UCC Release Notes

UCC v.2018.07

1 Introduction

This document provides the release notes for the UCC v.2018.07. Unified Code Count (UCC) is a code metrics tool that allows the user to count physical and logical software lines of code, compare and collect logical differentials between two versions of source code of a software product, and obtain Cyclomatic Complexity results. With the counting capabilities, users can generate the physical, logical SLOC counts, and other sizing information such as comment and keyword counts of the target program. The differencing capabilities allow users to count the number of added/new, deleted, modified, and unmodified logical SLOC of the current version in comparison with the previous version. The Cyclomatic Complexity results are based on McCabe's research on this metric.

This release supports various languages including Ada, ASP/ASP.NET, Assembly, Bash, C/C++, C Shell, COBOL, ColdFusion, ColdFusion Script, CSS, C#, DOS Batch, Fortran, HTML, IDL, Java, JavaScript, JSP, Makefile, MATLAB, Midas, NeXtMidas, Objective C, Pascal, Perl, PHP, Python, Ruby, Scala, SQL, VB, VBScript, Verilog, VHDL, XML, and X-Midas. It also supports physical counting of data files.

2 Compatibility Notes

UCC v.2018.07 is released in C++ source code that allows users to compile and run on various platforms. This release has been tested on Windows using MS Visual Studio, Cygwin, and MinGW, Macintosh OS X and on Unix/Linux using the g++ compiler.

The UCC v.2018.07 does not support PL/1 and Jovial, although these may be included in future releases. For the need of counting of code in these languages, users may consider using the CodeCount Tools Release 2007.07, which does not provide the differencing capability but uses the counting rules compatible to those of UCC v.2018.07.

3 Requirements

Minimum Software Requirements:

- Compiler: a compatible C++ compiler that can load common C++ libraries including IO and STL, such as MS Visual Studio, MinGW, and g++.
- Qt 5.7.0 or Qt 5.10.1 and Qt Creator 4.0.2: Optional. Required to use the GUI front-end and allows faster performance using threads.
- Boost C++ Library: Optional. Maybe used if not building with Qt. Gives faster performance when using threads.

- Operating systems: any platforms that can compile and run a C++ application. The software has been tested on Unix, Linux, Solaris, Mac OS X and Windows XP, 7, 8 and 10.

Minimum Hardware Requirements:

- RAM: 512 MB. Recommended: 1024 MB.
- HDD: 100 MB available. Recommended: 200 MB available.

4 Changes and Upgrades

This section describes main changes and upgrades to the tool since the release v.2017.01.

1) Bug Fixes:
   a. Fixed issue where UCC was not working for Scala.
   b. Fixed CAdaCouter.cpp bug that accessed invalid stack elements.
   c. Fixed issues with incorrect counting of quotes.
   d. Fixed issue with differencing not returning values for Unmodified lines.

2) Feature Enhancements:
   a. Added Maintainability Index calculation. This feature is limited to these languages: C/C++, Java and C#.
   b. Added Function-level Differencing. To be used using the command line argument "-funcDiff". This has to be used along with -d option.
   c. Re-implemented the extfile functionality in the GUI to allow using and saving files with custom extensions to language mappings.
   d. New compilation flag added for g++ in Unix makefile.

5 Performance Notes

Due to the addition of Maintainability Index calculations, we have seen that Release v2018.07 runs slightly slower than the Release v2017.01. The below results are from a test setup on an Intel(R) Core(TM) i7-6500U CPU clocked at 2.50 GHz laptop running Windows 10 64-bit with 12 GB of RAM.

Run Time Comparison

2017.01 Run Time

Test: Differencing between Linux-4.7.10 and Linux-4.8.10 source files.

1 threads

Result: 4337 seconds or 72 minutes 17 seconds total time

2 threads

Result: 4204 seconds or 70 minutes 4 seconds total time

5 threads

Result: 2517 seconds or 41 minutes 57 seconds total time

2018.04 Run Time

Test: Differencing between Linux-4.7.10 and Linux-4.8.10 source files.

1 threads

Result: 4848 seconds or 80 minutes 48 seconds total time

2 threads

Result: 4648 seconds or 77 minutes 28 seconds total time

5 threads

Result: 2667 seconds or 44 minutes 27 seconds total time

6 Known Issues and Limitations

#	Issue
1	For JavaScript code, the tool does not count the statement that is not terminated by a semicolon.
2	The tool only detects and handles C# and VB as code-behind languages for the ASP.NET.
3	Users have reported that when large numbers of files or files with large SLOC counts are run, UCC would take several hours to process, or would hang. To improve the performance, users may choose to use the **`–nodup`** flag, which disables duplicate file separation; duplicates are counted and reported along with original files. In the situation where UCC hangs, the problem is that the host computer has run out of memory. A workaround is to break the input file list into several lists and process in multiple runs. Additional work is being done in this area, and more improvements may be available in a future release. If you suspect your process is hanging due to memory limitations, it would be appreciated it if you would report the number of files, total file size, and the host computer's memory size to UnifiedCodeCount@gmail.com.
4	The UCC is designed to process well-formed, compilable code, and does not check to see if the provided files are compilable. Files that contain software that is not compilable, or is in non-standard format, may not process correctly.
5	The Fortran counter uses the FORTRAN90 and above format for the continuation character being an & at the end of the line. FORTRAN77 and lower versions used a non-zero character in column 6 as the continuation character. There are plans to develop a separate counter for FORTRAN77 and lower in the future.
6	Due to the recent addition of cyclomatic complexity metrics, UCC is able to handle slightly smaller loads than the 2013.04 version. For counting purposes, errors can be avoided by using the **`–nocomplex`** flag. The UCC development team is working on optimizing the performance to handle larger input.
7	When function differencing is enabled, UCC performance is considerably slower than when the option is disabled.
8	Due to additional maintainability index calculation, UCC runs slower than Release 2017.01.