<Objective-C> Final Report

<Summer 2016>

<08/03/2016>

# Team Members Information

<List the team members below by full names. In the “Attended Classes/Meetings Remotely?” sections, the answers should be one of: Never, Sometimes, Mostly, Always. >

|  |  |  |
| --- | --- | --- |
| Team Members | Attended Classes Remotely? | Attended Meetings Remotely? |
| XiangYu Li | Never | Never |
| Xiao Hu | Never | Never |
| Yu Jiang | Never | Never |

# Project Objective

The requirements/expectations of this project are:

1. Update Objective C counter implemented previously to be compatible with 2014.08zf version of UCC.
2. Add Cyclomatic Complexity, and any other new features that would apply to the language counter.
3. Consider implementation options to make this counter available to all users, without confusing with C++ counter.
   1. For instance, only specify Objective C specific extensions as default within UCC. If there are files that have shared extensions as other languages, users must specify to count as Objective C in extfile.
4. Also integrate Objective C counter to the 2015.12 version.

We completed all of the four tasks.

# Project Summary

In this Directed research project, we updated Objective-C counter implemented previously to be compatible with 2014.08zf and 2015.12 version of UCC. Meanwhile, we added the Cyclomatic Complexity 1, 2 and 3 for the Objective-C counter. We also solved the conflicts of extension files among Objective-C, C\_CPP, Matlab and NeXtMidas by specifying the extension file (see the detail in User Manual Section 10.4). Besides, we added the exclusive keywords of Objective-C for the Objective-C counter. Finally, we did the regression tests from 2011 test cases and new feature tests for 2014.08 and 2015.12 baselines.

## Test Summary

The major categories of tests that we wrote are:

Counting Test cases:

The UCC project is aimed at segregating and identifying the number of logical as compared to number of physical lines in a code base. The counting test cases were mainly aimed at testing various scenarios in which the number of logical lines might or might not vary with the physical lines of code.

Differencing test cases:

These test cases were aimed at testing the differencing module of Objective-C. The major test cases involved in this were testing whether the module is able to correctly identify the code difference in the files. The differencing module compares source files by specifying the directories where the files are located. It generates how the two files differ in code on the basis of “number of new lines”, “number of lines deleted”, “number of lines modified”. This allows us to get an idea as to how the files differ from each other and is highly useful to compare two different versions of the same file.

Stress Testing:

The stress testing focused on testing the code to its limits. The code mainly consisted of performing the count function on a code with 10,000+ lines which are normal in any development environment.

Keywords testing:

This test aimed at testing all the possible keywords that are present in the Objective-C language. This helps us to check if all the keywords are being identified by the code or not.

# Projects’ Decisions

<Describe the pros and cons of decisions you have made, and why you made the decision you did for the project. Could be design decisions, choice of algorithm, etc. The goal is to avoid future teams from choosing a wrong decision or wasting time doing research and analysis on something that has already been done before. >

There are different implementations of Code Counter in different versions, and we would like to use its super class implementation as standard discipline to tune Objective-C Counter from 2011 version.

The reason we made decisions is based on our multiple trials, we made UCC Objective-C Counter consistent with other Counters (C\_CPP Counter, Java Counter, etc.) in both 2014 and 2015 version respectively. Further development teams will only need to consider shared bugs in different counters.

The pros of our decisions:

1. Inherently be compatible to each version without introducing new bugs to baseline.

2. Easy to add new features based on well-tested baseline.

3. Reduce the duplication of code.

4. Increase the scalability and reusability of the project.

The cons of our decisions:

1. Objective-C is the strictly super set of C language. We cannot directly fix bugs in Objective-C Counter regardless of C Counter.

# Known Bugs/Issues

**Bug1:**UCC counter can only count “#define” and “# define”. It can’t count this line if there is two or more space between ‘#’ and “define” (like “# define”).

We tested this case. The result is that “# define” compiled and the UCC counter can’t count this line.

**Bug2:**

For the Data Declaration lines counter, the counter can’t count it with following format:

special character is close to the data type declaration (no space): (int), (void)

Example:

1. int main(int a){ }: out file would output only 1 int.

