AULA 10: Monitoramento da Qualidade Interna: Estratégias de código limpo.

Professora: **Rosana T. Vaccare Braga**

Agenda

- Relembrando o conceito de monitoramento da qualidade interna
- Convenção de nomenclatura
- Estratégias de código limpo
  - Princípios de código limpo
  - Refatoração de código
  - Análise estática de código
  - Práticas de desenvolvimento de equipe

Relembrando: Conceito de monitoramento da qualidade interna

- **Avaliação sistemática dos atributos e métricas internas do código, para identificar possíveis falhas, inconsistências ou oportunidades de melhoria na estrutura do software.**
- **É fundamental para garantir a qualidade e a confiabilidade do produto final, bem como para aprimorar o processo de desenvolvimento de software de forma contínua.**
  - **é possível identificar problemas precocemente, definir objetivos claros de qualidade, planejar melhorias no código-fonte e acompanhar os resultados alcançados.**

Conceito de monitoramento da qualidade interna

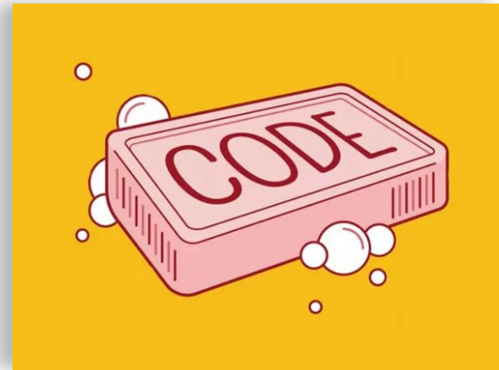
- **Algumas formas de monitorar a qualidade interna:**
  - **métricas de software**
  - **convenção de nomenclatura**
  - **estratégias de clean code**
  - **adoção de padrões de implementação, de projeto, etc.**
  - **e outros**

Convenções de nomenclatura

- **Conjunto de regras para nomear variáveis, funções e classes, que tornam o código mais legível e fácil de entender.**
- **Exemplos (nomeação de variáveis, classes, etc):**
  - Snake case: current_value, characters, big_calculator, right_side_enemies, blue_team
  - Pascal case: CurrentValue, Characters, BigCalculator, RightSideEnemies, BlueTeam
  - Camel case: currentValue, characters, bigCalculator, rightSideEnemies, blueTeam
  - Kebab case: current-value, characters, big-calculator, right-side-enemies, blue-team
  - Train case: Current-Value, Characters, Big-Calculator, Right-Side-Enemies, Blue-Team
  - Hungarian notation: lCurrentValue, rgCharacters, CBigCalculator, rgRightSideEnemies, rgBlueTeam

Estratégias de código limpo

- **Princípios de código limpo**
- **Técnicas de refatoração**
- **Análise estática de código**
- **Práticas de desenvolvimento de equipe**
- Teste de unidade
- Documentação de Código
- Padrões de projeto
- Ferramentas e tecnologias



Princípios de código limpo

- **O que é código limpo?**
  - É um código que é fácil de entender e modificar, bem organizado e segue práticas recomendadas de codificação. É um código que é escrito com um estilo consistente, é fácil de ler e entender, e que segue boas práticas de design de software.

"You know you are working on clean code when each routine you read turns out to be pretty much **what you expected**"

Ward Cunningham, criador do Wiki, do Fit e um dos signatários originais do Manifesto Ágil

Princípios de código limpo

- **Princípios de código limpo**
  - Dois conceitos que já foram abordados anteriormente, mas que também merecem destaque aqui: **Coesão** e **Acoplamento**.
  - **Um código com alta coesão e baixo acoplamento é mais fácil de manter, atualizar e estender, além de ser menos propenso a erros e bugs**. Por isso, é fundamental que os desenvolvedores priorizem esses conceitos ao escrever código para garantir a qualidade e a eficiência do sistema.

"I like my code to be **elegant** and **efficient**. Clean code does one thing well."

Bjarne Stroustrup, criador do C++

Princípios de código limpo

- **Princípios de código limpo**
  - **Simplicidade:** significa manter o código o mais simples possível, sem adicionar complexidade desnecessária. O código deve ser fácil de entender e manter, e deve evitar soluções excessivamente complexas para problemas simples.
  - **Legibilidade:** se refere à capacidade de ler e entender o código com facilidade. O código deve ser escrito em um estilo consistente, com nomes claros e descritivos para funções, variáveis e classes. O uso de comentários também pode ajudar a melhorar a legibilidade do código.

