Adoção da Ferramenta Checkstyle Para Melhoria da Qualidade do *manejo.app*

Adoption of the Checkstyle Tool to Improving manejo.app Quality

Keila Emy Taniguchi*, Gabriel Costa Silva†,

RESUMO

O Manejo Integrado de Pragas (MIP) é uma técnica para o controle de pragas, por meio do monitoramento constante da população de pragas em uma lavoura. O MIP permite a adequação da dosagem de inseticidas, o que possibilita a redução de custos associados ao controle de pragas. Para apoiar o Instituto de Desenvolvimento Rural do Paraná (IDR) e o Serviço Nacional de Aprendizagem Rural (SENAR) na aplicação do MIP junto a sojicultores Paranaenses, foi desenvolvido um aplicativo Web para coleta e análise de dados do MIP (*manejo.app*). Visando a melhoria da qualidade por meio da padronização do código do *manejo.app*, a ferramenta *Checkstyle* foi usada. A Checkstyle é capaz de verificar e indicar automaticamente trechos de código que violam boas práticas de desenvolvimento. A adoção da *Checkstyle* foi realizada em etapas. Cada etapa corresponde a um módulo do *manejo.app*. Os resultados obtidos mostraram que o código do *manejo.app* ficou mais claro de entender, melhorando consideravelmente a manutenabilidade da aplicação.

Palavras-chave: Manejo Integrado de Pragas da Soja, manejo.app, Checkstyle.

ABSTRACT

The Integrated Pest Management (IPM) consists in an pest control technique, based on constant pest population monitoring in the agriculture. The IPM allows the adequacy of insecticide dosage makes it possible to reduce costs associated with pest control. Thus, to support the Rural Development Institute of Parana (IDR from the Portuguese) and Rural Learning National Service (SENAR from the Portuguese) in IPM application with soy farmers in Parana state, it was developed a web application called *manejo.app* for IPM data acquisition and analysis. In order to improve quality by standardization of the applications' code, the *Checkstyle* tool has been used. The *Checkstyle* is able to verify and automatically point coding parts that violate good development pratices. The *Checkstyle* adoption has been applied in steps. Each step is related to one individual *manejo.app* application module. The results have shown that the *manejo.app* coding became clearer for undesrtanding, improving its application maintainability.

Keywords: Integrated Soy Pest Management, application, Checkstyle.

1 INTRODUÇÃO

O Manejo Integrado de Pragas da Soja (MIP-Soja) consiste no conjunto de medidas destinadas ao controle de pragas por meio do monitoramento de sua população. Este conjunto de medidas associa o controle de pragas à utilização adequada de defensivos agrícolas. Desta forma, garante um melhor aproveitamento dos inseticidas empregados, evitando seu uso em excesso (CONTE et al., 2020).

A dosagem adequada de inseticidas pode reduzir os custos associados ao controle de pragas. A redução de custos acontece por meio do aumento do tempo médio de aplicação dos defensivos agrícolas. Por consequência, reduz a frequência de aplicação necessária quando comparado às estratégias de controle de pragas que não é aplicado o manejo integrado.

* Engenharia de Computação, Universidade Tecnológica Federal do Paraná, Cornélio Procópio, Paraná, Brasil; keilaemy@gmail.com
† Universidade Tecnológica Federal do Paraná, Campus Cornélio Procópio; gabrielcosta@utfpr.edu.br

Para apoiar o IDR e SENAR no monitoramento de pragas junto aos sojicultores Paranaenses, foi desenvolvido o *manejo.app*. Usado por sojicultores Paranaenses desde 2019, o *manejo.app* aumenta a eficiência e eficácia na coleta e análise dos dados do MIP nas lavouras que adotam o MIP-Soja(DORIGATTI, 2020).

Como o *manejo.app* passa por modificações constantes, a manutenabilidade da aplicação ficou comprometida. Com o intuito de melhorar a qualidade no aplicativo, manutenabilidade, em particular, a ferramenta *Checkstyle* foi utilizada para ajustar a padronização do código-fonte da aplicação. A *Checkstyle* verifica se o código está em concordância com regras de codificação baseadas nas boas práticas de programação. Essas regras visam melhorar a qualidade, facilitando a manutenção do aplicativo, por meio de melhor legibilidade, entre outros.

Um subconjunto de pacotes do *manejo.app* foi usado como referência para a adoção da *Checkstyle*. Um pacote é um conjunto de classes, que representam objetos no mundo real. Cada pacote pode representar um módulo da aplicação. Os pacotes alvo deste projeto são: `entity`, `repository`, `service` e `view`.

