User

MANUAL

Automated Metric Collection Tool

Tortoises

Fall 2019

USER'S MANUAL

TABLE OF CONTENTS

Page #

1.0 GENERAL INFORMATION

A. GENERAL INFORMATION

1.1 Project Overview

In the working software development industry, it is difficult for management to accurately assess developers and compare their work with the overall progress of a project. There exists tools such as UCC, GCOV, and Cppcheck to assess the code produced by developers, producing a variety of metrics. The problem with the approach of using these tools is that they must be executed manually, one by one. The goal of this project was to eliminate that manual execution and gather all these metrics automatically. Once gathered, the metrics could be later accessed by management in order to add labor hours and generate different kinds of reports.

1.2 Project References

Included here are references that were used for the creation of this project and may be of use for future changes and development of this project.

https://csse.usc.edu/ucc_new/wordpress/
https://gcc.gnu.org/onlinedocs/gcc/Gcov.html
http://cppcheck.sourceforge.net/
https://jenkins.io/

1.3 Acronyms and Abbreviations

Included here are abbreviations that are going to be used within this document.

GCOV:	Test coverage program
UCC:	Unified Code Count
Cppcheck:	C/C++ analysis tool
SLOC:	Source lines of code
Git:	A version-control system for tracking changes in code during development

1.4 Tools Overview

The tools used in this project were UCC, GCOV, Cppcheck, and Jenkins. With conjunction from our tool they allowed for all the metrics to be created.

1.4.1 UCC

UCC is a counting tool that will scan source files for many different metrics, most importantly, lines of logical and physical SLOC. A SLOC is a line of code other than a comment that serves a function or a purpose within the code. SLOC's will be needed as a metric in order to calculate a developers SLOCs per

hour given their labor hours on the project. This is one of the most important metrics collected by this project since it can track a quantitative number to define how much work a developer is doing.

1.4.2 GCOV

GCOV is a compilation code analyzer that will generate metrics on how many times an exact line of code is executed throughout the program and provide percentage of code coverage. This is useful to tell how efficient the code that the developer has written, is.

1.4.3 Cppcheck

Cppcheck is a static code analyzer for C/C++ code. This tool will enable us to gather information on if the code submitted by the developer contain any memory leaks, mismatching allocation-deallocation, buffer overrun, and many more system critical errors that can cause headaches in the long run or compromise system security to certain attacks. This will allow the tracking of any errors or problems within the code so when trying to fix an issue, developers can see what push the error was created during. Allowing for quick locating and fixing of any errors or messages that Cppcheck may throw.

1.4.3 Jenkins

Jenkins is a free and open-source automation server that can be used for many things such as automated tasks and compiling projects in many instances at once. The feature that is taken advantage of, in Jenkins, is the continuous integration feature. Continuous integration is when multiple instances pull the code from the repository as soon as Jenkins detects new submission. This will enable us to execute the hook script whenever new code is submitted to the repository in many instances in order to provide the developer with instant feedback rather than wait for a scheduled time that Jenkins would run.

