Boas Práticas de Programação

Desenvolvendo código fonte com qualidade

Instrutor

Robson S. Rodrigues

- 20 anos de experiência com Engenheiro de Software
- Certificação Java, Web Component Developer, SQLServer
- Interesse em Engenharia e Arquitetura de Software com foco em boas práticas de programação
- Contato: robson.rodrigues@wmw.com.br

Agenda

- Código Fonte
- Clean Code
- Clean Tests
- Code Smells
- Mais Boas Práticas
- Exercícios
- Dúvidas, Críticas e Sugestões
- Referências

Código Fonte

Código Fonte

- Código fonte é um conjunto de instruções escritas por um desenvolvedor para que sejam executadas por um computador.
- A maioria do tempo gasto por um desenvolvedor não está na escrita do código fonte, mas sim na sua leitura, interpretação e no planejamento de como será realizada uma alteração ou implementação de nova funcionalidade.

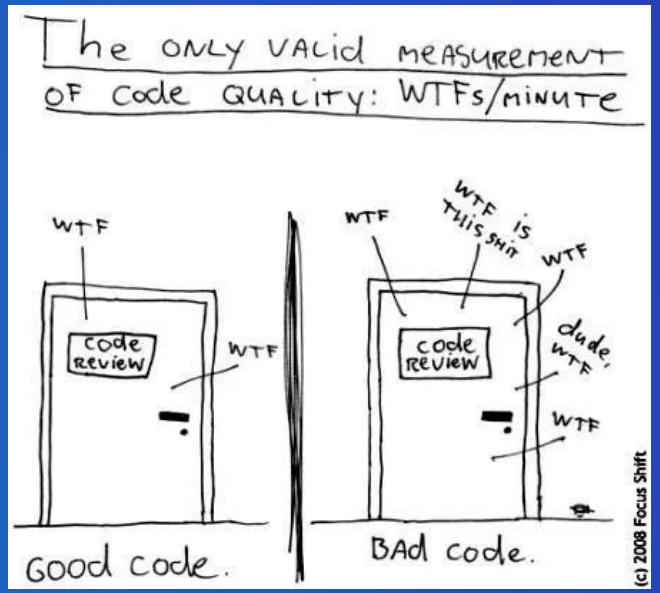
Código Fonte

- Desta forma, quanto mais claro e objetivo for o código fonte, mais produtivo será o trabalho do desenvolvedor.
- Este é um dos principais motivos pelo qual devemos escrever um código fonte de boa qualidade.

O que é Código Ruim

É um código:

- **Difícil de ler**
- **Difícil de entender**
- **Difícil de alterar**
- **Difícil de testar**



Motivos que levam a ter um Código Ruim

- Customização do produto para atender necessidades específicas do cliente sem ter uma arquitetura que suporte mudanças a longo prazo.
- Falta de capacitação da equipe.
- Priorização do prazo de entrega em detrimento da qualidade do produto.

Motivos que levam a ter um Código Ruim

- Desenvolver o produto sem foco na qualidade do código:
  - Não desenvolver o produto baseado em testes.
  - Não utilizar ferramentas de análise de qualidade de código.
  - Não ter processos de revisão e melhoria de código.

Consequências de ter um Código Ruim

- Demora mais tempo que o necessário para fazer uma alteração.
- É difícil escrever testes automatizados.
- É difícil de atualizar bibliotecas de terceiros.
- Falta de segurança se o produto vai realmente funcionar como deveria após uma alteração.

Consequências de ter um Código Ruim

- Não se sabe quais efeitos colaterais (side effects) podem acontecer depois de uma alteração.
- A equipe muitas vezes trabalha desmotivada, tem resistência ou medo de fazer melhorias no produto.
- Maior tempo para disponibilizar novas funcionalidades, diminuindo a competitividade do produto no mercado.

O que fazer para melhorar um Código Ruim

- Possuir uma arquitetura de software que esteja alinhada com modelo de negócio e que suporte mudanças a longo prazo (integração, customizações, campos e relatórios dinâmicos, hospedagem na nuvem, entre outros).
- Ter um tempo adequado para realizar as alterações.
- Investir na capacitação a equipe.

