3
Trabalhos Relacionados

Será feita uma breve apresentação e uma comparação entre os sistemas de críticas que serão usados como inspiração para a realização deste trabalho, esses são: JDT (JDT 2002-9) (Eclipse Java Development Tools), PMD (PMD 2002-9), Checkstyle (CheckStyle 2002-9), FindBugs (FindBugs 2002-9) e o NCL-Validator (Araújo et al. 2008). Foram considerados os seguintes critérios relacionados à usabilidade do sistema e adaptabilidade das regras:

1. Integração com o ambiente de desenvolvimento: o sistema fornece críticas em tempo real durante a edição do documento.

2. Permissão para modificação e especificação de novas regras: o sistema fornece mecanismos para especificação de novas regras ou modificação das regras pré-estabelecidas.

3. Permissão para edição e especificação de novas regras através do ambiente de desenvolvimento.

4. Fornecimento de uma linguagem declarativa para especificação das regras.

5. Sugestão de correções para os problemas.

6. Correção dos problemas de forma automática.

Ao final do capítulo é apresentado um quadro comparativo entre as ferramentas avaliadas, de acordo com os critérios apresentados anteriormente.

3.1 Eclipse Java Development Tools (JDT)

De fato, o JDT não é um sistema de críticas. Ele é apenas um *plugin* do Eclipse que fornece suporte ao desenvolvimento Java. As críticas feitas sobre o código são simplesmente mensagens do compilador apresentadas de forma gráfica e amigável. Porém, achamos relevante relacioná-lo nesta seção, pois o JDT serve como infraestrutura para os sistemas Checkstyle e PMD. Além disso, sua interface possui elementos que podem servir como inspiração para a elaboração do nosso sistema de críticas para NCL.

O JDT adiciona uma perspectiva (Figura 3.1) ao ambiente Eclipse que fornece suporte ao desenvolvimento Java, portanto ele já é o próprio ambiente de desenvolvimento. Como as mensagens de erro ou críticas advêm do compilador, não é possível fazer qualquer alteração nas regras.

Figura 3.1: Perspectiva adicionada pelo JDT

Uma funcionalidade bastante interessante fornecida pelo JDT é que além de apresentar as críticas provenientes do compilador, são mostradas sugestões de correções automáticas. No exemplo da Figura 3.2, é mostrado um aviso informando que a variável `helloWorld` foi declarada, mas não está sendo utilizada, e três sugestões de correção. As sugestões são exibidas em forma de três *hyperlinks*, que ao serem clicados, executam imediatamente a correção informada.

3.2 PMD

O PMD é uma ferramenta de verificação estática de código Java que procura por problemas potenciais. Esses problemas são apresentados para o usuário através de uma crítica dentro do próprio ambiente de desenvolvimento. O PMD possui integração com diversos outros aplicativos, como: JDeveloper, Eclipse, JEdit, JBuilder, BlueJ, CodeGuide, NetBeans/Sun Java Studio Enterprise/Creator, IntelliJ IDEA, TextPad, Maven, Ant, Gel, JCreator e Emacs.