2.0 SYSTEM SUMMARY

B. SYSTEM SUMMARY

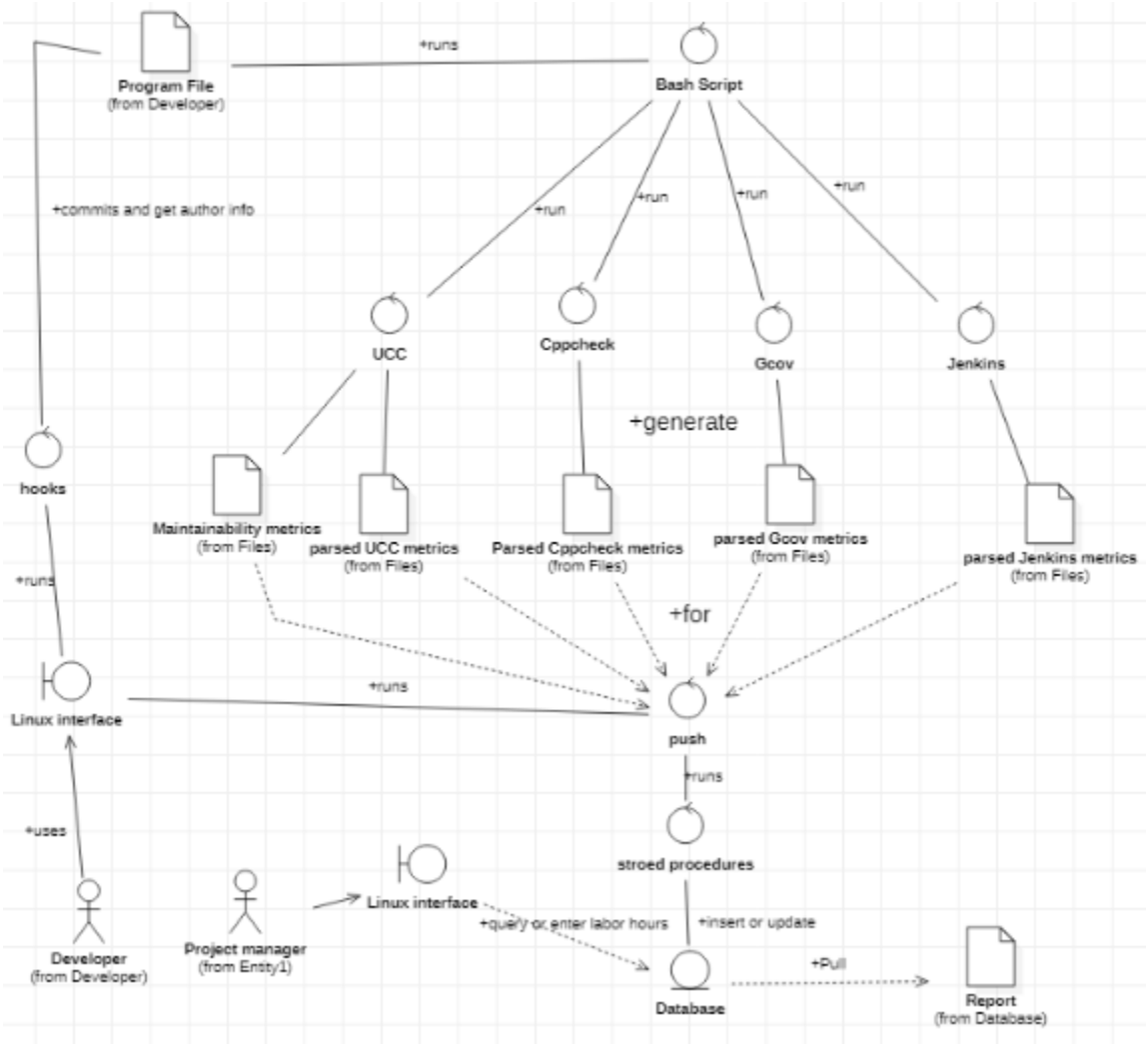
2.1 System Information

The project is very specific in the steps that it takes.

