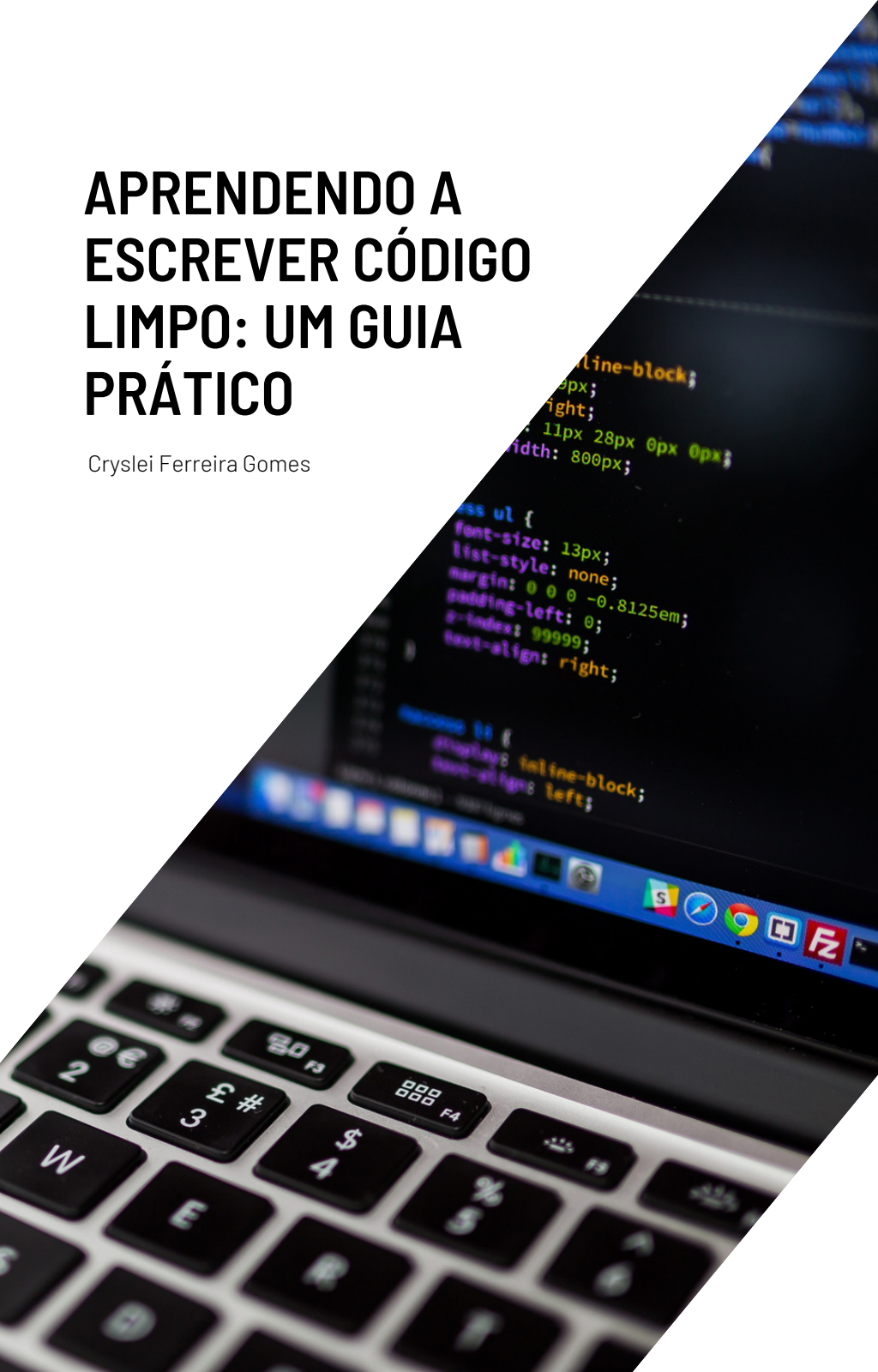
APRENDENDO A ESCREVER CÓDIGO LIMPO: UM GUIA PRÁTICO

Cryslei Ferreira Gomes



APRESENTAÇÃO

Código limpo é um estilo de codificação que se concentra em tornar o código mais fácil de ler, compreender e manter. Ele é baseado em boas práticas de programação e é projetado para garantir que o código seja confiável, escalável e mantível.

A importância do código limpo não pode ser subestimada, já que ele tem um impacto direto na qualidade e no sucesso do produto final. Código mal escrito e desorganizado pode ser difícil de manter e difícil de atualizar, o que pode levar a bugs e problemas de performance. Além disso, código mal escrito pode ser difícil de compreender para outros desenvolvedores, o que pode tornar o trabalho em equipe mais difícil e ineficiente.

Para criar código limpo, existem várias práticas recomendadas que você pode seguir. Aqui estão algumas das práticas mais comuns.

ÍNDICE

01

MANTENHA AS FUNÇÕES CURTAS E SIMPLES

AS FUNÇÕES DEVEM TER APENAS UMA ÚNICA RESPONSABILIDADE E DEVEM SER CURTAS O SUFICIENTE PARA SEREM LIDAS E COMPREENDIDAS FACILMENTE.
AQUI ESTÃO ALGUNS EXEMPLOS DA PRÁTICA DE MANTER FUNÇÕES CURTAS E SIMPLES:

EXEMPLO 1:

```
// Função não limpa
function calculateAndPrintResult(numbers) {
  const result = numbers.reduce((a, b) => a + b);
  console.log(`Resultado: ${result}`);
}

// Função limpa
function sum(numbers) {
  return numbers.reduce((a, b) => a + b);
}

function printResult(result) {
  console.log(`Resultado: ${result}`);
}

const result = sum([1, 2, 3, 4]);
printResult(result);
```

EXEMPLO 2:

```
// Função não limpa
function processData(data) {
  const processedData = data.map(d => d * 2);
  const filteredData = processedData.filter(d => d > 10);
  console.log(filteredData);
}

// Função limpa
function double(data) {
  return data.map(d => d * 2);
}

function greaterThan10(data) {
  return data.filter(d => d > 10);
}

const processedData = double([1, 2, 3, 4]);
const filteredData = greaterThan10(processedData);
console.log(filteredData);
```

Nos exemplos acima, a função limpa tem apenas uma responsabilidade, o que a torna mais fácil de ler, compreender e manter. Além disso, as funções limpas podem ser reutilizadas em outros lugares do código, tornando o código mais escalável e flexível.

02

USE NOMES SIGNIFICATIVOS PARA VARIÁVEIS E MÉTODOS

OS NOMES DE VARIÁVEIS E MÉTODOS DEVEM SER CLAROS E DESCRITIVOS, DE MODO QUE SEJAM FÁCEIS DE ENTENDER. AQUI ESTÃO ALGUNS EXEMPLOS DA PRÁTICA DE NOMEAR VARIÁVEIS E FUNÇÕES DE FORMA DESCRITIVA E CLARA:

EXEMPLO 1:

```
// Nomeação ruim
const d = new Date();

// Nomeação boa
const currentDate = new Date();
```

EXEMPLO 2:

```
// Nomeação ruim
function p(a, b) {
  return a * b;
}

// Nomeação boa
function multiply(a, b) {
  return a * b;
}
```

Nos exemplos acima, a nomeação clara e descritiva ajuda a tornar o código mais fácil de ler, compreender e manter. Além disso, nomes claros podem evitar erros comuns, como confundir variáveis ou usar a mesma variável para coisas diferentes.

03

DOCUMENTE O CÓDIGO DE MANEIRA CLARA E CONCISA

A DOCUMENTAÇÃO É IMPORTANTE PARA AJUDAR A COMPREENDER O CÓDIGO E AJUDAR OUTROS DESENVOLVEDORES A TRABALHAR COM ELE. AQUI ESTÃO ALGUNS EXEMPLOS DA PRÁTICA DE MANTER AS FUNÇÕES E VARIÁVEIS EM ESCOPOS APROPRIADOS:

EXEMPLO 1:

```
// Escopo global
let name = "John Doe";

function printName() {
  console.log(name);
}

// Escopo local
function getFullName(firstName, lastName) {
  const name = `${firstName} ${lastName}`;
  console.log(name);
}
```

EXEMPLO 2:

```
// Escopo global
let sum = 0;

function addToSum(value) {
  sum += value;
}

// Escopo local
function calculateSum(numbers) {
  let sum = 0;
  for (const number of numbers) {
    sum += number;
  }
  return sum;
}
```

Nos exemplos acima, manter as variáveis e funções em escopos apropriados ajuda a evitar conflitos e erros comuns, além de tornar o código mais fácil de ler e compreender. Além disso, usar escopos locais também pode aumentar a segurança do código, já que as variáveis e funções ficam restritas a um contexto específico.

04

SIGA A PRÁTICA DE ESCREVER COMENTÁRIOS CLAROS E ÚTEIS NO CÓDIGO

COMENTÁRIOS SÃO UMA PARTE IMPORTANTE DO CÓDIGO, POIS AJUDAM A EXPLICAR O QUE ESTÁ ACONTECENDO E COMO AS COISAS ESTÃO SENDO FEITAS. ALÉM DISSO, ELES TAMBÉM AJUDAM A DOCUMENTAR O CÓDIGO, O QUE É IMPORTANTE EM PROJETOS DE EQUIPE OU EM PROJETOS DE LONGO PRAZO.

AQUI ESTÃO ALGUMAS DICAS PARA ESCREVER COMENTÁRIOS EFICAZES:

- **MANTENHA OS COMENTÁRIOS CURTOS E DIRETOS AO PONTO. NÃO É NECESSÁRIO EXPLICAR TODO O CÓDIGO, APENAS O QUE É IMPORTANTE OU NÃO É ÓBVIO.**
- **USE COMENTÁRIOS PARA EXPLICAR AS DECISÕES DE PROJETO IMPORTANTES. POR EXEMPLO, SE VOCÊ ESTIVER USANDO UMA TÉCNICA ESPECÍFICA PARA SOLUCIONAR UM PROBLEMA, EXPLIQUE POR QUE VOCÊ A ESCOLHEU.**
- **ATUALIZE REGULARMENTE OS COMENTÁRIOS À MEDIDA QUE O CÓDIGO É ALTERADO.**
- **MANTENHA OS COMENTÁRIOS ACESSÍVEIS A TODOS OS MEMBROS DA EQUIPE, INDEPENDENTEMENTE DE SUA EXPERIÊNCIA.**
- **EVITE COMENTÁRIOS REDUNDANTES OU INÚTEIS. SE O CÓDIGO É CLARO E FÁCIL DE ENTENDER, É PROVÁVEL QUE OS COMENTÁRIOS NÃO SEJAM NECESSÁRIOS.**

Em resumo, escrever comentários claros e úteis no código é uma prática importante para tornar o código mais legível, compreendível e manutenível. Além disso, eles também ajudam a documentar o código, o que é importante para equipes de desenvolvimento e projetos de longo prazo.

05

REFATORE REGULARMENTE O CÓDIGO

REFATORAR O CÓDIGO É UMA TÉCNICA IMPORTANTE PARA MANTER O CÓDIGO LIMPO E ORGANIZADO. REFATORE REGULARMENTE PARA REMOVER REDUNDÂNCIAS E CORRIGIR PROBLEMAS DE ESTRUTURA. AQUI ESTÃO ALGUNS EXEMPLOS DA PRÁTICA DE USAR NOMES DESCRITIVOS PARA VARIÁVEIS, FUNÇÕES E OUTROS COMPONENTES DO CÓDIGO:

EXEMPLO 1:

```
// Nomes descritivos
const customerName = "John Doe";
const customerAddress = "123 Main St";
const customerEmail = "john.doe@example.com";

// Nomes menos descritivos
const cn = "John Doe";
const ca = "123 Main St";
const ce = "john.doe@example.com";
```

EXEMPLO 2:

```
// Nomes descritivos
function calculateArea(width, height) {
  return width * height;
}

// Nomes menos descritivos
function calc(w, h) {
  return w * h;
}
```

Nos exemplos acima, usar nomes descritivos ajuda a tornar o código mais legível e compreendível, especialmente quando outras pessoas precisam ler ou manter o código. Além disso, nomes descritivos também ajudam a evitar erros comuns, já que são mais fáceis de identificar e entender. Em geral, use nomes descritivos sempre que possível, pois eles ajudam a tornar o código mais fácil de ler, compreender e manter.

06

TESTE O CÓDIGO

TESTAR O CÓDIGO É IMPORTANTE PARA GARANTIR QUE ELE FUNCIONE CORRETAMENTE E PARA IDENTIFICAR PROBLEMAS ANTES QUE ELES CAUSEM PROBLEMAS. REALIZAR TESTES AUTOMATIZADOS, É UMA ÓTIMA MANEIRA DE PRATICAR O CÓDIGO LIMPO POR VÁRIOS MOTIVOS:

- **CONFIANÇA NO CÓDIGO: TESTES AUTOMATIZADOS AJUDAM A GARANTIR QUE O CÓDIGO ESTEJA FUNCIONANDO CORRETAMENTE, O QUE AUMENTA A CONFIANÇA NA QUALIDADE DO CÓDIGO.**

- **FEEDBACK RÁPIDO: TESTES AUTOMATIZADOS FORNECEM FEEDBACK IMEDIATO SOBRE A FUNCIONALIDADE DO CÓDIGO, O QUE PERMITE QUE OS ERROS SEJAM CORRIGIDOS RAPIDAMENTE.**

