DELPHI SONAR PLUGIN

# IMPLEMENTED FEATURES

1. Counting lines of code, statements, number of files
2. Counting number of classes, number of packages, methods, accessors
3. Counting number of public API (methods, classes and fields)
4. Counting comments ratio, comment lines (including blank lines)
5. CPD (code duplication, how many lines, block and in how many files)
6. Code Complexity (per method, class, file; complexity distribution over methods, classes and files)
7. LCOM4 and RFC
8. Code colorization
9. Unit tests reports
10. Assembler syntax in grammar
11. Include statement
12. Parsing preprocessor statements
13. Rules
14. Code coverage reports
15. Source code highlight for unit tests
16. “Dead” code recognition
17. Unused files recognition

# NOT IMPLEMENTED FEATURES (TODO)

1. Package tangle index

# CODE ASSUMPTIONS

1. Grammar has problems with recognizing certain identifiers. When we add whitespace (“ “) at several places, everything works fine. It was difficult to solve this problem from the grammar level, so it is solved in **DelphiSourceFileStream** class. It adds whitespaces where there are needed in parsed file, and grammar works fine. It is used for:
   1. Colon, “:” -> “ :”
   2. Array range, “x..y” -> “x .. y”
2. Grammar is NOT case insensitive, but Delphi code is. Plugin deals with it by **DelphiSourceFileStream** class, which feeds ANTLR parser lowercase characters (the “LA” method)
3. Number of classes includes: classes, records
4. Package is counted as a directory. Number of packages equals number of directories.
5. Preprocessor definitions between **{$if xxx}** and **{$ifend}** are removed (**DelphiSourceFileStream**).
6. Sources imported to Sonar are parsed through **DelphiSourceFileStream** class. It means, that the source will be lowercased and unknown preprocessor definitions will be cut out.

# GRAMMAR ASSUMPTIONS

All assumptions in grammar file are commented with **//ASSUMPTION** comment.

1. “ident” rule – was extended, now it uses some keywords that were used as variable names in source code, ex. “name”, “readonly”, “index”. If you use more keywords as variable names or function names, you should extend the rule further.
2. The grammar has problem with proper recognizing identifier that ends with “:” (“e:” does not work, does not recognize “e” as identifier), see **code assumption 1** for solution.
3. The grammar has problem with proper array range recognizing (“array[1..2]” does not work), see **code assumption 1** for solution.
4. The grammar is not case-insensitive, and it should be. For parsing files using ANTLRWorks, convert them to lowercase. Plugin deals with it automatically, by **AntlrNoCaseFileStream** class.

# MOST COMMON GRAMMAR PROBLEMS (FIXED):

Below is the list of the most common grammar problems. You can also look at file **GrammarTest.pas** to get more detailed information.

* 1. On e: exception do
  2. Name = procedure (x: integer) of object;
  3. Using keywords as variable names (ex. continue, message, name, db)
  4. Array[1..2]
  5. @address = someProcedure(x,y);
  6. (expression).namespace.ident = x;
  7. Default function argument values crashed AbstractAnalyser
  8. Inherited;
  9. if (tempstr[1] in ['0' .. '9', 'a' .. 'z'] = false) then
  10. str: string[3];
  11. function tfilewriter.writebytes(var ibytes; isize : dword) : boolean;
  12. DetailedDescription : array of Byte;
  13. ferrorlist : tlist<string>;
  14. Delphi complier does not require “;” at the end of except statement:

On e: exception do

Begin

End

# DEBUGGING GRAMMAR FROM ANTLRWORKS

If you want to debug grammar from AntlrWorks, do the following:

* 1. Convert file to be parsed to lowercase
  2. Apply all transformations mentioned in **code assumption 1**.
  3. Use more Java heap space while running AntlrWorks, ex. 1024M (java –Xmx1024m)

# DUNIT TESTS

You should put transformed DUnit xml files to directory specified in build.xml file, under sonar.surefire.reportsPath parameter. You can specify multiply directories separated by ‘,’. Then all specified directories will be parsed. The path can be either absolute or relative to the project main path.

The xml files should be renamed with “TEST-“ prefix, example: “dunit.xml” 🡪 “TEST-dunit.xml”.

TODO Transformation from Dunit format to SureFire format.

# CODE COVERAGE WITH AQTIME

CodeCoverageSensor class is responsible for running CC analysis. It creates a parser that connects to specified data base holding AQTime generated CC report. Connection properties can be specified in a build.xml or pom.xml file (see below). If parser can’t connect to the data base, analysis is skipped.

