Métodos de Desenvolvimento de Software

Construção de Software

Clean Code, Revisão de Código, Revisão em Pares, Refatoração, Programação em Pares e Soft Skills

Material Baseado nos livros:

- Swebok - v3
- Agile: Desenvolvimento de software com entregas frequentes e foco no valor de negócio. André Farias Gomes. Casa do Código.
- Código Limpo: Habilidades Práticas do Agile Software. Robert C. Martin. Atla Books.

“Qualquer idiota pode escrever código que um computador pode entender. Bons programadores escrevem código que seres humanos podem entender.”

(Martin Fowler)

Disciplinas de Construção

Swebok - versão 3

Disciplina de Construção

- O termo construção de software se refere à criação detalhada de software funcional por meio de uma combinação de codificação, verificação, teste de unidade, teste de integração e depuração.
- Fortemente vinculada as disciplinas de Requisitos, Design, Teste e Gerenciamento de Configuração

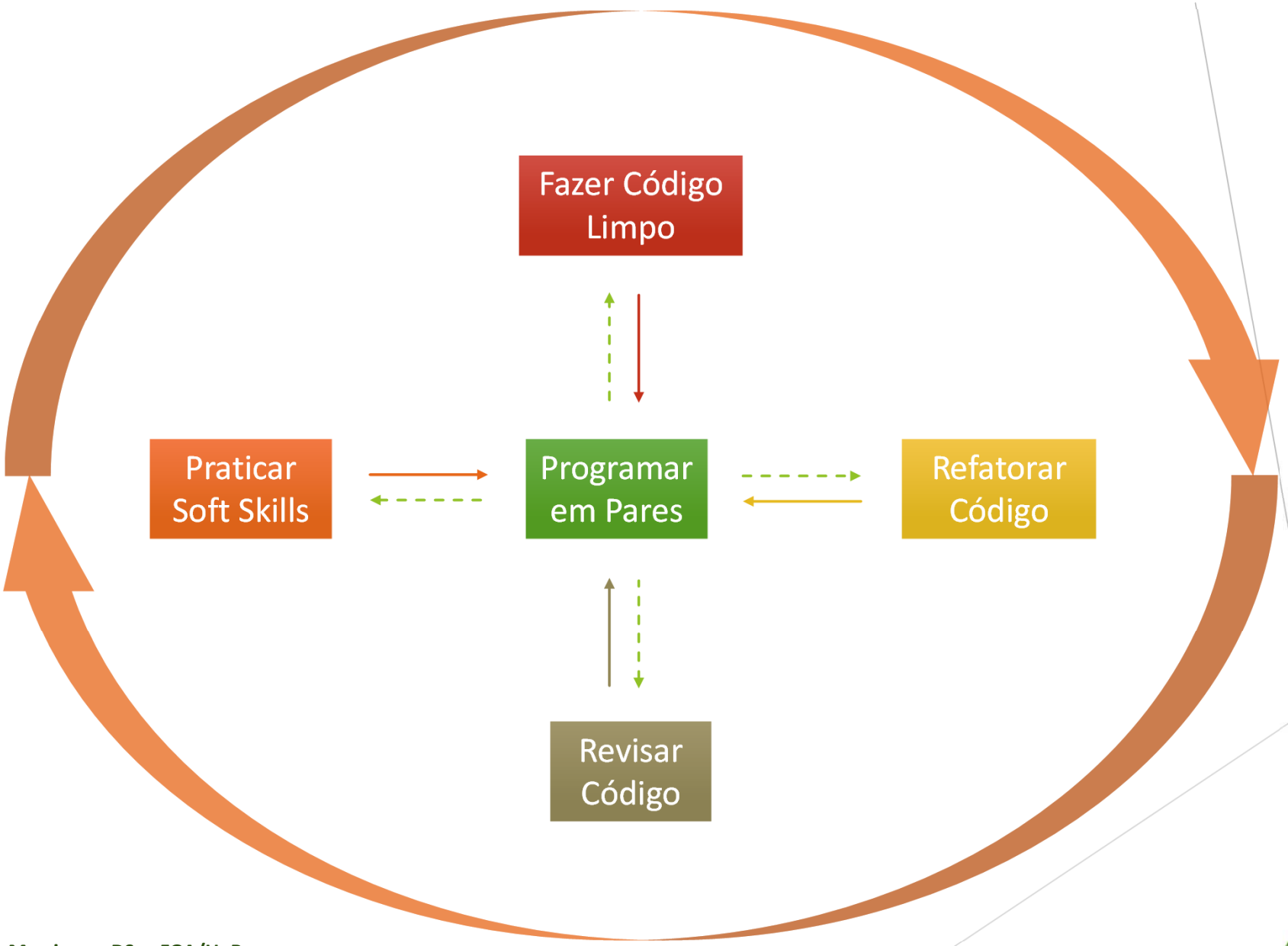
SWEBOK V3.0
Guide to the Software Engineering Body of Knowledge
IEEE computer society