2 MÉTODO

A *Checkstyle* é uma ferramenta de código fonte aberto que verifica automaticamente o código para que garantir sua padronização (CHECKSTYLE, 2021). A *Checkstyle* usa um conjunto de regras que são aplicadas ao código analisado. As regras são categorizadas em diferentes grupos, como pode ser observado na Figura 2. Neste projeto, apenas as categorias consideradas mais relevantes para o *manejo.app* foram consideradas.Em especial, as seguintes regras foram alvo de análise no *manejo.app*:

- *UnusedImports* (Importações não utilizadas/ categoria imports): Esta regra define que importações não utilizadas devem ser removidas.
- *AvoidStarImport* (Evite Estrelas na Importação/ categoria imports): Esta regra define que importações de pacote devem ser substituídas por importações de classes individuais. Assim, apenas as classes utilizadas efetivamente no código são importadas.
- *NeedBraces* (Precisa de Chaves/ categoria blocks): Esta regra define que em blocos condicionais e laços de repetição é necessário adicionar chaves para indicar seu início e fim.
- *NewlineAtEndOfFile* (Nova Linha no Final do Arquivo/ categoria misc): Esta regra determina que o código deve finalizar com uma linha vazia para que quando adicionadas novas informações no arquivo este não mostre linhas anteriores como alteradas.
- *LineLength* (Comprimento de Linha/ categoria size): Esta regra determina que as linhas de código devem ser limitadas a 80 caracteres, no máximo. .
- *FileTabCharacter* (Caractere de Tabulação de Arquivo)/ categoria whitespace): Esta regra determina que tabulações devem ser removidas e substituídas por espaços.
- *GenericWhitespace* (Espaço em Branco Genérico/ categoria whitespace): Esta regra determina que espaços em torno de “<” e ”>” devem ser removidos.
- *MethodParamPad* (Método Parâmetro de Preenchimento/categoria whitespace): Esta regra estabelece que espaços em torno de “(” e ”)” devem ser removidos em métodos.
- *WhitespaceAround* (Espaço em Branco ao Redor/categoria whitespace): Esta regra determina que não deve haver espaços em branco em torno de um token. Um token é um conjunto de palavras de uma linguagem. No caso a linguagem de programação Java alguns exemplos são os comandos for, final, throws, entre outros.

• *WhitespaceAfter* (Espaço em Branco Depois/ categoria whitespace): Esta regra determina que não deve haver espaços em branco após um token, exceto seguido de um ponto e vírgula.

As verificações foram realizadas em etapas, de forma de que cada pacote corresponde a uma etapa. As etapas foram executadas na seguinte ordem: `entity`, `repository`, `service`, e `view`. Essa ordem segue uma ordem de complexidade, do pacote menos complexo para o mais complexo. Para cada pacote, as classes que os compõem foram verificadas e individualmente corrigidas de acordo com a padronização estabelecida pela *Checkstyle*. A Seção seguinte detalha o processo de verificação e correção.

3 RESULTADOS

A inclusão da ferramenta *Checkstyle* no arquivo `pom.xml` foi o primeiro passo do projeto. O arquivo `pom.xml` faz parte dos recursos Maven.

A Figura 1 mostra a adição da *Checkstyle* ao projeto. As linhas 158 a 162 definem o plugin responsável por mostrar o relatório da verificação realizada pela ferramenta. Nas linhas 163 a 167 é adicionado o plugin responsável por gerar o relatório das verificações. Na linha 173 a 184 apresenta a adição da ferramenta como parte dos relatórios do projeto.

Figura 1 – Inclusão da Ferramenta Checkstyle ao arquivo pom.xml.

```
158          <plugin>
159              <groupId> org.apache.maven.plugins </groupId>
160              <artifactId> maven-site-plugin</artifactId>
161              <version>3.7.1</version>
162          </plugin>
163          <plugin>
164              <groupId> org.apache.maven.plugins </groupId>
165            <artifactId> maven-project-info-reports-plugin</artifactId>
166           <version>3.0.0</version>
167          </plugin>
168        </plugins>
169      </build>
170
171    <reporting>
172      <plugins>
173        <plugin>
174          <groupId> org.apache.maven.plugins </groupId>
175          <artifactId> maven-checkstyle-plugin</artifactId>
176          <version>3.1.1</version>
177          <reportSets>
178            <reportSet>
179              <reports>
180                <report> checkstyle </report>
181              </reports>
182            </reportSet>
183          </reportSets>
184        </plugin>
185      </plugins>
186    </reporting>
```

Fonte: Autoria Própria (2021).