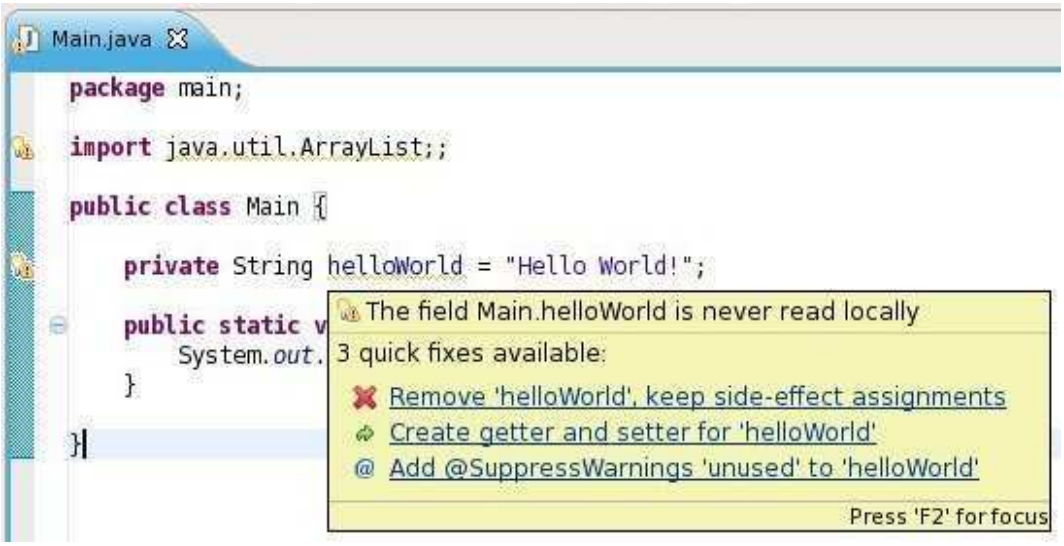

Figura 3.2: Crítica proveniente do compilador, informando que uma variável declarada não está sendo usada.

O PMD também permite especificação de novas regras e alteração das regras existentes dentro do próprio ambiente de desenvolvimento. Existem duas formas de se especificar as regras: usando Java ou declarativamente através do XPath. Além disso, ele ainda possui um ambiente simples de desenvolvimento de regras (Figura 3.3), que facilita a criação de regras usando XPath.

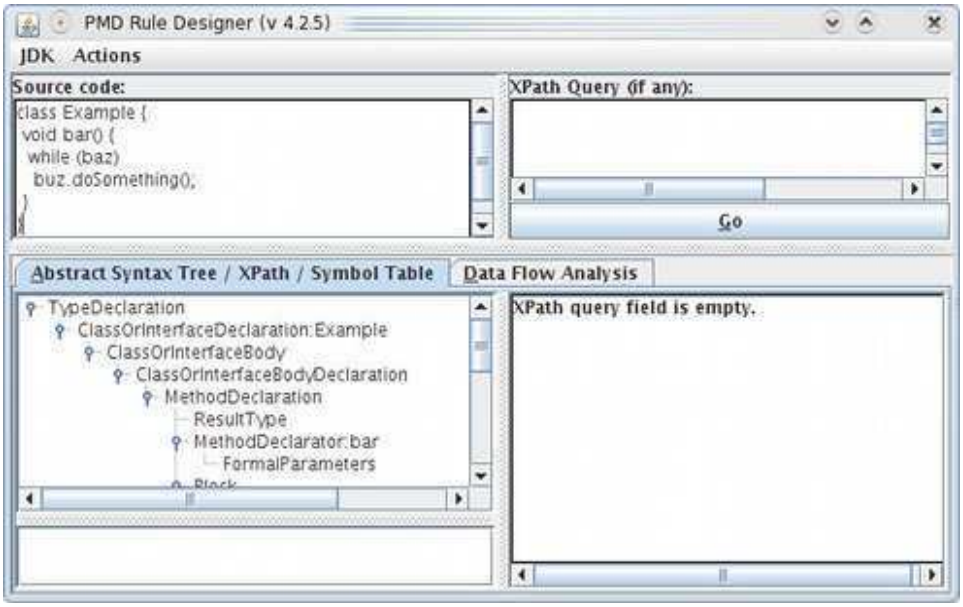

Figura 3.3: Ambiente de desenvolvimento de regras XPath do PMD

Na Figura 3.4 é mostrado um exemplo de sinalização de um erro em uma classe Java. Essa classe deseja sobrescrever o método `clone()` mas viola o contrato (ObjectAsASuperclass) estabelecido para implementação de métodos

`clone()`, pois não implementa a interface `java.lang.Cloneable` e não chama o método `clone()` de `java.lang.Object`. Observando a figura, é possível perceber que as críticas são integradas no ambiente e que são apresentadas sugestões para correção dos erros, mas nenhuma correção automática é fornecida.