Princípios de código limpo

- **Regras gerais**
  - **Siga as convenções padrão.**
  - **Mantenha o código simples. Quanto mais simples melhor.**
  - **Reduza a complexidade tanto quanto possível.**
  - **Regra dos escoteiros: deixe o acampamento mais limpo do que você o encontrou.**
  - **Sempre encontre a causa raiz de um problema.**

Vejam mais regras em:

https://gist.github.com/wojteklu/73c6914cc446146b8b533c0988cf8d29

Princípios de código limpo - Resumo

Clean Code

Introduction

Conventions

THE KEY PRINCIPLES

Meaningful Names

Functions

Classes

Comments

Formatting

Error Handling

Unit Tests

Resumo completo: https://erik-uus.gitbook.io/clean-code/

Princípios de código limpo - Meaningful names

Declarações com significado

O que a lista representa?

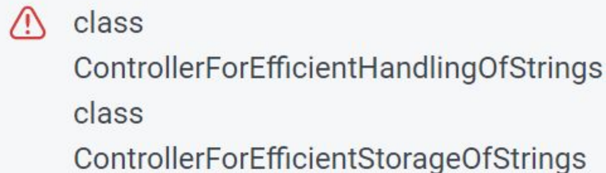
Evite a desinformação **(De que tipo é a lista de contas? String? Array de strings? Array de objetos?)**

https://erik-uus.gitbook.io/clean-code/

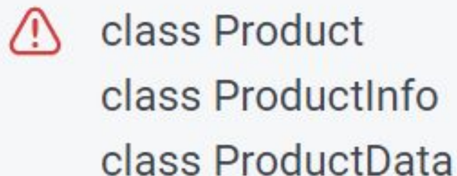
Princípios de código limpo - Meaningful names

Declarações com significado

Evite formatos semelhantes (Quanto tempo leva para detectar a diferença sutil?)


```
class
ControllerForEfficientHandlingOfStrings
class
ControllerForEfficientStorageOfStrings
```

Faça distinções significativas (Como esses nomes diferentes transmitem significados diferentes?)


```
class Product
class ProductInfo
class ProductData
```

https://erik-uus.gitbook.io/clean-code/

Princípios de código limpo - Meaningful names

Declarações com significado

Use nomes pronunciáveis

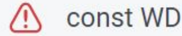
Adicione contexto usando prefixos (O que o estado representa? Condição ou país?)

https://erik-uus.gitbook.io/clean-code/

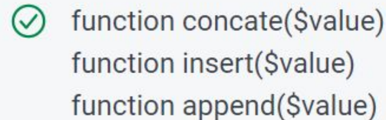
Princípios de código limpo - Meaningful names

Declarações com significado

Ajustar o comprimento de um nome ao tamanho de seu escopo (É óbvio que WD é um acrônimo para dias de trabalho por semana?)



Evite usar o mesmo nome para finalidades diferentes (O que significa add? Concatenar strings? Inserir um registro em uma tabela? Anexar um valor ao final de um array?)



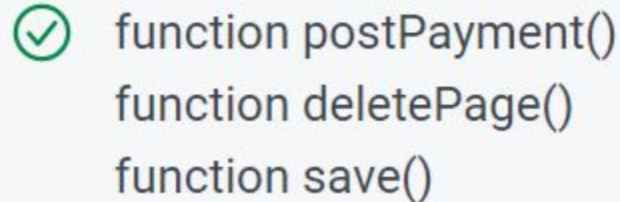
https://erik-uus.gitbook.io/clean-code/

Princípios de código limpo - Meaningful names

Declarações com significado

Use nomes do domínio do problema (O que significa o termo "document" no domínio de arquivos? As fotos são consideradas documentos?)

Métodos devem ter nomes de verbos



https://erik-uus.gitbook.io/clean-code/

Princípios de código limpo - Meaningful names

Declarações com significado

As classes devem ter nomes substantivos

```
class Customer
class WikiPage
class Account
```

Evite palavras soltas (no nome de uma classe)

```
Manager
Processor
Data
Info
```

https://erik-uus.gitbook.io/clean-code/

Princípios de código limpo - Funções

Functions should be small

Functions should hardly ever be 20 lines long

Functions should do one thing

Function is doing more than "one thing" if you can extract another function from it with a name that is not merely a restatement of its implementation.

etc…

Ver mais em https://erik-uus.gitbook.io/clean-code/

Princípios de código limpo - Classes

Classes should have one responsibility—one reason to change

Consider a class that compiles and prints a report. Such a class can be changed for two reasons. First, the content of the report could change. Second, the format of the report could change. These two things change for different causes. These two aspects of the problem are two separate responsibilities, and should be in separate classes.

Ver mais em https://erik-uus.gitbook.io/clean-code/

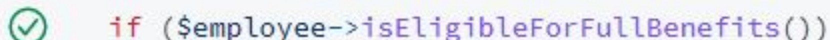
Princípios de código limpo - Comentários

Explain yourself in code

Rather than spend your time writing the comments that explain the mess you've made, spend it cleaning that mess.

```
// Check to see if the employee is eligible for full benefits

if ($employee->flag && ($employee->age > 65))
```



```
if ($employee->isEligibleForFullBenefits())
```

Ver mais em https://erik-uus.gitbook.io/clean-code/

Princípios de código limpo

Ao seguir esses princípios, os desenvolvedores podem escrever um código mais limpo, fácil de entender e de manter. O código limpo é mais fácil de testar, depurar e modificar, o que pode ajudar a reduzir o tempo e os custos de desenvolvimento e melhorar a qualidade do software.

“Clean code always looks like it was **written by someone who cares**.”

Michael Feathers, criador do livro “Working Effectively with Legacy Code"

Técnicas de refatoração

- **Como técnicas de refatoração podem ajudar a melhorar a qualidade interna do software?**
  - **As técnicas de refatoração são uma série de mudanças feitas no código para melhorar sua qualidade interna, sem alterar o comportamento externo do software. Essas mudanças ajudam a reduzir a complexidade do código, melhorar sua manutenibilidade e torná-lo mais fácil de entender.**
    - Segundo Sommerville, a manutenção da simplicidade é feita por meio da refatoração constante que melhora a qualidade do código, bem como por meio de projetos simples que não antecipam desnecessariamente futuras mudanças no sistema.

Técnicas de refatoração

- **Porque refatorar?**
  - **A refatoração é importante porque o software é constantemente modificado e evoluído ao longo do tempo, e a qualidade interna do código pode se deteriorar se não for cuidadosamente monitorada e mantida.**
- **Muitos dos princípios de clean code são equivalentes a refatorações**

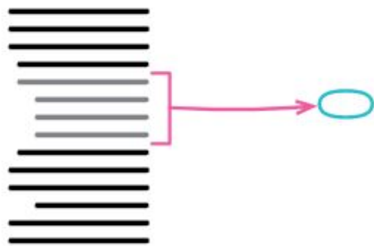
Mais sobre Refactoring: Livro Valente: https://engsoftmoderna.info/cap9.html

Técnicas de refatoração

- **Exemplos de refatoração**
  - reorganização da hierarquia de classes para eliminação de código duplicado
  - arrumação e renomeação de atributos e métodos, bem como a substituição do código com as chamadas para métodos definidos em uma biblioteca de programas.
- Ambientes de desenvolvimento de programas, como o Eclipse, incluem ferramentas de refatoração que simplificam o processo de encontrar dependências entre as seções do código e fazer as modificações no código global.

Técnicas de refatoração

- **Quais as técnicas de refatoração mais comuns?**
  - **Extração de método**: quando uma seção de código se torna muito grande e difícil de entender, pode ser útil extrair parte desse código em um método separado. Isso torna o código mais legível e fácil de entender.



```
function printOwing(invoice) {
  printBanner();
  let outstanding  = calculateOutstanding();

  //print details
  console.log(`name: ${invoice.customer}`);
  console.log(`amount: ${outstanding}`);
}
```

```
function printOwing(invoice) {
  printBanner();
  let outstanding  = calculateOutstanding();
  printDetails(outstanding);

  function printDetails(outstanding) {
    console.log(`name: ${invoice.customer}`);
    console.log(`amount: ${outstanding}`);
  }
}
```

https://refactoring.com/catalog/extractFunction.html

Técnicas de refatoração

- **Quais as técnicas de refatoração mais comuns?**
  - **Renomeação de variável:** variáveis mal nomeadas podem tornar o código difícil de entender. Renomear variáveis para nomes mais claros e descritivos pode ajudar a melhorar a legibilidade do código.
  - **Redução de duplicação de código:** duplicação de código é quando o mesmo trecho de código é usado em vários lugares no programa. Isso pode tornar o código mais difícil de manter e atualizar. Reduzir a duplicação de código através da criação de funções ou classes reutilizáveis pode ajudar a melhorar a qualidade interna do software.

Técnicas de refatoração

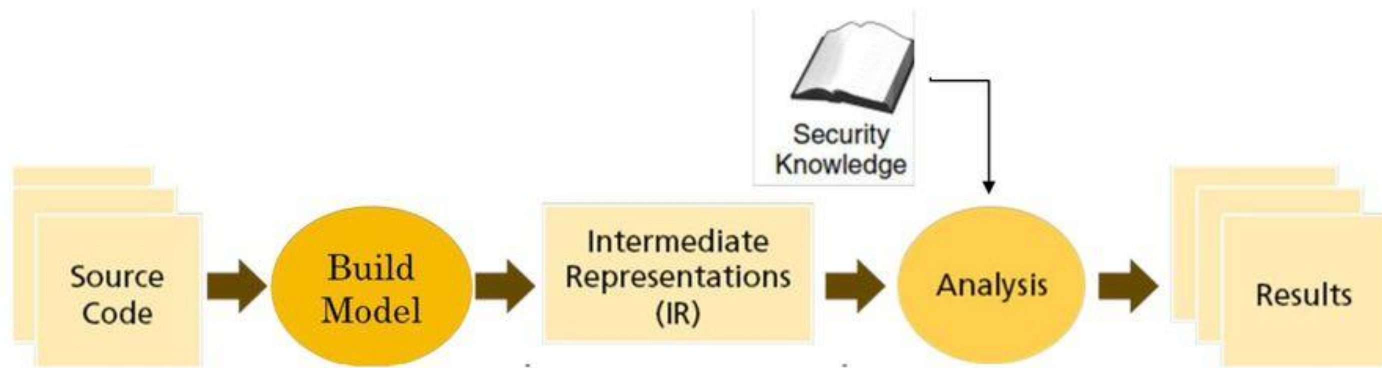
- **Quais as técnicas de refatoração mais comuns?**
    - **Extração de classe:** quando uma classe se torna muito grande e complexa, pode ser útil extrair parte dessa classe em uma nova classe separada. Isso pode tornar o código mais organizado e fácil de entender.
    - **Injeção de dependência:** quando o código é altamente acoplado, ou seja, as classes dependem fortemente umas das outras, a injeção de dependência pode ajudar a reduzir essa dependência. Isso torna o código mais modular e fácil de testar.

Essas são apenas algumas das muitas técnicas de refatoração disponíveis. Cada técnica tem seu próprio conjunto de benefícios e limitações, e cabe aos desenvolvedores escolher as técnicas mais adequadas para cada situação.

Vejam mais: https://refactoring.com/

Análise estática de código

- **O que é e como ela pode ajudar a monitorar a qualidade interna do software?**
    - É uma técnica usada para analisar o código-fonte de um programa sem executá-lo. Ela examina o código em busca de possíveis problemas de qualidade, como vulnerabilidades de segurança, erros de programação, violações de estilo de codificação e outros problemas relacionados à qualidade interna do software.



Análise estática de código

- **O que é e como ela pode ajudar a monitorar a qualidade interna do software?**
    - A análise estática de código pode ajudar a identificar e corrigir problemas de qualidade de software antes mesmo que o código seja compilado ou executado, o que pode economizar tempo e dinheiro no longo prazo.
- **Existem várias ferramentas de análise estática de código disponíveis, algumas das quais são de código aberto e outras comerciais. Essas ferramentas geralmente funcionam analisando o código-fonte do programa e identificando possíveis problemas.**

Análise estática de código

- **As ferramentas de análise estática de código normalmente fazem isso de uma das duas maneiras:**
    - **Análise baseada em regras:**
        - usam um conjunto de regras predefinidas para identificar problemas no código. Essas regras podem incluir coisas como proibições de certas funções, requisitos de comentários de código ou requisitos de estilo de codificação.
        - Essas ferramentas de análise estática de código são relativamente simples de usar, mas podem ser menos precisas do que as ferramentas de análise baseadas em modelos.

Análise estática de código

- **Análise baseada em modelos**:
  - usam algoritmos sofisticados para analisar o código e identificar possíveis problemas. Essas ferramentas usam modelos de análise de dados que são criados a partir de um grande número de amostras de código-fonte.
  - Elas podem ser mais precisas do que as ferramentas de análise baseadas em regras, mas podem exigir mais tempo e recursos para configurar e executar.

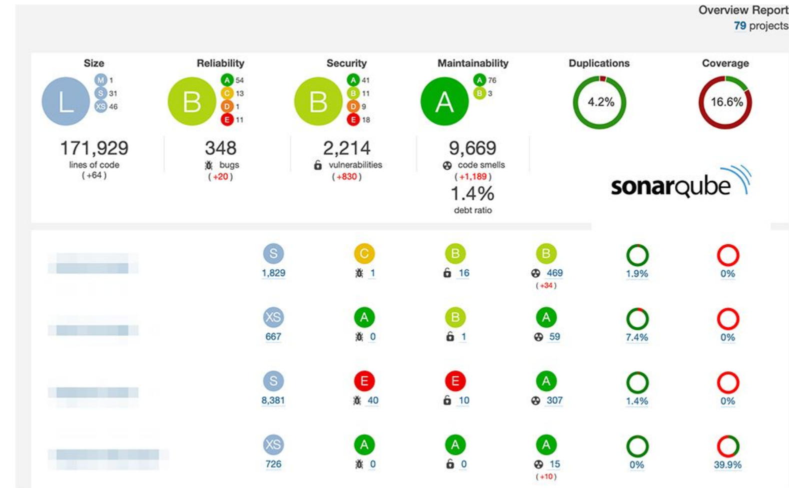
Ferramentas de análise estática de código

- **Algumas das ferramentas de análise de código estático mais comuns incluem o SonarQube, o PMD, o Checkstyle e o FindBugs.**
- **Essas ferramentas geralmente incluem uma interface de usuário amigável para que os desenvolvedores possam visualizar os resultados da análise de código estático e tomar as medidas necessárias para corrigir quaisquer problemas identificados.**
- **Link: comparação entre ferramentas:**

https://en.wikipedia.org/wiki/List_of_tools_for_static_code_analysis

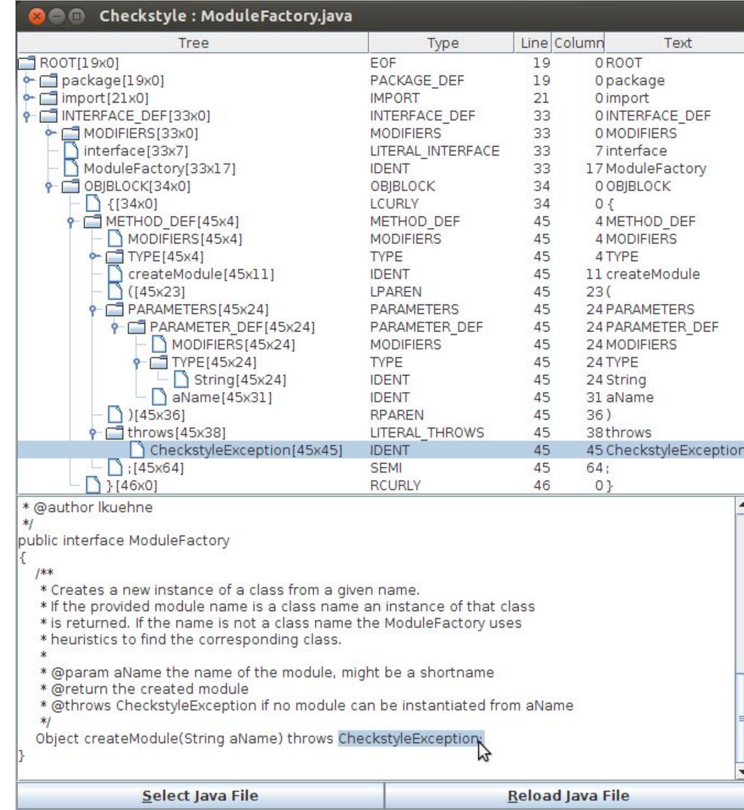
Ferramentas de análise de qualidade

O **SonarQube** **é uma ferramenta de análise estática de código-fonte de código aberto que** **identifica problemas de qualidade de código**, **como** **duplicação de código**, **vulnerabilidades de segurança** **e** **erros potenciais**. **É integrado com as ferramentas de desenvolvimento de software e oferece suporte a diversas linguagens de programação. Com a interface web, é possível monitorar a qualidade interna do software em tempo real e gerar relatórios de qualidade de código para desenvolvedores, gestores e clientes.**