Após a adição da ferramenta ao projeto, foi gerado o relatório para identificar o estado dos pacotes que compõem a aplicação. A Figura 2 mostra a análise gerada pela ferramenta para o pacote `entity`. Na tabela mostrada na Figura 2, a primeira coluna apresenta a categoria da regra analisada pela *Checkstyle*. A segunda coluna representa a regra a ser seguida. A terceira coluna mostra a quantidade de violações identificadas no código analisado. Por fim, a última coluna mostra a severidade das violações encontradas.

Figura 2 – Análise do pacote entity realizado pela ferramenta.

Category	Rule	Violations	Severity
blocks	NeedBraces	39	Error
coding	EmptyStatement	1	Error
	HiddenField	31	Error
	MagicNumber	41	Error
	SimplifyBooleanExpression	3	Error
	SimplifyBooleanReturn	7	Error
design	DesignForExtension	181	Error
	VisibilityModifier	17	Error
imports	AvoidStarImport	2	Error
	UnusedImports • processJavadoc: "false"	6	Error
javadoc	JavadocPackage	6	Error
	JavadocVariable	250	Error
	MissingJavadocMethod	202	Error
misc	FinalParameters	174	Error
	NewlineAtEndOfFile	3	Error
	TodoComment	4	Error
regexp	RegexpSingleline • format: "\s+$" • maximum: "0" • message: "Line has trailing spaces." • minimum: "0"	207	Error
sizes	LineLength • fileExtensions: "java"	166	Error
	ParameterNumber	2	Error
whitespace	MethodParamPad	34	Error
	WhitespaceAround	4	Error

Fonte: Autoria Própria (2021).

A Tabela 1 resume as violações encontradas nos pacotes analisados neste projeto. Sendo a primeira coluna apresenta as categorias que foram identificadas pela *Checkstyle*. A segunda coluna apresenta a regra a ser seguida para que o código seja padronizado. A terceira coluna mostra a quantidade de violações encontradas no projeto. A quarta coluna mostra a quantidade de violações após a conclusão das correções no projeto. Por fim a quinta coluna mostra a quantidade de correções realizadas no projeto.

Tabela 1 – Antes e depois da análise da ferramenta

Categoria	Regra	Violações Antes	Violações Depois	Correções Realizadas
Blocks	NeedBraces	42	0	42
Imports	AvoidStarImport	3	0	3
Imports	UnusedImports	30	2	28
Misc	NewlineAtEndOfFile	3	0	3
Sizes	LineLength	1067	75	992
Whitespace	FileTabCharacter	19	2	17
Whitespace	GenericWhitespace	7	0	7
Whitespace	MethodParamPad	38	0	38
Whitespace	WhitespaceAfter	1	0	1
Whitespace	WhitespaceAround	23	0	23

Fonte: Autoria Própria(2021).

A Figura 3 mostra um trecho do código da classe `AbstractController`. A parte superior da Figura mostra a linha 61, que viola a regra *LineLength*. Já a parte inferior da Figura mostra o mesmo trecho corrigido.

Figura 3 – Inclusão da Ferramenta Checkstyle ao arquivo pom.xml

```
61          StringBuilder returnURL = new StringBuilder("/dashboard/ur-dashboard.xhtml?");

61          StringBuilder returnURL =
62              new StringBuilder("/dashboard/ur-dashboard.xhtml?");
```

Fonte: Autoria Própria (2021).

A Figura 4 exemplifica uma correção realizada para a regra *NeedBraces*. Nesta observa-se um trecho da classe `Field` que viola tal regra. Na parte superior, a linha 68 mostra uma condicional (`if`) que não usa chaves. Para sua correção foi incluído o uso de chaves nas linhas 68 e 70, como mostra a parte inferior da Figura 4.

Figura 4 – Parcela Superior: Código antigo, Parcela Inferior: Código melhorado.

```
68          if (noSupervisorContainerCreated())
69              createSupervisorContainer();

68          if (noSupervisorContainerCreated()) {
69              createSupervisorContainer();
70          }
```

Fonte: Autoria Própria (2021).

A Figura 5 mostra um trecho do código da classe `SurveyService`. A parte superior da Figura mostra um trecho de código que viola a regra *AvoidStarImport*. Na parte superior, a linha 7 importa o pacote `base`, composto por sete classes. A parte inferior da Figura mostra o mesmo trecho de código após a correção. Observe que apenas uma classe foi importada. Dessa forma, as outras seis classes que estavam sendo importadas no pacote eram desnecessárias e assim enquadradas na regra *UnusedImports*.

Figura 5 – Parcela Superior: Código antigo, Parcela Inferior: Código melhorado.

```
7   import br.edu.utfpr.cp.emater.midmipsystem.service.base.*;

7   import br.edu.utfpr.cp.emater.midmipsystem.service.base.FieldService;
```

Fonte: Autoria Própria (2021).

A Figura 6 mostra um trecho da classe `FarmerController` e a violação da regra *WhitespaceAround*. Na parte superior da Figura, linha 72 esta regra é descumprida. Sua correção é mostrada na parte inferior da Figura.

Figura 6 – Parcela Superior: Código antigo, Parcela Inferior: Código melhorado.

```
72      protected String getItemName(){

72      protected String getItemName() {
```

Fonte: Autoria Própria (2021).

A Figura 7 exemplifica a violação da regra *MethodParamPad*. O trecho de código pertence a classe `RegionController`. Na parte superior da Figura 7, a linha 52 mostra um espaço que antecede o "(". Como a regra define, tal espaço deve ser removido, como mostra a parte inferior da Figura.

Figura 7 – Parcela Superior: Código antigo, Parcela Inferior: Código melhorado.

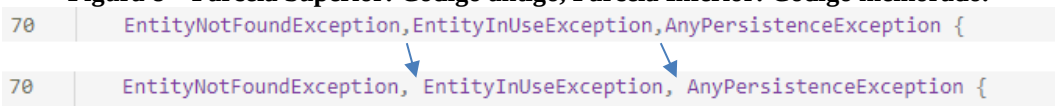
```
52      protected void doPrepareUpdate (Long anId) throws EntityNotFoundException {

52      protected void doPrepareUpdate(Long anId) throws EntityNotFoundException {
```

Fonte: Autoria Própria (2021).

A Figura 8 apresenta um trecho de código da classe `BladeReadingResponsibleEntityController`. A Figura mostra a violação da regra *WhitespaceAfter*. Na parte superior da Figura 8, a linha 70 mostra que após a virgula não há um espaçamento. A correção é mostrada na parte inferior da Figura, incluindo o espaçamento.

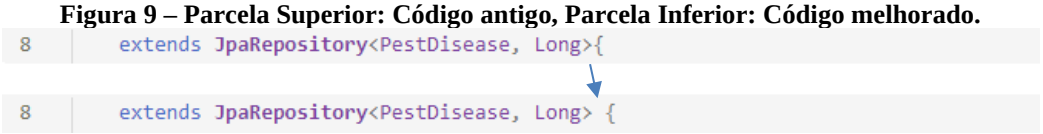
Figura 8 – Parcela Superior: Código antigo, Parcela Inferior: Código melhorado.



Fonte: Autoria Própria (2021).

A Figura 9 apresenta um trecho de código da classe `PestNaturalPredatorRepository`. A Figura mostra a correção referente a regra *GenericWhitespace*. Na parte superior da Figura 9, a linha 8 mostra que não há um espaço em branco após “>”. A parte inferior da Figura 9 mostra o trecho de código após a correção.

Figura 9 – Parcela Superior: Código antigo, Parcela Inferior: Código melhorado.



Fonte: Autoria Própria (2021).

4 CONCLUSÃO

A ferramenta *Checkstyle* se mostrou eficaz na melhoria da manutenabilidade do *manejo.app*. Dentre as dificuldades encontradas durante a execução do projeto, pode se citar que a utilização da ferramenta *Checkstyle* não apresentou funcionamento em conformidade com as instruções contidas em sua página. Deste modo, foi necessário a busca por soluções em páginas onde o tema fora discutido. Encontrada a solução do problema, as verificações da ferramenta foram executadas no projeto para todos os pacotes. Sendo assim, 1.156 melhorias foram adotadas no código do *manejo.app*.

Após a análise do projeto com suas devidas melhorias, foi possível notar a importância das boas práticas de desenvolvimento. Desta forma, tornando o código mais fluido e de fácil compreensão e, por fim, estabelecendo um padrão para as classes de um projeto.

REFERÊNCIAS

CHECKSTYLE, 2021. Site oficial da Ferramenta **Checkstyle**. Disponível em: https://checkstyle.sourceforge.io/. Acesso em: 20/06/2021.

CONTE, O.et al. **Resultados do manejo integrado de pragas da soja na safra 2019/2020 no Paraná**. Londrina: Embrapa Soja, 2020.

DORIGATTI, G. **Novo Software auxilia produtores do Paraná no manejo integrado de Pragas e Doenças**. Notícias Agrícolas. 26 mai. 2020. Disponível em: https://www.noticiasagricolas.com.br/. Acesso em: 29/07/2021.