You can specify exclude directories in build xml file, under sonar.delphi.codecoverage.excluded parameter. The files in specified directories will be not checked for code coverage.

# DELPHI SOURCE FILE STREAM CLASS

This class prepares source code for parsing. Its responsibilities are:

* Parse preprocessor definitions
* Add include files
* “Fix” the code as explained in (**code assumption 1**)**.**
* Feed ANTLR parser with lowercase characters (**code assumption 2**).

To properly do this things, the class firstly searches for areas that should not be parsed: comments and single quoted strings. Why? For example, you can have such piece of code:

X := 5; //{$include file.inc}

S := ‘my {$include file.inc} string’;

Without recognizing which areas to exclude, the parser would simply copy-paste the file “file.inc” into a comment or a string, and this would produce an error.

# ANT BUILD.XML / MAVEN POM.XML

There are few additional project parameters that you can set in Ant’s build.xml.

|  |  |  |
| --- | --- | --- |
| KEY | DESCRIPTION | EXAMPLE |
| sonar.delphi.sources.excluded | Excluded directories, they won’t be parsed and metrics won’t be calculated for them. Path’s can be relative to main project directory or absolute. Separate them with ‘,’.  Default value is “”. | <property key="sonar.delphi.sources.excluded " value="excluded\directory, another\excluded,c:\something" /> |
| sonar.delphi.codecoverage.excluded | Code coverage excluded directories list. Files in those directories will not be checked for code coverage. Default value is “”. | <property key="sonar.delphi.codecoverage.excluded" value="cc\test\excluded,c:\sth" /> |
| sonar.delphi.sources.include | Path to include directories for files included with {$include} or {$i} directive.  Default value is “”. | <property key="sonar.delphi.sources.include" value="includes\_dir,includes\_dir2" /> |
| sonar.delphi.sources.include.extend | Should we extend (copy-paste) include files? Possible values are “true” or “false”.  Default value is “true”. | <property key="sonar.delphi.sources.include.extend" value="true"> |
| sonar.delphi.sources.projects | Project file. If provided, will be parsed for include lookup path, project source files and preprocessor definitions. Default value is “”. | <property key=" sonar.delphi.sources.project" value="projects\myProject.dproj"> |
| sonar.delphi.sources.workgroup | Workgroup file. If provided, will be parsed, then all \*.dproj files found in workgroup file will be parsed. | <property key=" sonar.delphi.sources.workgroup" value="projects\All.groupproj"> |
| sonar.delphi.codecoverage.aqtime.jdbc.driver | Class name for JDBC driver. Default value is “net.sourceforge.jtds.jdbc.Driver” | <property key="sonar.delphi.codecoverage.aqtime.jdbc.driver" value="net.sourceforge.jtds.jdbc.Driver" /> |
| sonar.delphi.codecoverage.aqtime.jdbc.url | Database connection url. Default value is “”. **REQUIRED** | <property key="sonar.delphi.codecoverage.aqtime.jdbc.url" value="jdbc:jtds:sqlserver://localhost" /> |
| sonar.delphi.codecoverage.aqtime.jdbc.user | Database user name. Default value is “”. **REQUIRED** | <property key="sonar.delphi.codecoverage.aqtime.jdbc.user" value="admin" /> |
| sonar.delphi.codecoverage.aqtime.jdbc.password | Database user password. Default value is “”. **REQUIRED** | <property key="sonar.delphi.codecoverage.aqtime.jdbc.password" value="pass" /> |
| sonar.delphi.sources.tests | Path to unit tests source directories. DO NOT end directory path with backslash character. Default value is “”. **REQUIRED** | <property key="sonar.delphi.sources.tests" value="\Testing\Units,\Testing\Units2" /> |

# CUSTOM PMD RULES

If you want to make a custom PMD rule, you can do it in two ways:

WRITE YOUR OWN CLASS

Create rule class, that extends from org.sonar.plugins.delphi.pmd.rules.DelphiRule class. You will need to overload visit(DelphiPMDNode node, Object data) method. You will have the current ast tree node and whole ast tree to your disposal. If you want to save a violation, invoke addViolation(Object data, DelphiPMDNode node) method.

If you want to initialize some class variables at start of each file, overload the init() method. Look at the org.sonar.plugins.delphi.pmd.rules package for sample classes.