Ferramentas de análise de qualidade

O Checkstyle é uma ferramenta de análise estática de código-fonte em Java, que verifica a conformidade do código com um conjunto de regras de boas práticas de desenvolvimento de software. Ele busca problemas de formatação, convenções de nomenclatura, estilo de codificação, entre outros aspectos. Integrado com outras ferramentas de desenvolvimento, como Maven e Gradle, gera relatórios de erros e warnings e permite criar regras personalizadas.



Ferramentas de análise de qualidade

O PMD é uma ferramenta de análise estática de código-fonte que busca possíveis problemas de código em projetos de software. Oferece uma ampla variedade de regras de análise configuráveis para várias linguagens de programação, incluindo Java, JavaScript e XML. Pode ser integrado com outras ferramentas de desenvolvimento, como Maven e Ant, e gera relatórios detalhados sobre os problemas encontrados no código, ajudando os desenvolvedores a melhorar a qualidade e a manutenibilidade do software.

PMD

An extensible cross-language static code analyzer.

Download | Documentation

Latest Version: 7.0.0-rc1 (25-March-2023)
Release Notes | Source

QuickStart

See also Getting Started

Linux | MacOS | Windows | Windows (Chocolatey)

1. Download pmd-bin-7.0.0-rc1.zip
2. Extract the zip-archive, e.g. to `C:\pmd-bin-7.0.0-rc1`
3. Add folder `C:\pmd-bin-7.0.0-rc1\bin` to PATH, either
    1. Permanently: Using System Properties dialog > Environment variables > Append to PATH variable
    2. Temporarily, at command line: `SET PATH=C:\pmd-bin-7.0.0-rc1\bin;%PATH%`
4. Execute at command line: `pmd.bat check -d c:\src -R rulesets/java/quickstart.xml -f text`

Checkout the existing rules for Java.

Práticas de desenvolvimento de equipe

- **Como podem ajudar a garantir a qualidade interna do software?**
  - **São importantes para garantir a qualidade interna do software, pois promovem a colaboração e a comunicação entre os membros da equipe, ajudando a identificar e corrigir problemas de qualidade mais cedo no processo de desenvolvimento.**

Práticas de desenvolvimento de equipe

- **Quais são as práticas mais comuns?**
  - **Revisão de código:** envolve um processo em que um ou mais membros da equipe examinam o código produzido por outro membro da equipe. Isso ajuda a identificar problemas de qualidade, como bugs, vulnerabilidades e violações de padrões de codificação.
  - **Programação em pares:** envolve dois desenvolvedores trabalhando juntos em um computador para escrever código. Isso ajuda a identificar problemas de qualidade, além de promover a troca de conhecimento e experiência entre os membros da equipe.

Práticas de desenvolvimento de equipe

- **Quais são as práticas mais comuns?**
  - **Integração contínua:** é uma prática em que as alterações de código são integradas ao repositório principal várias vezes ao dia, permitindo que os membros da equipe trabalhem em conjunto e resolvam conflitos. Isso ajuda a identificar problemas de qualidade mais cedo no processo de desenvolvimento.
  - **Teste automatizado:** envolve a criação de testes automatizados que são executados automaticamente sempre que uma alteração de código é feita. Isso ajuda a identificar problemas de qualidade mais cedo no processo de desenvolvimento e permite que os desenvolvedores verifiquem se as alterações não introduziram problemas.

Práticas de desenvolvimento de equipe

- **Quais são as práticas mais comuns?**
  - **Stand-ups diários:** são reuniões curtas em que os membros da equipe relatam o que fizeram no dia anterior, o que estão trabalhando atualmente e quaisquer problemas que estejam enfrentando. Isso ajuda a garantir que todos na equipe estejam na mesma página e ajuda a identificar problemas de qualidade.

Próxima aula: Estratégias de código limpo

- Princípios de código limpo
- Técnicas de refatoração
- Análise de código estático
- Práticas de desenvolvimento de equipe
- Teste de unidade
- Documentação de Código
- Padrões de projeto
- Ferramentas e tecnologias