2. int main( int a){ }: out file would output 2 int.

**Bug3:**

UCC 2015 Objective-C Cyclomatic Complexity Function-Name parsing error locating in UCCFilesOut.cpp, it will parse function name again, and it masks our Objective-C Counter to parse Function-Name.

**Issue1**:

There are too many key words to add completely. We only added the Objective-C keywords as much as we can.

**Issue2:**

UCC 2014 Objective-C Cyclomatic Complexity FunctionName parsing error has be fixed, but regression test has not been proceeded.

**Issue3:**

UCC 2015 CBatchCounter has some deleting pointer syntax warnings, it may cause potential memory leaks.

# Development Notes

<In the table below, list the files and functions you added, modified, deleted, and give an explanation – such as, what new functionality did you add, what bugs did you fix, etc. Be sure to specify the class/file before the functions in the list. >

|  |  |  |  |
| --- | --- | --- | --- |
| **General Description of New Feature or Code Modification** | | | |
| Made the Objective-C counter compatible with 2014.08 and 2015.12 version | | | |
| **New Files** | **Description** | **New Functions** | **Description** |
| CObjCCounter.cpp | Add OBJC counter |  |  |
| CObjCCounter.h | Add OBJC counter |  |  |
| **Modified Files** | **Description** | **Modified Functions** | **Description** |
| MainObject.cpp | Initialize OBJC counter | MainObject() | Add the initialization line |
| MainObject.h | Include the OBJC head file | None | None |
| cc\_main.h | Add class type of enumeration | None | None |
| LangUtils.cpp | Initialize OBJC counter | Init\_CounterForEachLanguage() | Initialize OBJC counter |
| **Deleted Files** | **Description** | **Deleted Functions** | **Description** |
| None |  |  |  |

# Set Up and Improve Instructions

The OBJC shares the .h file with C\_CPP, the .m file with MATLAB and the .mm file with NeXtMidas. The user needs to use –extfile <extfilename> to specify the extension files which OBJC should associate with.

Extension file should be written as:

C\_CPP=

MATLAB=

NeXtMidas=

OBJC=.h,.m,.mm

It means all of the .h, .m and .mm extension files would only map to OBJC.

# Final Project Plan

<Give a week-by-week summary of project timeline. Be sure to specify when you began development, testing, research, analyses, and other activities. >

Week1: N/A

Week2 (6/2/16~6/8/16): Take familiar with framework and get clear about where to add funtionalities. Read the source code and documents of 2010 and 2011.

Week3 (6/9/16~6/15/16): Take familiar with framework and get clear about where to add funtionalities. Read the source code and documents of 2010 and 2011.

Week4 (6/16/16~6/22/16): Fix the linking error and make a test plan for 2014 version Objective C.

Week5 (6/23/16~6/29/16): Do regression test with 2014 baseline and make sure Objective C funtions work well.

Week6 (6/30/16~7/6/16): Do additional new tests for 2014 Objective C functions.

Week7 (7/7/16~7/13/16): Make the code compatible to 2015 and fix bugs.

Week8 (7/14/16~7/20/16): Do regression test with 2015 baseline and make sure Objective C funtions work well.

Week9 (7/21/16~7/27/16): Do new feature tests from 2014 for 2015 baseline.

Week10 (7/28/16~8/3/16): Final report.

:

# Team’s Strengths and Weaknesses

<Describe team members’ strengths and weaknesses, as well as how team worked together and the team’s strengths and weaknesses. >

Team members’ strengths:

1. on time

2. Learning agility and effective personal communication skills

3. Self-motivated and determined

4. Hardworking

Team members’ weaknesses:

1. Being a Debater

We meet regularly every week and work together.

# Final Deliverables Checklist

* Completed Final Report
* Copy of original/baseline source code (before modifications) – be sure it is the code you used to make modifications to
* Final source code with modifications
* Final version of Test Cases
* Final version of Defect Log
* Final version of all other documents
* Other. Explain: