



User Manual

UCC v.2009.10

**Copyright (C) 1998 - 2009
University of Southern California
Center for Systems and Software Engineering**

Table of Contents

USER MANUAL	I
TABLE OF CONTENTS	II
1. Introduction	1
1.1 Product Overview	1
1.2 System Requirements	1
1.2.1 Hardware	1
1.2.2 Software Operating Systems	1
1.2.3 Compilers Supported	2
2. Installation	2
2.1 Compilation	2
2.1.1 Visual Studio	2
2.1.2 g++	3
2.2 Execution	3
2.2.1 Command Line Specification	3
2.2.2 Details and Examples	4
2.3 Output Files	6
3. Counting Standards	7
4. Terminology Explanation	7
4.1 Basic Assumption and Definitions	7
4.1.1 File Extensions	7
4.1.2 Data Files	7
4.1.3 Source Files	7
4.1.4 SLOC Definitions and Counting Rules	7
4.1.5 TAB	8
4.1.6 Blank Line	8
4.1.7 Total Sizing	8
4.1.8 Keyword Count	8

1. Introduction

This document provides information for using the UCC tool version 2009.10.

1.1 Product Overview

Most of software cost estimation models including the COCOMO model require some sort of sizing of software code as an input. Ensuring consistency across independent organizations in the rules used to count software cost code is often difficult to achieve. To that end, the USC Center for Systems and Software Engineering (CSSE) has developed and released a code counting toolset called CodeCount to support sizing software code for historical data collection and reporting purposes. This toolset is a collection of tools designed to automate the collection of source code sizing information. It implements the popular code counting standards published by SEI [1] and adapted by the COCOMO model [2]. Logical and physical source lines of code (SLOC) are among the metrics generated by the toolset.

Unified CodeCount (UCC) is a unified and enhanced version of the CodeCount toolset. It is a code counting and differencing tool that unifies the source counting capabilities of the previous CodeCount tools and source differencing capabilities of the Difftool (which is now replaced by UCC). It allows the user to count, compare, and collect logical differentials between two versions of the source code of a software product. 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. 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 UCC tool is provided in C++ source code only, and may be used as is, or modified and further distributed subject to certain limitations. The user is responsible for compiling and using the executable version.

1.2 System Requirements

1.2.1 Hardware

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

1.2.2 Software Operating Systems

- Linux 2.6.9
- Unix
- Mac OS X
- Windows 9x/Me/XP/Vista
- Solaris

1.2.3 Compilers Supported

- MS Visual Studio 2003, 2005, 2008
- g++
- Eclipse C/C++

2. Installation{ XE "Installation" }

There is no setup package provided for installing the tool. The user can download the source files from the following URLs:

- a. CSSE's affiliate members: <http://sunset.usc.edu/csse/affiliate/private/>
- b. Public users: <http://sunset.usc.edu/research/CODECOUNT/index.html>

2.1 Compilation

The tool can be compiled using a C++ compiler. The compiler must support common C++ libraries including IO and STL. Below are typical steps for compiling the tool using Visual Studio or the g++ compiler. The procedure for compiling the tool using Eclipse C/C++ would be the same as typical C++ programs.

2.1.1 Visual Studio

On PC based machines, the user can use Visual Studio 2003, 2005 and 2008 to compile the source code by following procedure below:

1. Create an empty project of Project Type Visual C++ and Template Win32 Console Application. Type in the Project name and Select OK. In the Win32 Application Wizard, select Applications Settings and then select "Empty Project" check box.
2. Select Project/Add Existing Item. Locate and select all UCC source code files. Click on "Add" to add the selected files. This would add the existing code to the created project.
3. Open the Properties window and go to the Configuration mode page. The user should select "Release" mode for compiling. To choose the mode, Click on Build/Configuration Manager button. Select "Release" from the list "Active Solution configuration". Go to the C/C++ section and click "Precompiled Headers". Make sure the "Create/Use Precompiled Header" selection is "Not Using Precompiled Headers". (*)
4. Select Build Solution or use the shortcut Ctrl+Shift+B to compile.

Upon compilation, an executable file will be created in the *release* folder.

(*) Please note that Visual Studio by default uses the precompiled header option. The user will get error messages if this option is not turned off.

2.1.2 g++

The following command will compile source files stored in the folder *src*

```
g++ ./src/*.cpp -o UCC -DUNIX
```

Note: the command line option **-DUNIX** must be provided on UNIX system.