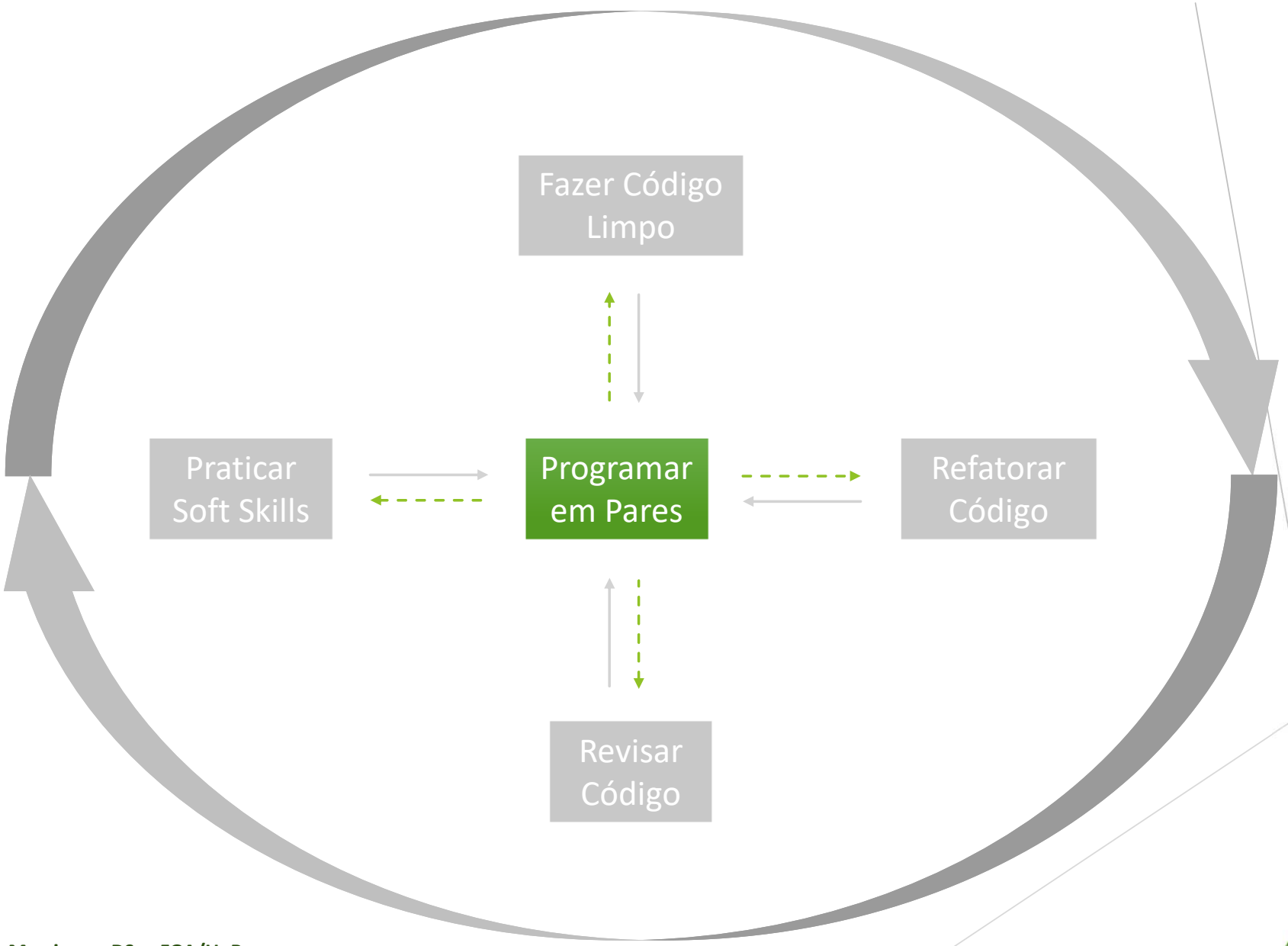
Disciplina de Construção (cont.)

- Fundamentalmente, a construção de software inclui
  - Minimizar a complexidade
  - Antecipar a mudança
  - Construir para verificar
  - Reuso
  - Padrões de construção

SWEBOK V3.0
Guide to the Software Engineering Body of Knowledge
IEEE computer society

Práticas e Técnicas


Fazer Código Limpo
Praticar Soft Skills
Programar em Pares
Refatorar Código
Revisar Código


Fazer Código Limpo
Praticar Soft Skills
Programar em Pares
Refatorar Código
Revisar Código

Programação em Pares

Programação em Pares

- Consiste na programação realizada por **duas pessoas** trabalhando para resolver, juntos, um problema
- Duas pessoas trabalham em um **mesmo computador**
- Geralmente, uma pessoa escreve o código (**condutor**) comentando sobre seu raciocínio, enquanto a outra analisa tudo e compartilha suas ideias (**navegador**). Assim, buscam chegar na melhor solução juntos
- O código é produto de **ambas as mentes**

Programação em Pares (cont.)

- **Por que** Programar em Pares?
  - É uma forma de **revisão de código contínua** e, geralmente, substitui a prática de revisão e inspeção de código.
  - A ideia é que se as revisões de código são uma coisa boa, então a revisão contínua é melhor.

Programação em Pares (cont.)

- Em XP toda a programação é feita por dois programadores, sentados lado a lado, na mesma máquina.
- As pesquisas sobre a programação em pares revelam que ela produz melhor código do que se os programadores trabalhassem isoladamente no mesmo tempo.
- À medida que os pares mudam, todos se beneficiam com o conhecimento especializado dos outros.
- Os programadores se instruem, melhoram as suas aptidões e se tornam mais valiosos para a equipe e para a empresa.

Programação em Pares (cont.)

