

# Notas sobre o capítulo 1 de Clean Code – by Lucas Trevizan

## Prefácio

O prefácio aborda sobre como pequenos detalhes impactam em um todo no desenvolvimento de software, e que a abordagem correta com esses “detalhes” é enfrenta-los de frente e diariamente para que ganhemos a confiança para enfrentar problemas maiores..

Além disso ele destaca o fato de que a maior parte do trabalho de um desenvolvedor é realizar manutenção de software mais do que fabricar um de fato, e isso é verdade. Por experiência própria eu digo que dificilmente criamos algo novo, geralmente estamos sempre dando manutenção e refatoração em algo que já existe.

Ele também introduz o conceito do 5S (uma filosofia japonesa focada na manutenção) que segue 5 princípios: **organização, arrumação, limpeza, padronização, disciplina**. Tornar seu código legível é tão importante quanto torna-lo executável. Um fato interessante é citado referente a endentação, a simples forma consistente dela era um indicador de baixo nível de bugs em um código.

Criar códigos limpos é uma tarefa árdua e conhecer conceitos teóricos não te privam da prática constante.

## O código

Códigos nunca deixarão de existir, por mais que haja um avanço constante na inteligência artificial. Códigos são especificações detalhadas, por mais alto nível que a linguagem seja ainda sim precisará ser escrita a nível detalhado para que o computador possa executa-la, nada é criado como mágica

## Código ruim

Um código ruim pode falir a empresa, se apressado o lançamento de um app o código pode virar uma bagunça e causar efeito bola de neve. Isso gera uma reflexão, por que escrever código ruim? muitos podem ser os motivos, pressa, melhor 1 na mão que 2 voando, etc....

## O custo de ter um código confuso

Um código ruim pode começar um projeto rapidamente, mas progredi-lo a passos de tartaruga e inevitavelmente acaba se tornando até mesmo "improgredível".

Devido a confusão do código ele torna-se impossível de se analisar, refatorar e adicionar novas funções sem que quebrem outras, trazer mais devs não resolve esse problema, então a produtividade chega a ser 0.

## O grande replanejamento

Depois que o código fonte fica um fiasco, os devs exigem um novo projeto replanejado, eu ainda não passei por essa fase, mas o que não me faltou foi a vontade de pedir pra fazer tudo do zero.

Então uma equipe é selecionada para o novo projeto, outra fica responsável pelo legado e começa uma corrida para que o projeto novo alcance o legado para poder substituí-lo. O problema é que depois de alguns anos o sistema novo se torna o legado, e o ciclo se inicia novamente, onde a maioria da equipe original do "antigo novo sistema" não está mais nele, e os novos devs que o assumiram já o acham uma bagunça. Então código limpo não se torna essencial apenas para reduzir custos, mas também para sua sobrevivência profissional.

## **Atitude**

O responsável por um código bom se tornar ruim, não é ninguém além de você mesmo. As vezes pegamos o caminho mais fácil e culpamos alguém, mas a grande e dura verdade é que você é o profissional que escreve o código e, portanto, a responsabilidade é sua. Seja honesto com sua opinião e visão frente as exigências do gerente, honestidade é tudo.

Nesse ponto eu coloco de novo minha experiência profissional como prova do argumento, por mais que a gente pegue um código péssimo, ainda sim existe a possibilidade de melhora-lo mesmo que um pouco e algumas vezes não o fazemos simplesmente para não ter o desgaste.

A única maneira de progredir mais rápido é sempre manter o código limpo.

## **A arte do código limpo?**

Podemos comparar o código limpo a um quadro, você consegue identificar um quadro bom e um ruim, mas você não consegue reproduzir um quadro bom, a menos que se esforce muito. É preciso desenvolver a "sensibilidade ao código" para saber a estratégia e ter a disciplina necessária para se escrever um código limpo.

## **O que é um código limpo?**

**Bjarne** o criador do C++ acredita que um código limpo deve ser "elegante", uma palavra que em suma representa naturalidade e estilo, portanto um código naturalmente entendível e belo de se ver. Um código ruim incita que ele se torne pior à medida que os outros trabalhem nele. Bjorn diz que o tratamento de erro deve ser completo, para se ter a maior atenção aos detalhes possíveis, Bjorn conclui que um código limpo faz bem apenas uma coisa, sem ambiguidade. Cada classe, função e módulo é responsável por uma única tarefa.

**Grady Booch** diz que um código limpo é simples e direto e como ler um livro bem escrito. Grady diz que um código limpo está repleto de "abstrações claras", um paradoxo lindo por assim dizer. Abstraímos o mundo real para dentro do nosso código (POO), ainda sim todas as suas funcionalidades devem ser claras e diretas, sem especulações, ela tem que conter apenas o necessário

**Dave Thomas** segue uma linha parecida com o Grady, ele também acredita que um código deve ser lido como um bom livro. Porém ele ressalta a importância de se possuir testes no código, segundo Dave um código só é limpo de verdade se possuir testes. E o código deve ser inteligível a seres humanos.

**Michael Feathers** resume o código limpo à frase "é um código feito por alguém que se importa".

**Ward Cunningham** ressalta a simplicidade do código e a naturalidade ao interpreta-lo enquanto o lê, assim como um livro, além de jogar a responsabilidade no programador fazendo lembrar que linguagens são ferramentas apenas e quem está por trás da beleza do que é escrito é o programador.

## **Somos autores**

Somos encorajados a pensar como escritores, quando escrevemos um código devemos pensar que haverá leitores para ele (de fato haverá), gastamos mais tempo lendo código do que escrevendo, o que nos leva de novo a mesma máxima, se você quer que seu código seja de fácil escrita então crie um código de fácil leitura.

## **A regra de escoteiro**

Deixe o código mais limpo do que quando você o encontrou, não precisam ser mudanças enormes, qualquer pequena melhoria já basta visto que constantemente você trabalhará no código e a cada pequena melhoria feita ele se torna melhor e consequentemente "antidegradável".

## **Conclusão**

O livro mostrará técnicas para se tornar um bom programador, mas apenas lê-lo não o tornará automaticamente. Ou seja, conceito sem prática é irrelevante.