Tortoise Requirements Definition Document

Prepared By: Nicholas Santoro

Organization: Rowan University

Date: 10/28/19

Version: *1.1*

Table of Contents

Preface

This document presents a complete understanding of the requirements of ASRC Federal to create a code-coverage metrics database. It covers their current process of creating metrics, reports, and storing this information. This document is intended for all of the stakeholders in the Tortoise metrics Database project. These stakeholders include: ASRC developers and ASRC Managers. This is the first version of this document.

1. Executive Summary

1.1 Introduction

The main objective of the Tortoise Metrics Database project is to create a system that automatically runs code-coverage metrics tools on source code and collects specific metrics to store in a MySQL database.

1.1.1 Project Objectives

The objectives for this project are as follows:

- Create a database used for generating reports regarding code-coverage metrics.
- Automatically collect metrics and store them in the database.
- Generate reports based on the information found in the database.

1.1.2 Key System Functions

The following key functions have been defined for the Tortoise Metrics Database project. The system will:

- Automatically run metrics tools when a developer pushes code to a Git source repository.
- Run the gcov, CPPCheck, and UCC metrics tools to gather code-coverage metrics on the code files. These tools are all open source and run on Linux systems.
- Store metrics generated by the Jenkins integration tool whenever the project is built.
- Parse the desired metrics information from the output of the code-coverage tools.
- Store the desired information in a MySQL database. The information in the database shall be able to be grouped by each push to Git and the corresponding delivery tag.
- Allow developers to add their labor hours after pushing code to a Git repository.
- The program shall run on Linux.

1.1.3 Major Benefits

The major benefits of this project will be:

- The simplification of the reporting process for managers. Managers will be able to create excel reports containing the code-coverage metrics that they require using the data stored in the metrics database.
- Upper management will be able to keep track of developer's progress on major time-sensitive projects.
- Easy calculation of Source Lines of Code (SLOC) count and SLOCs/hour.

2. Glossary of terms

Code-Coverage: Code-coverage is how much of the source code of a given program is executed during a testing routine.

CPPCheck: "CPPCheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. The goal is to detect only real errors in the code" [1].

gcov: "gcov is a test coverage program. Use it in concert with GCC to analyze your programs to help create more efficient, faster running code and to discover untested parts of your program. You can use gcov as a profiling tool to help discover where your optimization efforts will best affect your code. You can also use gcov along with the other profiling tool, gprof, to assess which parts of your code use the greatest amount of computing time." [2]

Git: Git is a version control system used to track and control source code changes in software development.

Git Push: A Git push is how a developer will move their changes from their local repository to their remote Git repository.

Jenkins: Jenkins is a java based build tool used for continuous integration.

Linux: An open-source operating system which is based upon UNIX.

MySQL: MySQL is an open-source relational database.

SLOC: Source Lines of Code.

UCC: "The Unified Code Counter (UCC) is a comprehensive software lines of code counter produced by the USC Center for Systems and Software Engineering. It is available to the general public as open source code and can be compiled with any ANSI standard C++ compiler." [3]

3. User Requirements

3.1 Overview

ASRC Federal wishes to be able to generate code-coverage metrics on all source code files without the supervisor having to run the metrics tools manually. ASRC Federal wishes to automate the process of generating code-coverage metrics on all source code files. ASRC expects the new system to automatically run the specified metrics tools (CPPCheck, gcov, and UCC) whenever a developer pushes their code to Git. The program shall be triggered by using a maximum of two additional flags in a Git push. Additionally, ASRC expects the new system to automatically collect this information into a MySQL database where it will be available to be exported via excel documents. The program is also expected to run on Linux.