- **Como** formar Parcerias?
  - A regra é: quando solicitado, você tem que dizer sim. Entretanto, você pode negociar o horário.
  - Os pares se formam naturalmente. Os gestores não se envolver na seleção dos pares.
  - As parcerias duram pouco. Quatro horas é um bom tempo para uma parceria. Às vezes uma parceria pode durar um dia. Muito raramente, pode durar uma semana.
  - Ao invés disso, as parcerias se formam por algumas horas e, então, se desfazem e recriam-se com parceiros diferentes.

Programação em Pares (cont.)

- **Quando** mudar de Parcerias?
  - Quando você estiver cansado do seu parceiro atual.
  - Quando você achar que o seu parceiro atual está demasiadamente cansado ou distraído para lhe ajudar.
  - Quando vocês dois ficarem empacados em um conceito.
  - Hora do almoço.
  - Hora de ir embora.
  - Ou geralmente, sempre que lhe apetecer.

Programação em Pares (cont.)

- **Atenção**
  - Questões de higiene e etiqueta
  - Incompatibilidades físicas e deficiências
  - A equipe está geograficamente distribuída
  - Meu parceiro se apodera do teclado e me ignora
  - Eu e meu parceiro discordamos a respeito de uma solução
  - Eu e meu parceiro falamos idiomas diferentes
  - Eu e meu parceiro trabalhamos em horários diferentes
  - Nossa equipe possui um número ímpar de programadores

Programação em Pares (cont.)

- **Atenção** (cont.)
  - Patologias das Parcerias
    - Parcerias exclusivas
    - Um cego guiando outro cego

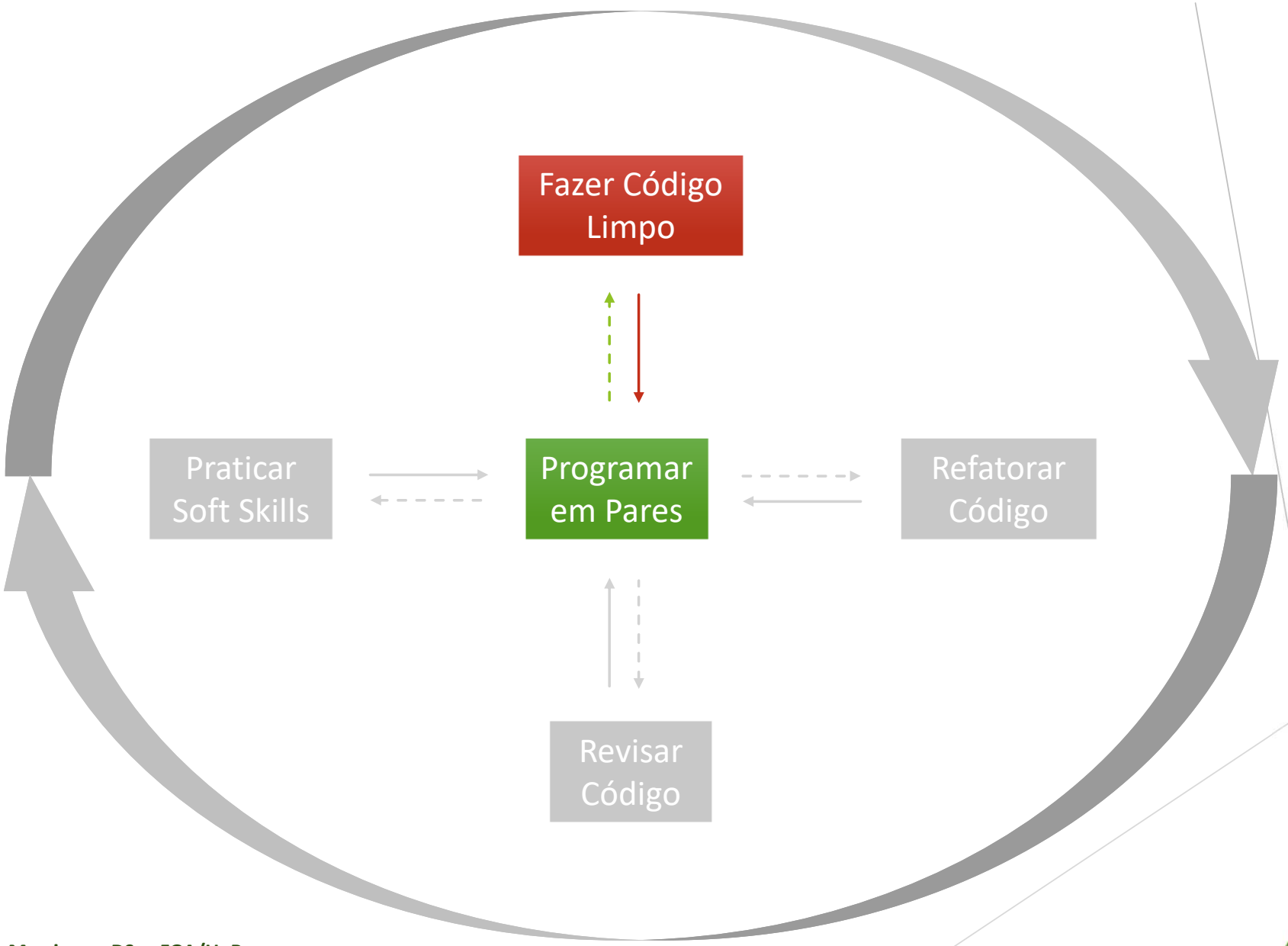
Programação em Pares (cont.)

