Tópicos Especiais em Plataformas Emergentes

TP3

Henrique Amorim Costa Melo - 17/0144488
Paulo Gonçalves Lima - 17/0122549
Pedro Vítor de Salles Cella - 17/0113060

Brasília, DF
2023

Dentre as 9 características mostradas foram escolhidas 5, para serem descritas e apresentadas melhor com exemplos e comparações, dentre as escolhidas estão:

- Simplicidade;
- Elegância;
- Modularidade (baixo acoplamento e alta coesão);
- Boas interfaces;
- Boa documentação.

Simplicidade

Simplicidade é uma característica desejável em código, pois facilita a manutenção e a compreensão do mesmo. A simplicidade é alcançada através de diversas técnicas, tais como:

- Estrutura: O código deve ser organizado de maneira clara e lógica, com funções e classes pequenas e bem definidas, que realizam tarefas específicas.

- Claridade: O código deve ser escrito de maneira clara e concisa, evitando a utilização de construções complexas ou desnecessárias.
- Coesão: As funções e classes devem ter alta coesão, ou seja, devem realizar apenas tarefas relacionadas entre si.

- Acoplamento: O código deve ter baixo acoplamento, ou seja, as partes do código devem ser independentes entre si, para que possam ser modificadas ou reutilizadas sem afetar outras partes do código.

- Outros efeitos: outras características desejáveis incluem a facilidade de teste e manutenção, e a facilidade de adição de novas funcionalidades.

Em resumo, a simplicidade em um código geralmente se traduz em um código mais fácil de entender, modificar e manter. Isso é especialmente importante em projetos de longa duração ou projetos com muitos desenvolvedores envolvidos.

A característica de simplicidade está intimamente relacionada aos maus-cheiros de código, pois a simplicidade é um dos principais objetivos na identificação e correção dos maus-cheiros. Martin Fowler, em seu livro "Refactoring: Improving the Design of Existing Code", define maus-cheiros como "indícios de que algo está errado no código e que precisa ser refatorado". Alguns dos maus-cheiros de código mais comuns relacionados à simplicidade são:

- Código duplicado: código repetido é um indicador de que o código não está organizado de maneira eficiente e pode ser simplificado.
- Funções longas: funções longas podem ser difíceis de entender e manter, e podem ser simplificadas dividindo-as em funções menores e mais específicas.

- Classes complexas: classes complexas podem ser difíceis de entender e manter, e podem ser simplificadas dividindo-as em classes menores e mais específicas.

- Variáveis com nomes ruins: variáveis com nomes ruins podem tornar o código difícil de entender, e podem ser simplificadas com nomes mais descritivos e claros.

Em resumo, a característica de simplicidade é um objetivo importante na identificação e correção de maus-cheiros de código, pois ajuda a tornar o código mais fácil de entender, modificar e manter. Isso é especialmente importante em projetos de longa duração ou projetos com muitos desenvolvedores envolvidos.

Uma operação de refatoração comum que pode ajudar a aumentar a simplicidade do código é a extração de método. A extração de método é a técnica de separar uma parte de um método existente em uma nova função, com um nome mais descritivo e específico. Essa técnica ajuda a aumentar a coesão e a diminuir o tamanho dos métodos, tornando-os mais fáceis de entender e manter.

Por exemplo, imagine que temos o seguinte método em uma classe:

```
class Pedido {
  void finalizarPedido(Pedido pedido) {
    // validação do pedido
    if (pedido.getItens().isEmpty()) {
      throw new IllegalArgumentException("Pedido sem itens");
    }
    // atualização do estoque
    for (Item item : pedido.getItens()) {
      estoque.decrementar(item.getProduto(), item.getQuantidade());
    }
    // gravação no banco de dados
    pedidoDao.gravar(pedido);
  }
}
```

Esse método tem mais de uma responsabilidade, validação do pedido, atualização do estoque e gravação no banco de dados. Para simplificar, podemos extrair cada uma dessas responsabilidades para funções separadas, como abaixo:

```
class Pedido {
  void finalizarPedido(Pedido pedido) {
    validarPedido(pedido);
    atualizarEstoque(pedido);
    gravarPedido(pedido);
  }
  void validarPedido(Pedido pedido) {
    if (pedido.getItens().isEmpty()) {
      throw new IllegalArgumentException("Pedido sem itens");
    }
  }
  void atualizarEstoque(Pedido pedido) {
    for (Item item : pedido.getItens()) {
      estoque.decrementar(item.getProduto(), item.getQuantidade());
    }
  }
  void gravarPedido(Pedido pedido) {
    pedidoDao.gravar(pedido);
  }
}
```

Modularidade

Modularidade é a característica de dividir um sistema em módulos ou componentes independentes, que podem ser facilmente compreendidos, testados e modificados sem afetar o funcionamento do sistema como um todo. A modularidade é importante porque ajuda a manter o código organizado, fácil de entender e fácil de manter.