After you write your class, modify the “rules.xml” and “default-delphi-profile.xml”, simply add your class to the list.

WRITE XPATH SENTENCE

You can write an XPath sentence which will parse the whole file. Open the “rules.xml” file, copy one of the XPath rules already available and change the “xpath” property to your XPath string. Don’t forget to add the rule to “default-delphi-profile.xml”. That’s it!

# IMPLEMENTED RULES

For full summary (description, examples) of custom PMD rules, do the following: execute Sonar -> Configuration -> Default Delphi Profile and browse the rules under Delphi profile.

List of implemented rules, as follows:

|  |  |
| --- | --- |
| ***RULE NAME*** | ***RULE PRIORITY*** |
| 1. *Then Try Rule (you should place ‘begin’ before ‘try’)*   ***Control structures should use begin-end*** | *3* |
| 1. *One Class Per File Rule (one class per file permitted)* | *3* |
| 1. *Inherited Method With No Code Rule (method that only inherits behavior)*   ***Overriding methods should do more than simply call the same method in the super class*** | *2* |
| 1. *Empty Except Block Rule (empty ‘except’ block, catching no exceptions)* | *1* |
| 1. *Empty Interface Rule (empty interface)* | *2* |
| *Empty Then Statement Rule (empty ‘then’ statement)* | *2* |
| 1. *Empty Begin Statement Rule (empty ‘begin’ .. ‘end’ statement)* | *2* |
| 1. *Empty Else Statement Rule (empty ‘else’ statement)* | *2* |
| 1. *Too Long Method Rule (method too long, above certain line limit)* | *3* |
| 1. *Too Many Arguments Rule (to many function/procedure arguments)* | *3* |
| 1. *Too Many Variables Rule (to many variables in a function/procedure)* | *3* |
| 1. *No Semi After Overload Rule (no semicolon ‘;’ after ‘overload’ keyword)* | *4* |
| 1. *No Function Return Type Rule (function with no return type specified)* | *3* |
| 1. *Interface Name Rule (interface name should begin with ‘I’ letter)*   ***Rename: Interface Name should comply with a naming convention*** | *4* |
| 1. *Class Name Rule (class name should begin with ‘T’ letter)*   ***Rename: Class Name should comply with a naming convention*** | *4* |
| 1. *Record Name Rule (record name should begin with ‘T’ letter)*   ***Rename: Record Name should comply with a naming convention*** | *4* |
| 1. *Catching General Exception Rule (catching Exception class in ‘except’ statement)* | *3* |
| 1. *Raising General Exception Rule (raising Exception class in ‘raise’ statement)* | *3* |
| 1. *Avoid Out Parameter Rule (using ‘out’ arguments)* | *4* |
| 1. *If True Rule (convert ‘if(x=true) then’ to ‘if (x) then’)* | *4* |
| 1. *If Not False Rule (convert ‘if not (x = false) then’ to ‘if (x) then’* | *4* |
| 1. *Public Field Rule (class fields should NOT be public)*   ***Note: Is it checking published fields???*** | *4* |
| 1. *Avoid Unused Method Parameters Rule (avoid making function arguments that are not used in body)* | *3* |
| 1. *No Interface Guid (no Guid at interface)* | *5* |
| 1. *If Assigned And Free (before calling ‘Free’ you don’t need to check for variable assignment ‘x <> nil’)* | *4* |
| 1. *Project File No Functions Rule (.dpr file should not contain procedures or functions)* | *3* |
| 1. *Project File No Variables Rule (.dpr file should not contain variables)* | *3* |
| 1. *Type Alias Rule (avoid using type aliases)* | *4* |
| 1. *Uppercase Reserved Keywords Rule (avoid using uppercase reserved keywords)*   ***Rename: Keywords should comply with a naming convention*** | *5* |
| 1. *Mixed Names Rule (avoid case mixing names of functions / variables)* | *4* |
| 1. *Unused Unit Rule (searches for units that are not referenced in other units)* | *3* |
| 1. *Unused Function Rule (searches for functions/procedures that are not used by other functions/procedures)*   ***Unused private method should be removed***  ***Method with greater visibility could be invoked by reflection*** | *3* |
| 1. *Constructor Without Inherited Statement Rule (constructor does not contain ‘inherited’ statement)* | *2* |
| 1. *Destructor Without Inherited Statement Rule (destructor does not contain ‘inherited’ statement)* | *2* |
| 1. *No ‘begin’ after ‘do Statement Rule*   ***Control structures should use begin-end*** | *4* |
| 1. *‘With’ after ‘do’ statement Rule* | *3* |
|  |  |