Vantagens	
Pode proporcionar mais atenção e foco no projeto por parte dos programadores durante a codificação	Favorece uma maior padronização do código, já que ele está sendo desenvolvido em dupla
Produz menos erros e bugs no código	Agiliza na criação de solução para falhas, pois são duas pessoas pensando no mesmo problema
Revisão de código é realizado em tempo real, por parte da dupla	Proporciona troca de conhecimento entre os programadores ajudando a criar um código mais criativo e eficaz
Cria um ambiente mais colaborativo entre o time de desenvolvimento	Tende a diminuir a necessidade de refatoração

Programação em Pares (cont.)

Desvantagens	
Pode aumentar o custo devido a utilização de dois programadores, ao em vez de um	As duas pessoas precisam estar engajadas no projeto e com o mesmo objetivo
Facilidade em fugir da prática	Conseguir escutar a opinião de outras pessoas sobre o seu código
Pode apresentar dificuldade na interação entre os parceiros de programação	Pode ser cansativa para as duas pessoas
Capacidade de comunicação para poder compartilhar de forma correta suas ideias e opiniões	Possíveis problemas pessoais entre a equipe pode gerar em um desconforto na hora do trabalho

Clean Code


Fazer Código Limpo
Praticar Soft Skills
Programar em Pares
Refatorar Código
Revisar Código

Clean Code

“Desenvolvedores profissionais escrevem código como profissionais, e código escrito por profissionais é código limpo!”

(Robert C. Martin)

Clean Code (cont.)

- Robert C. Martin, em seu livro “Clean Code” apresenta várias conceituações sobre o que é código limpo.
- Autores diferentes utilizam palavras diferentes para conceituar, como:
  - Bjarne Stroustrup (criador C++): “... *O código deve ser elegante*...”
  - Grady Booch: “... *O código limpo é simples e direto*...”
  - Dave Thomas: “... *Além do seu criador, outro desenvolvedor pode ler e melhorar um código limpo*...”
  - Michael Feathers: “... *Um código limpo sempre parece que foi escrito por alguém que se importava*...”

Clean Code (cont.)

- Ron Jeffries: "*... Efetue todos os testes, sem duplicação de código, expressa todas as ideias do projeto que estão no sistema, minimiza o número de entidades, como classes, métodos, funções e outras do tipo...*"
- Ward Cunningham (co-criador XP): "*... Você sabe que está criando um código limpo, quando cada rotina que vocês leia se mostra como o que você esperava...*"

Clean Code (cont.)

- Algumas características de Código Limpo:
  - Elegante, fácil de se compreender, e agradável de se ler
  - Simples e direto
  - Segue princípios de programação
  - Sem duplicidade
  - Bem testado
  - Parâmetros, Métodos, Funções, Classes e Variáveis possuem nomes significativos (que revela seu propósito)

Clean Code (cont.)

- Algumas características de Código Limpo (cont.):
  - Código é autoexplicativo (sem necessidade de comentários para compreender o código)
  - Código bem formatado (é possível ler o código sem precisar utilizar a barra de rolagem)
  - Métodos e Classes devem ter uma única responsabilidade (princípio da responsabilidade única)

Clean Code (cont.)

- A Regra do Escoteiro
  - Não basta escrever um bom código. Ele deve sempre ser mantido limpo.
  - *“Deixe a área do acampamento mais limpa do que como você a encontrou”*

Regras Gerais de Clean Code

- **Keep It Stupid Simple (KISS)**
  - Mantenha tudo o mais simples possível
  - Diz que devemos manter a lógica sempre o mais simples possível.
  - Evitar complexidade desnecessária, para que o código seja fácil de ler e fácil de evoluir ou de se dar manutenção.

```
public String diaSemana(int numeroDia){

    switch(numeroDia){
      case 0:
        return "Segunda";
      case 1:
        return "Terca";
      case 2:
        return "Quarta";
        ...

    }

}
```

=>

```
public String diaSemana(int numeroDia){

    String[] dias = {
      "Segunda",
      "Terca",
      "Quarta",
        ...
    };
    return dias[numeroDia];
}
```

Regras Gerais de Clean Code

- **Regra do Escoteiro**
  - "Deixe sempre o acampamento mais limpo do que você encontrou!".
  - Toda vez que um desenvolvedor mexer em um código, deverá deixá-lo mais bem organizado do que como o encontrou.
  - Se todos os desenvolvedores de um time seguirem esse princípio, aos poucos cada trecho de código poderá evoluir.

Regras Gerais de Clean Code

- **Regra do Escoteiro**
  - Sempre que um código for ser evoluído, o desenvolvedor deverá primeiro analisa-lo e refletir se não seria possível melhorar algum trecho.
  - Caso seja possível, a melhoria deve ser feita antes das novas alterações.

Regras Gerais de Clean Code

- **Causa Raiz**
  - Sempre procure a causa raiz do problema.
  - Não busque correções temporárias, que são rápidas e fáceis de fazer, mas que não corrigem o problema por completo.
  - Isso faz com que o problema volte a ocorrer e acabe causando um retrabalho ainda maior no futuro.
  - O ideal é sempre solucionar a causa raiz, mesmo que no momento do erro isso signifique gastar mais tempo.

