informa o resultado	Registro manual de foco, acionamento e feedback

Antes de transferir ou publicar

ID	Pergunta de conferência	Evidência a localizar
EN-01	Os comandos oficiais foram executados na cópia avaliada	Saída completa, código de saída e ambiente
EN-02	A integração foi observada com o servidor real	Registro HTTP e identificação do estado testado
EN-03	O retorno foi ensaiado no alcance permitido	Snapshot e restore em sandbox/recuperacao
EN-04	As limitações acompanham o artefato entregue	README, evidência e handoff consistentes
EN-05	Uma pessoa recebeu e reproduziu o contexto	Aceite real do recebedor ou pendência explícita
EN-06	A autoridade aceitou as lacunas compatíveis com o uso	Gate com alcance, responsáveis e gatilhos

Aplicação preenchida ao pacote desta edição

A execução automatizada registrada em 2 de novembro de 2026 aprovou dezoito testes e a validação de sintaxe e dados. Isso fornece evidência para partes de IM-02 a IM-05 e EN-01. O smoke HTTP verificou respostas públicas e o retry. O ensaio de recuperação permaneceu na sandbox. Esses resultados não transformam IM-06 ou EN-05 em aprovados: teclado, tecnologia assistiva, leitura com alunos e aceite de transferência exigem observação humana não registrada nessa execução.

Se o conjunto obrigatório da capacidade inclui REQ-004 e a evidência manual permanece pendente, a capacidade não pode ser apresentada como M5 plenamente verificada. Um estado global deve usar o menor nível sustentado entre os itens obrigatórios definidos; itens opcionais não entram nesse mínimo. Uma autorização limitada para exploração local precisa ser distinguida da aprovação de publicação do aplicativo para usuários externos. Não altere o checklist para obter o veredito desejado.

Como corrigir o próprio checklist

Se um item repetidamente não ajuda a detectar erro nem orientar decisão, investigue se está mal escrito, duplicado ou fora do escopo. Troque perguntas vagas como qualidade adequada por comportamentos observáveis. Preserve a justificativa da alteração para que o próximo ciclo saiba por que o controle mudou. O método se beneficia de checklists menores e precisos, desde que a redução não esconda compromissos.

Referências S06 S15 S19 Detalhes e limites na seção de referências.

Capítulo 56 Modelos de governança

Registros preenchidos para uma cadeia verificável

Os registros deste capítulo estão conectados à Oficina Caracol. Eles exemplificam conteúdo preenchido, não uma coleção de campos vazios. A execução automatizada citada ocorreu em 2 de novembro de 2026, com Node v24.22.1 e npm 11.9.0, e consta do relatório pratica/docs/evidencias/resultado-pratica.json e dos artefatos de evidência do pacote. Esse contexto identifica a execução; resultados futuros devem receber identificação própria. O material de julho é a fonte histórica do método, não a data desses testes novos.

A cadeia é simples: a decisão ADR-001 limita a arquitetura; REQ-001 a REQ-005 definem comportamentos; EV-OC-001 resume verificações e limites; GATE-001 decide o alcance; rollback e handoff explicam como retornar e continuar. O registro não inventa assinatura humana, hash de commit ou aceite inexistente. Onde a confirmação depende de uma pessoa, o estado permanece pendente.

README de entrada

Código 33 markdown

```
# Oficina Caracol
Finalidade: estudar catálogo público e reserva simulada com dados fictícios.
Versão do pacote: 2.0.0.
Contexto verificado: Node v24.22.1; npm 11.9.0; execução em 2026-11-02.
Dados: data/casos.json; contrato {id,titulo,descricao,ativo}.
API pública: somente {id,titulo,descricao} dos registros ativos.
Execução dentro de pratica: npm ci; npm test; npm run validate.
Servidor local: npm start; abrir http://127.0.0.1:3000.
Recuperação de estudo: npm run snapshot; npm run restore.
Alcance da recuperação: somente sandbox/recuperacao.
Regras e decisões: docs/decisao-001.md.
Resultados e limites: docs/evidencia-001.md e registros brutos.
Liberação: consultar docs/gate-001.md.
Bloqueios humanos: percurso manual de acessibilidade, leitura com alunos,
revisão independente e aceite da autoridade permanecem pendentes.
Próximo passo: reproduzir localmente e registrar as verificações humanas.
Não há dados pessoais, cadastro ou banco externo.
As reservas existem somente na memória de um processo.
Este pacote não constitui autorização de uso em produção.
```

Os comandos do README são uma rota de entrada. O ponto e vírgula nesta descrição separa etapas; quem executa deve observar o resultado de cada comando antes de continuar. O servidor permanece em execução até ser encerrado. A recuperação é um exercício separado e não substitui a restauração do código.

Registro de decisão arquitetural preenchido

Código 34 markdown

```
# ADR-001 Persistência e fronteiras do exemplo
Status: decisão didática adotada nesta edição.
Contexto: ensinar catálogo, validação, HTTP e retry com dependências externas zero.
Requisitos relacionados: REQ-001, REQ-002, REQ-003 e REQ-005.
Alternativas: aplicação estática sem servidor; servidor com banco e framework;
Node com APIs padrão e memória local.
Decisão: catálogo em JSON validado; reservas simuladas em memória;
resposta pública construída somente com id, titulo e descricao.
Justificativa: tornar os contratos observáveis sem credenciais e serviços externos.
Consequências: reiniciar apaga reservas; não há deduplicação entre processos;
não há autenticação, disponibilidade contínua ou garantia de produção.
Evidências relacionadas: EV-OC-001 e testes de catálogo, operações e servidor.
Revisar se: surgirem usuários reais, dados pessoais, persistência obrigatória
ou múltiplos processos. A revisão exigirá novo escopo e novos controles.
Autoria do registro didático: edição revista da Engenharia Caracol.
Revisão humana independente: não realizada.
```

Registro de evidência preenchido

Código 35 markdown

```
# EV-OC-001 Verificações do pacote local
Afirmação sustentada: os cenários automatizados definidos foram executados
e os resultados abaixo foram obtidos no pacote Oficina Caracol 2.0.0.
Execução: 2026-11-02; Node v24.22.1; npm 11.9.0.
Procedimentos: npm ci; npm test; npm run validate; smoke HTTP;
npm run snapshot; npm run restore; exercício antes e depois.
Resultado de npm test: saída 0; 18 testes aprovados; 0 falhas inesperadas.
Resultado de validate: saída 0; sintaxe de 14 arquivos JavaScript,
dados de 3 registros e 18 testes aprovados.
HTTP: GET /api/casos devolveu 200 e IDs site-acessivel e catalogo-local.
Retry: mesma chave e caso devolveram sim-1 nas duas respostas 200.
Conflito: a mesma chave para outro caso devolveu 409 e CONFLITO_CHAVE.
Recuperação: snapshot e restore saíram com 0 na sandbox.
Antes: uma falha intencional isolada detectou exposição de rascunho-interno.
Depois: os 6 testes de catálogo passaram.
Artefato bruto: relatório pratica/docs/evidencias/resultado-pratica.json e logs do pacote.
Limites: não prova ausência de defeitos; não verifica interação humana;
não prova segurança de produção, persistência ou concorrência distribuída.
Observação: houve aviso npm sobre http-proxy herdado do ambiente;
os comandos citados concluíram com os códigos registrados.
Responsabilidade humana de revisão e aceite: pendente.
```

Observe a diferença entre teste falho e resultado inesperado: exercicio:antes foi criado para detectar uma implementação defeituosa isolada. Sua saída 1 era esperada e precisa permanecer visível. Esconder essa falha apagaria justamente a demonstração de que o teste protege a regra. Os seis testes do exercício depois já fazem parte da suíte principal de dezoito. Não os some como se fossem testes independentes.

Gate preenchido com impedimentos explícitos

Código 36 markdown

```
# GATE-001 Estado da edição de estudo
Escopo avaliado: pacote executável local com dados fictícios e documentação.
Evidência técnica: EV-OC-001 e resultados brutos identificados.
REQ-001, REQ-002, REQ-003 e REQ-005: cenários automatizados aplicáveis aprovados.
REQ-004: estrutura inspecionada; teclado, leitor de tela e alunos pendentes.
Riscos residuais: reservas voláteis; recuperação só sandbox; cobertura limitada.
Veredito para publicação do aplicativo para usuários externos: BLOQUEADO.
Motivo: verificações humanas e aceite da autoridade ainda não registrados.
Exploração local: os resultados técnicos sustentam o exercício descrito;
não equivalem a aprovação pedagógica ou operacional para terceiros.
Próxima ação: executar o roteiro manual e obter revisão e aceite identificados.
Responsável por assumir o aceite: ainda não designado nominalmente.
Prazo de liberação: não definido; a falta de prazo não elimina o bloqueio.
Reabrir: mudança relevante em contrato, código, dados ou contexto de uso.
Não declarar M5 completo da capacidade com REQ-004 pendente.
```

Um gate bloqueado é um resultado útil: mostra precisamente o que a automação permite afirmar e qual decisão continua aberta. Após a verificação humana, produza uma revisão do registro, vinculando as novas evidências. Não altere silenciosamente a fotografia anterior para sugerir que a aprovação já existia.

