Several changes are made to fix old bugs. Some of the bugs are fixed by referring to 2019 Spring team, and some are fixed by myself. If you have any questions about the modification or would like to find any existing bugs, it will be very useful to look at reports from 2019 Summer team and 2019 Spring team.

The goal is to fix bugs existing in UCCG 1.3.1 to prepare for the new release. Below are the bugs that have been fixed:

**Bug 1:**

import sun.awt.windows.ThemeReader;

When compiling, the line above will fail, and it is in src/ucc/main/ProcessController.java. The modification is to slience this line, and it will be good.

**Bug 2:**

Some of data in outfile\_cyclomatic\_cplx will have new object name instead of function name which is displayed in the function name region. It is shown below:

**A screenshot of a cell phone

Description automatically generated**

The modification is to change line 786, which is:

**A screenshot of a cell phone

Description automatically generated**

Adding a “new” keywork to the if condition, the bug will be finxed.

New version:

**A picture containing device

Description automatically generated**

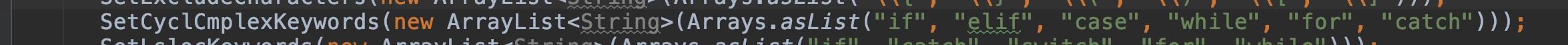
**Bug 3:**

**A close up of a piece of paper

Description automatically generated**

The CC1 count for Python file is not always true. The reason is that in PythonLanguageProerties.java file, in line 57, the setcyclCmplexKeywords is not true. Python is not using try-catch but using try-except, so by changing “catch” to “except” the bug will be fixed.

Before:

****

After:

**A close up of a logo

Description automatically generated**

**Bug 4:**

CC1 count for .py file with specific function name may be incorrect. For example,

A screenshot of a cell phone

Description automatically generated

Changeme appears only once in EES1.py but the CC1 count for that was 2

A close up of a mans face

Description automatically generated

Actually both CalculateCyclomaticComplexity and GetTrueIndentationSize in PythonCounter.java file increase the count.

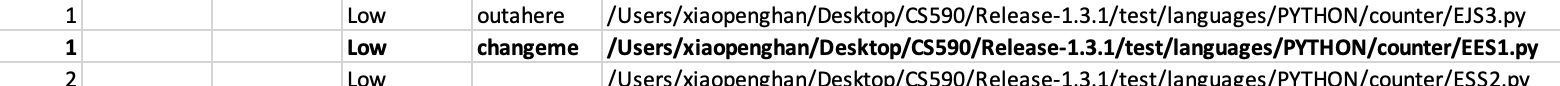
The way to fix this is to comment one line in GetTrueIndentationSize which is around line 1070.

After that it will be

**A close up of a logo

Description automatically generated**

And the result will be correct below:

****

**Bug 5:**

****

VHDL language use wrong keyword for cyclomatic complexity, it should be “elsif” instead of “elseif”

After change it will be (line 61 in VHDLLanguageProperties) :

****

**Bug 6:**

Regarding to previous Integration report, more keywords are added to some of the languages.