- **MAIOR FACILIDADE NA MANUTENÇÃO: TESTES AUTOMATIZADOS AJUDAM A MANTER O CÓDIGO LIMPO, JÁ QUE QUALQUER MUDANÇA FUTURA NO CÓDIGO PODE SER TESTADA RAPIDAMENTE.**

- **MAIOR CAPACIDADE DE EVOLUÇÃO: COM TESTES AUTOMATIZADOS EM VIGOR, É MAIS FÁCIL ADICIONAR NOVOS RECURSOS E FUNCIONALIDADES AO CÓDIGO, JÁ QUE AS MUDANÇAS PODEM SER TESTADAS RAPIDAMENTE.**

- **MAIOR FACILIDADE NA INTEGRAÇÃO COM OUTROS SISTEMAS: TESTES AUTOMATIZADOS TAMBÉM AJUDAM A GARANTIR QUE O CÓDIGO POSSA SER INTEGRADO COM OUTROS SISTEMAS COM MAIS FACILIDADE, JÁ QUE ERROS E PROBLEMAS PODEM SER IDENTIFICADOS RAPIDAMENTE.**

Em resumo, realizar testes automatizados é uma ótima maneira de praticar código limpo, pois ajuda a garantir que o código esteja funcionando corretamente, forneça feedback rápido sobre erros, seja fácil de manter, evoluir e integrar com outros sistemas.

07

TRABALHE EM EQUIPE

TRABALHAR EM EQUIPE É UMA ÓTIMA MANEIRA DE PRATICAR O CÓDIGO LIMPO POR VÁRIOS MOTIVOS:

1. **MAIOR PERSPECTIVA: TRABALHAR EM EQUIPE PERMITE QUE DIFERENTES PESSOAS, COM DIFERENTES HABILIDADES E PERSPECTIVAS, TRABALHEM JUNTAS NO MESMO PROJETO. ISSO PODE LEVAR A SOLUÇÕES MAIS CRIATIVAS E EFICIENTES AOS PROBLEMAS DE CODIFICAÇÃO.**
2. **FEEDBACK MÚTUO: AO TRABALHAR EM EQUIPE, É POSSÍVEL OBTER FEEDBACK CONSTANTE DOS COLEGAS SOBRE O CÓDIGO, O QUE AJUDA A IDENTIFICAR PROBLEMAS E MELHORAR A QUALIDADE DO CÓDIGO.**
3. **DIVISÃO DE TAREFAS: TRABALHAR EM EQUIPE PERMITE QUE AS TAREFAS SEJAM DIVIDIDAS ENTRE OS MEMBROS, O QUE PODE AJUDAR A GARANTIR QUE O PROJETO SEJA CONCLUÍDO MAIS RAPIDAMENTE E COM MAIS EFICIÊNCIA.**
4. **APRENDIZADO MÚTUO: AO TRABALHAR EM EQUIPE, É POSSÍVEL APRENDER COM OS COLEGAS SOBRE NOVAS TÉCNICAS E MELHORES PRÁTICAS DE CODIFICAÇÃO.**
5. **MAIOR COLABORAÇÃO: TRABALHAR EM EQUIPE INCENTIVA A COLABORAÇÃO E O TRABALHO EM CONJUNTO.**

REFERÊNCIAS

MARTIN, R. C. (2008). CLEAN CODE: A HANDBOOK OF AGILE SOFTWARE CRAFTSMANSHIP. UPPER SADDLE RIVER, NJ: PRENTICE HALL.

FOWLER, M. (1999). REFACTORING: IMPROVING THE DESIGN OF EXISTING CODE. ADDISON-WESLEY PROFESSIONAL.

HUNT, A., & THOMAS, D. (1999). THE PRAGMATIC PROGRAMMER. ADDISON-WESLEY PROFESSIONAL.

GAMMA, E., HELM, R., JOHNSON, R., & VLISSIDES, J. (1994). DESIGN PATTERNS: ELEMENTS OF REUSABLE OBJECT-ORIENTED SOFTWARE. ADDISON-WESLEY PROFESSIONAL.

MARTIN, R. C. (2002). AGILE SOFTWARE DEVELOPMENT, PRINCIPLES, PATTERNS, AND PRACTICES. UPPER SADDLE RIVER, NJ: PRENTICE HALL.

FREEMAN, S., & PRYCE, N. (2009). GROWING OBJECT-ORIENTED SOFTWARE, GUIDED BY TESTS. ADDISON-WESLEY PROFESSIONAL.

MARTIN, R. C. (2017). CLEAN ARCHITECTURE: A CRAFTSMAN'S GUIDE TO SOFTWARE STRUCTURE AND DESIGN. PEARSON EDUCATION.