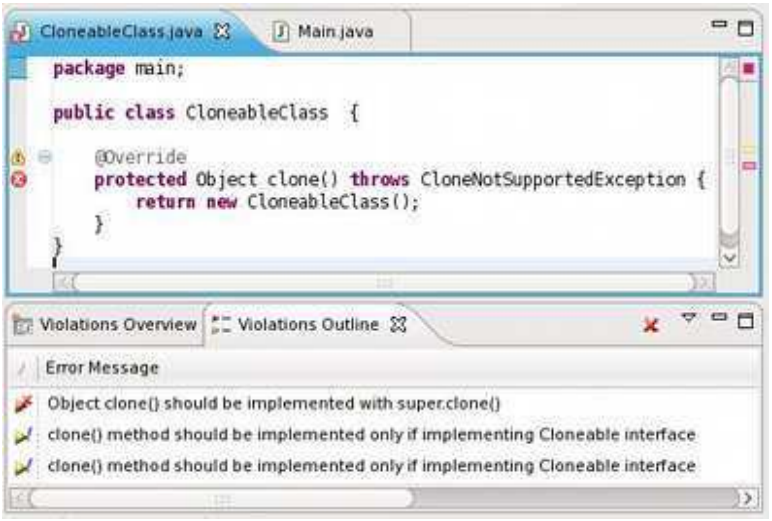

Figura 3.4: Exemplo de sinalização de erros do PMD

3.3 CheckStyle

O CheckStyle é muito similar ao PMD em vários aspectos. Possui integração a diversas IDEs e ferramentas de *build* tal qual: Eclipse, WSAD, IntelliJ, NetBeans, BlueJ, iIDE, Emacs IDE, jEdit, Vim, Krysalis Centipede, Maven, Sonar, QALab e Borland JBuilder. A Figura 3.5 mostra o CheckStyle em uso. Nota-se que ele critica não apenas potenciais erros, mas também questões de estilo, como, por exemplo, o não uso do Javadoc.

Como diferenças positivas relativas ao PMD, podemos citar que o CheckStyle permite que alguns dos problemas encontrados sejam corrigidos de forma automática (Figura 3.6). As diferenças negativas são que o CheckStyle não permite a criação de regras por meio de uma linguagem declarativa e integrada ao ambiente de desenvolvimento. A especificação de novas regras é somente feita por meio da linguagem Java através de Visitors (Gamma et al. 1995). Felizmente, o CheckStyle possui um ambiente de desenvolvimento de regras que facilita a criação e os testes das novas regras criadas. A Figura 3.7 mostra uma Árvore Sintática Abstrata gerada pelo ambiente.

