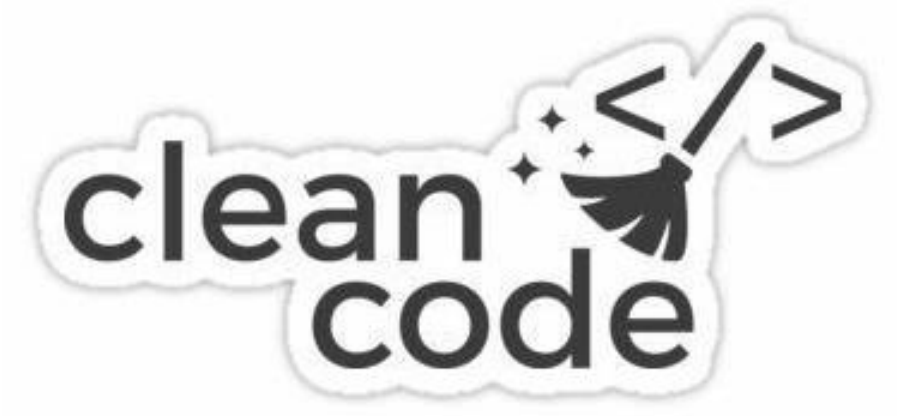
COTI INFORMÁTICA
ESCOLA DE NERDS

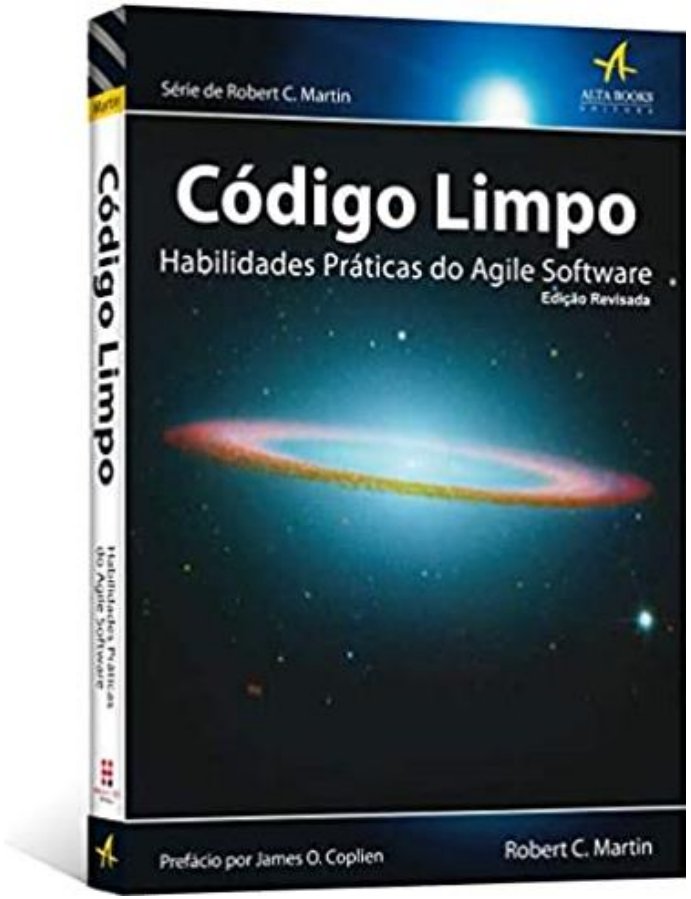
Quais são os princípios do **código limpo**?



As técnicas do **Clean Code** apareceram pela primeira vez no livro “Clean Code: A Handbook of Agile Software Craftsmanship“, lançado em 2008.

Ele foi escrito por Robert Cecil Martin, conhecido na comunidade como Uncle Bob. O autor atua na área de desenvolvimento desde 1970 e é um dos profissionais por trás do Manifesto Ágil, lançado em 2001.

