Automatic Software for Judging Java Source Code Quality

SAT301 Final Year Project Oral Presentation

Guanyuming He Supervisor Thomas Selig

Department of Computing, School of Advanced Technology
Xi'an Jiaotong-Liverpool University

10 April, 2024 / EE110

Introduction

Background

- Increasing number of students in programming courses.
- People stared using automatic tools to evaluate programming coursework.
- Almost all of these tools only evaluate the correctness, so the gap is no code quality evaluation tool in educational context.

What I will present

- A system to fill the gap.
- How it works.
- The results on some Java code.
- Discussion & Future Works

Objectives

How do I judge code quality?

- Judge code quality from **multiple** prospects: spacing, indentation, naming, length, comments, ...
- **Deterministic** & **explainable**, thus arguably **fair** (compare with Machine Learning).

How do I engineer such a system?

- Successfully develop a system to judge **Java** source code quality.
- Make the System **extensible and customisable** to adapt to different code quality standards.

Methodology: Two major methods

Text analysis

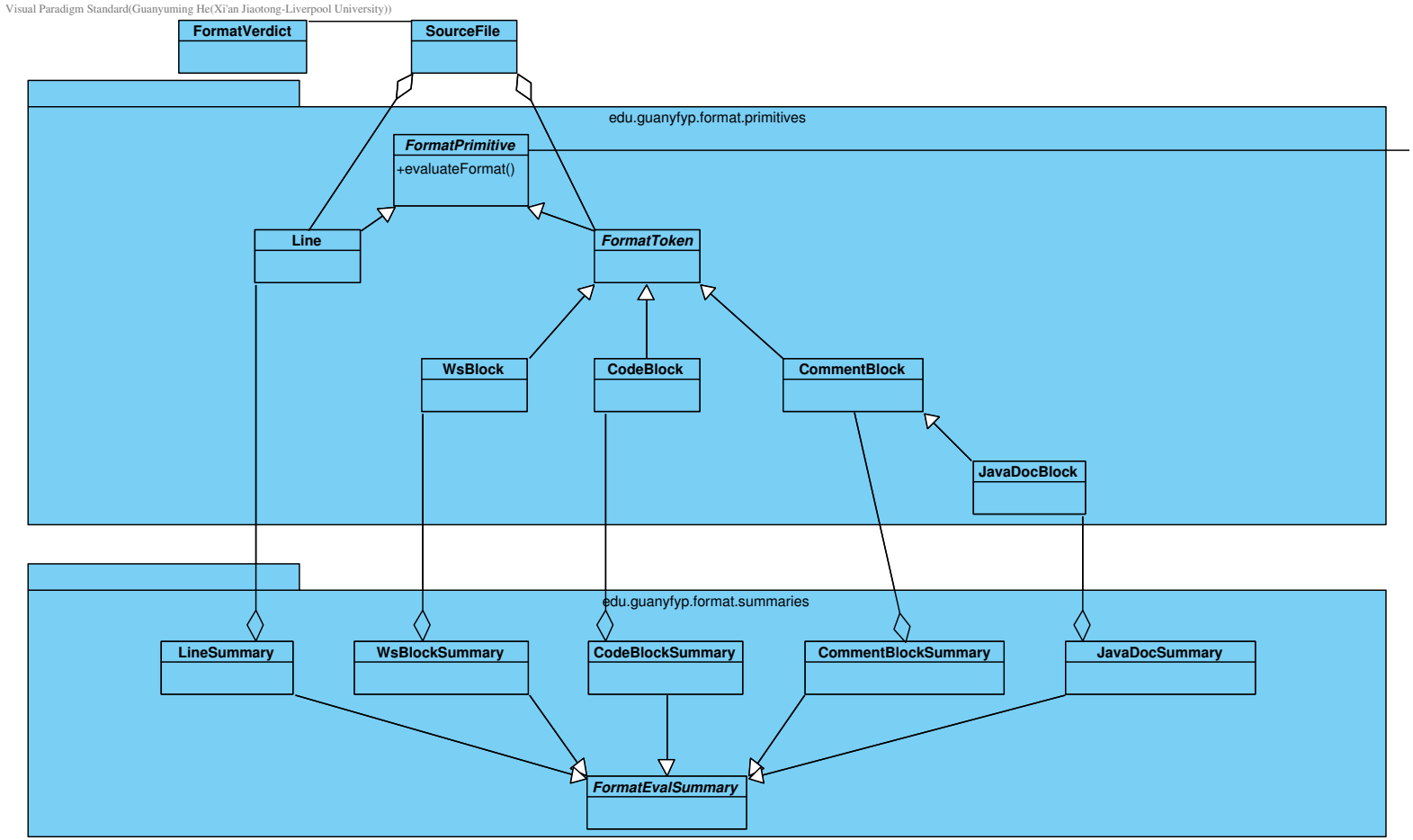
- Divide source code file into tokens.
- Find tokens before, after. Find tokens based on line and/or index.
- Calculate how long tokens (tabs, etc.) look.

