Padrões e Desenho de Software

Professores:
José Luis Oliveira
Sérgio Matos

# Clean Code

Hugo Paiva, 93195


DETI
Universidade de Aveiro
05-06-2020

# Conteúdo

# 1 Clean Code

## 1.1 Introdução

*Clean Code* é o conceito de um código fácil de entender e susceptível a mudanças que ganhou relevância em 2008, quando *Robert Cecil Martin* o mostrou ao mundo através do seu livro *Clean Code: A Handbook of Agile Software Craftsmanship*. Nesse livro, *Robert* apresenta ao detalhe as técnicas e princípio ideais para o desenvolvimento de software de fácil compreensão.

Um sistema é algo que nunca está terminado, que necessita sempre de ser atualizado, quer seja devido à implementação de novas funcionalidade, resolução de problemas ou, até, devido a se ter tornado obsoleto. Ao longo destes ciclos de desenvolvimento, de modo a reduzir os custos de manutenção, é imperial a utilização de código limpo. Uma fraca qualidade de código leva a uma grande carga cognitiva, sendo necessárias mais horas de trabalho para a resolução de problemas. O problema é tão relevante que o livro refere um rácio de 10 leituras de código até começar a escrita.

Um código limpo é, portanto, algo que leva tempo, atenção e dedicação, a mesma razão pela qual muitas das vezes não é utilizado na indústria.

## 1.2 Princípios e Boas Práticas

Estando profundamente associado à complexidade de um sistema, ou seja, quanto mais limpo está o código, menos complexidade o sistema tem, é natural que o estudo dos sintomas desta complexidade seja um fator relevante para a escrita de um código limpo.

Com isto em mente, *Robert* definiu alguns princípios e boas práticas para chegar ao conceito de *Clean Code*. De entre os referidos no livro, destacaram-se os seguintes:

- Nomes Significativos
- Organização de Funções
- Organização das Classes e Estruturas de Dados
- Comentários Expressivos
- Formatação do Código
- Tratamento de Erros
- Testes Limpos

### 1.2.1 Nomes Significativos

Tal como o próprio princípio diz, os nomes devem ser significativos e de grande importância para manter um código compreensível. Independentemente do tipo de nome (funções, variáveis, métodos, etc.), segundo este princípio, os nomes devem seguir dois pontos principais:

- Ir diretos ao ponto, passando a sua ideia central
- Em caso de necessidade, utilizar nomes grandes sem preocupações, garantindo a sua compreensão

Dito isto, são de evitar exemplos deste tipo, onde é pouco perceptível qual o contexto do problema:

```
int[] f; // frutas

for (int l=0; l<50; l++) {
    if (f[l] == 1) {
        f[l] = 2;
    }
}
```

Ao invés, deve-se utilizar abordagens do género:

```
int fruitsAtHome;

final int NUMBER_OF_FRUITS = 50;
for (int l=0; l<NUMBER_OF_FRUITS; l++) {
    if (fruitsAtHome[l] == ROTTEN) {
        fruitsAtHome[l] = TRASH;
    }
}
```

Existem, no entanto, outros aspetos que devem ser tido em consideração. Evitar símbolos e emojis, utilizar nomes pronunciáveis, utilizar verbos em métodos e nomes em classes e utilizar sempre as mesmas palavras para um determinado contexto (escolher get ao invés de fetch) são alguns destes aspetos.

### 1.2.2 Organização de Funções

Segundo o autor deste conceito, existem duas regras para a criação das funções:

- As funções devem ser pequenas
- As funções devem ser ainda mais pequenas

O objetivo com este trocadilho de regras é manter as funções com o mínimo de funcionalidades possíveis, permitindo uma menor complexidade ao longo do programa e, utilizando os nomes significativos, o código deverá estar organizado de maneira a que qualquer pessoa consiga ver todos os percursos ao longo da execução do programa facilmente. Aliás, durante o livro, é referido que as funções apenas devem fazer uma coisa.

Deve-se, também, utilizar menos argumentos, evitando, novamente, o aumento de complexidade.

```
Circle makeCircle(Point center, double radius);
```

A declaração desta função é, sem dúvida, mais clara que a seguinte.

```
Circle makeCircle(double x, double y, double radius);
```

Basicamente, continua a tendência de manter a menor complexidade possível, evitando efeitos secundários para além do objetivo das funções, repetições de código, *output* de vários argumentos e utilizar tratamento de erros.

## 1.3 Organização das Classes e Estruturas de Dados

## 2 Bibliografia


[1] https://gist.github.com/wojteklu/73c6914cc446146b8b533c0988cf8d29

[2] https://garywoodfine.com/what-is-clean-code/

[3] https://www.infoq.com/br/articles/clean-code-book-review/

[4] http://ceur-ws.org/Vol-2066/isee2018paper06.pdf

[5] https://simpleprogrammer.com/clean-code-principles-better-programmer/

[6] https://x-team.com/blog/principles-clean-code/

[7] https://codingsans.com/blog/clean-code

[8] https://www.hostgator.com.br/blog/clean-code-o-que-e/

[9] https://www.butterfly.com.au/blog/website-development/clean-high-quality-code-a-guide-on-how-to-b