| *Control flow statements "if", "for", "while", "case" and "try" should not be nested too deeply(max:3)* |  |
| *Global Methods* |  |
| *Too Long File (file too long, above certain line limit)* |  |
| *Global variables (global variables is evil) (skip form reference variables)* |  |
| *Nested Method(nested method make the code difficult to read and you can't reuse this code, prefers a private method)* |  |
| *Method Name Rule (method name should follow Pascal Case)* |  |
| *Too many methods (the class/interface/record has too many methods)* |  |
| *Too many attributes* |  |
| *Magic Number should not be used.*  -1, 0 and 1 are not considered magic numbers. |  |
| *FixInsight - Empty Finally Block* |  |
| *FixInsight - Assignment right hand side is equal to its left hand side* |  |
| *FixInsight -* *THEN statement is equal to ELSE statement* |  |
| *FixInsight -* *Variable is assigned twice successively* |  |
| *FixInsight -* *Unreachable code* |  |
| *FixInsight -* *Values on both sides of the operatior are equal* |  |
| *FixInsight -* *Object created inside Try block* |  |
| *FixInsight -* *Related "if/else if" statements should not have the same condition* |  |
| *FixInsight - Format parameter count mismatch* |  |
| *FixInsight - Loop iterator could be out of range (missing -1?) Note: remember of Strings and Enums* |  |
| *FixInsight - Suspect Free call* |  |
| *FixInsight - Variable hides a class field, method or property* |  |
| *FixInsight - Empty method* |  |
| *FixInsight - Empty method on abstract class should be abstract* |  |
| *FixInsight - Parenthesis might be missing around IN operand* |  |
| *FixInsight -* *Return value of function might be undefined* |  |
| *FixInsight -* *Destructor without an override directive* |  |
| *FixInsight -* *Generic interface declared with a GUID* |  |
| *FixInsight -* *Pointer type name should start with 'P'* |  |
| *FixInsight -* *Field names should comply with a naming convention*  *Following Object Pascal Style Guide* |  |
| *FixInsight -* *Nested WITH statement* |  |
| *SONAR - "TODO" tags should be handled* |  |
| *SONAR - "FIXME" tags should be handled* |  |
| *SONAR -* *Deprecated code should be removed eventually* |  |
| *SONAR - Unit names should comply with a naming convention* |  |
| *SONAR -* *Do not call Destroy directly. Call Free instead.* |  |
| *SONAR -* *Statements should be on separate lines* |  |
| *SONAR - Control structures should use begin-end*  ***Put begin..end after THEN, ELSE, DO*** |  |
| *SONAR -* *Tabulation characters should not be used* |  |
| *SONAR - Try-except blocks should not be nested* |  |
| *SONAR - Exception types should not be tested using in try..except blocks*  *The specific type should be catched.* |  |
| *SONAR - keywords should not be used as variable/method names* |  |
| *SONAR - Return of boolean expressions should not be wrapped into an "if-then-else" statement* |  |
| *SONAR - Local Variables should not be declared and then immediately returned or raised* |  |
| *SONAR - Literal boolean values should not be used in condition expressions.*  ***Replace: If True Rule and If not False Rule*** |  |
| Place begin statements on their own line.  **After THEN, ELSE, DO statements** |  |
| *SONAR - The members of an interface declaration or class should appear in a pre-defined order*  *Class and Instance fields, Constructors, Methods, Properties* |  |
| *SONAR - Access levels should appear in a pre-defined order*  *Private, Protected, Public, Published* |  |
| Classes should not be too complex (max: 200) |  |
| Method should not be too complex (max: 10) |  |
| Use Assigned method to test for a nil pointer |  |
| SONAR - Comments should not be located at the end of lines of code |  |
| SONAR - Unused private fields should be removed |  |
| SONAR - Collapsible "if" statements should be merged |  |
| SONAR - Method parameters and caught exceptions should not be reassigned |  |
| SONAR - IP addresses should not be hardcoded |  |
| SONAR - Goto statements should not be used |  |
| SONAR - "case" clauses should not have too many lines (max: 5) |  |
| SONAR - Exceptions should not be thrown in finally blocks |  |
| Deprecated elements should be followed by a recommendation message |  |
| Insufficient line coverage by unit tests (minimumLineCoverageRatio: 65.0) |  |
|  |  |