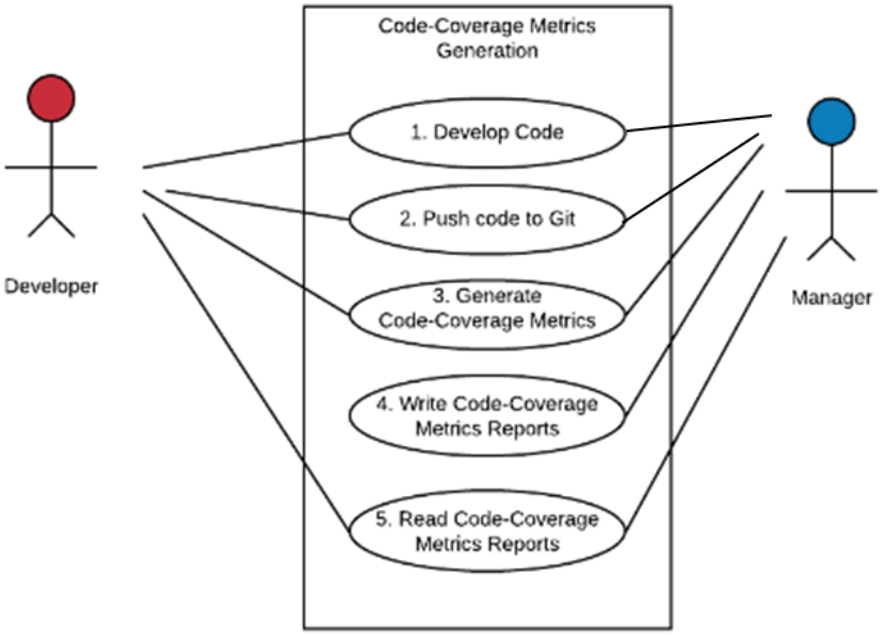
1. The developer uses a Linux interface
2. They develop their code and make any changes that they feel the need to do.
3. They then get ready to commit their changes.
    a. Here they give their commit a tag called “metrics”
    b. They then commit their code to the git repository
4. While this commit is happening, a hook is automatically activated
    a. Jenkins notices a change within the repository and will run its metrics collection as well
5. The hook calls upon the files that will make the metric collection tools run
6. Each metric tool runs and parses its data so that it is ready to be read to the database
7. The data is then stored into the database with the corresponding commit
8. The project manager can later come in and give each commit labor hours
9. They are also able to generate reports
    a. Single commit report
    b. Date range report
    c. All commits report
10. Anyone can read these reports since they generated into an excel file

2.2 Data Flows

Below is a data flow chart to show exactly how the system works and what is going on behind the scenes during each step of the project.



2.3 User Access Levels



2.3.1 Developers

Developers can:

- Develop and make changes to code
- Prepare code to be pushed
- Push these changes to Git

This will then automatically generate all the metrics and store them into the database,

2.3.2 Project Managers

Project managers can do everything a developer can do, plus:

- Give a certain commit, how many hours were worked on that commit
- Generate three types of reports
    - Single commit report – All of the metrics and information about a single commit
    - Date range report – All of the metrics and information about commits within a certain date range
    - All commit report – All of the metrics and information about all commits that are in the database currently

3.0 INSTALLATION

C. INSTALLATION

3.1 Initial Setup

Each machine must already have access to a Linux server and the Git repositories. Below shows where each file needs to go for the project to run.

On each repo on each developer machine:
pre-push goes in $PROJECT_DIRECTORY/.git/hooks
gen_ucc_xml goes in $PROJECT_DIRECTORY/.git/hooks/pre-push-commands

On management machines:
labor_hours goes into /usr/bin
report goes into /usr/bin

For Jenkins:
jenkins.sh goes into /var/lib/jenkins

All these files need execution permission flag set.

On each developer machine to setup metrics database connection this must be run:

mysql_config_editor set --login-path=metrics
--host=[IP_ADDRESS] --user=[USERNAME] --password

3.2 Tool Installation

Each tool will need to be installed correctly on each machine in order to be used.

3.2.1 UCC Installation

Within the UCC manual there are specific instructions to install it since there is not setup package provided for installing the tool.
You can download the source files from: http://csse.usc.edu/ucc
After this you must compile the tool and upon compilation, an executable will be created in the *Release* folder.
This will install UCC and allow you to use it with any code.

3.2.2 GCOV Installation

GCOV should already be part of the Linux server, but if it is not, enter the command:
sudo apt-get install gcov
This will install GCOV and allow you to use it with any compiled and run code.