Partial Syntax (parsing) & semantics analysis

- Divide tokens into code, spaces, JavaDoc, and other comments.
- Divide code into exact types. E.g. class name, parameter name, keyword, operator ...
- Understand the meaning of code to some extent.

Methodology: System Architecture

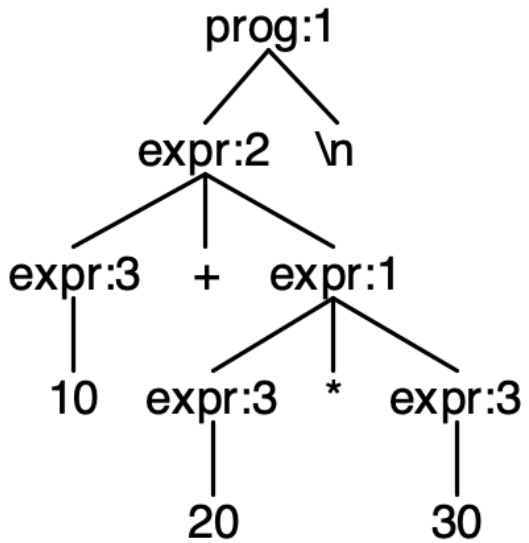
The core part is shown below (Other classes are not shown)



Methodology: Software Reuse: ANTLR 4

- ANTLR version 4 is reused to handle the syntax (parsing) & semantics analysis. It turns source code into a **parsing tree** according to a grammar.
- When walking on the tree, ANTLR notifies if a syntax construct has been entered/exited.

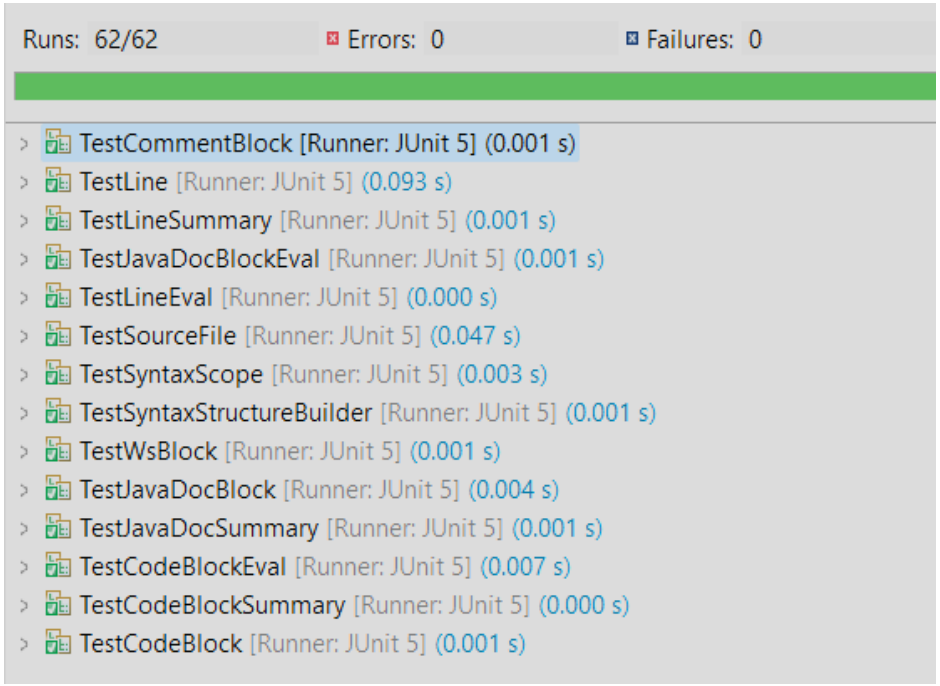
```
grammar Expr;
prog:    (expr NEWLINE)* ;
expr:    expr ('*'|'/') expr
     |   expr ('+'|'-') expr
     |   INT
     |   '(' expr ')'
     ;
NEWLINE : [\r\n]+ ;
INT     : [0-9]+ ;
```



