Changes in JHawk 6

Contents

Changes in JHawk6 3
Line counting 4
System Level 4
Compilation Unit Level 4
Package Level 4
Class Level 4
Method Level 4
Comment counting 5
Statement counting 6
Expression counting 6
Variable Declarations 7

Changes in JHawk6

We have made some changes to the way that metrics are collected in JHawk6 vs JHawk5. In most cases these are bug fixes but in some cases they are changes in philosophy. The changes are summarised below and covered in more detail in subsequent sections of this document:

- The main change in JHawk6 is that we are now using an ANTLR based parser rather than the JavaCC based parser that we used before. This was partly because JavaCC is no longer being developed and partly because the code produced was difficult to maintain. The ANTLR based parser is slower and uses more memory but we believe that the benefits that we get in accuracy and easier maintenance outweigh the disadvantages. Analysis of large file sets is still very fast – for example the 22,000 java files in the sources for Eclipse 4.2 are analysed down to the method level in less than 15 minutes. We, of course, hope to improve these figures over time but we feel that the current performance is adequate.
- We have made changes to the JHawk Metric Interchange Format which is used for XML output. We have made no major changes in the user interface or the CSV and HTML output formats.
- We now support the analysis of Java files written for versions of Java from 1.4 up to Java 8.
- Halstead figures are now only collected at method (and static block) level and at class level. The Class level figures for are based (where it makes sense) on the totals of the Halstead metrics gathered at the method level plus those collected at the class level – the class statement itself, annotations and member declarations (they are collected for classes, interfaces and enumerations).
- The number of Java statements will be slightly different in some cases as we resolved a bug in that calculation. See details in section below.
- The number of Java expressions will be different in many cases as we have changed our philosophy in relation to the way that we count them – for a further explanation see below
- We have improved the counting of code and comment lines to increase the information available (see the section below).
- Return value classes and primitive types are now accurate as a result of a bug fix.
- A number of bugs related to the counting of variable references were fixed (see the section below).
- In some circumstances local method calls were not counted correctly. This is now fixed.
- In some cases some methods in classes were not being detected. This is now fixed.

We feel that these changes allow us to maintain JHawk's position as the most reliable metrics calculation engine on the market. We have also made a lot of changes under the covers that we hope will allow us to expand the functionality of JHawk in the future.

Line counting

System Level

- Total lines enclosed in the system – calculated by adding up all the line records for all of the compilation units in the system
- Comments in the system
- Blank lines in the system
- Comments outside the system (this will always be zero)

Compilation Unit Level

- Total lines in the compilation unit (including comments, blank lines , lines of code)
- Comments inside the compilation unit
- Blank lines inside the compilation unit
- Comments outside the compilation unit (this will always be zero)

Package Level

- Total lines enclosed declared at the package level (including the declaration, import declarations, comments, blank lines and comments outside the package). This also includes the total lines of all the classes declared as being members of this package

Class Level

- Total lines enclosed by the class (including the declaration, comments, blank lines and comments outside the class)
- Comments inside the class
- Blank lines inside the class
- Comments outside the class

Method Level

- Total lines enclosed by the method (including the declaration, comments, blank , lines and comments outside the method)
- Comments inside the method
- Blank lines inside the method
- Comments outside the method

Comment counting

The numbers of each type of comment are counted as well as the number of lines that each comment occupies in the code. Four types of comment are counted:

1. Formal comments

```
/**
    • Formal (or Javadoc) comments
**/
```

2. Multi-line comments

```
/*
 *  A multi line comment
*/
```

3. Single line comments (on their own line)

```
// On my own line
```

4. Single line comments (at the end of the line of code)

```
Int x = 1; //at the end of a line
```

Only lines in comments of types 1,2 and 3 are counted. Note that in previous versions of JHawk the end of line comments were counted as comment lines - we keep a note of these comments as POST_CODE_COMMENTS if you wish to retrieve them.

Statement counting

Proper counting of switch statements

```
1.      switch (element.getElementType()) {
2.              case IElementDescriptor.TYPE:
3.                      signature = ((IReferenceTypeDescriptor)element).getSignature();
4.                      break;
5.              case IElementDescriptor.METHOD:
6.                      signature = ((IMethodDescriptor)element).getSignature();
7.                      name = ((IMethodDescriptor)element).getName();
8.                      break;
9.              case IElementDescriptor.FIELD:
10.                     name = ((IFieldDescriptor)element).getName();
11.                     break;
12.             default:
13.                     break;
        }
```

This would previously have been counted as 9 statements as the case and the following line were always counted as a single statement. We felt that the second statement should be counted on its own as the case could stand on its own and therefore was really also a statement.

Expression counting

In JHawk we count expressions. The definition of expressions in Java is very precise and is described in the Java language definition (which you can find here). JHawk counts all expressions except Primary expressions. A Primary expression would be something like a variable e.g. in the statement:

```
return retVal;
```

retVal is a Primary expression so that the statement above would contain no expressions. The Java language definition describes a statement as a Statement Expression. JHawk counts a Statement Expression as a Statement but not as an expression.

The following examples illustrate pieces of code that are recognised as expressions by JHawk.

We are familiar with expressions as a mathematical concept e.g. in a calculation so if we take a Java statement that involves a calculation we can see how it can be built up of expressions:

```
return(a*b)*Math.PI;

(a*b)*Math.PI      //The overall expression
a*b                // The first sub-expression
```

In total this statement contains two expressions. As Math.PI is a constant it is not treated as an expression.

In the next statement we have a number of method calls, each of which is an expression:

```
System.out.println(stack.pop());

println()    //The first expression
pop()        //The second expression
```

System.out is not treated as an expression as it is a constant (like Math.PI in the previous statement).

In this statement again we have a number of method calls, each of which is an expression; and a parameter which is a variable and is therefore not an expression:

```
System.out.println(stack.getElement(i));
println()          //The first expression
getElement(i)      //The second expression
```

In the following statement we see a conditional expression and an assignment:

```
if(myheight>0)item=value[myheight-1];
myheight>0               //Conditional Expression
item=value[myheight-1]   //Assignment expression
value[myheight-1]        //Expression to access variable value
myheight-1               //Arithmetic expression
```

In this assignment statement there are three expressions:

```
value[myheight++]=item;

value[myheight++]=item  //Assignment expression
value[myheight++]       //Expression to access variable
myheight++              //Arithmetic expression
```

Variable Declarations

- Variables declared in for loops were not always counted correctly before.
- If a variable is declared more than once in the same method each declaration is now counted.
- In some cases classes (e.g. casts, instanceof expressions) were counted as variables.

.

Interface changes

ParserBasicRecordInterface

- The numberOfCommentLines method was removed

ParserSystemRecordInterface

- The String[] allPackageNames method was removed
- The getCompilationUnit(String) method was added
- The getCompilationUnits method was added