|  |  |  |
| --- | --- | --- |
| **General Description of New Feature or Code Modification** | | |
| **Modified Files** | **Modified**  **Line number** | **Description** |
| **ADALanguageProperties.java** | Line 60 | - "task", "terminate", "then", "type", "until", "use", "with", "if")));  + "task", "terminate", "then", "type", "until", "use", "with", "if", "case", "elsif", "end", "loop"))); |
| **CCPPLanguageProperties.java** | Line 58 | - "virtual", "void", "volatile", "wchar\_t")));  + "virtual", "void", "volatile", "wchar\_t", "FILE"))); |
| **CSharpLanguageProperties.java** | Line 43 | - "finally", "new", "return", "super", "this", "default")));  + "finally", "new", "return", "super", "this", "default", "as", "base", "case", "checked", "do", "else", "fixed",  + "for", "foreach", "goto", "if", "lock", "sizeof", "stackalloc", "switch"))); |
| CSharpLanguageProperties.java | Line 59 | - "toDegrees", "ulp")));  + "toDegrees", "ulp", "ilogb"))); |
| CSharpLanguageProperties.java | Line 51 | - SetLogOpKeywords(new ArrayList<String>(Arrays.asList("log", "log10", "log1p", "log2")));  + SetLogOpKeywords(new ArrayList<String>(Arrays.asList("log", "log10", "log1p", "log2", "logb"))); |
| **DOSBatchLanguageProperties.java** | Line 43 | - SetCondKeywords(new ArrayList<String>(Arrays.asList("if", "for")));  - SetLogicalOps(new ArrayList<String>(Arrays.asList("==", "equ", "neq", "lss", "leq", "gtr", "geq")));  + SetCondKeywords(new ArrayList<String>(Arrays.asList("if", "for", "choice")));  + SetLogicalOps(new ArrayList<String>(Arrays.asList("==", "&&", "||", "equ", "neq", "lss", "leq", "gtr", "geq"))); |
| **FortranLanguageProperties.java** | Line 47 | - SetOtherExecKeywords(new ArrayList<String>(Arrays.asList("break", "continue", "goto", "return", "exit")));  + SetOtherExecKeywords(new ArrayList<String>(Arrays.asList("break", "continue", "goto", "return", "exit", "backspace", "call",  + "close", "cycle", "do", "elseif", "entry", "exit", "forall", "format", "if", "inquire", "open", "pause",  + "print", "read", "return", "rewind", "select case", "stop", "where", "write"))); |
| **FortranLanguageProperties.java** | Line 61 | - SetExcludeKeywords(new ArrayList<String>(Arrays.asList("endif", "else", "enddo", "end", "case default", "case")));  + SetExcludeKeywords(new ArrayList<String>(Arrays.asList("endif", "else", "do", "enddo", "end", "case default", "case", "endblockdate", "endfile",  + "endfunction", "endinterface", "endmodule", "endprogram", "endselect", "endsubroutine", "endtype", "endwhere", "endforall",  + "endassociate", "endnum"))); |
| **IDLLanguageProperties.java** | Line 39 | - new ArrayList<String>(Arrays.asList("if", "else if", "switch", "case", "for", "foreach", "while")));  + new ArrayList<String>(Arrays.asList("if", "else if", "repeat", "else", "switch", "case", "for", "foreach", "while"))); |