Um dos principais erros que os programadores cometem é acreditar que, uma vez que o código está pronto e funcionando, não precisa mais de revisão.
Porém, **um sistema nunca está totalmente finalizado**, sempre existe a necessidade de atualizações e novas funcionalidades. Além disso, o código envelhece e pode se tornar obsoleto. É nesse cenário que o Clean Code se encaixa.



1 – Nomes são muito importantes

A definição de nome é essencial para o bom entendimento de um código.
Aqui, não importa o tipo de nome, seja ele: Variável; Função; Parâmetro; Classe ou Método.

Ao definir um nome, é preciso ter em mente **2 pontos principais**:

** Ele deve ser preciso e passar logo de cara sua ideia central. Ou seja, deve ir direto ao ponto;
** Não se deve ter medo de nomes grandes. Se a sua função ou parâmetro precisa de um nome extenso para demonstrar o que realmente representa, é o que deve ser feito.

2 – Regra do escoteiro

Há um princípio do escotismo que diz que, uma vez que você sai da área em que está acampando, você deve deixá-la mais limpa do que quando a encontrou.

Trazendo a regra para o mundo da programação, a regra significa **deixar o código mais limpo do que estava antes de mexer nele**.

3 – Seja o verdadeiro autor do código

O ser humano é acostumado a pensar de forma narrativa , portanto, o código funciona da mesma forma. Logo, ele é uma história e, como os programadores são seus autores, **precisam se preocupar na maneira com que ela será contada**.

Em resumo, para estruturar um código limpo, é necessário criar funções simples, claras e pequenas. Existem 2 regras para criar a narrativa via código:

** As funções precisam ser pequenas;
** Elas têm de ser ainda menores.

Não confunda com os termos “nome” e “função”. Como dissemos no primeiro princípio, nomes grandes não são um problema. Já as funções precisam ser as menores possíveis.

4 – DRY (Don’t Repeat Yourself)

Esse princípio pode ser traduzido como “não repita a si mesmo”. Essa expressão foi descrita pela primeira vez em um livro chamado *The Pragmatic Programmer* e se aplica a diversas áreas de desenvolvimento, como:

Banco de Dados;
Testes;
Documentação;
Codificação.

O DRY diz que cada pedaço do conhecimento de um sistema deve ter uma representação única e **ser totalmente livre de ambiguidades**. Em outras palavras, define que não pode existir duas partes do programa que desempenhem a mesma função.

5 – Comente apenas o necessário

Esse princípio afirma que comentários podem ser feitos, porém, se forem realmente necessários. Segundo Uncle Bob, os comentários mentem. E isso tem uma explicação lógica.

O que ocorre é que, **enquanto os códigos são constantemente modificados, os comentários não**. Eles são esquecidos e, portanto, deixam de retratar a funcionalidade real dos códigos.

Logo, se for para comentar, que seja somente o necessário e que seja revisado juntamente com o código que o acompanha.

6 – Tratamento de erros

Tem uma frase do autor Michael Feathers, muito conhecido na área de desenvolvimento, que diz que as coisas podem dar errado, mas, quando isso ocorre, os programadores são os responsáveis por **garantir que o código continuará fazendo o que precisa**.

Ou seja: saber tratar as exceções de forma correta é um grande e importante passo para um programador em desenvolvimento.

7 – Testes limpos

Testar, na área de programação, é uma etapa muito importante. Afinal, **um código só é considerado limpo após ser validado através de testes –** que também devem ser limpos.

Por isso, ele deve seguir algumas regras, como:

Fast: O teste deve ser rápido, permitindo que seja realizado várias vezes e a todo momento;
Independent: Ele deve ser independente, a fim de evitar que cause efeito cascata quando da ocorrência de uma falha – o que dificulta a análise dos problemas;
Repeatable: Deve permitir a repetição do teste diversas vezes e em ambientes diferentes;
Self-Validation: Os testes bem escritos retornam com as respostas true ou false, justamente para que a falha não seja subjetiva;
Timely: Os testes devem seguir à risca o critério de pontualidade. Além disso, o ideal é que sejam escritos antes do próprio código, pois evita que ele fique complexo demais para ser testado.