Alguns dos efeitos da modularidade no código incluem:

- Estrutura: O código é dividido em módulos ou componentes que são fáceis de entender e navegar.

- Claridade: A modularidade ajuda a manter o código claro, pois cada módulo ou componente tem uma única responsabilidade.

- Coesão: Cada módulo ou componente tem alta coesão, ou seja, realiza apenas tarefas relacionadas entre si.

- Acoplamento: A modularidade ajuda a manter o acoplamento baixo, ou seja, os módulos ou componentes são independentes uns dos outros, para que possam ser modificados ou reutilizados sem afetar outros módulos ou componentes.
- Outros efeitos: outras características desejáveis incluem a facilidade de teste, manutenção e adição de novas funcionalidades.

Em resumo, a modularidade é uma característica importante para o código, pois ajuda a manter o código organizado, fácil de entender e fácil de manter. Isso é especialmente importante em projetos de longa duração ou projetos com muitos desenvolvedores envolvidos.

A característica de modularidade está intimamente relacionada aos maus-cheiros de código, pois a modularidade é um dos principais objetivos na identificação e correção dos maus-cheiros. Alguns dos maus-cheiros de código mais comuns relacionados à modularidade são:

- Acoplamento excessivo: acoplamento excessivo pode tornar o código difícil de modificar e manter, e pode ser simplificado através da melhoria da modularidade do código.

- Código duplicado: código repetido é um indicador de que o código não está organizado de maneira eficiente e pode ser simplificado através da modularidade.

- God Class: é uma classe muito grande, com muitas responsabilidades e muitas dependências, o que torna ela difícil de ser testada, modificada e compreendida.

Em resumo, a característica de modularidade é um objetivo importante na identificação e correção de maus-cheiros de código, pois ajuda a tornar o código mais fácil de entender, modificar e manter, além de tornar o código mais escalável e reutilizável.

Uma operação de refatoração comum que pode ajudar a aumentar a modularidade do código é a extração de classe. A extração de classe é a técnica de separar uma parte de uma classe existente em uma nova classe, com uma responsabilidade específica. Essa técnica ajuda a aumentar a coesão e a diminuir o tamanho das classes, tornando-as mais fáceis de entender e manter, além de diminuir o acoplamento entre as classes.

Por exemplo, imagine que temos a seguinte classe:

```
class GerenciadorDeProdutos {
  void adicionarProduto(Produto produto) {
    // validação do produto
    if (produto.getNome().isEmpty()) {
      throw new IllegalArgumentException("Produto sem nome");
    }
    // atualização do estoque
    estoque.adicionar(produto);
    // gravação no banco de dados
    produtoDao.gravar(produto);
  }
}
```

Esse método tem mais de uma responsabilidade, validação do produto, atualização do estoque e gravação no banco de dados. Para aumentar a modularidade, podemos extrair cada uma dessas responsabilidades para classes separadas, como abaixo:

```
class GerenciadorDeProdutos {
  void adicionarProduto(Produto produto) {
    validadorProduto.validar(produto);
    estoque.adicionar(produto);
    produtoDao.gravar(produto);
  }
}
class ValidadorProduto {
  void validar(Produto produto) {
    if (produto.getNome().isEmpty()) {
      throw new IllegalArgumentException("Produto sem nome");
    }
  }
}
class Estoque {
  void adicionar(Produto produto) {
    // lógica para adicionar o produto no estoque
  }
}
class ProdutoDao {
  void gravar(Produto produto) {
```

```
    // lógica para gravar o produto no banco de dados
  }
}
```

Com essa refatoração, cada classe tem uma única responsabilidade e o acoplamento entre as classes é menor. Isso torna o código mais fácil de entender e manter, além de tornar o código mais escalável e reutilizável. Essa operação de refatoração é uma das muitas técnicas que podem ser usadas para aumentar a modularidade do código e melhorar sua qualidade.

Boa Documentação

A boa documentação é a característica de ter um conjunto completo e preciso de documentação associado ao código, que inclui comentários, documentação de código, manuais e outros recursos. A boa documentação é importante porque ajuda a entender o código, facilitando a sua manutenção e evolução.

Alguns dos efeitos da boa documentação no código incluem:

- Estrutura: A documentação ajuda a entender a estrutura do código e como as diferentes partes do código estão relacionadas.

- Claridade: A boa documentação ajuda a explicar o propósito e a funcionalidade do código, tornando-o mais fácil de entender.

- Coesão: A documentação ajuda a entender a coesão do código, ou seja, como as diferentes partes do código estão relacionadas entre si.