3.2.3 Cppcheck Installation

Cppcheck should already be part of the Linux server, but if it is not, enter the command:
sudo apt-get install cppcheck
This will install Cppcheck and allow you to use it with any C/C++ code.

3.3 Script Installation

As stated above, script installation will require the following steps:
On each repo on each developer machine:
pre-push goes in $PROJECT_DIRECTORY/.git/hooks
gen_ucc_xml goes in $PROJECT_DIRECTORY/.git/hooks/pre-push-commands

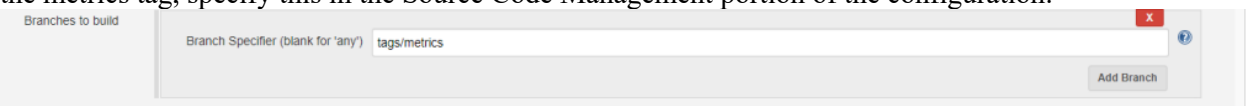
On management machines:
labor_hours goes into /usr/bin
report goes into /usr/bin

For Jenkins:
jenkins.sh goes into /var/lib/jenkins

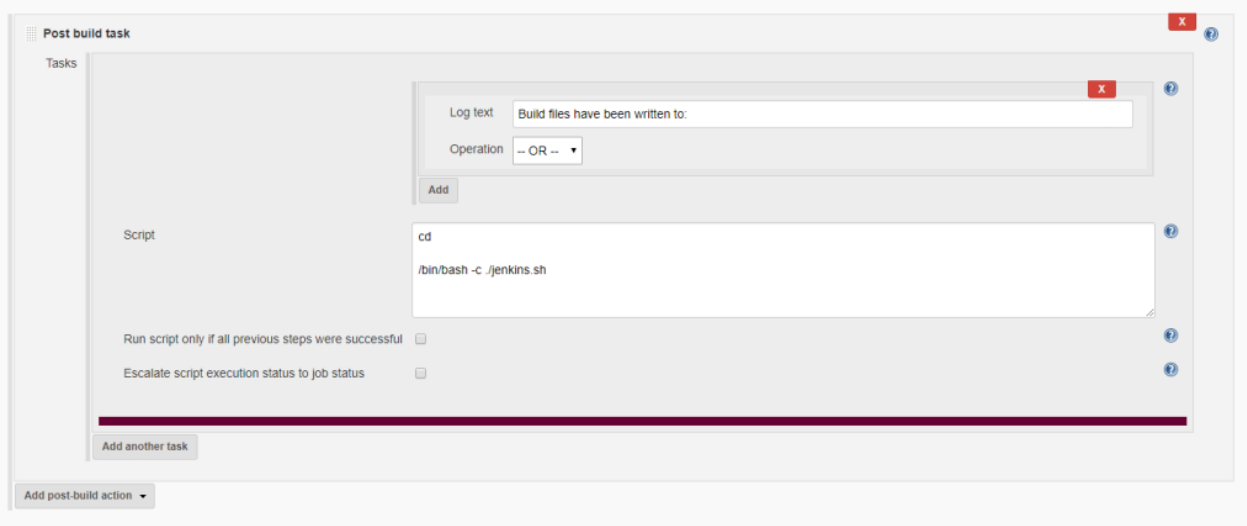
All these files need execution permission flag set.

3.4 Jenkins Installation

Jenkins has a file called jenkins.sh which will need to be edited to replace [USERNAME] [PASSWORD] [IP_ADDRESS] with the correct database connection information. This file is secure. After doing this, you will need to set Jenkins up via their web browser application. To make Jenkins only build on commits with the metrics tag, specify this in the Source Code Management portion of the configuration:



To execute the Jenkins script that will generate the metric of a build being successful or not, configure a post build task as the image show, the Log text field should contain text that would only appear in the build log should a build finish successfully. This is different per project and cannot be explicitly stated in the user manual.