Rollback preenchido e delimitado

Código 37 markdown

```
# Plano de retorno da demonstração
Gatilho: o exercício de recuperação precisa restaurar os dados fictícios da sandbox.
Pré-condição: ler o procedimento e manter o trabalho em ambiente de estudo.
Fonte preservada: data/casos.json não é alvo dos comandos de recuperação.
Snapshot: npm run snapshot produz sandbox/recuperacao/snapshot.json.
Restore: npm run restore produz sandbox/recuperacao/atual.json.
Resultado observado em 2026-11-02: ambos concluíram com saída 0,
3 registros e SHA-256 coincidente no relatório de execução.
Confirmação adicional: o teste de recuperação verifica preservação do catálogo fonte.
Fora do alcance: código, dados de produção e reservas em memória.
Se o componente defeituoso for outro: parar, preservar evidência
e planejar um retorno específico; não chamar este script de solução universal.
Comunicação: registrar componente afetado, ação escolhida, resultado e limites.
Ensaio em ambiente de produção: não realizado e não autorizado pelo exemplo.
```

Handoff preenchido para o próximo ciclo

Código 38 markdown

```
# Transferência da Oficina Caracol
Finalidade: permitir reprodução e continuação do estudo local.
Estado recebido: suíte automatizada verificada; validações humanas pendentes.
Arquitetura: catálogo JSON; regras em catalogo.mjs e operacoes.mjs;
HTTP em servidor.mjs; interface em public.
Executar: seguir README e registrar a versão local do Node.
Fontes de estado: contrato, ADR-001, EV-OC-001 e GATE-001.
Acessos: nenhum cadastro ou credencial externa necessário ao exercício.
Risco de continuidade: reiniciar o processo apaga reservas simuladas.
Retorno: consultar rollback.md e respeitar a fronteira da sandbox.
Próximo trabalho: verificar teclado e feedback, revisar com iniciantes
e apresentar os resultados à autoridade responsável pela edição.
Recebedor nominal e confirmação de reprodução: pendentes.
O registro está preenchido; o aceite humano ainda não ocorreu.
```

Essa última linha impede uma confusão frequente: entregar o texto de handoff é uma ação do emissor; alguém conseguir continuar o trabalho é uma observação sobre a transferência. São eventos diferentes e exigem registros diferentes.

Referências S07 S08 S14 S15 S19 Detalhes e limites na seção de referências.

Capítulo 57 Prompts para a Vida Relacional

Um pedido operacional contém contexto e limite de prova

Prompts orientam uma ferramenta de linguagem; não substituem requisitos, execução nem responsabilidade. Vida Relacional designa aqui a colaboração entre pessoas, registros, código e ferramentas, em que uma mudança precisa manter seus vínculos com o que já foi construído. O projeto concreto destes pedidos é a Oficina Caracol, não uma aplicação com credenciais ou dados reais. Copie o pedido pertinente, forneça o pacote e confirme que a ferramenta pode acessar os caminhos indicados.

Antes de prometer um resultado, a ferramenta deve declarar o que conseguiu ler e executar. Sem terminal, pode analisar texto fornecido; não pode inventar saída de npm test. Sem navegador, pode inspecionar HTML e JavaScript; não pode registrar percurso real por teclado. Uma conclusão de inventário pode decorrer de leitura, mas uma conclusão sobre execução precisa de execução identificada.

Inventário sem alteração

Código 39 text

```
Contexto: recebi Oficina Caracol 2.0.0 para estudo local.
Raiz de trabalho: pratica. Não altere arquivos.
Leia README.md, package.json, data/casos.json, src, public, tests e docs.
Liste os caminhos realmente lidos e indique os que não estiverem disponíveis.
Identifique comandos existentes, contrato dos dados, rotas, testes e limitações.
O contrato de casos é {id,titulo,descricao,ativo}; a saída pública contém
apenas id,titulo,descricao dos ativos. Reservas são simuladas em memória.
Separe encontrado, inferido e não localizado; forneça caminho e trecho
para cada afirmação relevante. Não afirme execução se fez somente leitura.
Entregue mapa de responsabilidades, lacunas e próximo teste proposto.
Não instale dependências nem crie arquivos.
```

Implementação com fronteira

Código 40 text

```
Contexto: Oficina Caracol, REQ-003, mensagem de catálogo vazio.
Objetivo: no app do navegador, representar array público vazio com mensagem
explícita; distinguir vazio de carregamento e de erro.
Leia public/app.js, public/index.html, public/styles.css e os contratos antes de mudar.
Altere somente o arquivo necessário entre public/app.js e public/index.html.
Se precisar de outro caminho, explique a necessidade antes de ampliar o escopo.
Preserve botões nativos, região de status e foco visível já existentes.
Não crie cadastro, dados pessoais, banco externo ou dependência.
Use os comandos existentes npm test e npm run validate se houver terminal.
Entregue diff, motivo, resultados reais e limitações. Inspeção de código
não é prova de teclado ou leitor de tela; mantenha essa verificação pendente.
Se não houver acesso ao pacote, devolva proposta isolada com esse limite,
sem apresentá-la como alteração aplicada.
```

Correção com regressão demonstrada

Código 41 text

```
Contexto: Oficina Caracol e REQ-002, inativos não podem ser revelados.
Use npm run exercicio:antes para observar a falha intencional isolada.
Registre o código de saída e explique por que 1 é esperado nesse comando.
Compare o defeito do exercício com src/catalogo.mjs e seus consumidores.
A implementação defeituosa não deve ser copiada para src.
Execute npm run exercicio:depois e a suíte normal. Identifique quais
resultados sustentam a correção e quais não cobrem integração ou interface.
Se encontrar outro defeito real, reproduza-o em teste pertinente, faça
a menor correção e mostre falha antes e aprovação depois, sem inventar saídas.
Não silencie a falha esperada do exercício nem altere o critério para passar.
Descreva como retornar sua mudança e liste os limites restantes.
```

Auditoria com afirmações proporcionais

Código 42 text

```
Audite a cópia fornecida da Oficina Caracol contra REQ-001 a REQ-005.
Leia requisitos, decisões, código, testes, resultados e gate.
Para cada achado, informe caminho, trecho, fato observado, impacto
e ação proposta. Diferencie defeito demonstrado, hipótese e melhoria.
Se houver terminal, execute npm test e npm run validate e guarde as saídas.
Se houver servidor, confira GET /api/casos e o retry de reserva.
Sem os meios necessários, marque a verificação como não realizada.
Não use o relatório de outra versão como prova da cópia atual.
Não transforme inspeção de HTML em aprovação de acessibilidade.
Emita LIBERADO, RESSALVAS ou BLOQUEADO para um escopo declarado;
justifique lacunas e identifique quem poderia aceitar o risco.
Não invente aceite humano nem libere publicação validada com pendências impeditivas.
```

Integração com fronteiras de efeito

Código 43 text

```
Verifique a integração da Oficina Caracol em ambiente local de estudo.
Confirme primeiro o contrato vigente em README, src e tests.
A partir de npm start, confirme a resposta pública de GET /api/casos.
Envie POST /api/reservas com chave teste-pedido-001 e casoId site-acessivel.
Repita exatamente a mesma intenção com a mesma chave e compare resultados.
Depois use essa chave com casoId catalogo-local: o contrato prevê 409.
Identifique os resultados como sua execução local; não fixe sim-1 como
ID universal, pois o processo pode ter recebido outras reservas.
Explique que o servidor mantém estado apenas em memória e encerre-o ao fim.
Não execute testes contra serviços externos ou dados reais.
Atualize evidência e proposta de gate somente com observações realizadas.
Teste manual de teclado e feedback permanece etapa humana separada.
```

Crítica de uma resposta sem prova

Resposta inadequada: “Revisei o código e todos os testes passaram; o sistema está acessível e pronto para produção.” Há três saltos: leitura foi promovida a execução; execução foi promovida a validação de uso; demonstração local foi promovida a produção. A correção é pedir os comandos, saídas, cópia examinada e limites. Se não existirem, a ferramenta deve substituir a frase por uma descrição do que realmente fez.

Resposta proporcional: “Inspecionei os arquivos fornecidos e identifiquei tratamento explícito de vazio. Não executei a aplicação nem o percurso de teclado. Recomendo as verificações indicadas; não posso afirmar aprovação delas.” Quando houver execução real, acrescente os resultados e mantenha a conclusão no alcance observado. O valor do prompt está em tornar a colaboração examinável, inclusive quando a resposta honesta é que falta evidência.

Referências S04 S06 S15 S19 Detalhes e limites na seção de referências.

Parte X Conclusão

Consolidar autonomia e responsabilidade

Capítulo 58 A engenharia como memória

Lembrar o que permite continuar

A qualidade de um projeto não está apenas no código atual. Ela também está na capacidade de explicar por que esse código existe, quais compromissos ele cumpre e como alguém pode modificá-lo ou recuperá-lo. Memória de engenharia é informação selecionada para sustentar essas ações. Acumular arquivos sem autoridade não preserva esse caminho; pode escondê-lo.

A Oficina Caracol oferece um exemplo pequeno dessa memória. O contrato delimita o que um caso pode conter; a decisão explica por que as reservas são simuladas em memória; os testes protegem cenários; a evidência registra o que uma execução observou; o gate separa exploração local de publicação externa. Esses registros só ajudam enquanto seus vínculos continuam legíveis e suas afirmações correspondem ao estado examinado.

Retornar sem apagar o percurso

Quando uma hipótese é refutada, não é necessário fingir que a decisão anterior jamais existiu. Identifique a nova evidência, declare o que mudou e substitua a autoridade vigente com uma ligação para o histórico pertinente. O retorno fica organizado porque a equipe conhece o ponto de partida, a causa da revisão e o próximo compromisso.

Esse cuidado não obriga a conservar toda conversa. A memória útil distingue resultado bruto, interpretação, decisão e registro superado. Ela também reconhece o que não sabe. Uma anotação de teste manual pendente informa mais que um selo de qualidade sem observação. Um gate bloqueado pode ser a descrição mais precisa do momento e permitir que o próximo trabalho seja pequeno e claro.

Uma medida prática de continuidade

Entregue a cópia do projeto a alguém que não acompanhou a conversa. Essa pessoa consegue localizar a execução, explicar a limitação de reservas e escolher a próxima verificação? Enquanto esse ensaio não acontecer, a transferência permanece proposta, não comprovada. A finalidade da documentação é tornar possível essa continuidade, com esforço proporcional ao risco e sem depender da memória do autor.