- Acoplamento: A documentação ajuda a entender o acoplamento entre as diferentes partes do código, ou seja, como as diferentes partes do código estão relacionadas entre si.

- Outros efeitos: a boa documentação ajuda a guiar novos desenvolvedores, facilita o trabalho em equipe, além de ajudar na manutenção e evolução do código.

Em resumo, a boa documentação é uma característica importante para o código, pois ajuda a entender o código, facilitando a sua manutenção, assim como entender o que de fato um código faz e como ele deve ser instalado e utilizado.

A característica de boa documentação está diretamente relacionada aos maus-cheiros de código. Alguns dos maus-cheiros de código mais comuns relacionados à documentação são:

- Código sem comentários: Código sem comentários pode ser difícil de entender e manter, especialmente para novos desenvolvedores ou para aqueles que trabalham no código depois de um período de tempo.

- Comentários obsoletos ou desatualizados: Comentários obsoletos ou desatualizados podem ser confusos e enganosos, e podem ser removidos ou atualizados para garantir que a documentação seja precisa e útil.

- Código com nomes de variáveis e funções ambíguos: Código com nomes de variáveis e funções ambíguos pode ser difícil de entender e manter, e pode ser refatorado para garantir que os nomes sejam claros e precisos.
- Código com classes ou métodos sem documentação: Classe ou métodos sem documentação podem ser difíceis de entender e usar, e pode ser refatorado para incluir comentários e documentação.

Em resumo, a característica de boa documentação é uma parte importante na identificação e correção de maus-cheiros de código, pois ajuda a garantir que o código seja fácil de entender e usar, especialmente para novos desenvolvedores ou para aqueles que trabalham no código depois de um período de tempo.

Uma operação de refatoração comum que pode ajudar a aumentar a documentação do código é a adição de comentários. A adição de comentários é a técnica de incluir informações claras e precisas sobre o código, incluindo comentários no código, documentação externa e outras formas de comunicação. Esse tipo de refatoração ajuda a garantir que o código seja fácil de entender e usar, especialmente para novos desenvolvedores ou para aqueles que trabalham no código depois de um período de tempo.

Por exemplo, imagine que temos o seguinte trecho de código:

```
int result = 0;
for (int i = 0; i < list.size(); i++) {
    result += list.get(i);
}
```

Esse trecho de código não tem comentários, é difícil saber qual é sua função e como ele funciona. Para aumentar a documentação, podemos adicionar comentários, como abaixo:

```
int result = 0;
//iteração para somar todos os elementos da lista
for (int i = 0; i < list.size(); i++) {
    //soma o elemento atual na variavel result
    result += list.get(i);
}
```

Com a adição de comentários, o código ficou mais fácil de entender e usar, especialmente para novos desenvolvedores ou para aqueles que trabalham no código depois de um período de tempo. Essa operação de refatoração é uma das muitas técnicas que podem ser usadas para aumentar a documentação do código e melhorar sua qualidade.

Elegância

A característica de elegância se refere ao conjunto de qualidades que tornam o código atraente e fácil de entender. Ela é frequentemente associada a soluções simples e elegantes para problemas complexos. O código elegante é geralmente bem estruturado, fácil de ler e compreender, e tem um baixo acoplamento e alta coesão. Alguns dos efeitos da elegância no código incluem:

- Estrutura: O código elegante tem uma estrutura clara e consistente, com classes e métodos bem definidos e organizados.
- Claridade: O código elegante é fácil de ler e compreender, e usa nomes de variáveis e funções claros e precisos.
- Coesão: O código elegante tem alta coesão, ou seja, as classes e métodos têm uma única responsabilidade e trabalham juntos de forma coesa.
- Baixo acoplamento: O código elegante tem baixo acoplamento, ou seja, as classes e métodos dependem o mínimo possível uns dos outros, o que facilita a manutenção e a extensão do código.
- Simplicidade: O código elegante é simples e direto, evitando soluções complexas ou desnecessárias.
- Reutilizabilidade: O código elegante é fácil de ser reutilizado, pois está bem estruturado e modularizado.

Em resumo, a característica de elegância é um conjunto de qualidades que tornam o código atraente e fácil de entender. O código elegante é geralmente bem estruturado, fácil de ler e compreender, e tem um baixo acoplamento e alta coesão.

A característica de elegância está diretamente relacionada aos maus-cheiros de código, pois a elegância é uma das principais qualidades que devem ser buscadas na refatoração de código para corrigir maus-cheiros. Alguns dos maus-cheiros de código mais comuns relacionados à elegância são:

- Código complexo: Código complexo é difícil de entender e manter, e pode ser refatorado para torná-lo mais simples e elegante.
- Código confuso: Código confuso é difícil de entender e manter, e pode ser refatorado para torná-lo mais claro e legível.
- Código com classes ou métodos muito longos: Classes ou métodos muito longos são difíceis de entender e manter, e podem ser refatorados para torná-los mais curtos e elegantes.
- Código com classe ou métodos com múltiplas responsabilidades: Classe ou métodos com múltiplas responsabilidades são difíceis de entender e manter, e podem ser refatorados para torná-los mais coesos e elegantes.

Uma operação de refatoração comum que pode ajudar a aumentar a elegância do código é a extração de método. A extração de método é a técnica de dividir um método grande e complexo em múltiplos métodos menores e mais simples, cada um com uma responsabilidade específica. Isso ajuda a garantir que o código tenha uma estrutura clara e consistente, seja fácil de ler e compreender, e tenha uma alta coesão.

Por exemplo, imagine que temos o seguinte trecho de código:

```
public void processData(List<Integer> list) {
    int result = 0;
    for (int i = 0; i < list.size(); i++) {
        int current = list.get(i);
        if (current > 10) {
            result += current;
        } else if (current < -10) {
            result -= current;
        } else {
            result *= current;
        }
    }
    //muitas outras linhas de código
}
```

Esse método é grande e complexo, com várias responsabilidades diferentes (iterar sobre a lista, verificar se o valor é maior ou menor que 10 e atualizar o

resultado) e muitas outras linhas de código. Para torná-lo mais elegante e fácil de entender, podemos usar a técnica de extração de método para dividi-lo em vários métodos menores, como abaixo:

```
public void processData(List<Integer> list) {
    int result = 0;
    for (int i = 0; i < list.size(); i++) {
        int current = list.get(i);
        result = updateResult(current, result);
    }
    //muitas outras linhas de código
}

private int updateResult(int current, int result) {
    if (current > 10) {
        return result + current;
    } else if (current < -10) {
        return result - current;
    } else {
        return result * current;
    }
}
```

Com a extração de método, o código ficou mais fácil de entender e manter, pois cada método tem uma responsabilidade específica e é mais curto. Essa operação de refatoração é uma das muitas técnicas que podem ser usadas para aumentar a elegância do código e melhorar sua qualidade.

Boas Interfaces

A característica de boas interfaces se refere ao conjunto de qualidades que tornam as interfaces de um sistema fáceis de usar e entender. Boas interfaces são geralmente intuitivas, fáceis de aprender e usar, e fornece aos usuários a capacidade de realizar tarefas de forma eficiente. Alguns dos efeitos de boas interfaces no código incluem:

- Consistência: Boas interfaces são consistentes, ou seja, usam os mesmos elementos de interface e convenções para realizar tarefas similares, o que facilita a aprendizagem e o uso.

- Simplicidade: Boas interfaces são simples e diretas, evitando elementos desnecessários ou confusos.
- Feedback: Boas interfaces fornecem feedback claro e imediato para as ações dos usuários, o que ajuda a garantir que as tarefas estejam sendo realizadas corretamente.

Alguns dos maus-cheiros de código mais comuns relacionados às boas interfaces são:

- Classes God: Classes God são classes que possuem muitas responsabilidades e muitos métodos, tornando-as difíceis de entender e manter. Essas classes podem ser refatoradas para torná-las mais coesas e ter interfaces mais limadas
- Métodos longos : Métodos longos são difíceis de entender e manter, e podem ser refatorados para torná-los mais curtos e elegantes, e também para serem mais claros e precisos em suas interfaces.

Uma operação de refatoração comum que pode ajudar a melhorar a qualidade das interfaces é a extração de interface. A extração de interface é a técnica de criar uma nova interface a partir de uma classe existente, que encapsula os métodos e propriedades que são importantes para as outras classes. Isso ajuda a garantir que as interfaces sejam claras, precisas e fáceis de usar.

Por exemplo, imagine que temos uma classe Car que possui métodos para iniciar e parar o motor, mudar as marchas, acelerar e frear. Essa classe pode ser refatorada para ter uma interface Driveable que contém somente os métodos que são importantes para outras classes usarem.

```
class Car {
    public void startEngine() { ... }
    public void stopEngine() { ... }
    public void changeGear(int gear) { ... }
    public void accelerate(int speed) { ... }
    public void brake(int speed) { ... }
}

interface Driveable {
    public void startEngine();
    public void stopEngine();
    public void accelerate(int speed);
    public void brake(int speed);
}
```

Com a extração de interface, as outras classes podem usar apenas os métodos que são importantes para elas, sem precisar conhecer todos os detalhes da implementação da classe Car. Isso torna as interfaces mais fáceis de usar e entender. A extração de interface é uma das muitas técnicas que podem ser usadas para melhorar a qualidade das interfaces e aumentar a elegância do código.