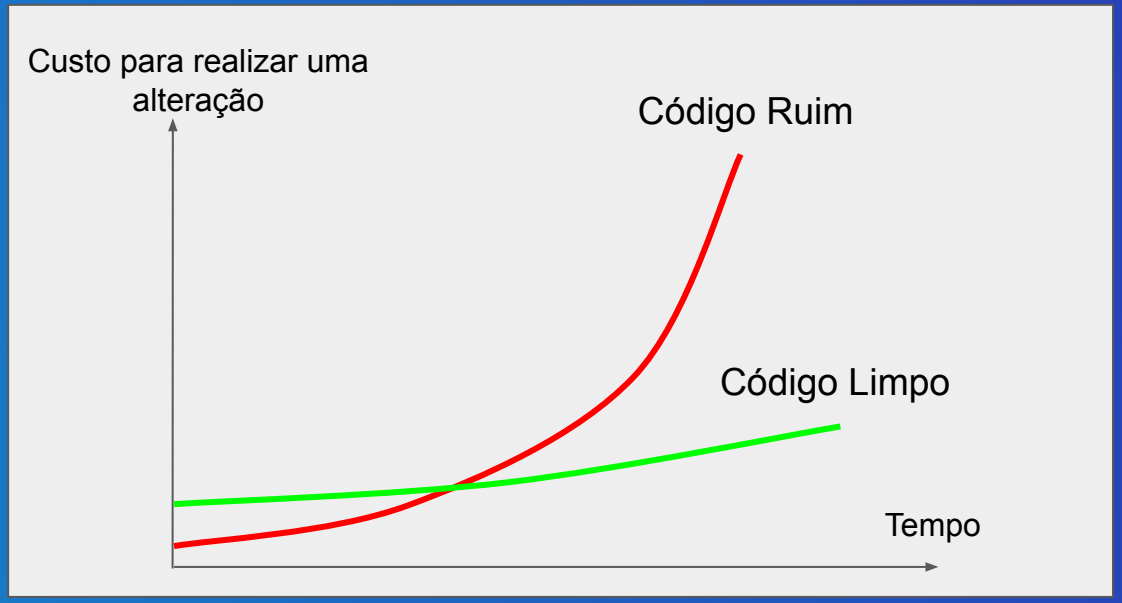
O que fazer para melhorar um Código Ruim

- Investir em qualidade de código, utilizando técnicas de codificação limpa (Clean Code).
- Investir em testes automatizados (TDD).
- Investir numa arquitetura de software limpa (Clean Architecture).
- Utilizar ferramentas de análise de qualidade de código (Sonar).

O que fazer para melhorar um Código Ruim

- Fazer programação em pares (Pair Programming)
- Fazer revisões de código (Code Review).
- Fazer melhorias no código fonte atual (Refactoring).
- Usar uma ferramenta para monitorar a cobertura de testes (Test Coverage).

Razões para investir em qualidade de código



Quanto pior a qualidade do código, maior o custo de manutenção ao longo do tempo

Produtividade ao Longo do Tempo

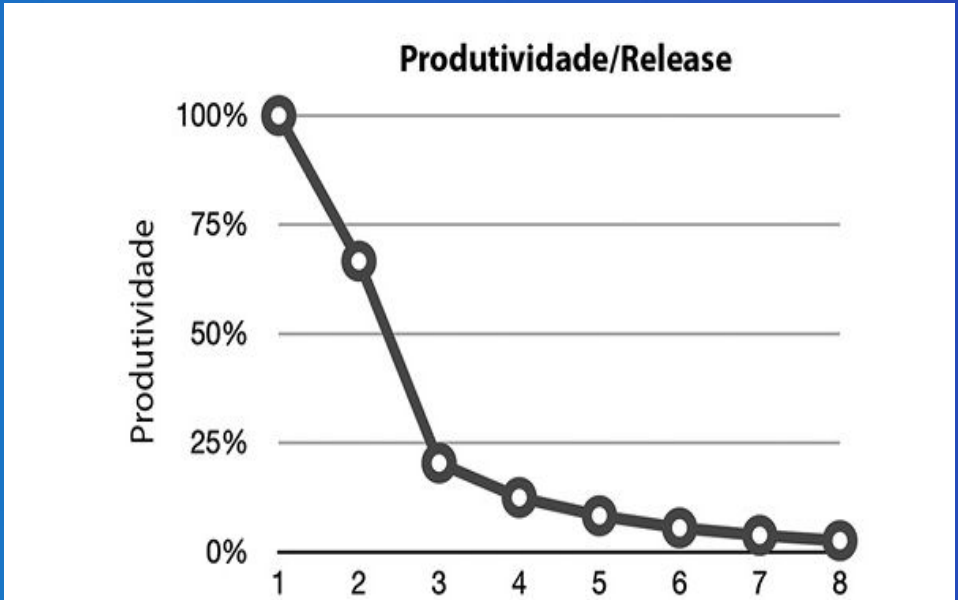


Figura 1.4 Produtividade por release

Fonte: Livro Clean Architecture - Robert Martin (Uncle Bob)