2.2 Execution

2.2.1 Command Line Specification

```
ucc [-d [-i1 fileListA.txt] [-i2 fileListB.txt] [-t #]] [-cf] [-tdup #] [-trunc #] [-dir dirA  
[dirB] filespecs]
```

Where,

- d If specified will run the differencing function.
- i1 fileListA.txt Filename containing filenames in the *Baseline A* to be compared. The file is in the plain text format, one filename per line.
- i2 fileListB.txt Filename containing filenames in the *Baseline B*. Normally, *Baseline B* is the newer or the current version of the program, as compared to *Baseline A*. The file format is same as *Baseline A*.
- t # Specify the modification threshold, the percentage of common characters between two lines of code to be compared over the length of the longest line. If two lines have the percentage of common characters equal or higher than the specified threshold, they are matched and counted as modified. Otherwise, one SLOC deleted and one SLOC added are counted. The valid values range from 0 to 100 and default to 60 (same as *-t 60*).
- dir Specify directories containing files to be counted and/or compared. If this argument is provided, the input list files (see below) are ignored.

dirA	Name of the directory. If the option <i>-d</i> is provided, dirA is the directory of <i>Baseline A</i> . Otherwise, it specifies the directory to be counted (with the counting function).
dirB	Name of the directory of <i>Baseline B</i> , only if the option <i>-d</i> is given.
filespecs	Specifications of file extensions to be counted/compared; wildcard chars ? * are allowed. Use a space between two <i>filespecs</i> , for examples, *.cpp *.c
-cf	Support handling Rational ClearCase filenames. The Rational ClearCase application appends version information to the filename, starting from '@@'. This option allows the UCC tool handle the original filename instead of the ClearCase-modified filename.
-tdup #	Specify the threshold percentage for duplicated files of the same name. This specifies the maximum percent difference between two files of the same name in a baseline to be considered duplicates. By default, this threshold is zero, so the files must be identical by logical SLOC – spacing and comments are not considered.
-trunc #	Truncate threshold, specifying the maximum number of characters allowed in a logical SLOC. Additional characters will be truncated. The default value is 10,000, and zero is for no truncation.

Note: if no argument is given the tool reads and counts all files listed in the text file *filelist.dat*. Details on this file are described in Section 2.2.2.1.

2.2.2 Details and Examples

2.2.2.1 Counting Source Files

The counting function is executed using the command line with no *-d* switch. There are two alternative ways to specify the source files of the target program: using the *-dir* switch and using the file list (*filelist.dat*).

1. Using the *-dir* command line switch

```
UCC -dir project1 *.cpp *.h *.c *.hpp
```

This command allows the tool to count all C/C++ source files contained in the folder *project1*.

2. Using *filelist.dat*

```
UCC
```

This command allows the tool to find the file *filelist.dat* in the working directory and count all source files listed in it.

The file *filelist.dat* contains a list of source files to be counted, one filename per line. You can create the file *filelist.dat* using one of the following commands

- Unix:
ls -l <filespecs> > filelist.dat
or, use the following to obtain full directory/pathname specification
find [*Directory*] -name '<filespecs>' > filelist.dat
to append this file with additional filenames use:
find [*Directory*] -name '<filespecs>' >> filelist.dat
- MS-DOS:
dir/B > filelist.dat [*Directory*]\<*filespecs*>
or, use the following to obtain full directory/pathname specification
along with files in all subdirectories:
dir/B/S > filelist.dat [*Directory*]\<*filespecs*>
to append this file with additional filenames use:
dir/B/S > filelist.dat [*Directory*]\<*filespecs*>

Where,

- *Directory* is the directory name relative to the current working directory. *Directory* is optional, and it is not given, the working directory is implied.
- *filespecs* is the file specifications, and wildcard chars ? * are allowed. Use a space between two filespecs, for examples, *.cpp *.c

2.2.2.2 Differencing Baselines

In this function, source files in the baselines will be matched and compared to determine the counts for SLOC added, deleted, modified, or unmodified.

To run this function, UCC must be called with the *-d* switch.

1. Using List Files

Compare source files of Baseline A and Baseline B contained in files *fileListA.txt* and *fileListB.txt*.

UCC -d

By default, the files *fileListA.txt* and *fileListB.txt* contains source filenames in *Baseline A* and *Baseline B*, respectively. You can specify different filenames by using the *-i1* and *-i2* command line switches.

UCC -d -i1 fileA.txt -i2 fileB.txt

(Since the format of these files is the same as *filelist.dat*, you can use the commands described above to create them).

2. Using the `-dir` command line switch

UCC -d -dir code1.0 code1.2 *.cpp *.c *.hpp *.h

This command allows the tool to match and compare all C/C++ source files contained in the folder *code1.0* and *code1.2*.

2.3 Output Files

Filename	Function	Description
<LANG>_outfile.txt	Counting	Main count results for source files of <LANG>. <LANG> is the name of the language of the source files, e.g., C_CPP for C/C++ files and Java for Java files.
outfile_cplx.txt	Counting	Complexity results.
Duplicates-outfile_cplx.txt	Counting	Complexity results for duplicated files.
DuplicatePairs.txt	Counting	A text file listing matches between a file and its duplicate file.
Duplicates-<LANG>_outfile.txt	Counting	List of duplicate files for the language <LANG>.
outfile_diff_results.txt	Differencing	Main differencing results in the plain text format.
outfile_diff_results.csv	Differencing	Main differencing results in .csv format that can be opened using MS Excel.
Baseline-<A B>-<LANG>_outfile.txt	Differencing	Count results for source files of <LANG> for <i>Baseline A</i> and <i>Baseline B</i> .
Baseline-<A B>-<LANG>_cplx.txt	Differencing	Complexity results for <i>Baseline A</i> and <i>Baseline B</i> .
MatchedPairs	Differencing	A text file listing matches between files in <i>Baseline A</i> and <i>Baseline B</i> .
Duplicates-<A B>-<LANG>_outfile.txt	Differencing	List of duplicate files in <i>Baseline A</i> and <i>Baseline B</i> .
Duplicates-<A B>-<LANG>_cplx.txt	Differencing	Complexity results for duplicate files in <i>Baseline A</i> and <i>Baseline B</i> .
error_log_<mmddyyyy>_<time>	Both	Log file listing errors that occur at the time specified by <mmddyyyy> and <time>

3. Counting Standards

Refer to the counting standard documents included in the release.

4. Terminology Explanation

4.1 Basic Assumption and Definitions

4.1.1 File Extensions

The tool determines the language in a source file using its file extension. This version supports the following languages and file extensions:

Languages	File Extensions
C/C++	.cpp, .c, .h, .hpp, .h, .cc, .hh
C#	.cs
Java	.java
SQL	.sql
Ada	.ada, .a, .adb, .ads
Perl	.pl, .pm
ASP, ASP.NET	.asp, .aspx
JSP	.jsp
CSS	.css
HTML	.htm, .html, .shtml, .stm, .sht, .oth, .xhtml
PhP *	.php
JavaScript	.js
VB	.vb, .frm, .mod, .cls, .bas
VbScript	.vbs

(*) **Note:** this version does not support differencing PhP source code; the differencing results do not include the counts for PhP files.

4.1.2 Data Files

Data files shall contain only blank lines and data lines. Data lines are counted using the physical SLOC definition.

4.1.3 Source Files

Source code files may contain blank lines, comment lines (whole or embedded), compiler directives, data lines, or executable lines. Source code files have to be compiled successfully to ensure the integrity of the inclusive syntax.

4.1.4 SLOC Definitions and Counting Rules

Please refer to the counting standard documents.

4.1.5 TAB

A Tab character is treated as a blank character upon input.

4.1.6 Blank Line

A blank line is defined as any physical line of the source file that contains only blank, Tab, or form feed characters prior to the occurrence of a carriage return (EOLN).

4.1.7 Total Sizing

The total sizing of analyzed source code files in terms the SLOC count contains the highest degree of confidence. However, the sizing information pertaining to the sub classifications (compiler directives, data lines, executable lines) has a somewhat lower level of confidence associated with them.

Misclassifications of the sub classifications of SLOC may occur due to:

- (1) user modifications to the UCC tool,
- (2) syntax and semantic enhancements to the parsed programming language,
- (3) exotic usage of the parsed programming language, and
- (4) integrity of the host platform execution environment.

Additionally, in some programming languages a single SLOC may contain attributes of both a data declaration and an executable instruction simultaneously. These occurrences represent events beyond the control of the UCC tool designer and may cause the inclusive parsing capabilities of the tool to misclassify a particular SLOC. For these reasons, the counts of sub-classifications should be regarded as an approximation and not as a precise count. In only the physical SLOC definition does the sum of the sub-classification counts equal the total physical SLOC count.

4.1.8 Keyword Count

The search for any programming language specific keywords over a physical line of code for purposes of incrementing the tally of occurrences shall include the detection of multiple keywords of the same type, e.g., two occurrences of the keyword READ on the same physical line.

The search for any programming language specific keywords over a physical line of code for purposes of incrementing the tally of occurrences shall include the detection on multiple keywords of different types, e.g., occurrences of keywords READ and WRITE on the same physical line.

Keywords found within comments (whole or embedded) or string literals shall not be included in the tally count.

References

- [1] R.E. Park, “Software Size Measurement: A Framework for Counting Source Statements”, Technical Report CMU/SEI-92-TR-20 ESC-TR-92-020, 1992

- [2] B. Boehm, C. Abts, S. Chulani, “Software development cost estimation approaches: A survey”, *Annals of Software Engineering*, 2000.

- [3] V. Nguyen, S. Deeds-Rubin, T. Tan, B. Boehm, “A SLOC Counting Standard”, *CSSE Tech Report*. DOI = <http://csse.usc.edu/csse/TECHRPTS/2007/usc-csse-2007-737/usc-csse-2007-737.pdf>