



# Clean Code

Habilidades práticas do software ágil

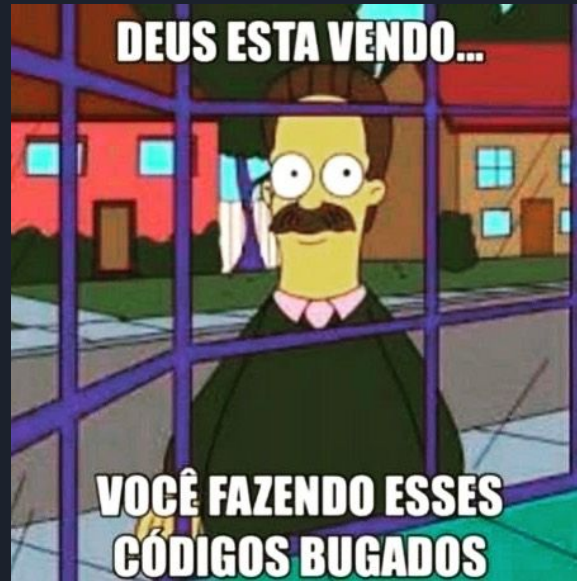
- Robert C. Martin



# O que é Clean Code?

- Uma filosofia e também técnicas de desenvolvimento que visam facilitar a escrita e leitura do código para obtermos melhores resultados.

Por que escrever códigos melhores?



"A proporção de leitura por escrita de um código é de 10:1"

- Robert C. Martin

# Dívida técnica





# 1. Nomes são muito importantes

O nome de uma variável ou método deve descrever exatamente o que a entidade representa.

- ❑ Não tenha medo de nomes grandes.
- ❑ O nome deve ser claro e objetivo.
- ❑ Evite misturar idiomas, exemplo: **getProdutos**.
- ❑ Métodos devem ser representados por verbos ou frases verbais: `enviarNotaFiscal`, `cadastrar`.
- ❑ Classes devem ser representadas por substantivos: `Cliente`, `Produto`, `NotaFiscal`.



# 1. Nomes objetivos

Teste longo que depende de várias funções

```
changeTipo() {  
    this.emissaoNfe.misc.natureza = null;  
    this.getNatureza();  
}
```

Nomes claros e objetivos

```
alterouTipoOperacaoNfe() {  
    this.emissaoNfe.misc.natureza = null;  
    this.obterNaturezasOperacaoNfe();  
}
```

# 1. Nomes

Nomes dizem exatamente o que o elemento representa

```
let atual = produtoEstoque?.atual || 0;  
let valor = 0;
```

```
switch (tipoOperacao) {  
  case '+':  
    valor = digitado;  
    break;  
  case '-':  
    valor = -digitado;  
    break;  
  default:  
    valor = digitado - atual;  
    break;  
}
```

```
let estoqueAtualDepositoProduto = estoqueDepositoProduto?.atual || 0;  
let valorMovimentadoNaOperacao = 0;
```

```
switch (tipoOperacaoRealizada) {  
  case '+':  
    valorMovimentadoNaOperacao = valorInventario;  
    break;  
  case '-':  
    valorMovimentadoNaOperacao = -valorInventario;  
    break;  
  default:  
    valorMovimentadoNaOperacao = valorInventario - estoqueAtualDepositoProduto;  
    break;  
}
```





## 2. Evite comentários

Os comentários mentem, enquanto os códigos são constantemente modificados, os comentários podem ser esquecidos e deixar de retratar o que o código faz.

- ❑ Se de fato existe a necessidade de comentários, provavelmente algo está errado com o código.
- ❑ Utilize a dica sobre nomes, e tente insira nos nomes das elementos o que os mesmos se propõe a fazer.