Regras Gerais de Clean Code

- **Utilize nomes descritivos (significativos)**
  - Nomear bem as variáveis, funções ou classes facilita a leitura e o entendimento do código.
  - Nomes confusos fazem com que os desenvolvedores percam mais tempo tentando entender para que serve cada trecho de código.

```
//O que é x?
x = 0;

//Total de que?
total = 0;

//Agora sim!
totalDeRegistros = 0;
```

Regras Gerais de Clean Code

- **Siga as Convenções**
  - Se no início do projeto foram definidas convenções siga-as.
  - Coisas como por exemplo: nomes de variáveis, ou escrever constantes em maiúsculo, ou mesmo convenções de idioma.
  - Seguir as convenções pré-estabelecidas ajuda a manter o código fácil de entender para todos os membros de uma equipe, inclusive os mais novos, além de criar um padrão reconhecível.
  - Para seguir tal princípio, basta tomar o cuidado de seguir as convenções ao escrever novos trechos de código. E para os trechos já implementados, verificar se todos seguem os mesmos padrões.

Regras Gerais de Clean Code

- **Siga as Convenções**

```
//Em inglês sem regra de escrita
void recordclient()

//Em português utilizando camelCase
void realizarCompra()

//Em português utilizando PascalCase
void AtualizarEstoque()
```

Cada uma das três funções descritas utiliza convenções diferentes.

=>

```
void registrarCliente()

void realizarCompra()

void atualizarEstoque()
```

O mais correto, portanto, seria escolher uma das convenções e segui-la consistentemente

Regras Gerais de Clean Code

- **Don’t Repeat Yourself (DRY)**
  - “Não se repita“
  - Evite repetições de código.
  - Permita reusabilidade.
  - Para seguir esse princípio os desenvolvedores devem procurar no projeto, manualmente ou através de ferramentas automatizadas, por trechos de código que sejam frequentemente repetidos. Essa repetição deve então ser removida.

Regras Gerais de Clean Code

- **Escreva funções pequenas e com apenas um objetivo**
  - Funções menores e com poucas responsabilidades são mais fáceis de serem entendidas e reutilizadas.
  - Também fica mais fácil encontrar quaisquer problemas e mudar comportamentos.
  - Se uma função possuí várias responsabilidades, é ideal separar essas responsabilidades entre múltiplas funções.

Regras Gerais de Clean Code

- **Escreva funções pequenas e com apenas um objetivo**

```
//Ruim
public void RealizarPedido()
{
    // Cadastra o cliente
    // Aplica o desconto
    // Atualiza o estoque
    // Salva o pedido
}

//Bom
public void CadastrarCliente() { ... }
public void AplicarDesconto() { ... }
public void AtualizarEstoque() { ... }
public void RegistrarPedido() { ... }
```

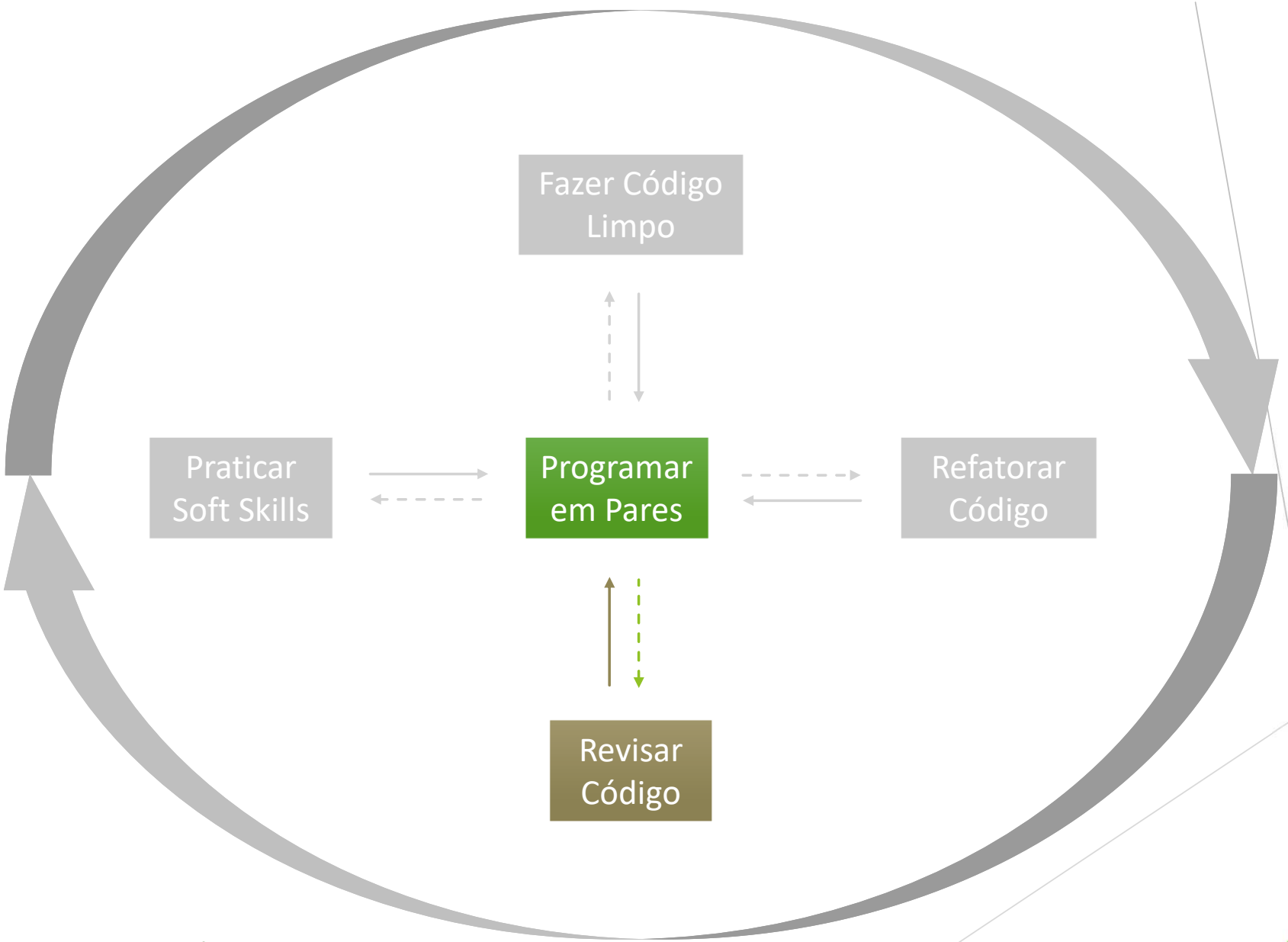
Regras Gerais de Clean Code

- **Dica**
  - No caso de projetos grandes já em andamento, às vezes pode ser difícil analisar tudo manualmente para ver se os princípios estão sendo seguidos.
  - Para facilitar esse trabalho, podem ser utilizadas as chamadas ferramentas de análise de código estático, que vão conseguir identificar a maior parte dos problemas mencionados automaticamente.

Regras Gerais de Clean Code

- **Considerações**
  - Em seu livro, Robert Martin (2014), cita diversos outros princípios. Alguns deles se referindo a aspectos mais complexos da programação, como por exemplo paralelismo. E também a aspectos que não são diretamente do código, como por exemplo estrutura de pastas e arquivos.
  - Os princípios do Clean Code são amplamente aceitos pela comunidade desde que foram estabelecidos, e se seguidos à risca certamente tendem a propiciar um aumento significativo na qualidade do produto em questão, dito que esta pode ser diretamente influenciada pela qualidade interna do software.

Revisão de Código e Revisão em Pares


Fazer Código Limpo
Praticar Soft Skills
Programar em Pares
Refatorar Código
Revisar Código

Revisão de Código

- O que é?
  - Identificação e prevenção de erros
  - Feedback
- Qual a importância?
  - Incentiva o trabalho em equipe
  - Compartilha e dissemina o conhecimento entre os membros da equipe
  - Economia e qualidade

Revisão de Código

É preciso **TRABALHAR EM EQUIPE**

EVITAR encontrar culpados e/ou debochar das pessoas que cometeram erro

Revisão de Código (cont.)

- Como fazer?
  - Entenda o que foi feito
  - Conhecer o objetivo do código
  - Um ou mais desenvolvedores devem ler o código fonte com o objetivo de assegurar a sua qualidade e aprender com o processo
  - Pode ser individual ou coletiva
  - Simular situações
  - Revise fazendo perguntas coerentes
  - Realizar perguntas coerentes

Revisão de Código (cont.)

- Vantagens
  - Incentiva o trabalho em equipe
  - Aumenta a qualidade da entrega
  - Compartilha o conhecimento
  - Aumenta a responsabilidade coletiva

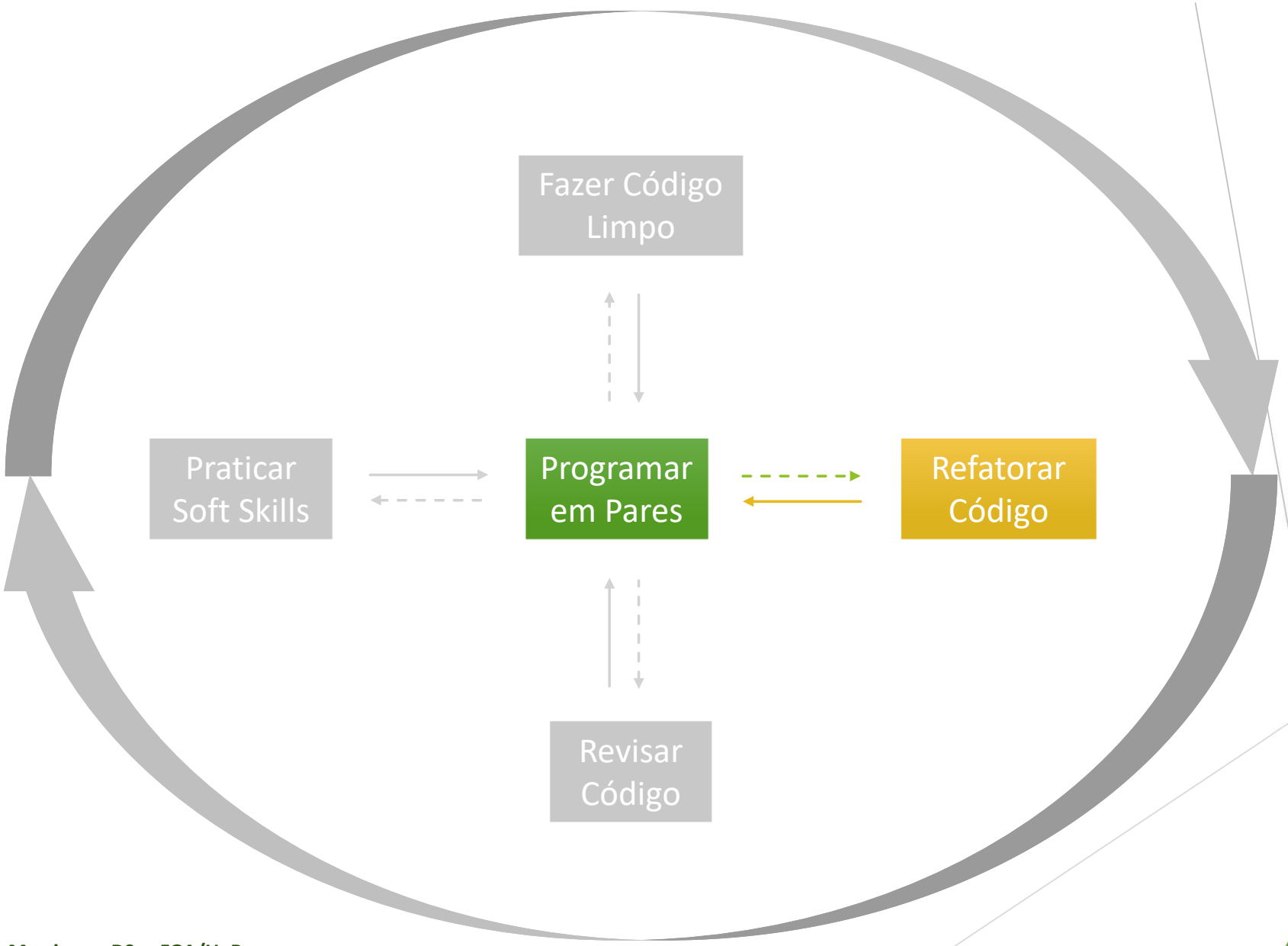
Revisão de Código (cont.)

- Boas práticas
  - Registro de revisão e feedback
  - Comunicação constante
  - Identificar padrões
  - Sem hierarquia
  - Tempo e calma para revisar
  - Entender o contexto do código

Revisão em Pares

- As pessoas que irão revisar não podem ter participado diretamente da programação.
- O desenvolvedor, ao começar o projeto, escolhe um par para compartilhar ideias do projeto e o que deve e como deve ser feito o mesmo.
- O par deve ajudar a projetar a solução, para contribuir com seu conhecimento e ideias.
- Depois ficam responsáveis pela primeira revisão do código.

Refatoração


Fazer Código Limpo
Praticar Soft Skills
Programar em Pares
Refatorar Código
Revisar Código

Refatoração

- Refatorar é **alterar a estrutura do código sem alterar o seu comportamento**.
- É uma prática que permite ao desenvolvedor **melhorar o design do código**, tornando-o mais limpo e fácil de se compreender.
- É essencial para **prevenir que o código se deteriore**.
- A refatoração deve ser intimamente realizada **junto com teste**, visando garantir que o comportamento do código não está sendo alterado.

Refatoração (cont.)

- Objetivos da Refatoração
  - Atualizar a estrutura interna do código e deixá-la de acordo com as mudanças feitas;
  - Deixar o código mais limpo e organizado;
  - Ajudar no tratamento de bugs no código;
  - Aumentar a qualidade do código como um todo, facilitando o andamento e rapidez do projeto diretamente.

Refatoração (cont.)

- **Por que** refatorar?
  - Para melhorar o design e a legibilidade do código.
  - Existem várias metas específicas:
    - O código deve passar em todos os seus testes.
    - Ele deve ser o mais expressivo possível que você possa fazer.
    - Ele deve ser o mais simples possível que você possa fazer.
    - Ele não deve ter nenhuma redundância.

Refatoração (cont.)

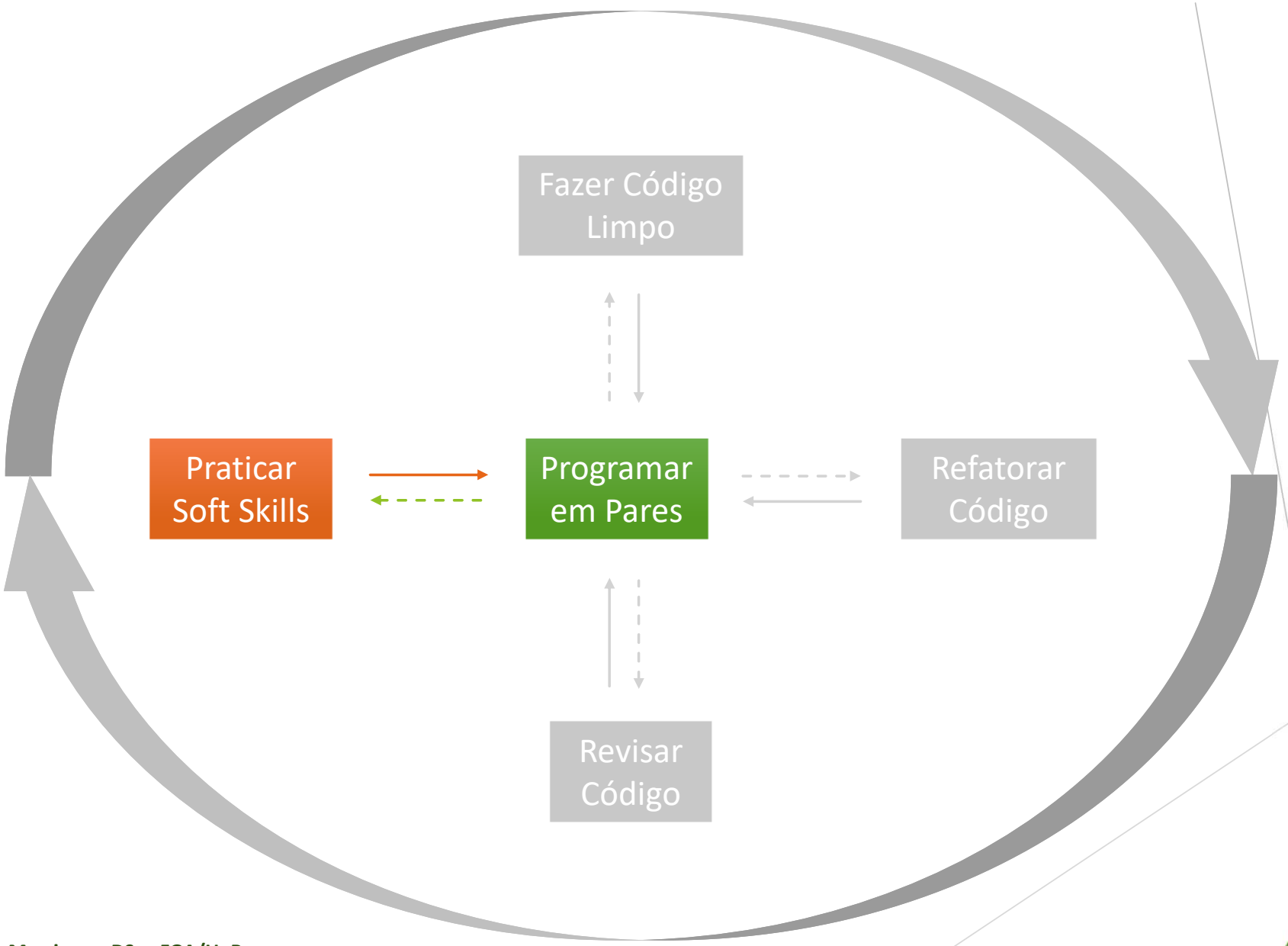
- **Quando** deve-se refatorar?
- Observe as cinco características de um produto íntegro:
  - **Simplicidade**
    - Geralmente, quanto mais simples o design do código, melhor. Padrões de projeto, quanto bem aplicados, são uma ótima maneira de implementar soluções simples para problemas complexos
  - **Clareza**
    - O código deve ser facilmente compreendido por todos os membros da equipe
  - **Eficácia**
    - O código deve atingir o objetivo pelo qual foi criado

Refatoração (cont.)

- Observe as cinco características de um produto íntegro (cont.):
  - **Sem repetição**
    - O mesmo código não deve estar em dois lugares diferentes
  - **Utilidade**
    - Toda funcionalidade deve ser útil para o seu usuário. Quando isso passa a não acontecer, a funcionalidade deixa de ser necessária, tem-se então, desperdício para mantê-la. Em alguns casos é melhor retirá-la do código

Não deixe que o sol se ponha com código ruim.

Soft Skills


Fazer Código Limpo
Praticar Soft Skills
Programar em Pares
Refatorar Código
Revisar Código

Soft Skills

- Habilidades não técnicas que lidam com interações interpessoais e intrapessoais
- Estão relacionadas com traços de personalidade e inteligência emocional

Soft Skills (cont.)

- Alguns exemplos:
  - Comunicação
  - Empatia
  - Adaptação
  - Resiliência
  - Liderança
  - Colaboração
  - Autocontrole
  - Proatividade

Soft Skills (cont.)

- Como desenvolver Soft Skills?
  - Reconhecer potencialidades
  - Feedback
  - Autoconhecimento
  - Praticar

Exercício

Programação em pares, revisão de código, clean code e refatoração

Exercício – PARTE 1

1. Cada estudante deve escolher uma funcionalidade a ser desenvolvida
2. Dado um tempo de desenvolvimento (25min), será realizada uma revisão por pares
   - Observar a utilização das regras de clean code
     - Nomes significativos, estruturas simples, sem repetições,
     - Responsabilidade única, resolução de problemas (causa raiz),
     - Reutilização de código, comentários, formatação.
   - Identificar necessidade de refatoração

Exercício – PARTE 2

1) Escolham duplas para programação em pares
2) Definam uma funcionalidade a ser desenvolvida
3) Iniciem o desenvolvimento
   - Observar a utilização das regras de clean code
     - Nomes significativos, estruturas simples, sem repetições,
     - Responsabilidade única, resolução de problemas (causa raiz),
     - Reutilização de código, comentários, formatação.
   - Identificar necessidade de refatoração

Exercício – PARTE 3

1) Reportem quais foram as diferenças identificadas entre as práticas
   - Facilidades
   - Dificuldades
   - Qualidade no código
   - Etc.

Métodos de Desenvolvimento de Software

Construção de Software

Clean Code, Revisão de Código, Revisão em Pares, Refatoração, Programação em Pares e Soft Skills

Material Baseado nos livros:

- Swebok - v3
- Agile: Desenvolvimento de software com entregas frequentes e foco no valor de negócio. André Farias Gomes. Casa do Código.
- Código Limpo: Habilidades Práticas do Agile Software. Robert C. Martin. Atla Books.