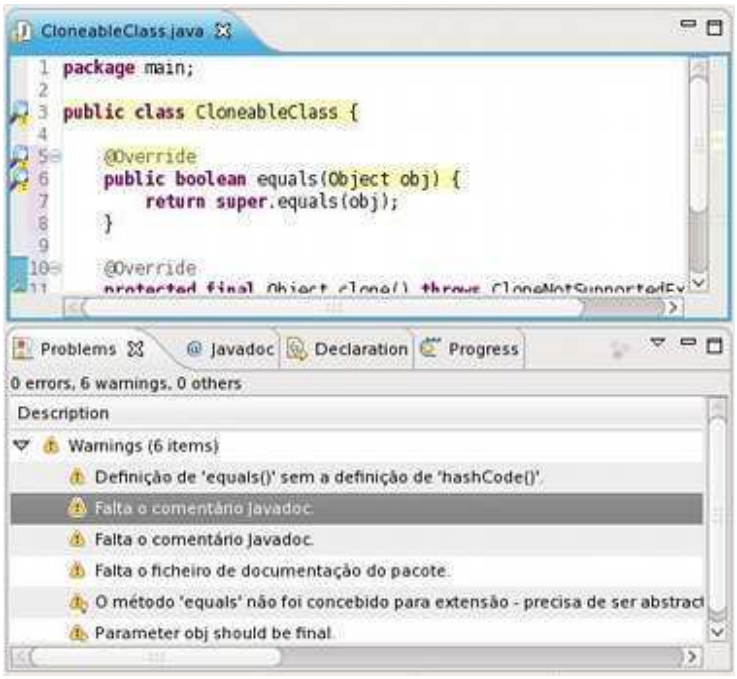

Figura 3.5: Exemplo do CheckStyle em uso

Figura 3.6: Correção automática do CheckStyle

3.4 FindBugs

FindBugs é um projeto que possui suporte de diversas instituições como a Universidade de Maryland, Sun e Google. Assim como os outros, é um verificador estático de código, porém não analisa questões de estilo. Possui uma interface gráfica escrita em Swing e uma outra integrada ao ambiente de desenvolvimento Eclipse.

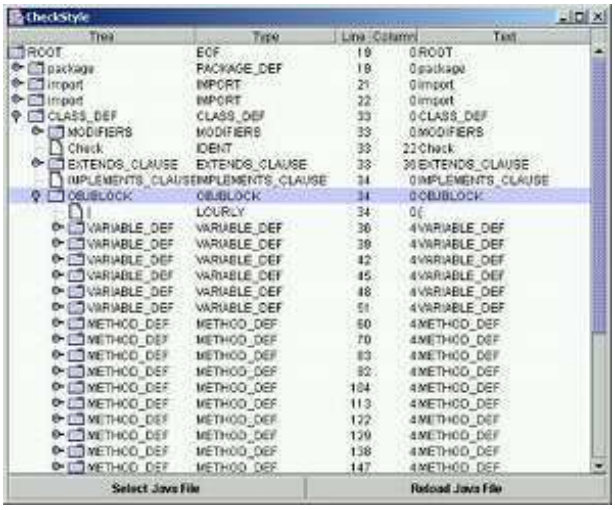

Figura 3.7: Ambiente de criação de regras do CheckStyle

A sinalização de possíveis erros é mostrada na Figura 3.8. Cada erro encontrado possui uma informação detalhada sobre o ocorrido e como repará-lo, porém nenhuma forma de reparo automático é oferecido.

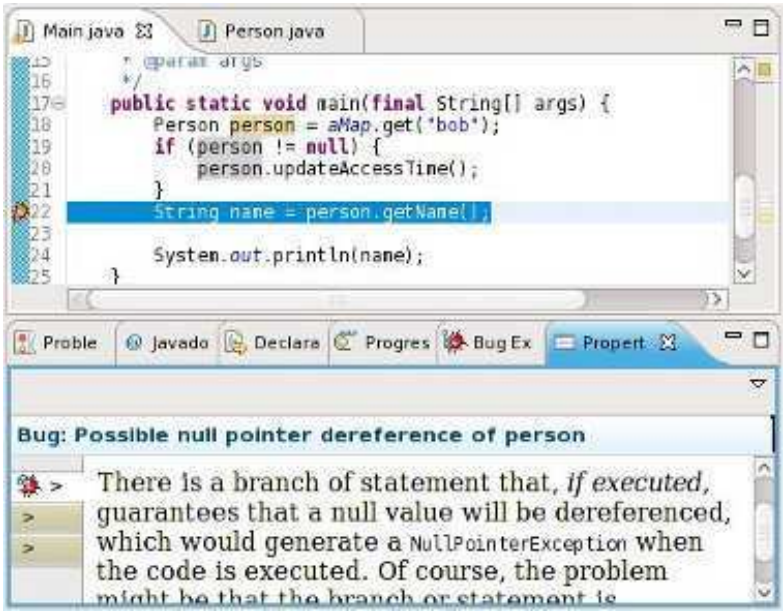

Figura 3.8: Integração do FindBugs com o Eclipse

Para a adição de novas regras, não existem formas declarativas de especificação. Assim como no CheckSyle tudo é feito em linguagem Java por meio de Visitors. Diferentemente dos outros analisadores, o FindBugs age sobre o Java bytecode e não sobre o código fonte do programa. Desta forma, escrever uma nova regra para o FindBugs é significativamente mais complexo que escrever para os anteriores.

3.5 NCL-Validator

O NCL-Validator é a ferramenta feita com o mesmo propósito que o NCL-Inspector, que é prover validações para documentos NCL. Porém a sua arquitetura não é flexível e para adicionar novas regras é necessário recompilar todo o seu código. Além disso, não permite a especificação de regras através de uma linguagem declarativa.

Um de seus pontos fortes é possuir integração com o ambiente NCL-Eclipse e Composer. Assim a sinalização dos erros pode ser feita em tempo real. Porém, o NCL-Validator/NCL-Eclipse não possui nenhum mecanismo de correção automática dos problemas e também não sugere correções. A Figura 3.9 mostra o NCL-Validator sendo usado dentro do Composer.

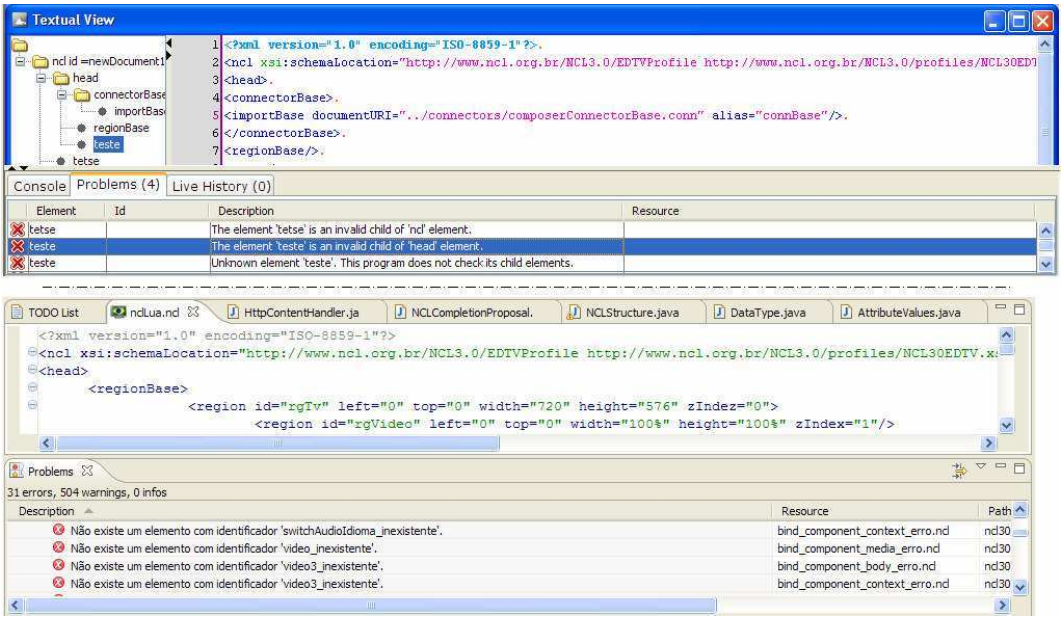

Figura 3.9: NCL-Validator sendo usado dentro do Composer.

3.6 Considerações

O PMD possui uma forma bastante fácil de criação de novas regras, além de possibilitar a criação de regras através de uma linguagem declarativa. O CheckStyle fornece correções automáticas para os problemas. O FindBugs realiza a verificação utilizando o Java Bytecode, ao invés de usar o código fonte, isto é, utiliza uma forma diferente de olhar para o programa. O NCL-Validator

avalia código NCL e ainda é integrado ao ambiente de desenvolvimento NCL-Eclipse.