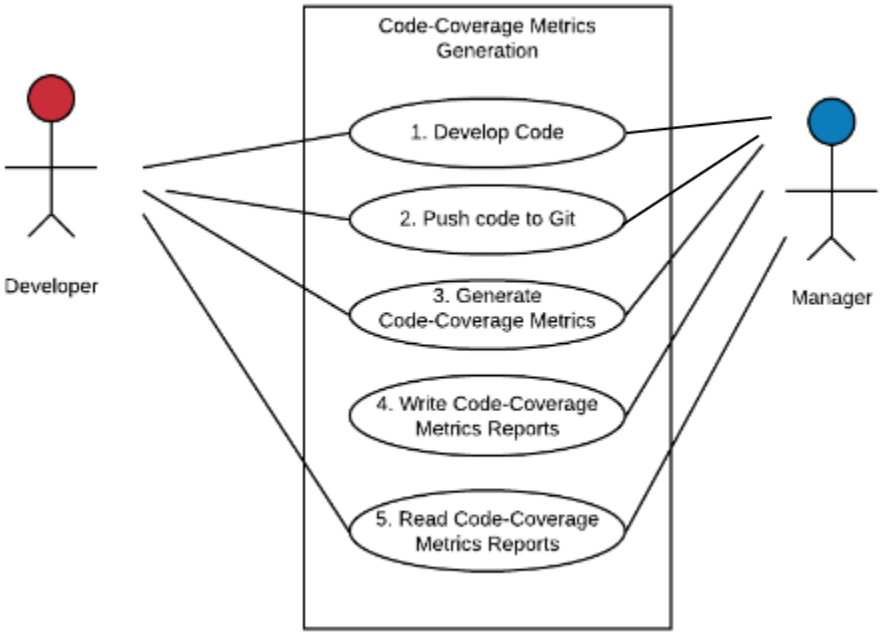


Jenkins requires the git plugin as well as the post build task plugin

4.0 USING THE SYSTEM

D. USING THE SYSTEM

There are two types of users when it comes to this project. A developer, who can develop code, push code to Git, and indirectly generate all the metrics that this project is able to collect. A project manager, who can do everything a developer can do, as well as add labor hours to a commit and create the metric reports.



4.1 Developer Use

The developer is the main drive to make this project run. They are the initializer when it comes to metrics collection since it runs upon their setup and commit. For the developer to be able to allow the metrics tools to run in the background, see C. INSTALLATION.

Since GCOV is a finicky program, the code must be compiled with certain flags and run at least once for its metrics to be generated:
g++ -ftest-coverage -fprofile-arcs SOURCE_FILES
./MAIN_FILE
It will then run perfectly fine.

Otherwise, they simply need to add the metrics tag to their commit. This can be done by:
git tag -f metrics
They then can push their code to the repository:
git push –tags origin master
It will then require them to log in.

This is all the developer needs to do. The program will run everything else automatically. When the project manager creates the reports, they may share them with the developer if they wish.

4.2 Project Manager Use

Since the project manager can do everything that the developer can do, see 4.1 Developer Use for more information. As for extra capabilities, see below:

4.2.1 Labor Hour Tool

For the project manager to add labor hours to each commit they must have installed the labor hours script correctly, See 3.1 Initial Setup.

To add labor hours to a single commit, the project manager must type:
./labor_hours
Here they will be prompted to log in.
Which will then bring up a list of all the commits with no labor hours currently stored.
They will then type the corresponding number to the commit that they would like to add hours to.
Finally, they can type how many hours they would like to add to this commit.
After typing the number hours, the labor hours will be stored with that commit.

For example:

```
ubuntu@ip-172-31-48-19:~/HookCode$ ./labor_hours
Email: ubuntu@ip-172-31-48-19.ec2.internal
Pulling commits for ubuntu@ip-172-31-48-19.ec2.internal
+-----------+------------------------------------------+---------------------+-----------------+-------------+
| COMMIT_ID | COMMIT_HASH                              | COMMIT_DATE         | FILES_COMMITTED | AUTHOR_NAME |
+-----------+------------------------------------------+---------------------+-----------------+-------------+
|        27 | c9e16ed50545772afe5d2398336951311a4a808f | 2019-11-10 18:26:52 |              17 | Ubuntu      |
|        29 | c8210c105fbf83c3693944f24a6b8bbcff78c330 | 2019-11-11 19:53:43 |              17 | Ubuntu      |
|        30 | 5d39e0030d61a60286985af3af8d7473011ead74 | 2019-11-11 20:54:19 |              17 | Ubuntu      |
|        37 | 92f4481ff492f31331593206828212 0f141282db | 2019-11-25 00:20:17 |              17 | Ubuntu      |
|        38 | fa017a9eb69080736cf03de67e41ecbed958204f | 2019-11-25 02:36:49 |              17 | Ubuntu      |
|        39 | 3ee0b6cde6d66ac7d741387bc62268254d86e793 | 2019-12-09 21:16:41 |              16 | Ubuntu      |
|        40 | 35fbd9349b08edfe57644f6634728ac60c5d77a6 | 2019-12-09 21:19:40 |              16 | Ubuntu      |
|        41 | 0dfbb18fa09311eb9e6972bd79acc0eb7214849a | 2019-12-09 21:20:25 |              16 | Ubuntu      |
|        42 | b7d502d5eacf3e504515a967f46825100f4e07a8 | 2019-12-10 00:17:58 |              16 | Ubuntu      |
|        43 | d4d6f42284a194a3faed496850578641db4ebfe9 | 2019-12-10 00:34:06 |              17 | Ubuntu      |
|        46 | 865e59be1e9895b4fe709ed1446bff2c3a23fca2 | 2019-12-11 02:47:09 |              40 | Ubuntu      |
|        47 | 075595ab29a8e83cb18f519d967a2c0483c8e98b | 2019-12-11 04:00:50 |              37 | Ubuntu      |
|        48 | 9e7238aa5887c88b0f4a8265a9a917d9a26f611e | 2019-12-11 04:02:39 |              37 | Ubuntu      |
|        49 | 5a050b2abfa0ecb3cd9c93b410874b438cf5e528 | 2019-12-11 04:04:49 |              18 | Ubuntu      |
|        50 | 064cd04bd81f0a365bb44223ebe556c19ac61d33 | 2019-12-11 13:38:21 |              17 | Ubuntu      |
+-----------+------------------------------------------+---------------------+-----------------+-------------+
Which commit would you like to enter hours for? Please enter commit_ID: 27
Enter your labor hours: 8
+-----------+---------------------+-------------+-------------+
| commit_ID | commit_date         | author_name | labor_hours |
+-----------+---------------------+-------------+-------------+
|        27 | 2019-11-10 18:26:52 | Ubuntu      |           8 |
+-----------+---------------------+-------------+-------------+
```

4.2.2 Report Generation

For the project manager to generate reports they must have installed the report script correctly, See 3.1 Initial Setup.

To generate a report, the project manager must type:
./report
They will be asked which type of report they would like to generate:

1.) All metrics for 1 commit
2.) Metrics for all commits within a certain date range

3.) All commits.

After choosing 1, 2, or 3, they will be prompted to enter a commit hash

They can get the commit hash from the labor hours list in 4.2.1 Labor Hour Tool

They can also use MySQL Workbench:

SELECT * FROM Commit_info;

This will get a list of all the commits in the database.

After entering a commit hash, a report will be generated.

For example:

```
ubuntu@ip-172-31-48-19:~/HookCode$ ./report
Which type of report would you like? Choose a number from the menu below.
1.) All metrics for 1 commit
2.) Metrics for all commits within a certain date range
3.) All commits.
1
You have chosen to create a report based on a commit hash.
Please enter a commit_hash:
c9e16ed50545772afe5d2398336951311a4a808f
Report generated in file name 'single_12-16-2019.csv'.csv
```