Produtividade ao Longo do Tempo

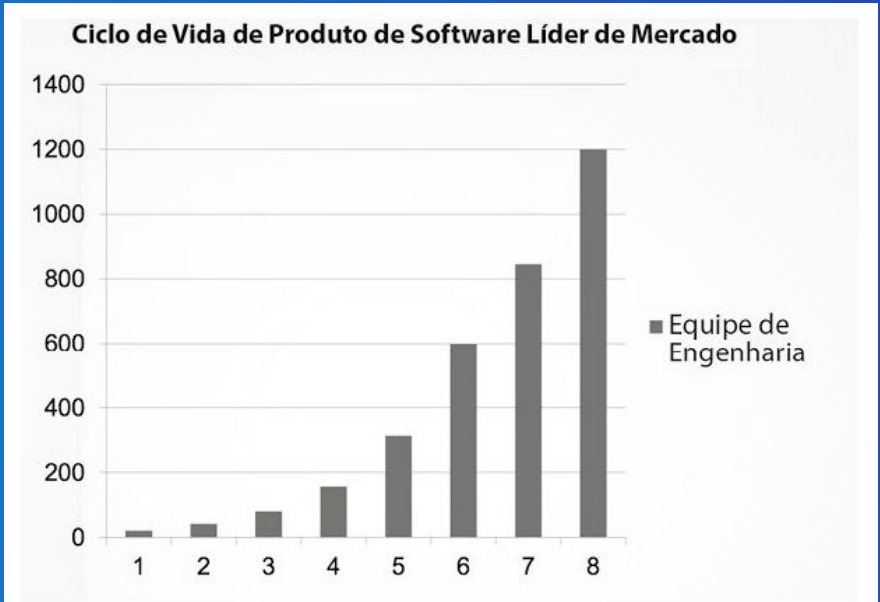


Figura 1.1 Crescimento da equipe de engenharia

Reprodução autorizada de uma apresentação de slides de Jason Gorman

Fonte: Livro Clean Architecture - Robert Martin (Uncle Bob)

O que é um Código Limpo

É um código:

- Fácil de ler
- Fácil de entender
- Fácil de alterar
- Fácil de testar

Benefícios de ter um Código Limpo

- Facilita a manutenção do produto.
- Facilita escrever testes automatizados.
- Facilita manter as bibliotecas de terceiros atualizadas.
- A equipe trabalha motivada e sente-se segura para fazer alterações e executar melhorias no código fonte do produto.

Benefícios de ter um Código Limpo

- As entregas são realizadas com maior segurança e de forma mais ágil.
- O tempo para responder às necessidades do mercado diminui bastante.

Conclusão

- Quanto melhor a qualidade do código, mais fácil mantê-lo limpo.
- Quanto pior a qualidade do código, mais caro é investir na limpeza do mesmo.
- Invista na qualidade de código desde o primeiro dia.

Clean Code

Clean Code

Nomenclaturas

Clean Code - Nomenclaturas

- Use nomes claros.
- Evite usar abreviações.
- Use constantes para valores fixos.
- Use nomes de constantes em letras maiúsculas.
- Evite usar o mesmo nome de variável para dois propósitos diferentes.
- Utilize o contexto atual para nomear uma variável.

Clean Code - Nomenclaturas

- Nomes de métodos devem expressar a ação pretendida.
- Escolha uma nomenclatura para métodos e classes e mantenha a consistência entre nomes.
- Não economize na hora de escrever o nome dos métodos e variáveis.

Clean Code

Comentários

Clean Code - Comentários

- Comentários não ajudam a melhorar um código ruim.
- Prefira escrever um código claro e conciso a escrever um comentário.
- Um comentário tende a ficar desatualizado. Muitas vezes a funcionalidade é alterada e o comentário não é atualizado.

Clean Code - Comentários

Comentários válidos:

- APIs (Javadoc).
- Uma melhoria a ser feita (TODO:).
- Explicar no código alguma limitação causada pelo uso de algum componente externo.

Clean Code

Métodos

Clean Code - Métodos

- Cada método deve fazer apenas uma ação.
- Evite escrever métodos muito grandes (> 20 linhas).
- Evite escrever muitas condições aninhadas (> 1 nível).
- Evite criar métodos com muitos parâmetros (> 3 params).
- Evite ter flags como parâmetros.

Clean Code

Tratamento de Erros

Clean Code - Tratamento de Erros

- Sempre dispare exceções não checadas (RuntimeException).
- Converta exceções checadas (Exception) para não checadas (RuntimeException).
- Crie classes de exceções específicas para cada contexto.
- Sempre propague a stack original quando fizer um catch de uma exceção (try...catch(OriginalException e) {throw new MinhaException(e)})

Clean Code - Tratamento de Erros

- Nunca ignore uma exceção (try..catch vazio) ou simplesmente retorne zero, null ou qualquer outro valor padrão quando ocorrer uma exceção, pois isto pode mascarar um erro que deveria ser tratado.
- Evite try..catch de blocos de código muito grandes.
- Evite try..catch muito genéricos. Ex: try..catch(Exception e).
- Lógicas muito complexas dentro do catch.