| **IDLLanguageProperties.java** | Line 41 | - Arrays.asList("and", "or", "not", "xor", "lt", "gt", "ge", "le", "eq", "ne", "~", "&&", "||")));  + Arrays.asList("and", "or", "not", "xor", "lt", "gt", "ge", "le", "eq", "ne", "~", "&&", "||", ">", "<", ">=", "<=",  + "^"))); |
| **PascalLanguageProperties.java** | Line 38 | - SetCondKeywords(new ArrayList<String>(Arrays.asList("if", "case", "try", "for", "while", "repeat", "with")));  + SetCondKeywords(new ArrayList<String>(Arrays.asList("if", "case", "try", "for", "while", "repeat", "with", "else"))); |
| **PascalLanguageProperties.java** | Line 40 | - Arrays.asList("=", "<>", ">", "<", ">=", "=<", "not", "and", "or", "xor", "shl", "shr")));  + Arrays.asList("=", "<>", ">", "<", ">=", "<=", "not", "and", "or", "xor", "shl", "shr"))); |
| **PascalLanguageProperties.java** | Line 47 | - "cardinal", "char", "class", "comp", "complex", "const", "double", "extended", "file",  + "cardinal", "char", "class", "comp", "complex", "const", "currency", "double", "extended", "file", |
| **PerlLanguageProperties.java** | Line 48 | - "last", "new", "next", "return", "self", "switch", "try", "unless", "until", "while")));  + "last", "new", "next", "return", "self", "switch", "try", "unless", "until", "while", "eval", "sub"))); |
| **PythonLanguageProperties.java** | Line 44 | - "raise", "return", "try", "while", "with", "yield")));  + "raise", "return", "try", "while", "with", "yield", "elif", "else", "if"))); |
| **RubyLanguageProperties.java** | Line 49 | - "unless", "until", "when", "while", "yield")));  + "unless", "until", "when", "while", "yield", "continue", "default", "die", "switch", "try"))); |
| **SQLLanguageProperties.java** | Line 45 | - "truncate", "union", "update", "where")));  + "truncate", "union", "update", "where", "CHECK", "CONNECT", "CONTINUE", "ELSIF", "EXIT", "FOR", "GOTO",  + "IF", "LOCK", "LOOP", "MINUS", "RAISE", "RETURN", "START", "UNIQUE", "USE", "VIEW", "WHEN", "WHILE",  + "dbms\_output.put\_line"))); |
| **SQLLanguageProperties.java** | Line 53 | - "year")));  + "year", "number", "cursor", "exception"))); |
| **ScalaLanguageProperties.java** | Line 48 | - "new", "return", "super", "override", "if", "else")));  + "new", "return", "super", "override", "if", "else", "case", "def", "do", "for", "match", "this", "while"))); |
| **ScalaLanguageProperties.java** | Line 51 | - "String", "Boolean", "Unit", "Null", "Nothing", "Any", "AnyRef", "var", "val")));  + "String", "Boolean", "Unit", "Null", "Nothing", "Any", "AnyRef", "var", "val", "abstract",  + "Array", "class", "extends", "HashMap", "HashSet", "implements", "LinkedHashMap", "LinkedList",  + "object", "override", "sealed", "static", "TreeMap", "Vector"))); |
| **VerilogLanguageProperties.java** | Line 51 | - "notif0", "notif1")));  + "notif0", "notif1","$scale", "$shm\_open", "$shm\_probe", "$stime", "$stop", "$strobe", "$time", "$timeformat", "$write"))); |

**Bug 7:**

Some code are not included in the function, but in the test case it is missing. Using a single space as a function name is designed in this situation. The solution is to comment some lines that remove useful statements.

Before change:

A screenshot of a cell phone

Description automatically generated

After change:

A picture containing text

Description automatically generated

Before change:

A screenshot of text

Description automatically generated

**Bug 8:**

Function counter is not working properly for Python. The reason is that CountFileSLOC() count the PSLOC, but when doing PerformPreProcrssing() method, all leading spaces will be deleted. However, GetTrueCyclomatic() will also use the indentation to determine whether the function has ended or not. The other problem is that when the indentation changes, the counter will think that the next code will not belong to the old function, and it will start to count to next “def” function. As a result, the result will not show correctly.

|  |  |  |
| --- | --- | --- |
| **General Description of New Feature or Code Modification** | | |
| **Modified Files** | **Modified Functions** | **Description** |
| PythonCounter.java | CountFilePSLOC | **Before, it only uses one String to save the line:**  line = ro.line;  **Add another tempLine to save the original line:**  // Add a tempLine to save the indentation from pre-processing (Add by 2019 Spring Defects Fixing Team)  String tempLine = line;  line = ro.line;  *(line 124-126)*  **Add leading space before writing to BufferedWriter that passes lines to CountFileLSLOC():**  // Add the indentation for countint the cyclomatic complexity (Add by 2019 Spring Defects Fixing Team)  line = tempLine.substring(0, tempLine.indexOf(tempLine.trim())) + line;  *(line 310-311)* |
| PythonCounter.java | GetTrueCyclomaticComplexity | **Add a stack to save the outer function kndx and tempCount, push when meets a new def and pop when meets first line that has no greater indentation:**  int kndx = 0;  int tempCount = 0;  // Using stack to save the outer function (Add by 2019 Spring Defects Fixing Team)  Stack<Integer> kndxStack = new Stack<>();  Stack<Integer> tempCountStack = new Stack<>();  (adds line 1036-1038)  **Change the if statement in line 1035 from:**  if (cyclomaticComplexityObj.keyword.get(k).trim().startsWith("def"))  **Into:**  // If there is a function or return from inner function (Modified by 2019 Spring Defects Fixing Team)  if ((cyclomaticComplexityObj.keyword.get(k).trim().startsWith("def")  && cyclomaticComplexityObj.indentation.get(k) != -999) || tempCountStack.size() < kndxStack.size())  (adds line 1043-1045)  **Instead of getting kndx from k, resume kndx when necessary:**  // Save the def index  kndx = k;  // Resume the kndx if there is any (Add by 2019 Spring Defects Fixing Team)  if (tempCountStack.size() < kndxStack.size())  {  kndx = kndxStack.pop();  }  (line 1050-1055)  **End the loop when there is another def:**  // If there is function in function, end the loop (Add by 2019 Spring Defects Fixing Team)  if (cyclomaticComplexityObj.keyword.get(k).trim().startsWith("def"))  {  break;  }  // Increment the count for the def  tempCount++;  (line 1068-1073)  **Push current kndx and tempCount into stack:**  **Before:**  // Handle last entry if it meets the criteria  if (cyclomaticComplexityObj.indentation.get(kndx) < cyclomaticComplexityObj.indentation.get(k))  {  // Increment the count for the def  tempCount++;  // Set the indentation to -999 as we go so we know which ones we've counted  cyclomaticComplexityObj.indentation.set(k, -999);  }  **After:**  // Handle last entry if it meets the criteria  if (cyclomaticComplexityObj.indentation.get(kndx) < cyclomaticComplexityObj.indentation.get(k))  {  // If there is function in function, save current kndx and tempCount in stack (Add by 2019 Spring Defects Fixing Team)  if (cyclomaticComplexityObj.keyword.get(k).trim().startsWith("def"))  {  kndxStack.push(kndx);  tempCountStack.push(tempCount);  }  else  {  // Increment the count for the def  tempCount++;  // Set the indentation to -999 as we go so we know which ones we've counted  cyclomaticComplexityObj.indentation.set(k, -999);  }  }  (line 1084-1101)  **Pop tempCount out when ends a function:**  // Reset our index  k = kndx;  // Set the indentation to -999 as we go so we know which ones we've counted  cyclomaticComplexityObj.indentation.set(k, -999);  // If there is an outer funtion, resume the tempCount (Add by 2019 Spring Defects Fixing Team)  if (!kndxStack.isEmpty() && kndxStack.peek() != kndx)  {  tempCount = tempCountStack.pop();  k = kndxStack.peek();  }  (line 1048-1053) |

After using these strategies, most of the counting in Python will be correct. However, I notice that there are still some rows that have big differences between two versions. Like the table below:

A screenshot of a cell phone

Description automatically generated

Also, some main functions are recognized as a single space in Java version than in C++ version.

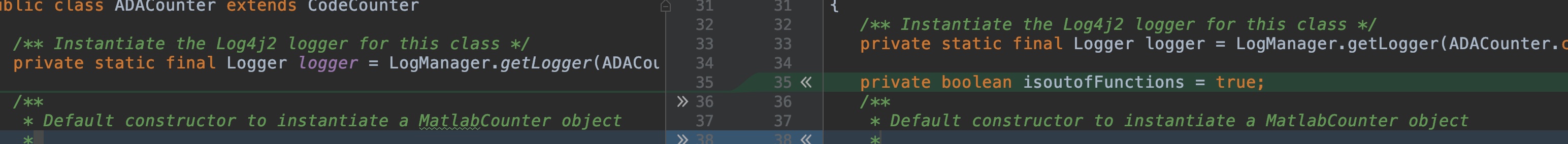
I’m trying to understand why this happened, and try to find how to fix the bug.

**Bug 9:**

This part is to integrated and modified three bugs for ADACounter. Firstly, the total functions are different between Cpp and Java versions. Secondly, the CC1 is different between two versions. Thirdly, the function name is different between two versions.

The modification is below:

(Left part is the original code that needs to be modified, and right part is the new code that will be integrated to the UCCG 1.3.1)

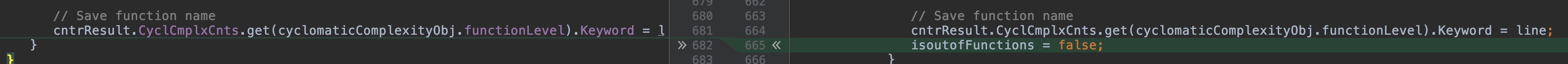


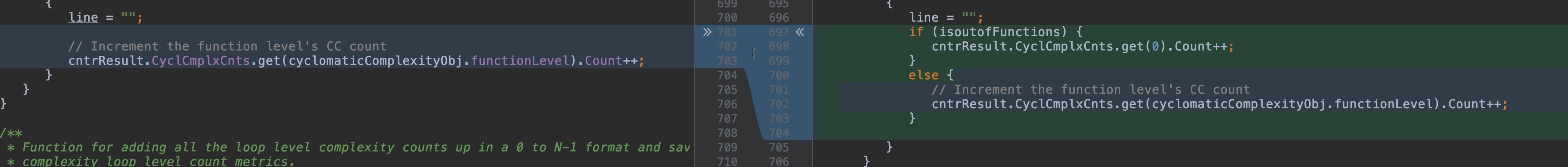
A close up of a sign

Description automatically generated

A picture containing electronics

Description automatically generated





A screenshot of a cell phone

Description automatically generated

**Bug 10:**

The counting rules from documents show that for web counters, there is no need to count duplicate lines if those lines are belong to other languages. The line does the operation is shown below:

A close up of a device

Description automatically generated

However, in ASP there are still bugs regarding counting physical and logical SLOC.

The reason is that ASPCounter counts asp codes, but in HTMLCounter, some asp codes are counted twice, like the one below:

**A picture containing building, text, photo

Description automatically generated**

In MultiLanguageHandler, some ASP starting tags and ending tags have all been set up.

A close up of text on a black background

Description automatically generated

However, when encountering ending tag %>, the ASP closure may not be terminated yet.

Like the example below:

A screenshot of a cell phone

Description automatically generated

It is just a tag inside the asp tag, but the system recognized it as the ending signal.

2019 Spring team treat that bug as a Java version deficiency, and partially fixed It by removing counting <% tags in asp. In the counting standards, there’s no file related to asp counting, and as a result, I decided to count only codes with asp tags as asp code, and the rest of the codes will be HTML.

Revision:

Before:

In MultiLanguageHandler.java:

A close up of text on a black background

Description automatically generated

After:

A screenshot of a cell phone

Description automatically generated

**Bug 11:**

HTML counts some of the asp codes which have already been calculated in ASPCounter, and I modified codes to filter these kinds of code. The modification is in HTMLCounter.java

Before revision:

A screenshot of a cell phone

Description automatically generated

After revision:

A screenshot of a cell phone

Description automatically generated

Results:

A screenshot of a computer

Description automatically generated

The total physical SLOC counted from ASPCounter and HTMLCounter will match the results in C++ version UCC. However, there is still problem that the logical SLOC counts are not correct.

**Bug 12:**

Javascript has some wrong math keywords in src/langprops/JavaScriptLanguageProperties.java. Every keyword should start with Math. The modification is below:

Before:



After:

A close up of a screen

Description automatically generated

**Bug 13:**

ColdFusion has no output file, and it is shown an error below:

A close up of text on a black background

Description automatically generated

When looked at the code, there’s something wrong with the en. The original code is below:

A close up of a sign

Description automatically generated

In deleteItemList, the deleteItem should be wrong item for the newline. The endPosition should already be the substring ending position, and we should not use endPosition + 1 for the substring method. Thus, the modified code is shown below:

A close up of a keyboard

Description automatically generated

By modifying the code, there will be some ColdFusion output files, and the result is correct comparing with the results in C++ version test results.

**Bug 14:**

ColdFusionScript cannot generate any output files. When ColdFusion’s bug has been fixed, ColdFusionScript still can not generate any output files. However, when running the UCC in C++, that version can’t generate any output files as well. The script will be devided into other languages because of the UCC Java language’s feature. A close look at the code should be in src/ucc/counters/handlers/MultiLanguageHandler.java. As a result, the bug is marked solved.

**Bug 15:**

A screenshot of a cell phone

Description automatically generated

From the above code, only files with a suffix like ‘.cfm’ will be counted as ColdFusion file, but actually suffix ‘.cfc’ is also file with ColdFusion language.

The modification for counting files with suffix ‘.cfc’ is below:

The modified file is ColdFusionLanguageProperties

Before:

A picture containing device

Description automatically generated

After:

