**Análise e Justificativa para a Refatoração do Jabberpoint**

**Aluno: Mateus Valença**

**Introdução**

Este relatório apresenta uma análise aprofundada do sistema do Jabberpoint, com o objetivo de identificar problemas de qualidade de código e justificar a necessidade de refatoração.

**Descrição do Sistema**

O Jabberpoint é um software de apresentações em slides composto por diversas classes, como a classe Slide, que representa um slide com seus respectivos atributos e métodos, e a classe MenuController, responsável pelos controles do menu na tela do usuário, como “Exit”, “Help” e “Open”.

**Estudo das Métricas**

As seguintes métricas foram utilizadas como forma de identificar possíveis problemas do código. A seguir, apresentamos as métricas selecionadas e uma descrição de cada uma delas:

**Métrica: Line Length (Comprimento da Linha)**

Esta métrica avalia o comprimento das linhas de código em caracteres. O objetivo é evitar linhas muito longas que dificultam a leitura e compreensão do código.

**Métrica: MissingJavadocMethod (Método Javadoc Faltando)**

Esta métrica avalia se um método não possui um comentário Javadoc.

**Métrica: FinalParameters (Parâmetro Final)**

Esta métrica verifica se os parâmetros para métodos, construtores, blocos catch e for-each são finais.

**Métrica:** [**ModifierOrder**](https://checkstyle.sourceforge.io/checks/modifier/modifierorder.html#ModifierOrder)**(Ordem do modificador)**

Esta métrica verifica se a ordem dos modificadores está de acordo com as sugestões na especificação da linguagem Java, § 8.1.1, 8.3.1, 8.4.3 e 9.4.

**Métrica: MissingPackageInfo(Falta de informações do pacote)**

Essa métrica verifica se há um arquivo chamado "package-info.java" em cada pacote do seu código-fonte.

**Métrica: TrailingWhitespace(Espaço em branco à direita)**

Esta métrica verifica se há espaços em branco adicionais no final de uma linha de código.

**Métrica: MissingCtorJavadoc(Falta de comentário Javadoc)**

Essa métrica verifica se há um comentário JavaDoc ausente ou inadequado no construtor de uma classe.

**Métrica: UtilityClassWithPublicConstructor(Classe utilitária com construtor público)**

Esta métrica verifica se uma classe utilitária possui um construtor público ou com visibilidade de pacote.

**Análise dos Problemas Identificados**

Com base nas métricas escolhidas, foram identificadas os seguintes problemas:

1. Métrica: **Line Length (Comprimento da Linha)**

Problema: Linhas de código excedem 80 caracteres e isto pode prejudicar a manutenção.

Exemplo: A linha 84 da classe “TextItem” possui 104 caracteres.

1. Métrica: **MissingJavadocMethod (Método Javadoc Faltando)**

Problema: A ausência do método Javadoc dando a descrição da funcionalidade dos métodos.

Exemplo: Na linha 17, da classe “Style” falta um comentário Javadoc explicando a funcionalidade do atributo.

1. Métrica:  **FinalParameters (Parâmetro Final)**

Problema: Parâmetros de construtores não são finais.

Exemplo: O parâmetro “scale” na linha 54, da classe “Style” deve ser final.

1. Métrica: [**ModifierOrder**](https://checkstyle.sourceforge.io/checks/modifier/modifierorder.html#ModifierOrder)**(Ordem do modificador)**

Problema: Modificadores fora da ordem sugerida pela JLS.

Exemplo: O modificador “static” na linha 24, da classe “SlideViewerFrame” está fora da ordem sugerida pela JLS.

1. Métrica: **MissingPackageInfo(Falta de informações do pacote)**

Problema: Código-fonte de classes sem o arquivo "package-info.java" no pacote.

Exemplo: Está faltando o "package-info.java" no código-fonte da classe “Presentation”

1. Métrica: **TrailingWhitespace(Espaço em branco à direita)**

Problema: Linhas no código-fonte com espaços em branco à direita.

Exemplo: A linha 4, da classe “BitmapItem” está com espaços em branco à direita.

1. Métrica: **MissingCtorJavadoc(Falta de comentário Javadoc)**

Problema: Faltando comentário Javadoc no construtor de uma classe

Exemplo: O método “keyPressed” da classe “KeyController” não existe um documento Javadoc.

1. Métrica: **UtilityClassWithPublicConstructor(Classe utilitária com construtor público)**

Problema: Classes utilitárias com construtores públicos ou com problema de visibilidade de pacotes.

Exemplo: A classe utilitária “JabberPointApplication” possui construtor público.

**Com base nas classes, podemos observar os seguintes problemas:**

1. Classe “JabberPointApplication”:

* TrailingWhitespace (Espaço em branco à direita): A classe possui linhas com espaços em branco à direita.

Exemplo: A linha 4 desta classe possui espaços em branco.

* UtilityClassWithPublicConstructor(Classe utilitária com construtor público): Esta classe é utilitária e possui construtor público.
* MissingJavadocMethod (Método Javadoc Faltando): Não possui comentário Javadoc no seu método “main”.
* Line Length (Comprimento da Linha): A linha 35 desta classe possui 118 caracteres.
* FinalParameters (Parâmetro Final): O parâmetro “args” na linha 17 desta classe deveria ser final.

1. Classe “KeyController”:

* TrailingWhitespace (Espaço em branco à direita): A classe possui linhas com espaços em branco à direita.

Exemplo: A linha 7 desta classe possui espaços em branco.

* MissingJavadocMethod (Método Javadoc Faltando): Faltam comentários Javadoc em métodos desta classe.

Exemplo: O método “Presentation” não possui comentários Javadoc

* FinalParameters (Parâmetro Final): Alguns parâmetros desta classe deveriam ser finais.

Exemplo: O parêmetro “keyEvent” na linha 23 da classe deveria ser final.

* MissingCtorJavadoc(Falta de comentário Javadoc): O método “keyPressed” da classe “KeyController” não existe um documento Javadoc.

**Plano de Refatoração**

Com base nos problemas identificados na análise das métricas de código do Jabberpoint, propomos o seguinte plano de ação para a refatoração:

**Análise e compreensão do sistema:**

Realizar uma revisão completa do código-fonte para compreender a estrutura, funcionalidades e dependências do sistema.

Identificar os principais componentes dessas classes, como métodos e atributos, e entender suas responsabilidades e interações.

**Definição de metas e objetivos:**

Estabelecer metas claras para a refatoração, como em todo método colocar um comentário Javadoc descrevendo a funcionalidade do método, definir um número de caracteres por linha, 80 como sugere a JLS e definir os critérios de sucesso para a refatoração, como métricas específicas que devem ser alcançadas.

**Refatoração do código:**

Identificar os pontos críticos do sistema que apresentam problemas, como longas linhas de código, ausência de descrição de certos métodos, inadequações com as convenções de boas práticas da JLS ou atribuições e declarações errôneas de métodos e variáveis

Exemplo: Não deixar linhas em branco à direita no código-fonte.

Exemplo: Deixar o modificador “static” na linha 24, da classe “SlideViewerFrame” na ordem sugerida pela JLS.

Exemplo: Definir o parâmetro “scale” na linha 54, da classe “Style” como final.

**Testes unitários:**

Criar testes unitários abrangentes para as principais funcionalidades do Jabberpoint.

Garantir que os testes cubram diferentes cenários e casos de uso.

Exemplo: O método saveFile() pode ser testado, para garantir que realmente está salvando o arquivo do usuário e se não ocorre um possível corrompimento do conteúdo do arquivo.

**Revisão de código:**

Realizar revisões de código entre os membros da equipe para garantir a qualidade das alterações realizadas.

Identificar possíveis melhorias adicionais e sugerir ajustes ou otimizações no código.

Exemplo: Revisar a classe “MenuController” pois ela tem muitos métodos e funcionalidades e o plugin “Checkstyle” identifica 75 erros que se corrigidos a classe estaria ótima para possíveis modificações.

**Integração contínua:**

Configurar um sistema de integração contínua para automatizar a compilação, testes e análise de código do projeto.

Utilizar ferramentas de análise estática, como Checkstyle e PMD, para identificar problemas de código em tempo real e fornecer feedback imediato aos desenvolvedores.

Exemplo: Utilizar os plugins Checkstyle para verificar se o código-fonte da projeto possui comentários Javadoc.

**Validação e monitoramento:**

Realizar testes de integração e validação do sistema refatorado para garantir que todas as funcionalidades estejam funcionando corretamente.

Monitorar o desempenho e a estabilidade do sistema após a refatoração, identificando possíveis problemas e corrigindo-os prontamente.

Exemplo: Realizar testes de integração após a refatoração da classe “BitmapItem” para identificar uma provável melhora no desempenho da classe.

**Conclusões**

A análise do projeto Jabberpoint identificou inúmeros problemas na qualidade da escrita do código e violações de boas práticas na programação. Esses problemas irão causar uma provável dificuldade na manutenibilidade e escalabilidade do código.

Com a implementação do plano de refatoração proposto, espera-se melhorar bastante a qualidade do código e da sua legibilidade, tornando assim o código bem mais legível e com uma melhor reutilização deste código, facilitando futuras alterações.