Xi'an Jiaotong-Liverpool University 西交利物浦大學

Unit Test Results

The whole system is thoroughly tested with JUnit.




```
ArrayList<FormatTokenTestProperties> line14 = new ArrayList<>();
int i = 0;
line14.add(new FormatTokenTestProperties(CodeBlock.class, "public", 0, 0, 14, i++));
line14.add(new FormatTokenTestProperties(WsBlock.class, " ", 6, 6, 14, i++));
line14.add(new FormatTokenTestProperties(CodeBlock.class, "class", 7, 7, 14, i++));
line14.add(new FormatTokenTestProperties(WsBlock.class, " ", 12, 12, 14, i++));
line14.add(new FormatTokenTestProperties(CodeBlock.class, "MixtureClass", 13, 13, 14, i++));
line14.add(new FormatTokenTestProperties(WsBlock.class, " ", 25, 25, 14, i++));
line14.add(new FormatTokenTestProperties(CodeBlock.class, "extends", 26, 26, 14, i++));
line14.add(new FormatTokenTestProperties(WsBlock.class, " ", 33, 33, 14, i++));
line14.add(new FormatTokenTestProperties(CodeBlock.class, "ABC", 34, 34, 14, i++));
line14.add(new FormatTokenTestProperties(WsBlock.class, " ", 37, 37, 14, i++));

ArrayList<FormatTokenTestProperties> line15 = new ArrayList<>();
line15.add(new FormatTokenTestProperties(CodeBlock.class, "{", 0, 0, 15, 0));
// Note that here I deliberately put a tab in the empty line.
ArrayList<FormatTokenTestProperties> line16 = new ArrayList<>();
line16.add(new FormatTokenTestProperties(WsBlock.class, "\t", 0, 0, 16, 0));

expected_tokens.add(line14);
expected_tokens.add(line15);
expected_tokens.add(line16);


ArrayList<FormatTokenTestProperties> line17 = new ArrayList<>();
line17.add(new FormatTokenTestProperties(WsBlock.class, "\t", 0, 0, 17, 0));
line17.add(new FormatTokenTestProperties(CommentBlock.class, "// A field", 4, 1, 17, 1));

ArrayList<FormatTokenTestProperties> line18 = new ArrayList<>();
int i = 0;
line18.add(new FormatTokenTestProperties(WsBlock.class, "    ", 0, 0, 18, i++));
line18.add(new FormatTokenTestProperties(CodeBlock.class, "public", 4, 4, 18, i++));
line18.add(new FormatTokenTestProperties(WsBlock.class, " ", 10, 10, 18, i++));
line18.add(new FormatTokenTestProperties(CodeBlock.class, "final", 11, 11, 18, i++));
line18.add(new FormatTokenTestProperties(WsBlock.class, " ", 16, 16, 18, i++));
line18.add(new FormatTokenTestProperties(CodeBlock.class, "int", 17, 17, 18, i++));
line18.add(new FormatTokenTestProperties(WsBlock.class, " ", 20, 20, 18, i++));
line18.add(new FormatTokenTestProperties(CodeBlock.class, "m", 21, 21, 18, i++));
line18.add(new FormatTokenTestProperties(WsBlock.class, " ", 22, 22, 18, i++));
line18.add(new FormatTokenTestProperties(CodeBlock.class, "=", 23, 23, 18, i++));
line18.add(new FormatTokenTestProperties(WsBlock.class, " ", 24, 24, 18, i++));
line18.add(new FormatTokenTestProperties(CodeBlock.class, "0", 25, 25, 18, i++));
line18.add(new FormatTokenTestProperties(CodeBlock.class, ";", 26, 26, 18, i++));
```

Xi'an Jiaotong-Liverpool University 西交利物浦大學

Results on Bad Code 1

Spacing/New Line Around | JavaDoc | Indentation | Spacing | Naming | Name Length | Line Length

```
1  /**
2   * This Java source file contains badly formatted code
3   *
4   * @return Should not be here.
5   * @author Guanyuming He
6   */
7  public class badly_named_class {
8      private int i; // too short
9      // badly indented
10         private int Integer2;
11     // badly indented
12 private String sssssssssssssssssssssssssssssssssssssssss; // too long
13
14     /**
15      * ctor 1
16      * @param missing
17      */
18     public badly_named_class(int i, int integer2,String str) // space after ,
19     { // inconsistent scope style
20         // badly indented. space around ; and =
21             this.i = i;this.Integer2=integer2;
22         this.ssssssssssssssssssssssssssssssssssssssssss =str;
23     }} // the last } is not supposed to be here.
24 // This line is too Looooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooooong
```

Results on Bad Code 2

```
Verdict:
identifierLengthProblemFrequency = 0.3125
identifierNamingProblemFrequency = 0.1875
spacingProblemFrequency = 0.2
inconsistentScopeStyleFrequency = 0.0
badJavaDocFrequency = 1.0
lineLengthProblemFrequency = 0.041666668
lineIndentationProblemFrequency = 0.16666667
hasCommentsAtAll = true
Summary toString() is not implemented yet.
Summary toString() is not implemented yet.
Summary toString() is not implemented yet.
Summary toString() is not implemented yet.
Summary toString() is not implemented yet.
```

Extensibility & Customisability

Extensibility

- Inherit from one of the concrete `FormatPrimitive`
- Override `evaluateFormat()`
- Put results into fields and add observers for them

Customisability

```
906 ////////////////////// Settings //////////////////////
907
908⊖    public static final class Settings
909     {
910         // Default values come from the Java coding convention by oracle.
911
912         // Identifier settings
913         public int longestIdentifierLength = 15;
914         public int shortestIdentifierLength = 2;
915         public NamingStyle desiredClassNamingStyle = NamingStyle.PASCAL_CASE;
916         public NamingStyle desiredMethodNamingStyle = NamingStyle.CAMEL_CASE;
917         public NamingStyle desiredVariableNamingStyle = NamingStyle.CAMEL_CASE;
918         public NamingStyle desiredConstantNamingStyle = NamingStyle.UPPERCASE_UNDERSCORE;
919
920         // Punctuation settings
921         public boolean checkPunctuationSpacesAround = true;
922
923         // Operator settings
924         public boolean checkOperatorSpacesAround = true;
925         public boolean checkSpaceAroundIncDec = true;
926     }
927
928     public static final Settings settings = new Settings();
929
```

Xi'an Jiaotong-Liverpool University 西交利物浦大學

Discussion & Future Works

Comparison with other code quality judging works

- Most state-of-the-art works use Machine Learning.
- Some works use similar methods, but are for the industry.
- However, my work may not be as strong as some industry solutions.
- Besides, my work hasn't been optimised for speed.

Future Works

- Consider integrating it into some existing auto-graders.
- Consider making it support evaluating more languages.

Conclusion

- Gap: automatic code quality evaluation in education context.
- Method: Text Analysis & (Partial) Parsing & Semantic Analysis
- Goal: Fair, Deterministic, Extensible, Customisable.
- Thoroughly Tested by Unit Test
- Results: Can find many kinds of problems.

Thank You

Feel free to ask Questions.