Referências S07 S08 Detalhes e limites na seção de referências.

Capítulo 59 O código como consequência e prova

A ideia precisa encontrar o comportamento

Chamar o código de consequência não reduz sua importância. Significa que a implementação responde a um problema e a compromissos identificados. A partir daí, ela também oferece resistência às nossas afirmações: um requisito bonito pode não estar implementado; uma decisão coerente pode falhar quando encontra dados inválidos; uma função correta pode ser integrada de maneira errada. O código executado ajuda a descobrir essas diferenças.

Na Oficina Caracol, declarar idempotência não cria a propriedade. A operação precisa associar chave e intenção, repetir o resultado para a mesma intenção e recusar a reutilização incompatível. A prova disponível é delimitada: os cenários executados mostram esse comportamento no registrador em memória e no servidor de um processo. Não demonstram persistência após reinício nem coordenação entre processos.

Um exemplo isolado que pode ser examinado

O trecho seguinte usa a API real do pacote e pode ser executado em contexto de módulo JavaScript dentro de pratica. Ele também foi executado isoladamente na preparação desta revisão, sem falhas nas asserções. Os registros do pacote apresentam a suíte mais ampla; não se deve confundir este ensaio específico com uma verificação de todo o sistema.

Código 44 javascript

```
import assert from 'node:assert/strict';
import { criarRegistradorDeReservas } from './src/operacoes.mjs';
const r = criarRegistradorDeReservas();
const pedido = { chave: 'conclusao-001', casoId: 'site-acessivel' };
const primeira = r.reservar(pedido);
const repetida = r.reservar(pedido);
assert.deepEqual(repetida, primeira);
assert.equal(r.contar(), 1);
assert.throws(
  () => r.reservar({ chave: 'conclusao-001', casoId: 'catalogo-local' }),
  erro => erro.codigo === 'CONFLITO_CHAVE'
);
assert.equal(r.contar(), 1);
```

As asserções comparam resultados e efeitos; não basta observar duas mensagens visualmente iguais. Ainda assim, o teste tem entradas e contexto definidos. A integração, a interface e a operação exigem suas próprias verificações. Não existe um salto legítimo de este caso passou para tudo está certo.

A engenharia continua após uma aprovação

Uma aprovação autoriza um avanço sob condições. Se o contrato ou o contexto mudar, a evidência pode deixar de sustentar a conclusão anterior. A resposta adequada é revisar seu alcance, repetir as verificações afetadas e atualizar a decisão. O código não encerra o método; ele materializa um compromisso que continua examinável.

Referências S15 S19 Detalhes e limites na seção de referências.

Capítulo 60 Autonomia do desenvolvedor júnior

Autonomia se observa em decisões delimitadas

Autonomia não é deixar uma pessoa sozinha diante de uma ambiguidade. É oferecer contexto, fronteiras, critérios e meios para que ela execute, avalie o resultado e reconheça quando precisa de ajuda. Um desenvolvedor júnior demonstra progresso quando consegue explicar uma mudança, identificar dependências afetadas, selecionar verificações pertinentes e limitar suas afirmações ao que observou.

A Engenharia Caracol de Graciliano Tolentino propõe uma trilha para organizar esse trabalho. A leitura do método e a conclusão da Oficina Caracol não garantem competência profissional nem equivalência com a experiência de outra pessoa. Essa avaliação exige tarefas variadas, revisão, tempo de prática e observação em contextos reais compatíveis com a responsabilidade assumida. O livro torna o percurso mais examinável; não substitui a experiência.

Uma progressão concreta de responsabilidade

Situação	Ação que demonstra avanço	Limite que precisa reconhecer
Reproduzir a base	Executa comandos e identifica o que falhou	Não modifica tudo antes de compreender a execução
Fazer uma mudança pequena	Relaciona requisito, diff e teste	Não amplia autorização por conveniência
Tratar um defeito	Reproduz, corrige e demonstra regressão	Não atribui causa sem evidência
Integrar uma capacidade	Confere o caminho completo e seus estados	Não confunde função isolada com fluxo pronto
Transferir o trabalho	Explica estado, riscos e próximo passo	Não inventa o aceite do recebedor

Como usar a Oficina para verificar o aprendizado

Uma avaliação inicial pode pedir que o aluno acrescente um caso fictício válido, demonstre que o inativo não aparece e explique uma rejeição por contrato inválido. Uma avaliação posterior pode pedir que investigue uma reserva repetida, identifique o efeito único e descreva o que acontece ao reiniciar o processo. As tarefas avaliam raciocínio e execução; decorar os nomes de maturidade não basta.

O avaliador deve pedir explicação e observar o percurso, evitando confundir texto persuasivo com compreensão. Se uma ferramenta gerou a correção, o aluno precisa explicar por que ela trata o defeito e quais efeitos pode introduzir. Consultar documentação e usar ferramentas é parte do trabalho; transferir a elas uma certeza que não foi verificada impede aprender com o resultado.

Pedir ajuda também é uma habilidade

Uma boa solicitação de ajuda inclui objetivo, cópia examinada, comportamento esperado, resultado observado, tentativas e limites. Esse relato reduz o trabalho de reconstrução de quem ajudará. Em contrapartida, quem orienta precisa distinguir o que está ensinando, o que decidiu e o que continua incerto. O crescimento acontece quando a responsabilidade aumenta junto com a capacidade de demonstrar decisões e de reconhecer fronteiras.

Referências S01 S19 Detalhes e limites na seção de referências.

Capítulo 61 Pessoas IA e o futuro do método

Capacidade ampliada exige continuidade

Ferramentas de inteligência artificial ampliam a velocidade de leitura, comparação e geração. Também podem produzir respostas plausíveis sobre arquivos que não leram e testes que não executaram. O problema não se resolve com confiança irrestrita nem com rejeição automática. A colaboração precisa de fontes identificadas, fronteiras de alteração e evidências proporcionais ao que será afirmado.

A Oficina Caracol mostra essa relação em escala de estudo: a ferramenta pode propor uma mudança, mas a execução registra o que aconteceu; a evidência delimita a conclusão; uma pessoa com responsabilidade definida decide sobre riscos e uso. Uma resposta convincente não exerce, por si, autoridade sobre conteúdo, publicação ou revisão humana.

O próprio método precisa aceitar revisão

Controles que não reduzem risco, não preservam memória e não melhoram a execução devem ser investigados e simplificados. Se o registro custa mais que a decisão que protege, reveja o formato. Se o checklist não detecta uma classe de erro que apareceu, ajuste a pergunta. Se um nível de maturidade é atribuído sem evidência, corrija a avaliação em vez de defender o rótulo.

As referências bibliográficas fornecem caminhos de fundamentação. Elas não representam aprovação externa da Engenharia Caracol, leitura integral presumida ou cobertura automática de um currículo. Cada relação entre capítulo e fonte precisa informar o tema pertinente e o alcance de consulta. A expansão de uma bibliografia e o teste de uma proposta didática são trabalhos diferentes.

Estado honesto desta edição

Esta é uma edição revista para revisão e uso de estudo com os limites descritos. Os resultados automatizados do pacote foram registrados; a revisão humana independente, a avaliação com alunos e os aceites necessários não devem ser declarados realizados sem seus próprios registros. Os capítulos principais oferecem procedimentos, exemplos e exercícios conforme a função didática. O kit organiza consulta e os capítulos de conclusão sintetizam compromissos; não se afirma que todos possuem o mesmo molde.

Novas verificações humanas devem acrescentar contexto e observações precisas. Um percurso de teclado examinado em certo navegador não certifica conformidade integral com WCAG nem substitui a experiência de pessoas com deficiência. Uma descoberta posterior pode exigir voltar ao requisito, ao código ou ao próprio texto. Essa possibilidade de retorno organizado é uma força do método.

Compromisso de trabalho

Compreender antes de implementar. Preservar antes de reconstruir. Explicitar as dependências. Testar antes de declarar o comportamento. Registrar o que permite continuar. Aceitar o que a evidência demonstra e reconhecer o que permanece aberto. Retornar sem desorganizar e entregar sem perder a responsabilidade pelas afirmações.

Referências S04 S06 S12 Detalhes e limites na seção de referências.

Glossário

As definições ajudam a consultar o vocabulário usado nesta edição. O exemplo indica uma aplicação concreta; a remissão aponta onde o conceito aparece no percurso, sem substituir o estudo do capítulo.

Acessibilidade

Condições que permitem a pessoas com diferentes capacidades perceber, compreender e operar a interface. Sua verificação depende do contexto e não é certificada por um único teste automático.

Exemplo Percorrer os botões do catálogo por teclado e registrar se o foco e o feedback são perceptíveis.

Consultar capítulos 5.

Acoplamento

Dependência entre partes do programa: mudar uma pode exigir conferir ou alterar a outra. O objetivo é tornar essa ligação necessária e compreensível.

Exemplo A página depende do formato de casos devolvido pelo servidor.

Consultar capítulos 16.

ADR

Registro de uma decisão arquitetural que explica contexto, alternativas, escolha e consequências. Serve para compreender por que a solução foi adotada e quando revê-la.

Exemplo A decisão de usar APIs padrão, sem bibliotecas externas, fica em pratica/docs/decisao-001.md.

Consultar capítulos 8.

API

Conjunto de operações e regras pelas quais uma parte do programa oferece serviços a outra. Uma API pode ser local ou usar mensagens pela rede.

Exemplo A interface consulta a API do servidor para obter os casos públicos.

Consultar capítulos 5.

Argumento

Valor entregue a uma função quando ela é chamada. É diferente do parâmetro, que é o nome usado na definição para receber esse valor.

Exemplo Ao consultar um caso pelo identificador, o identificador informado é um argumento.

Consultar capítulos 4.

Arquivo

Unidade de informação guardada com um nome e um caminho. Pode conter texto, código, dados ou outro conteúdo.

Exemplo pratica/data/casos.json contém os registros fictícios do catálogo.

Consultar capítulos 1.

Array

Coleção ordenada de valores do JavaScript, acessados por posições que começam em zero. Não é o mesmo que um único objeto com propriedades nomeadas.

Exemplo O catálogo recebe um array com vários objetos de caso.

Consultar capítulos 2.

Auditoria

Exame de afirmações e evidências contra critérios declarados. Seu resultado deve registrar achados e limites, sem presumir que a inspeção encontrou todos os problemas.

Exemplo Conferir se a afirmação de ocultar inativos é sustentada pela resposta da API e pelos testes.

Consultar capítulos 8.

Backlog

Lista organizada do trabalho considerado para o projeto, com motivo, prioridade, dependências e critérios de conclusão. Um item listado ainda não é uma entrega pronta.

Exemplo O item que corrige a exposição de inativos permanece aberto até sua prova integrada.

Consultar capítulos 19.

Baseline

Medição de referência usada para comparar o resultado de uma mudança. Só permite comparação útil quando cenário, ambiente e medida são identificados.

Exemplo Medir a listagem antes de uma alteração e comparar o mesmo cenário depois dela.

Consultar capítulos 40.

Booleano

Tipo de valor que admite verdadeiro ou falso. Em JavaScript, true e false sem aspas são booleanos; o texto entre aspas é outro tipo.

Exemplo O campo ativo precisa ser booleano; o texto sim não satisfaz esse contrato.

Consultar capítulos 2.

Build

Preparação de um conjunto de arquivos que pode ser identificado e utilizado. Pode existir sem compilar JavaScript, desde que se saiba exatamente o que será executado.

Exemplo Identificar quais arquivos do catálogo compõem a versão que será entregue.

Consultar capítulos 42.

Caminho

Indicação de onde um arquivo ou diretório está localizado. Um caminho relativo depende da pasta usada como referência.

Exemplo Dentro de pratica, src/catalogo.mjs aponta para o módulo de catálogo.

Consultar capítulos 1.

Capacidade

Comportamento útil que o sistema consegue oferecer e que pode ser demonstrado. Não equivale apenas a um arquivo escrito ou a uma tarefa marcada.

Exemplo Consultar a lista pública sem receber casos inativos é uma capacidade verificável.

Consultar capítulos 6.

Ciclo

Conjunto de trabalho que produz e verifica uma capacidade. A prova do ciclo reúne as unidades em um comportamento integrado.

Exemplo O ciclo de listagem reúne contrato dos dados, seleção, apresentação e verificação do percurso.

Consultar capítulos 6.

Cliente

Parte que inicia uma solicitação a um serviço. Não significa necessariamente uma pessoa que comprou um produto.

Exemplo O navegador atua como cliente ao pedir a lista de casos ao servidor.

Consultar capítulos 21.

Código de saída

Valor numérico com que um processo informa seu resultado ao sistema que o iniciou. Nas verificações do pacote, zero indica sucesso e um valor diferente de zero indica falha.

Exemplo Um teste reprovado deve impedir que a validação seja interpretada como aprovada.

Consultar capítulos 6.

Coesão

Relação entre elementos que trabalham juntos para uma finalidade reconhecível. Ajuda a decidir o que deve permanecer na mesma unidade.

Exemplo Validação dos casos reúne regras sobre os mesmos dados, em vez de espalhá-las por arquivos sem motivo.

Consultar capítulos 16.

Commit

Registro identificado de uma revisão no histórico do Git. Guarda uma referência para recuperar e comparar o trabalho registrado.

Exemplo Associar uma execução de testes à revisão do catálogo que ela verificou.

Consultar capítulos 6.

Condição

Expressão usada para decidir se um caminho de execução será seguido. O resultado da avaliação determina qual bloco será executado.

Exemplo A seleção inclui um caso quando seu campo ativo é verdadeiro.

Consultar capítulos 3.

Const

Declaração do JavaScript que impede reatribuir outro valor ao mesmo vínculo. Não congela o interior de um objeto nem de um array.

Exemplo Um caso declarado com const ainda pode ter sua propriedade titulo alterada se não houver outro controle.

Consultar capítulos 1.

Continuidade

Conservação do contexto necessário para outra pessoa ou ferramenta retomar o trabalho. Inclui decisões, estado atual, evidências, limites e próximo passo.

Exemplo O registro de passagem explica qual gate permanece bloqueado e qual verificação falta.

Consultar capítulos 8.

Contrato

Descrição explícita do que uma parte recebe, devolve e faz nas condições previstas. Pode incluir tipos, restrições e respostas de falha.

Exemplo O contrato dos casos exige id, titulo e descricao não vazios e ativo booleano.

Consultar capítulos 2.

Credencial

Informação usada para demonstrar identidade ou autorização, como uma senha ou um token. Uma credencial exposta pode continuar válida mesmo após ser apagada do arquivo atual.

Exemplo Um registro de evidência não deve conter o valor de um token real.

Consultar capítulos 6.

Critério de aceite

Condição observável que determina se um comportamento atende ao requisito. Precisa permitir comparar entrada, resultado esperado e resultado observado.

Exemplo Dado um caso inativo, a listagem pública não deve entregá-lo.

Consultar capítulos 12.

CSS

Linguagem que define a apresentação visual de documentos web, como cores, espaçamento e foco visível. A aparência não substitui as regras de seleção dos dados.

Exemplo pratica/public/styles.css apresenta os cartões e o indicador de foco do catálogo.

Consultar capítulos 5.

Decomposição

Divisão de uma capacidade em partes compreensíveis e verificáveis. A divisão precisa preservar como essas partes serão reunidas.

Exemplo Separar contrato, seleção e apresentação, mantendo uma prova integrada da listagem.

Consultar capítulos 18.

Dependência

Algo necessário para outra parte funcionar ou ser verificada. Pode ser um módulo, um contrato, um recurso de execução ou um pacote externo.

Exemplo A página depende do formato da resposta da API, mesmo sem usar bibliotecas externas.

Consultar capítulos 1.

Diff

Comparação que mostra diferenças entre versões ou entre um arquivo atual e uma revisão registrada. Ajuda a delimitar o que mudou, mas não demonstra sozinho que a mudança funciona.

Exemplo Revisar o diff para confirmar que uma correção de seleção não alterou estilos sem relação.

Consultar capítulos 6.

Diretório

Pasta que reúne arquivos e outros diretórios. O diretório de trabalho é a referência que o terminal usa para caminhos relativos.

Exemplo Os comandos oficiais são executados dentro do diretório pratica.

Consultar capítulos 1.

Endpoint

Ponto de acesso a uma operação de uma API. Em HTTP, sua identificação envolve o método e o caminho, além das regras da operação.

Exemplo A consulta GET ao caminho /api/casos entrega a lista pública.

Consultar capítulos 21.

Erro

Diferença ou falha que impede compreender a instrução, executar uma ação ou atender ao comportamento previsto. Seu tratamento depende da causa e do efeito.

Exemplo Um ativo escrito como texto viola o contrato, mesmo que o arquivo JSON esteja bem formado.

Consultar capítulos 1.

Escopo

Limite do que uma capacidade, versão, prova ou autorização abrange. Também declara o que ainda precisa de outra decisão.

Exemplo O pacote simula reservas em memória; seu escopo não equivale a um serviço real de reservas.

Consultar capítulos 9.

Estado

Situação de um dado, componente ou processo em determinado momento. Uma operação pode mudar esse estado, que precisa ser observado no contexto correto.

Exemplo O contador de reservas é um estado em memória que se perde ao reiniciar o processo.

Consultar capítulos 2.

Evidência

Informação identificada que sustenta uma afirmação dentro de limites. Deve permitir reconhecer versão, procedimento, resultado e alcance da conclusão.

Exemplo Uma resposta da API demonstra quais registros foram entregues naquela execução.

Consultar capítulos 6.

Exceção

No código, sinalização de uma situação anormal que interrompe o fluxo normal até ser tratada ou propagada. Em decisões de aceite, a palavra também designa um desvio explicitamente aceito e justificado, que não pode esconder uma falha impeditiva.

Exemplo validarCasos lança TypeError para dados inválidos; uma ressalva editorial aceita no gate tem outro sentido de exceção.

Consultar capítulos 19, 30.

Foco

Elemento da interface que receberá a próxima ação de teclado. Sua posição precisa ser perceptível para que a pessoa saiba onde está operando.

Exemplo Ao usar Tab, o botão de reserva deve apresentar um indicador de foco visível.

Consultar capítulos 5.

Função

Conjunto nomeado de instruções que realiza uma tarefa e pode receber entradas e devolver um resultado. Separar sua finalidade permite verificar a regra sem depender da tela.

Exemplo listarAtivos recebe casos e devolve os registros públicos ativos.

Consultar capítulos 4.

Gate

Decisão registrada sobre uma passagem, com escopo, critérios, evidência, autoridade e consequência. Pode liberar, liberar com ressalvas ou bloquear o avanço pretendido.

Exemplo A publicação que declara teclado verificado fica bloqueada enquanto essa evidência necessária não existir.

Consultar capítulos 8.

Git

Ferramenta de controle de versões que registra o histórico de arquivos e permite comparar revisões. Pode ser usada localmente, sem publicar o trabalho.

Exemplo Guardar uma revisão do catálogo antes de uma mudança facilita comparar o que foi alterado.

Consultar capítulos 6.

GitHub

Serviço que pode hospedar repositórios Git e apoiar colaboração. Não é a própria ferramenta Git nem é necessário para fazer commits locais.

Exemplo Um repositório da Oficina pode ser publicado no serviço, mas o exercício local de histórico não depende disso.

Consultar capítulos 6.

Granularidade

Tamanho escolhido para as unidades de trabalho ou de revisão. Precisa equilibrar compreensão local e integração, sem fragmentar por hábito.

Exemplo Agrupar uma regra e seus testes pode ser mais útil que criar uma tarefa para cada linha.

Consultar capítulos 18.

Handoff

Passagem do trabalho acompanhada das informações necessárias para retomá-lo. Inclui estado, decisões, riscos, evidências e próximo passo.

Exemplo O documento de passagem informa como iniciar o catálogo e qual verificação manual permanece pendente.

Consultar capítulos 47.

Hash

Valor calculado a partir de um conteúdo para ajudar a identificá-lo e detectar diferenças. Não demonstra sozinho que o conteúdo é correto, seguro ou autêntico.

Exemplo Comparar o hash registrado de um pacote ajuda a conferir se o conjunto recebido mudou.

Consultar capítulos 28.

Homologação

Avaliação de um resultado contra critérios de aceite e uso declarados. Não se confunde com apenas copiar arquivos para um destino.

Exemplo Conferir o catálogo integrado e decidir se atende ao incremento antes de aceitar sua entrega.

Consultar capítulos 42.

HTML

Linguagem que descreve a estrutura e os elementos de uma página web. Não determina, sozinha, toda a aparência e o comportamento.

Exemplo pratica/public/index.html contém a estrutura da lista, dos botões e da região de mensagens.

Consultar capítulos 5.

HTTP

Protocolo de troca de mensagens entre cliente e servidor. Uma solicitação informa operação e destino; a resposta inclui status, cabeçalhos e conteúdo.

Exemplo O navegador consulta /api/casos e recebe os registros públicos em JSON.

Consultar capítulos 5.

Idempotência

Propriedade de uma operação que, repetida nas condições definidas, conserva o mesmo efeito pretendido. Na simulação, a chave identifica uma mesma intenção e impede criar outra reserva para a repetição correspondente.

Exemplo Repetir a mesma chave e o mesmo caso devolve o mesmo resultado lógico sem aumentar o contador.

Consultar capítulos 23, 35.

Implantação

Colocação de uma versão em execução no destino previsto, também chamada deploy. Não prova automaticamente o aceite nem a capacidade de recuperar uma falha.

Exemplo Iniciar a versão identificada do catálogo no ambiente de destino e conferir seu funcionamento.

Consultar capítulos 35.

Integração contínua

Prática de integrar alterações com frequência e conferir o conjunto por uma construção e verificações repetíveis; também é chamada CI. Uma sequência automática isolada não demonstra que a equipe integra frequentemente nem que todos os requisitos foram aprovados.

Exemplo Ao integrar uma mudança do catálogo, repetir a validação do conjunto e tratar a falha antes de aceitar novo avanço.

Consultar capítulos 31.

Interface

Ponto de interação entre partes de um sistema ou entre pessoa e programa. Seu contexto define quais entradas, saídas e ações precisam ser compreendidas.

Exemplo A página do catálogo permite consultar serviços e acionar a reserva simulada.

Consultar capítulos 1.

Java

Linguagem de programação distinta de JavaScript. Seus nomes parecidos não tornam instruções e ambientes intercambiáveis.

Exemplo A Oficina executa JavaScript com Node e navegador; seus arquivos não são programas Java.

Consultar capítulos 1.

JavaScript

Linguagem usada nos exemplos para expressar dados, decisões e ações. O ambiente de execução determina quais recursos estão disponíveis.

Exemplo O Node executa a seleção de casos no servidor e o navegador executa a apresentação da página.

Consultar capítulos 1.

JSON

Formato textual para representar dados com regras de escrita próprias. Um texto JSON bem formado ainda pode violar o contrato do projeto.

Exemplo casos.json precisa ter os campos e tipos previstos, além de pontuação válida.

Consultar capítulos 2.

Laço

Estrutura que repete instruções para itens de uma coleção ou enquanto uma condição se mantém. É preciso acompanhar o estado a cada passagem.

Exemplo Percorrer os casos para reunir somente os títulos dos ativos.

Consultar capítulos 3.

Licença

Conjunto de permissões e condições aplicáveis ao uso e à distribuição de uma obra ou software. Precisa ser conferido antes de incorporar material de terceiros.

Exemplo Uma biblioteca proposta para cartões exige análise de licença antes de entrar no projeto.

Consultar capítulos 28.

Lockfile

Arquivo gerado pelo gerenciador para registrar uma resolução de instalação correspondente ao manifesto. Contribui para a repetição, mas não substitui a identificação do ambiente nem os testes.

Exemplo package-lock.json deve corresponder a package.json antes da conferência com npm ci.

Consultar capítulos 6.

Manifesto

Registro que declara elementos e seus papéis; o significado depende do contexto. O manifesto do pacote descreve comandos e dependências, enquanto o de governança identifica artefatos, versões e estados.

Exemplo package.json orienta o npm; docs/manifesto.json distingue documentos vigentes, candidatos e históricos.

Consultar capítulos 6, 24.

Maturidade

Estado de evidência de um item, e não percentual de esforço ou nota da pessoa. Na escala M0–M8, descrição, implementação, teste e integração precisam de provas diferentes.

Exemplo Uma capacidade com item obrigatório ainda em M4 não alcança M6 pela média dos demais.

Consultar capítulos 19.

Mock

Substituto controlado de uma dependência usado durante uma verificação. Ajuda a criar situações repetíveis, mas não demonstra a integração real da parte substituída.

Exemplo Uma resposta de falha controlada pode exercitar a mensagem da interface sem provar o servidor real.

Consultar capítulos 31.

Módulo

Parte de código com responsabilidade identificável e elementos que podem ser importados por outras partes. O arquivo deve tornar claro o que oferece e do que depende.

Exemplo src/catalogo.mjs oferece validação e seleção de casos para outros módulos.

Consultar capítulos 1.

Navegador

Programa que abre páginas web e executa seus recursos de interface. Seu ambiente é diferente do Node usado no servidor.

Exemplo Abrir http://127.0.0.1:3000 para operar a interface da Oficina.

Consultar capítulos 1.

Node.js

Ambiente que executa JavaScript fora do navegador e oferece recursos como arquivos, servidor HTTP e testes. É distinto da linguagem que executa.

Exemplo O comando npm start inicia o servidor da Oficina usando Node.

Consultar capítulos 1.

Null

Valor de JavaScript usado para representar uma ausência explícita de resultado ou referência. É diferente de undefined, que pode indicar um valor não atribuído ou uma posição inexistente.

Exemplo obterCasoAtivo devolve null quando não encontra um caso público correspondente.

Consultar capítulos 4.

Objeto

Valor que reúne propriedades relacionadas, acessadas por nomes. Suas propriedades podem ser alteradas mesmo quando o vínculo foi declarado com const, salvo outra restrição.

Exemplo Um caso reúne id, titulo, descricao e ativo.

Consultar capítulos 2.

Observabilidade

Possibilidade de compreender o estado e as ações do sistema a partir de sinais que ele fornece. Medir ou registrar algo não resolve automaticamente o problema revelado.

Exemplo Usar registros e resultados para identificar por que uma consulta ao catálogo falhou.

Consultar capítulos 46.

Parâmetro

Nome usado na definição de uma função para receber uma entrada. O valor informado na chamada é seu argumento.

Exemplo No contrato de listarAtivos, casos é o nome da entrada que recebe a coleção.

Consultar capítulos 4.

Persistência

Conservação de dados para além da execução em memória que os criou. Reiniciar um processo pode apagar dados que não foram persistidos.

Exemplo As reservas simuladas da Oficina somem ao reiniciar e não demonstram persistência.

Consultar capítulos 5.

Pipeline

Sequência definida de tarefas e verificações executadas segundo uma ordem e consequências conhecidas. Não precisa depender de um serviço externo.

Exemplo A validação reúne conferência de sintaxe, dados e testes antes de permitir avanço.

Consultar capítulos 53.

Porta

Número que identifica um ponto de comunicação de um serviço em um endereço. Uma porta já ocupada pode impedir iniciar outro servidor no mesmo local.

Exemplo O servidor de estudo atende localmente na porta 3000.

Consultar capítulos 13.

Prioridade

Importância relativa de um item no contexto e na versão definidos. Precisa considerar impacto e compromisso, sem apagar suas dependências.

Exemplo Corrigir a exposição de inativos é P1; uma melhoria visual reversível pode ser P3.

Consultar capítulos 14.

Promise

Objeto do JavaScript que representa o resultado futuro de uma operação, que pode ser concluída ou rejeitada. Permite organizar a espera por ações cujo término não é imediato.

Exemplo A consulta da interface ao servidor aguarda uma resposta antes de apresentar a lista ou a falha.

Consultar capítulos 5.

Proporcionalidade

Escolha de controles suficientes para as consequências e incertezas do contexto. Não significa eliminar toda verificação em projetos pequenos.

Exemplo O catálogo fictício mantém testes e gate, embora não exija os mesmos controles de um serviço real com dados pessoais.

Consultar capítulos 26.

RACI

Tabela de papéis: R executa, A responde pela decisão, C é consultado e I é informado. Ajuda a explicitar autoridade sem presumir que executar uma tarefa permite decidir qualquer mudança.

Exemplo A pessoa que implementa a tela não decide sozinha transformar o catálogo fictício em coleta de dados pessoais.

Consultar capítulos 9.

Rastreabilidade

Ligação identificável entre requisito, decisão, implementação, verificação e entrega. Permite saber por que uma prova existe e quando precisa ser revista.

Exemplo REQ-002 liga a seleção de ativos aos testes do módulo e da resposta pública.

Consultar capítulos 21.

README

Documento de entrada que explica finalidade, preparação, execução, verificações e limites do projeto. Deve permitir começar sem depender da conversa anterior.

Exemplo pratica/README.md indica a pasta de execução e o endereço local do catálogo.

Consultar capítulos 6.

Recuperação

Ação de restabelecer uma condição aceitável após uma falha. Pode exigir corrigir causa, restaurar dados e repetir verificações, conforme o efeito ocorrido.

Exemplo No laboratório, recuperar a cópia de dados e demonstrar que ela voltou ao conteúdo esperado.

Consultar capítulos 4.

Refatoração

Mudança na estrutura interna que preserva o comportamento considerado no escopo. Uma correção que muda o resultado exigido precisa ser reconhecida como mudança de comportamento.

Exemplo Separar a seleção da apresentação sem mudar quais casos públicos são devolvidos.

Consultar capítulos 14.

Regressão

Efeito indesejado de uma mudança sobre comportamento que já deveria funcionar. Reexecutar testes pertinentes ajuda a detectá-lo, sem garantir ausência de todos os defeitos.

Exemplo Após alterar a descrição de casos, conferir se a regra de ocultar inativos continua funcionando.

Consultar capítulos 6.

Repositório

Conjunto de arquivos e histórico controlado por uma ferramenta de versões, como Git. Pode existir localmente sem hospedagem em um serviço.

Exemplo O histórico do projeto permite localizar a revisão do catálogo associada à evidência.

Consultar capítulos 6.

Requisição

Mensagem enviada pelo cliente para solicitar uma operação ao servidor. Inclui informações necessárias para interpretar o pedido.

Exemplo A interface envia uma requisição para consultar os casos públicos.

Consultar capítulos 5.

Requisito

Necessidade ou condição que o sistema deve atender no contexto definido. Precisa ser traduzido em comportamento e critérios verificáveis.

Exemplo REQ-002 determina que registros inativos não sejam revelados pela listagem pública.

Consultar capítulos 3.

Restauração

Reaplicação de uma cópia de referência para recuperar o conteúdo ou estado previsto. Sua abrangência deve ser conhecida antes de executar a ação.

Exemplo O comando de restore do exercício opera somente em sandbox/recuperacao e não sobrescreve o catálogo fonte.

Consultar capítulos 42.

Retorno

Resultado devolvido por uma função; no método, a mesma palavra também nomeia a revisão controlada de trabalho anterior. O contexto deve esclarecer qual sentido está sendo usado.

Exemplo listarAtivos devolve a lista pública; um retorno Caracol reabre evidências afetadas por uma mudança.

Consultar capítulos 4, 8.

Retry

Nova tentativa de uma operação após uma falha ou ausência de resposta. Precisa considerar limite, tipo de falha e possibilidade de repetir efeitos.

Exemplo Ao tentar novamente a mesma reserva simulada, conservar a chave da intenção original.

Consultar capítulos 35.

Risco

Possibilidade de uma falha e suas consequências no contexto. Probabilidade, impacto, detecção e reversibilidade ajudam a escolher o tratamento.

Exemplo Enviar um inativo na resposta pública ameaça REQ-002 e exige uma prova na fronteira HTTP.

Consultar capítulos 6.

Rollback

Retorno a uma versão ou condição anterior como forma de conter uma mudança, quando seus efeitos permitem isso. Não desfaz automaticamente exposição de credenciais nem toda alteração de dados.

Exemplo O roteiro de rollback registra quando reverter uma mudança do catálogo e quais verificações repetir.

Consultar capítulos 46.

Runtime

Ambiente que executa o programa e fornece os recursos disponíveis. A compatibilidade precisa ser conferida na versão efetivamente utilizada.

Exemplo Node executa o servidor; o navegador executa a interface e fornece outras APIs.

Consultar capítulos 13.

Sandbox

Área isolada destinada a experimentar efeitos com alcance limitado. O isolamento precisa existir tecnicamente, não apenas no nome da pasta.

Exemplo snapshot e restore do laboratório operam em sandbox/recuperacao com dados fictícios.

Consultar capítulos 6.

Schema

Descrição da forma esperada de um dado, com campos, tipos e restrições. A validação verifica se uma entrada atende a essa descrição.

Exemplo O esquema dos casos exige campos conhecidos, IDs únicos e ativo booleano.

Consultar capítulos 41.

Segredo

Informação cujo valor deve permanecer restrito, como um token de acesso real. Registrá-lo num exemplo pode criar uma exposição em vez de uma evidência útil.

Exemplo Os documentos da Oficina usam dados fictícios e não devem conter valores de credenciais.

Consultar capítulos 21.

Servidor

Programa que recebe solicitações e oferece respostas segundo operações definidas. Não é necessariamente uma máquina física separada.

Exemplo src/servidor.mjs atende a API e os arquivos estáticos permitidos da Oficina.

Consultar capítulos 1.

Snapshot

Cópia de referência criada para observar ou recuperar um estado. Não comprova recuperação até que seu uso e seu alcance sejam verificados.

Exemplo O exercício guarda uma referência dos dados no sandbox antes de demonstrar restore.

Consultar capítulos 6.

Stack

Conjunto de tecnologias escolhido para construir e executar o projeto. Sua adequação depende dos requisitos e das consequências da escolha.

Exemplo A Oficina usa JavaScript, Node e APIs padrão do navegador, sem dependências externas.

Consultar capítulos 27.

String

Valor usado para representar texto em JavaScript. Um texto que parece número ou booleano continua sendo texto enquanto não houver uma conversão apropriada.

Exemplo O titulo de um caso é string; o campo ativo não deve receber a string true.

Consultar capítulos 2.

Terminal

Interface para executar comandos e ler seus resultados. O diretório atual influencia onde caminhos e arquivos serão procurados.

Exemplo Abrir o terminal dentro de pratica antes de executar os comandos do pacote.

Consultar capítulos 1.

Teste

Execução ou avaliação de um cenário para comparar o observado com o esperado. Seu alcance depende das entradas, das partes envolvidas e do critério usado.

Exemplo Fornecer um caso inativo e conferir que ele não é devolvido pela listagem pública.

Consultar capítulos 1.

Teste automatizado

Verificação executada por software com condições e comparação definidas. Não substitui toda avaliação humana nem comprova requisitos fora dos cenários incluídos.

Exemplo npm test executa as verificações automatizadas fornecidas no pacote.

Consultar capítulos 20.

Teste de integração

Verificação da colaboração entre partes e de suas fronteiras reais no escopo declarado. Distingue-se do teste unitário por incluir essa colaboração relevante.

Exemplo Iniciar o servidor de teste com dados controlados e conferir a resposta HTTP da listagem.

Consultar capítulos 15.

Teste de sistema

Avaliação do produto integrado em relação aos requisitos. Um percurso de ponta a ponta, às vezes chamado e2e, precisa declarar de qual entrada até qual resultado percorre o sistema e quais partes ficaram substituídas.

Exemplo Abrir o catálogo no navegador, usar o teclado e conferir o resultado servido pelos componentes integrados.

Consultar capítulos 38.

Teste manual

Verificação em que uma pessoa executa ações e registra observações contra critérios. Deve informar passos, contexto e limites, em vez de apenas dizer que parece bom.

Exemplo Percorrer controles com teclado e registrar foco e mensagens observadas.

Consultar capítulos 52.

Teste unitário

Verificação de uma pequena unidade com entradas e dependências controladas. Não demonstra sozinho o percurso completo entre navegador, servidor e dados.

Exemplo Exercitar listarAtivos com uma coleção conhecida sem iniciar a interface.

Consultar capítulos 38.

Timeout

Limite de tempo após o qual o cliente deixa de esperar uma resposta. Não demonstra que o servidor não executou a operação nem cancela automaticamente seu efeito.

Exemplo Uma reserva pode ter sido registrada mesmo que a resposta não tenha chegado ao cliente dentro do prazo.

Consultar capítulos 35.

Undefined

Valor de JavaScript que pode indicar ausência de atribuição ou acesso a uma posição inexistente. É diferente de uma ausência explícita representada por null.

Exemplo Ler uma posição fora do array de casos pode produzir undefined.

Consultar capítulos 2.

Unidade

Porção revisável de trabalho com finalidade, entrada, resultado e critério próprios. Seu tamanho deve preservar contexto e permitir integração.

Exemplo Uma unidade implementa a regra de seleção e seus testes; o ciclo comprova sua colaboração com a página.

Consultar capítulos 6.

URL

Endereço que identifica um recurso e como acessá-lo. Uma forma textual válida não garante que o destino exista ou seja o esperado.

Exemplo http://127.0.0.1:3000 identifica o endereço local usado no exercício.

Consultar capítulos 41.

Validação

Comparação de uma entrada com regras explícitas para aceitar, rejeitar ou sinalizar o dado. Estar bem escrito no formato JSON não basta para passar pelo contrato.

Exemplo validarCasos rejeita ativo com tipo incorreto e IDs duplicados.

Consultar capítulos 2.

Variável

Vínculo entre um nome e um valor usado pelo programa. As regras da declaração determinam se esse vínculo pode receber outro valor.

Exemplo Um contador de consultas usa um nome para guardar o total que muda após cada ação.

Consultar capítulos 2.

Verificação

Conferência de um resultado contra condições definidas. Registrar que ela ocorreu é diferente de demonstrar que seu resultado foi aceitável.

Exemplo A execução que revela um inativo na resposta é uma verificação reprovada, não uma aprovação do requisito.

Consultar capítulos 6.

Versionamento

Controle de revisões identificáveis para acompanhar, comparar e recuperar mudanças. Números de versão e histórico ajudam a localizar trabalho, mas não substituem evidências de funcionamento.

Exemplo Associar os arquivos do catálogo e seu gate à mesma revisão examinada.

Consultar capítulos 28.

Versionamento semântico

Convenção que atribui significado a partes de uma versão conforme a compatibilidade de uma interface pública. Só é útil quando o projeto segue o contrato declarado; não elimina testes.

Exemplo Ao considerar uma dependência futura, avaliar a mudança anunciada e repetir a compatibilidade em vez de confiar apenas no número.

Consultar capítulos 28.

Referências e relação com os capítulos

As referências abaixo identificam versão, localização consultada e limite de acesso. Consulta parcial não equivale à leitura integral de uma obra. As regras autorais do Caracol, os casos fictícios e o código da Oficina não são transcrições desses documentos. Os endereços levam à fonte ou à página de acesso, sem redistribuir livros e normas externas.

S01 Guide to the Software Engineering Body of Knowledge

IEEE Computer Society. 2024 com atualização 4.0a de 2025.

Trecho consultado Apresentação e indicação da revisão 4.0a de 25 de setembro de 2025

Situa áreas de conhecimento consolidadas. A combinação e os estados Caracol pertencem à proposta do autor.

Acesso e limite Consulta parcial à página oficial; o livro integral não foi incorporado nem distribuído.

Abrir a referência Consulta em 2 de novembro de 2026

S02 ISO IEC IEEE 12207 Systems and software engineering Software life cycle processes

ISO IEC IEEE. 2026 segunda edição.

Trecho consultado Abstract público e identificação da edição de abril de 2026

O resumo descreve processos ao longo do ciclo de vida e não impõe um modelo específico. Não é declaração de conformidade com a norma.

Acesso e limite Consulta parcial ao resumo e aos metadados oficiais; texto normativo integral não consultado.

Abrir a referência Consulta em 2 de novembro de 2026

S03 Secure Software Development Framework SSDF Recommendations for Mitigating the Risk of Software Vulnerabilities

Souppaya Murugiah Scarfone Karen Dodson Donna NIST. NIST SP 800-218 versão 1.1 fevereiro de 2022.

Trecho consultado Visão geral dos grupos e práticas PO.1 PO.2 e PO.4 da tabela 1 páginas impressas 5 a 9

Referência para incluir segurança no desenvolvimento e tratar verificações como práticas contextualizadas.

Acesso e limite Consulta parcial às práticas de requisitos papéis e critérios de segurança no documento oficial. Não houve avaliação de cumprimento integral.

Abrir a referência Consulta em 2 de novembro de 2026

S04 Secure Software Development Practices for Generative AI and Dual Use Foundation Models An SSDF Community Profile

Booth Harold e colaboradores NIST. NIST SP 800-218A julho de 2024.

Trecho consultado Abstract e escopo de produtores de modelos sistemas e adquirentes

Não converte um prompt de programação em conformidade com um perfil destinado a sistemas e modelos de IA.

Acesso e limite Consulta parcial ao resumo oficial e identificação. Leitura complementar para contextos com modelos de IA.

Abrir a referência Consulta em 2 de novembro de 2026

S05 Application Security Verification Standard

OWASP Foundation. ASVS versão 5.0.0 maio de 2025.

Trecho consultado Introduction e identificação da versão estável

Referência para formular verificações de segurança web; o catálogo de estudo não é certificado pelo ASVS.

Acesso e limite Consulta parcial ao projeto e à identificação da edição. Requisitos individuais precisam ser selecionados e verificados no projeto real.

Abrir a referência Consulta em 2 de novembro de 2026

S06 Web Content Accessibility Guidelines WCAG 2.2

World Wide Web Consortium. Recomendação de 12 de dezembro de 2024.

Trecho consultado Critérios 1.4.3 1.4.4 1.4.10 2.1.1 2.4.7 3.1.1 3.3.2 e 4.1.3

Orienta os roteiros de interface e a necessidade de verificar resultados. Os roteiros do exemplo cobrem um subconjunto.

Acesso e limite Consulta parcial aos critérios de contraste redimensionamento refluxo teclado foco idioma rótulos e mensagens de estado. Não é auditoria completa de conformidade.

Abrir a referência Consulta em 2 de novembro de 2026

S07 Pro Git

Chacon Scott Straub Ben. Segunda edição 2014 versão web consultada em 2026.

Trecho consultado Seções 2.2 Recording Changes to the Repository e 2.4 Undoing Things

Apoia comparação e registro de versões. Histórico de código não substitui revogação de uma credencial exposta.

Acesso e limite Consulta parcial a status diff staging commits e cuidados ao desfazer alterações. As operações de restauração do projeto são próprias da sandbox e não derivam de uma garantia de recuperação por Git.

Abrir a referência Consulta em 2 de novembro de 2026

S08 Using architectural decision records to streamline decision making during development

Kunce Darius Goby Dominik AWS. Guia web sem edição fixa.

Trecho consultado Introdução ADR process e Best practices

Apoia contexto alternativas decisão consequências e histórico do ADR. Os documentos preenchidos da Oficina são exemplos autorais.

Acesso e limite Consulta parcial às páginas do processo e de preservação de decisões substituídas.

Abrir a referência Consulta em 2 de novembro de 2026

S09 Continuous Integration

Fowler Martin. Artigo web atualizado em 2024.

Trecho consultado Definição e práticas de integração frequente com build e testes

Integração contínua é uma prática de integração e verificação frequente. A mera existência de uma ferramenta de pipeline não a demonstra.

Acesso e limite Consulta parcial ao artigo; sem reprodução extensa.

Abrir a referência Consulta em 2 de novembro de 2026

S10 Semantic Versioning

Preston Werner Tom e colaboradores. Especificação 2.0.0 tradução brasileira.

Trecho consultado Especificação itens 1 3 6 7 e 8

Apoia comunicação de mudanças na API. O número 2.0 deste livro identifica a edição editorial e não certifica uma API do método.

Acesso e limite Consulta parcial às regras de API pública imutabilidade de versão publicada e incrementos.

Abrir a referência Consulta em 2 de novembro de 2026

S11 Managing the Development of Large Software Systems

Royce Winston W. IEEE WESCON 1970 pp 1 a 9 reimpressão ICSE 1987 pp 328 a 338.

Trecho consultado Introdução e figuras 2 a 4 da cópia digital

A leitura histórica inclui a crítica ao teste tardio e relações de iteração; não reduz o artigo a defender uma sequência rígida sem retorno.

Acesso e limite Consulta parcial ao texto original reproduzido em fac símile hospedado por terceiro. Acesso à cópia não significa licença de redistribuição; o PDF externo não integra o pacote.

Abrir a referência Consulta em 2 de novembro de 2026

S12 Manifesto para Desenvolvimento Ágil de Software

Beck Kent e demais signatários. 2001 tradução brasileira.

Trecho consultado Declaração de valores e ressalva sobre valor dos itens comparados

Contextualiza colaboração software em funcionamento e adaptação. Não atribui aos signatários aprovação do Caracol.

Acesso e limite Consulta à declaração curta oficial; os princípios são leitura complementar.

Abrir a referência Consulta em 2 de novembro de 2026

S13 Removing sensitive data from a repository

GitHub Docs. Documentação web.

Trecho consultado About removing sensitive data e About sensitive data exposure

Credencial real exposta deve ser invalidada ou rotacionada; remover um arquivo da versão atual não elimina todas as cópias.

Acesso e limite Consulta parcial às seções de revogação rotação e alcance de cópias.

Abrir a referência Consulta em 2 de novembro de 2026

S14 npm ci

npm Documentation. Documentação web.

Trecho consultado Description e observações sobre coerência e opções do lockfile

A instalação limpa exige coerência entre manifesto lockfile e opções. O arquivo de lock contribui para repetibilidade sem provar sozinho todo o ambiente.

Acesso e limite Consulta parcial à documentação e aos resultados indexados da página oficial.

Abrir a referência Consulta em 2 de novembro de 2026

S15 Making retries safe with idempotent APIs

Featonby Malcolm Amazon Builders Library. Artigo técnico web.

Trecho consultado Retrying and side effects e Reducing client complexity

Apoia o exemplo autoral de mesma chave mesma intenção e conflito para conteúdo diferente. O registrador em memória não oferece garantia distribuída.

Acesso e limite Consulta parcial às seções sobre intenção chave de requisição e efeitos de repetição.

Abrir a referência Consulta em 2 de novembro de 2026

S16 Certified Tester Foundation Level Syllabus

International Software Testing Qualifications Board. Versão 4.0.1 2024.

Trecho consultado Seções 2.2 Test Levels and Test Types 2.2.3 Confirmation Testing and Regression Testing e 5.2 Risk Management

Apoia distinguir fronteira de teste e finalidade de reexecução. Não é acreditação do livro ou certificação do leitor.

Acesso e limite Consulta parcial aos níveis finalidade de regressão e relação entre risco de produto e esforço de teste.

Abrir a referência Consulta em 2 de novembro de 2026

S17 Refactoring

Fowler Martin. Site do autor.

Trecho consultado Definição de refatoração e catálogo Extract Function

Preservar comportamento observável é o foco; a escolha de proteção por testes depende do contexto, não de uma categoria única obrigatória.

Acesso e limite Consulta parcial à definição e ao exemplo de extração de função.

Abrir a referência Consulta em 2 de novembro de 2026

S18 JavaScript Guide e Getting started Your first website

MDN Web Docs. Documentação web.

Trecho consultado Grammar and types Control flow and error handling Functions e módulos de primeira página

Referência técnica para conceitos da linguagem e separação entre estrutura apresentação e interação. Não é curso integral reproduzido do MDN.

Acesso e limite Consulta parcial a sintaxe tipos controle funções e introdução web. Exemplos do livro são originais.

Abrir a referência Consulta em 2 de novembro de 2026

S19 Node.js Documentation

Node.js Project. Linha 24 documentação web consultada em 2026.

Trecho consultado Test runner Running tests from the command line HTTP createServer e Server.listen

Apoia execução dos módulos testes HTTP e arquivos. A versão efetivamente usada está registrada no pacote, sem prometer igualdade em todo sistema operacional.

Acesso e limite Consulta parcial às APIs utilizadas. Teste local realizado com Node v24.22.1; a página latest pode avançar para outra correção da linha 24.

Abrir a referência Consulta em 2 de novembro de 2026

Mapa de uso das referências

Cap.	Assunto	Fontes
01	Computador arquivos editor e primeira execução	S18, S19
02	Valores variáveis objetos e coleções	S18
03	Decisões repetições e acompanhamento da execução	S18
04	Funções módulos contratos e tratamento de erros	S18, S19
05	Página navegador interação e comunicação com o servidor	S18, S19, S06
06	Testes depuração versões e primeiro ciclo completo	S07, S16, S19
07	O que significa programar	S01
08	A identidade da Engenharia Caracol	S01, S07, S08
09	Papéis e autoridades	S01, S03
10	Linguagem natural estruturada	S01
11	Compreender profundamente o problema	S01
12	Escopo e critérios de aceite	S01, S06
13	Inventário do estado atual	S07, S14, S19
14	Preservar reparar refatorar ou reconstruir	S08, S17
15	Fonte única de verdade	S01, S07
16	Responsabilidade única e coesão	S01, S17, S19
17	Waterfall e Caracol	S11, S12, S16
18	Ciclos unidades e granularidade	S01, S16
19	Maturidade M0 a M8	S16, S03
20	Gates e vereditos	S03, S06, S16
21	Evidência e rastreabilidade	S16, S07, S03
22	Retorno controlado	S07, S16, S03
23	Raiz diretórios e porta de entrada	S07, S19
24	Manifesto numeração e mapa geral	S07, S03
25	Backlog prioridades e pronto	S01, S16
26	Risco e proporcionalidade	S16, S03
27	Tecnologias e compatibilidade	S08, S18, S19
28	Dependências e lockfile	S14, S10, S03
29	Comandos reproduzíveis	S14, S19, S13
30	Da especificação para o código	S18, S19
31	Uma unidade por vez sem perder o todo	S09, S19
32	Protocolo de IA	S07, S19
33	Revisão de código produzido por IA	S07, S19
34	Alucinações arquiteturais	S19
35	Erros e estados de falha	S15, S18, S19
36	Integração duplicação e mudanças	S17, S09
37	Estratégia de testes por risco	S16
38	Níveis de teste e proteção contra regressões	S16, S19
39	Segurança e privacidade desde o desenho	S03, S05, S13

Cap.	Assunto	Fontes
40	Desempenho e acessibilidade em condições observáveis	S06
41	Dados conteúdo e configuração como comportamento	S18, S03
42	Build entrega e decisão final	S19, S07
43	Auditoria documental arquitetural e de código	S01, S03
44	Dívida técnica e refatoração com propósito	S17
45	Migração versão e depreciação sem surpresa	S10, S15
46	Observabilidade e resposta a incidentes	S15, S19
47	Continuidade institucional e transferência do projeto	S13, S19
48	Excesso de documentação	S07, S08
49	Paralisia por análise	S01
50	Fragmentação e visão do todo	S07, S17
51	Falsa segurança e suficientemente bom	S06, S19
52	Perfil Essencial	S06, S14, S19
53	Perfil Estruturado	S08, S09, S19
54	Perfil Crítico e crise	S03, S07, S19
55	Checklists essenciais	S06, S15, S19
56	Modelos de governança	S07, S08, S14, S15, S19
57	Prompts para a Vida Relacional	S04, S06, S15, S19
58	A engenharia como memória	S07, S08
59	O código como consequência e prova	S15, S19
60	Autonomia do desenvolvedor júnior	S01, S19
61	Pessoas IA e o futuro do método	S04, S06, S12

Registro da edição e correspondência histórica

A edição fundacional foi preservada. Esta revisão substitui a prosa de molde por explicações e casos próprios, corrige os critérios de aprovação, acrescenta a entrada de programação e entrega os artefatos para estudo. A tabela permite reencontrar o assunto mesmo quando a numeração antiga era ambígua.

Número em julho	Página em julho	Número novo	Assunto preservado ou revisto
1	5	7	O que significa programar
2	7	8	A identidade da Engenharia Caracol
3	9	9	Papéis e autoridades
4	11	10	Linguagem natural estruturada
5	15	11	Compreender profundamente o problema
6	17	12	Escopo e critérios de aceite
7	19	13	Inventário do estado atual
8	21	14	Preservar reparar refatorar ou reconstruir
9	23	15	Fonte única de verdade
10	25	16	Responsabilidade única e coesão
11	29	17	Waterfall e Caracol
12	31	18	Ciclos unidades e granularidade
13	33	19	Maturidade M0 a M8
14	36	20	Gates e vereditos
15	38	21	Evidência e rastreabilidade
16	40	22	Retorno controlado
17	44	23	Raiz diretórios e porta de entrada
18	46	24	Manifesto numeração e mapa geral
19	48	25	Backlog prioridades e pronto
20	50	26	Risco e proporcionalidade
21	52	27	Tecnologias e compatibilidade
22	54	28	Dependências e lockfile
23	56	29	Comandos reproduzíveis
24	60	30	Da especificação para o código
25	62	31	Uma unidade por vez sem perder o todo
26	64	32	Protocolo de IA
27	66	33	Revisão de código produzido por IA

Número em julho	Página em julho	Número novo	Assunto preservado ou revisto
28	68	34	Alucinações arquiteturais
29	70	35	Erros e estados de falha
30	72	36	Integração duplicação e mudanças
31	76	37	Estratégia de testes por risco
32	78	38	Níveis de teste e proteção contra regressões
33	80	39	Segurança e privacidade desde o desenho
34	82	40	Desempenho e acessibilidade em condições observáveis
35	84	41	Dados conteúdo e configuração como comportamento
36	86	42	Build entrega e decisão final
37	90	43	Auditoria documental arquitetural e de código
38	92	44	Dívida técnica e refatoração com propósito
39	94	45	Migração versão e depreciação sem surpresa
40	96	46	Observabilidade e resposta a incidentes
41	98	47	Continuidade institucional e transferência do projeto
42	102	48	Excesso de documentação
43	104	49	Paralisia por análise
44	106	50	Fragmentação e visão do todo
45	108	51	Falsa segurança e suficientemente bom
46	112	52	Perfil Essencial
47	114	53	Perfil Estruturado
48	116	54	Perfil Crítico e crise
44	118	55	Checklists essenciais
45	119	56	Modelos de governança
46	120	57	Prompts para a Vida Relacional
47	123	58	A engenharia como memória
48	123	59	O código como consequência e prova
49	123	60	Autonomia do desenvolvedor júnior
50	123	61	Pessoas IA e o futuro do método

Verificações executadas e pendências

O pacote prático foi executado em Node v24.22.1 e npm 11.9.0, sem dependências externas. O registro mais recente informa 18 testes automatizados aprovados. Foram exercitados instalação limpa, validação de dados e sintaxe, catálogo HTTP, idempotência, conflito e recuperação em sandbox. O exercício de implementação incorreta falha de propósito; a variante corrigida passa. Os logs preservam comandos, resultados e limitações.

O navegador não iniciou no ambiente de execução usado para esta revisão. Assim, não houve verificação interativa de teclado, leitor de tela, tela móvel ou zoom. A presença de HTML semântico e de testes de API não encerra esses critérios. O roteiro manual está no pacote e a pendência permanece visível no gate de publicação do aplicativo.

Não foram realizados revisão técnica humana independente nem teste de aprendizagem com iniciantes. Os capítulos oferecem critérios e soluções para estudo, mas a efetividade didática precisa ser medida em aplicação com leitores. A análise de um projeto local não autoriza seu uso como sistema crítico, nem estende a recuperação da sandbox a dados de produção.

O arquivo controle_revisao/matriz_correcao.csv relaciona cada apontamento da análise anterior ao tratamento, à evidência e ao estado final. Observações sobrepostas permanecem identificadas; o total de registros não representa defeitos independentes. O novo sumário, a sequência de capítulos, o idioma e a renderização são conferidos como parte da produção desta edição.

Protocolo de avaliação com leitores

Para validar a progressão, convide leitores com experiência declarada, preserve suas tentativas e proponha uma tarefa diferente do exemplo. Primeiro observe se conseguem localizar e executar o projeto; depois peça uma mudança de regra, um diagnóstico de falha e a transferência para outra pessoa. Registre tempo, pedidos de ajuda, erros de interpretação e critérios atendidos. O objetivo é descobrir onde o texto precisa melhorar, não induzir aprovação.

Após a aplicação, revise os capítulos em que leitores diferentes falham pelo mesmo motivo. Uma nova rodada deve verificar se a mudança reduziu o obstáculo. Até que esse processo seja executado, descreva a obra como edição revista com validação técnica documentada e avaliação didática externa pendente.

Competência	Tarefa observável	Evidência a conservar
Execução	Iniciar o projeto em ambiente preparado pelo próprio leitor.	Comandos ambiente resultado e obstáculos.
Compreensão	Explicar ativo booleano versus texto e vazio versus inválido.	Resposta própria e exemplo executado.
Mudança	Incluir uma capacidade nova preservando os inativos ocultos.	Diff testes novos e regressão.
Diagnóstico	Localizar a causa de uma falha fornecida sem apagar proteção.	Hipótese reprodução correção e reteste.
Continuidade	Outra pessoa executa e explica o estado do projeto.	Registro da transferência e lacunas encontradas.