## 2. Comentários

Comentário não agrega nenhuma informação relevante

```
/**  
 * Método para selecionar produtos  
 * @param txtPesquisa  
 */  
async selecionarProdutos(txtPesquisa?: string): Promise<any[]> {  
  
    this.textoPesquisando = txtPesquisa;  
  
    // ...  
}
```



## 2. Evite comentários desnecessários

Comentário não agrega nenhuma informação relevante

```
/*                                -\_(ツ)_/^-                                */  
nfe.vendedor.valorComissao = Number(  
    (nfe.valorTotal * (nfe.vendedor.percentualComissao / 100)).toFixed(3).slice(0, -1)  
);  
/*                                -\_(ツ)_/^-                                */
```



### 3. Regra do escoteiro

Deixe sempre o acampamento mais limpo do que você encontrou.

- ❑ Faça melhorias pontuais.
- ❑ Evite grandes refatorações.



### 3. Regra do escoteiro

#### Números mágicos e comentário

```
var contingencia = true;

if (nfe.modelo == '55') {
    contingencia = filial.configuracaoObj().tpCon == 6 || filial.configuracaoObj().tpCon == 7; //Se 6 ou 7 contingencia
} else {
    contingencia = filial.configuracaoObj().tpConNfc == 9;
}
```

### 3. Regra do escoteiro

Usando constantes e comentário removido

```
Nfe.MODELO = { NFE: 55, NFCE: 65 };
```

```
Nfeconfiguracao.CONTINGENCIA_NFE = { NORMAL: 1, SVC_AN: 6, SVC_RS: 7 };
```

```
Nfeconfiguracao.CONTINGENCIA_NFCE = { NORMAL: 1, CONTINGENCIA: 9 };
```

```
var contingencia = true;
```

```
if (nfe.modelo == Nfe.MODELO.NFCE) {
```

```
    contingencia = (filial.configuracaoObj().tpCon == NfeConfiguracao.CONTINGENCIA_NFE.SVC_AN ||  
        filial.configuracaoObj().tpCon == NfeConfiguracao.CONTINGENCIA_NFE.SVC_RS);
```

```
} else {
```

```
    contingencia = filial.configuracaoObj().tpConNfc == NfeConfiguracao.CONTINGENCIA_NFCE;
```

```
}
```



## 4. Funções claras e objetivas

Funções devem por padrão fazer somente uma coisa.

- ❑ As funções precisam ser pequenas
- ❑ Elas têm de ser ainda menores
- ❑ O ideal é que não receba parâmetros, e quando receber tem que ser o mínimo possível.
- ❑ Evite que uma função altere valor de outras funções.
- ❑ Caso a função seja muito grande, desconfie.

## 4. Funções claras e objetivas

Extraindo responsabilidades para funções menores

```
// Evite
async function salvarPedidoVenda(pedido){
  // Cria receita financeira ...
  // Atualiza estoque ...
}
```

```
// Utilize
async function criarReceitaFinanceira(pagamento){
  // Cria receita financeira
}

async function atualizarEstoque(pedido){
  // Atualiza estoque
}

async function salvarPedidoVenda(pedido){
  await criarReceitaFinanceira(pedido.pagamento);
  await atualizarEstoque(pedido.itens);
}
```





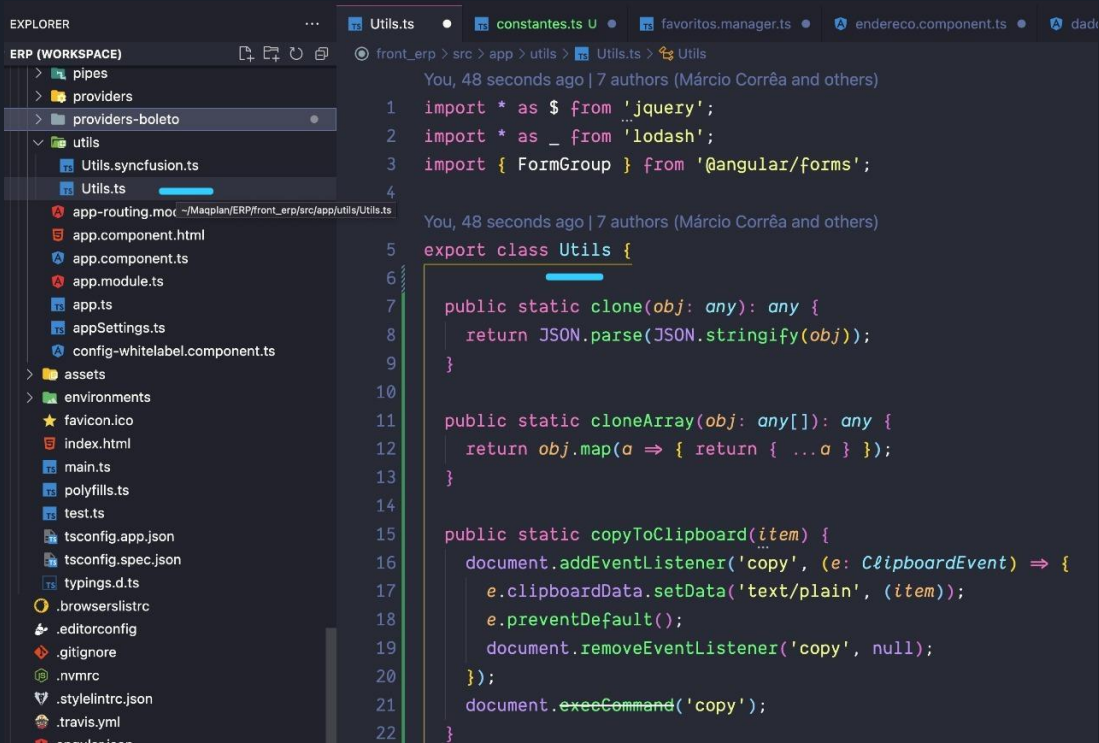
## 5. Não se repita ( Don't repeat yourself - DRY )

Cada parte do projeto deve ter uma representação única.

- ❑ Não pode existir duas partes do projeto que desempenhem a mesma função.
- ❑ Isole partes que podem ser reaproveitadas.
- ❑ Mesmo que a repetição possa parecer inofensiva, ela pode ser um problema à medida que o software cresce.

## 5. Não se repita

Isole partes que podem ser reaproveitadas



```
EXPLORER
ERP (WORKSPACE)
  > pipes
  > providers
  > providers-boleto
  > utils
    Utils.synctfusion.ts
    Utils.ts
  app-routing.module.ts
  app.component.html
  app.component.ts
  app.module.ts
  app.ts
  appSettings.ts
  config-whitelabel.component.ts
  assets
  environments
    favicon.ico
    index.html
    main.ts
    polyfills.ts
    test.ts
    tsconfig.app.json
    tsconfig.spec.json
    typings.d.ts
    .browserslistrc
    .editorconfig
    .gitignore
    .npmrc
    .stylelintrc.json
    .travis.yml
    README.md

front_erp > src > app > utils > Utils.ts
You, 48 seconds ago | 7 authors (Márcio Corrêa and others)
1 import * as $ from 'jquery';
2 import * as _ from 'lodash';
3 import { FormGroup } from '@angular/forms';
4
You, 48 seconds ago | 7 authors (Márcio Corrêa and others)
5 export class Utils {
6
7   public static clone(obj: any): any {
8     return JSON.parse(JSON.stringify(obj));
9   }
10
11   public static cloneArray(obj: any[]): any {
12     return obj.map(a => { return { ...a } });
13   }
14
15   public static copyToClipboard(item) {
16     document.addEventListener('copy', (e: ClipboardEvent) => {
17       e.clipboardData.setData('text/plain', (item));
18       e.preventDefault();
19       document.removeEventListener('copy', null);
20     });
21     document.execCommand('copy');
22   }
```



## 6. Trate seus erros

As coisas podem dar errado, e quando assim acontece, precisamos garantir que o código faça o que precise ser feito.

- ❑ Sempre que necessário use try-catch-finally.
- ❑ Usar exceções em vez de códigos de retorno.
- ❑ Cada exceção lançada deve fornecer o contexto suficiente para determinar a origem do erro.



## 6. Trate seus erros: Retornos genéricos

Não existe tratamento de erro, erros genéricos

```
FilialModel.findOne(filter, function (err, filial) {  
  if (err) return callback(err);  
  if (!filial) return callback(new Error("Filial não cadastrada!"));  
  callback(null, filial);  
});
```

## 6. Trate seus erros: Retornos específicos

### Classes construtoras de erros

```
exports.ErrorBuilder = class extends Error {
  constructor(message, status = 400, name = 'Error', err = null) {
    super(message)
    this.status = status
    this.name = name

    if (err) this.stack = err?.stack
  }
}

exports.NotFoundException = class extends Error {
  constructor(message) {
    super(message || '0 registro não foi encontrado.')
    this.status = 404
    this.name = 'Não encontrado'
  }
}
```



## 6. Trate seus erros: Resultado

Agora com o seu devido tratamento

```
try {
  FilialModel.findOne(filter, function (err, filial) {
    if (err) return callback(new ErrorBuilder(err));
    if (!filial) return callback(new NotFoundException("Filial não cadastrada!"));
    callback(null, filial);
  });
} catch (error) {
  throw new Error(error);
}
```



## 7. Siga as convenções

Mesmo que não concorde com os padrões do projeto, siga-os!

- ❑ Idioma.
- ❑ Organização das pastas.
- ❑ Constantes em maiúsculo.



## 7. Siga as convenções

Idiomas diferentes em um único arquivo

```
public static tratarStringParaRegex(str: string): string {  
    str = str.replace(/\\/ig, '\\\\').replace(/\'/ig, '\\\\');  
    return str;  
}  
  
public static onlyNumbers(str: string) {  
    if (str && str.length > 0) {  
        let numb = str.trim().match(/\d/g);  
        return numb.join('');  
    }  
    return str;  
}
```





## 8. Testes claros e limpos

Trate seus testes como parte fundamental do seu código, não secundária.

- ❑ Praticar o TDD.
- ❑ Teste deve ser rápido.
- ❑ Deve ser independente.
- ❑ Permitir a repetição em ambientes diferentes.
- ❑ Testes bem escritos retornam respostas como `true` e `false`.

## 8. Testes

Teste longo que depende de várias funções

```
describe('Despesa', () => {
  before(() => {
    cy.loginERP()
    cy.wait(4000)
    menu.acessarFinanceiro()
  })

  beforeEach(() => {
    menu.acessarFinanceiroDespesa()
  })

  it('Editar Despesa informações completas', () => {
    Despesa.preencherInformacoesCompletas();
    // ACESSAR JANELA DE EDIÇÃO
    // INFORMAR BENEFICIARIO
    // INFORMAR DATAS
    // INFORMAR CENTRO DE CUSTO
    // INFORMAR CONTA CONTABIL
    // INFORMAR NUMERO DO DOCUMENTO
    // SALVAR DESPESA
  });
});
```



## 8. Testes

### Testes rápidos e independentes

```
it('Deve acessar janela de cadastro', () => {  
  Despesa.acessarJanelaDeCadastro();  
});  
  
it('Deve preencher beneficiário', () => {  
  Despesa.preencherBeneficiario();  
});  
  
it('Deve informar datas despesa', () => {  
  Despesa.informarDatas();  
});  
  
it('Deve preencher informações do centro de custo', () => {  
  Despesa.informarCentroDeCusto();  
});  
  
it('Deve preencher conta bancária', () => {  
  Despesa.informarConta();  
});  
  
it('Deve preencher número do documento', () => {  
  Despesa.informarNumeroDocumento();  
});  
  
it('Deve salvar a despesa', () => {  
  Despesa.salvarDespesa();  
});
```

"Qualquer tolo escreve código que um computador entenda.  
Bons programadores escrevem código que humanos possam entender."

- Martin Fowler