Clean Code

Formatação do Código

Clean Code - Formatação do Código

- Usar convenções para escrita do código.
- Habilite as ações automáticas ao salvar o código na IDE:
  - Formatação do código fonte (apenas linhas alteradas).
  - Organização de imports.
  - Ações adicionais.
- Elimine os warnings mostrados pela IDE.

Clean Tests

Clean Tests

- Os testes fazem parte do código fonte do produto.
- Da mesma forma que o código fonte, um teste deve ser fácil de entender e fácil de alterar.
- Aplique todas as regras de boas práticas de codificação também nos na escrita dos testes.

Clean Tests

- Cinco princípios básicos a serem seguidos ao escrever um teste:
  - Testes devem rodar de forma rápida.
  - Testes devem rodar de forma independente.
  - Testes devem ter o mesmo resultado independente do ambiente de execução.
  - Testes devem ser escritos junto com o código a ser testado.
  - Testes devem ser objetivos.

Code Smells

Code Smells

“Um **code smell** *é uma indicação superficial do que geralmente corresponde a um problema maior no sistema”* -- Kent Beck / Martin Fowler (Livro Refactoring)

Code Smells - Código Fonte

- Código duplicado.
- Classes ou métodos muito grandes.
- Variáveis globais.
- Métodos estáticos.
- IFs aninhados (complexidade ciclomática alta).
- Comentários no código.

Code Smells - Código Fonte

- Valores fixos no código fonte sem uso de constantes.
- Dependência cíclica entre pacotes.
- Hierarquia de classes muito grande.
- Superclasse depende de uma subclasse.
- Método protegido sendo usado por outra classe dentro do mesmo pacote.

Code Smells - Código Fonte

- Usar um tipo específico quando pode ser utilizado um tipo genérico. Por exemplo declarar uma variável com o tipo HashMap ao invés do seu tipo mais genérico que é o Map.
- Não usar “final” nos parâmetros dos métodos, permitindo a alteração dos valores recebidos dentro do método.
- Comprar tipos diferentes com “==” ou “equals”.
- Não prever NullPointerException na comparação de variáveis.

Code Smells – Testes

- Dificuldade para testar um método ou classe.
- Poucos testes automatizados.
- Testes que demoram muito para executar.
- Testes que sempre precisam ser configurados antes de serem executados.
- Classes de teste muito grandes. Com muitos métodos de testes ou com métodos de teste com muitas linhas.
- Precisar escrever testes para métodos privados.

Mais Boas Práticas

Mais Boas Práticas

- KISS: Keep It Simple
- DRY: Don't Repeat Yourself
- YAGNI: You Ain't Gonna Need It

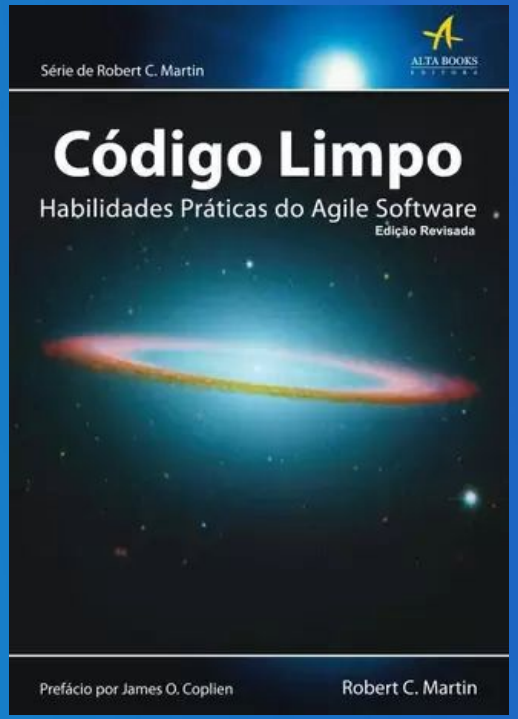
Exercícios

Exercícios

- **Habilitar a formatação de código na IDE (Save Actions)**
- **Identificar os problemas nos componentes abaixo e corrigi-los:**
  - Nomenclaturas
  - Métodos
  - Comentários
  - Tratamento de Erros

Dúvidas, Críticas e Sugestões?

Referências



Rodrigo Branas
https://www.youtube.com/c/RodrigoBranas

Clean Code with Robert C. Martin // Live #58

Clean Code // Live #35

Playlist Clean Code:
https://youtube.com/playlist?list=PLQCmSnNFVYnSpfpwwQGO8QHQ3CcizaZsV

Obrigado