Unified Code Count – Java (UCC-J)

Release Notes

v.2018.05

May 2018

1. Introduction

This document provides the release notes for UCC-Java (UCC-J). Unified Code Count – Java (UCC-J) 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, Bash, C/C++, C Shell, COBOL, ColdFusion, ColdFusion Script, CSS, C#, DOS Batch, Fortran, Go, HTML, Java, JavaScript, JSP, Makefiles, MATLAB, NeXtMidas, Pascal, Perl, PHP, Python, R, Ruby, Scala, SQL, VB, VBScript, Verilog, VHDL, XML and XMidas. It also supports physical counting of data files.

This version is based on UCC-G version 1.1.1 from IAI (Integrity Applications Incorporated).

2. Compatibility Notes

UCC-J is released in Java source code that allows user to compile and run on various platforms. This release has been tested on Windows using Eclipse, MacOS and Linux environments using JDK 8.

UCC-J is also made available as an executable JAR file.

3. Requirements

Minimum Software Requirements

Below software packages need to be pre-installed for UCC-J to run:

- Operating Systems: any platforms that have the Java VM version 8 and run a Java application. The software has been tested on MacOS, Linux, Solaris and Windows.
- Java Runtime Environment version 8 or higher
- Eclipse Mars or later
- Log4j 2

Minimum Hardware Requirements

- The following hardware capabilities are needed to use UCC-J:
- RAM: 512MB

- Recommended JVM RAM Allocation for running UCC- G on baselines up to 1,000 files (~200 MB)
  - Minimum: 64MB
  - Recommended: 128MB
- Recommended JVM RAM Allocation for running UCC-J on baselines larger than 10,000 files (~2GB)
  - Minimum: 256MB
  - Recommended: 512MB

- Hard Disk Free Space:
  - 2x the space of the file list or directory for a counter run (for a 100MB file list, 200MB is required)
  - 2x the space of each file list or directory for a differencer run (for a 100MB directory A and a 150MB directory B, 500MB is required)

4. Features

This section describes the features of UCC-J in comparison with UCC C++ version and the features which are yet to be implemented.

UCC-J performs most of the functions that the latest UCC C++ version performs.

UCC -J supports –

1. Counting Capabilities – UCC-J allows users to measure the size of information of a baseline source program by analyzing and producing count for logical and physical SLOC, comments, compiler directive SLOC, keywords and complexity measures.

2. Differencing Capabilities – UCC-J allows user to measure and compare the differences between two baselines of source programs. These differences are measured in terms of the number of logical SLOC added/new, deleted/modified and unmodified.

3. Duplicate Matching – Two files are matched if they have the same filename regardless of which directories they belong to. Two files that have the same filename are matched if they have the least uncommon characters in their directory names. This feature allows users to handle to the situation where files are moved from one directory to another or the directory structure is changed. The remaining files are matched according to an algorithm that makes the most likely match.

4. Reports – A variety of reports are produced. The default report format is .csv, which will directly open into Excel, but plain text reports with the extension.txt can be specified by using the -ascii switch.

5. Counting and Differencing Directories – UCC-J allows users to count or compare source files by specifying the directories where the files are located. The capability eliminates difficulties in creating the file list that users may have encountered in the previous versions of the CodeCount toolset.

6. Various Programming Languages supported – The counting and differencing capabilities accept the source code written in the languages mentioned in Section 1.

7. Command Arguments – The tool accepts the user's settings via command arguments. Specifics of the command arguments are detailed in the UCC-J User Manual.

8. Complexity Count – UCC-J provides complexity counts for all source code files. The complexity counts include the number of math, trig, logarithm functions, calculations, conditionals, logicals, preprocessors, assignments, pointers and nested loops.

9. Cyclomatic Complexity Count – UCC-J provides cyclomatic complexity to measure the count of the number of linearly independent paths through the source code. The complexity would be 1 if there is no decision points (IF statements or FOR loops). Cyclomatic complexity is not supported for all the languages in UCC-J but support will be made available for all in the future.

10. Support for many file extensions – UCC-J determines the language used in a source file using the file extension. Please refer to the UCC-J User Manual for the list of languages and the file extensions supported for each.

- UCC-J GUI interface has not yet been implemented and will be worked on future releases.
- These are switches not implemented in UCC-J which are a part of the UCC C++ version
    1. -threads
    2. -visualdiff
    3. -legacy
    4. -ramlimit
    5. -nouncounted
    6. -nowarning
    7. -cc4enable
- COBOL and Objective C counting and differencing are not supported in UCC-J as compared to its support in the UCC C++ version.

5. Changes and Upgrades

This section describes main changes and upgrades to the tool since the release v.2017.01 and UCC-G v 1.1.1.

1) Bug Fixes:

- Corrected Physical Executable SLOC counts for all languages
- Corrected data keywords for Pascal
- Corrected executable keywords for Java
- Corrected conditional keywords handling for ADA, C++, C#, Java, Scala, SQL, and VB
- Corrected defect of escape quote being ignored in some cases for C++, C#, Java, and JSP

2) Feature Enhancements:

- Go Language support
- R Language support
- Maintainability Index support for C++ input

6. Known Issues and Limitations

#	Issue
1	For Ada code, the tool does not count “package body<> is” and “use<>” as data declaration statements.

Some anomalies with UCC-J have been discovered when compared with UCC C++ but the developers and testers have not found the exact root cause for these. Listed below.

Language	Anomaly
Ada	• Data declaration count • Execution count • Assignment count
ASP	• Blank line count • Execution count • Total # lines count
Assembly	• Comment count
Bash	• Comment count • Compiler directive • Assignment count • Calculation count

CPP	• Logical count
C#	• Execution count
DOS	• Assignment count • Preprocessor count
Java	• Calculation count • Conditional count
Pascal	• Calculation count
Perl	• Calculation count
Ruby	• Execution count
Scala	• Execution count
SQL	• Execution count
VHDL	• Execution count
Verilog	• Execution Count