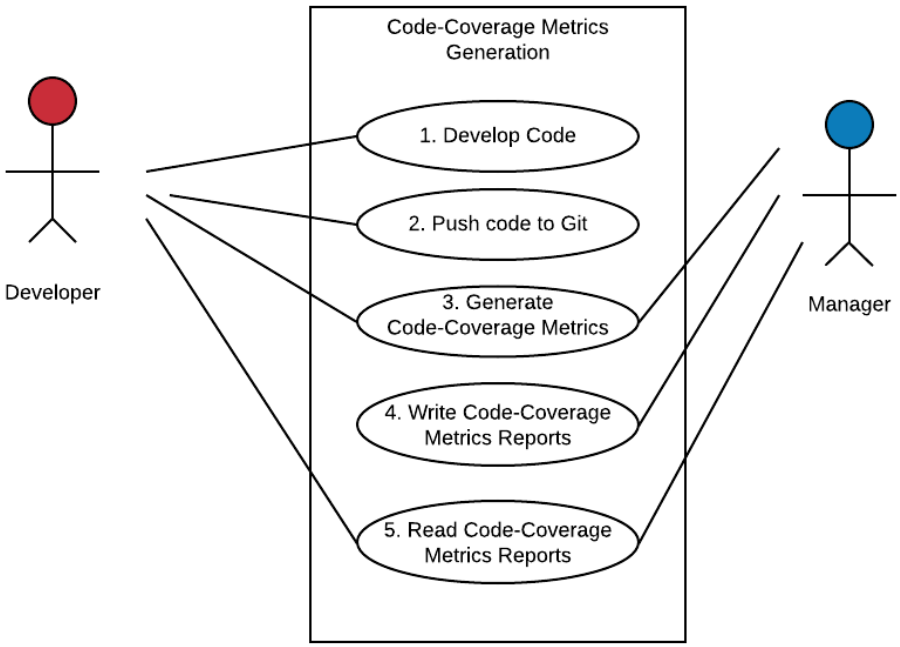
The new system should also be able to generate reports on demand using the data stored in the database. A weakness of the current manual process is that the code-coverage tools must be run individually, and the output must be manually parsed in order to gather the valuable metrics. The new system shall improve this process by automatically running the metrics tools stated above, and then automatically parsing the output in order to gather valuable metrics more efficiently.

3.2 Current Process

Managers at ASRC Federal manually run the tools CPPCheck, gcov, and UCC on files in order to generate the necessary metrics to build reports. The manual collection of metrics is done utilizing excel spreadsheets and macros. The metrics are not generated for every file in every project. The labor hours are stored on an existing, separate MySQL database. The labor hours are then combined with SLOC measurements in order to generate SLOCs/hour metrics. All of the metrics that are generated or calculated are then stored in excel spreadsheets in a reportable format. The Jenkins integration tool is used when building projects. The results from the Jenkins tool are not stored in the same excel spreadsheets as the metrics generated from the other metrics tools.

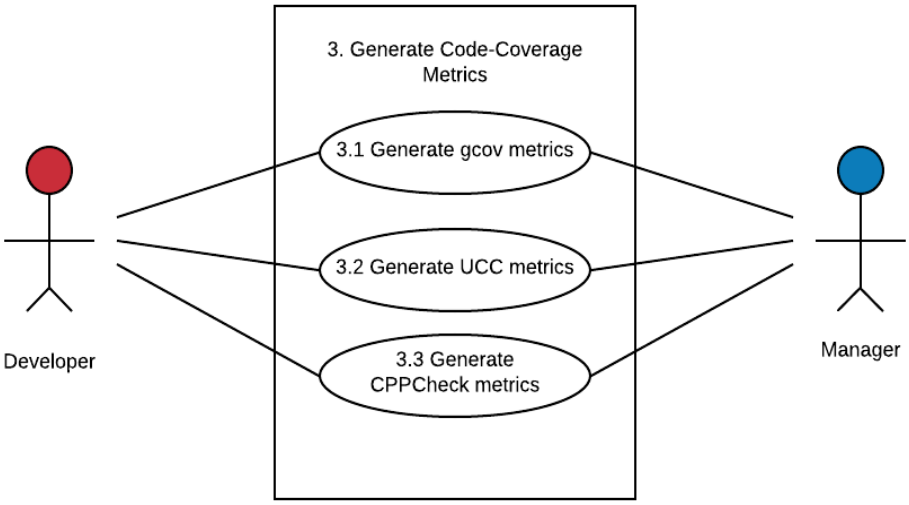
3.2.1 Context Diagram of the Current Business Process

Below is an example of a diagram depicting the current business process.



3.2.2 Current Use Case for Metrics Generation

Below is a decomposition of use case 3 in the diagram in section 3.2.1. It depicts the process of generating the code-coverage metrics.



3.3 Proposed Process

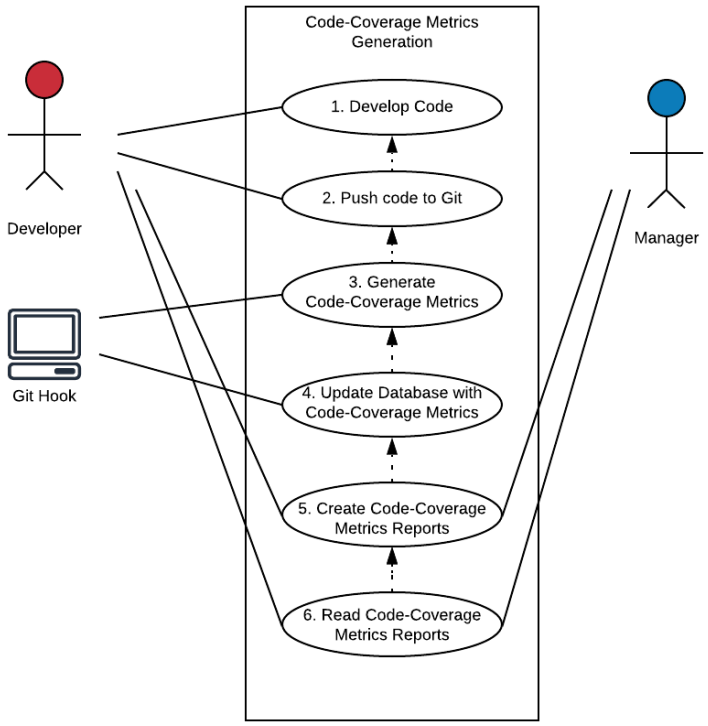
The proposed solution is to automate the process of code-coverage metrics generation. The developer will start the process by pushing his or her code to Git on a Linux machine. This will trigger the tools CPPCheck, gcov, and UCC to be run on the source code being pushed. The tools will generate metrics about the source code that was pushed to Git. The output from the tools will then be parsed in order to gather all of the specified metrics in an easy to understand format. If a Git push is unsuccessful for any reason, the metrics tools will not be run on that source code and no metrics will be generated.

The formatted output files will then be inserted into the MySQL database. The individual metrics will be inserted into the appropriate columns and the formatted output file will also be inserted into its own column in the database. Information about the push will also be stored in the database. The author of the push, the files which were push, what project the files belong to and all the information about the push will be gathered in a table in the database. The code-coverage information will then be available to report on.

After the metrics are created, a developer will be able to enter the labor hours they spent working on a project. The developer will then be able to choose the push they were working on from a list and enter the number of labor hours they spent working on the project. This information will be inserted into the metrics database in the correct position. The labor hours are necessary to compute the SLOCs/hour code-coverage metric.

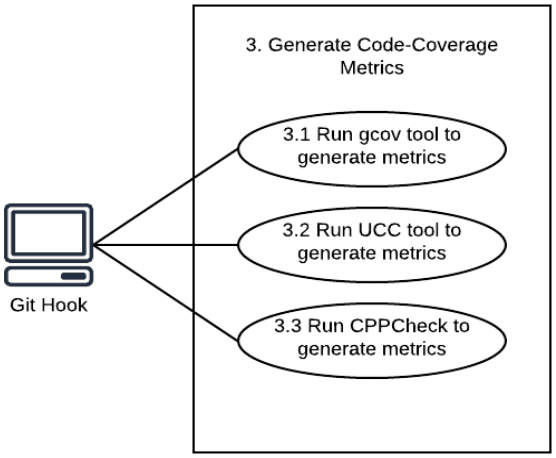
3.3.1 Context Diagram of the Proposed Solution Process

Below is an example of a diagram depicting the proposed solution process.



3.3.2 Proposed Use Case for Metrics Generation

Below is a diagram depicting the process of generating the code-coverage metrics.



4. System Requirements

4.1 Overview

When a developer pushes his or her source code to a Git repository, shall automatically be generated for all of the files which were pushed to Git.

After the metrics are stored, a manager or developer at ASRC should be able to export excel reports. The reports will be used to gauge the progress of a project in addition to the overall quality of the code in the project. The reports can be generated for an individual file or for an entire project. The managers will also be able to create SQL queries to create custom reports if none of the standard reports contain the metrics they desire.

In order for managers to be able to create these reports, the database must contain all of the information being reported. The data for the reports will come from the output of the tools CPPCheck, gcov, and UCC. The metrics data will be specific to the source code file that the tools analyzed. The tools will be called automatically after a push to Git by using a Git hook. In the hook, there will be commands calling the metrics tools. The output from the tools will then be parsed and stored in the database. In addition to the metrics data, the database will also contain:

- Information about the author who wrote the file (name, email address);
- General information about the Git push (date, push description, push tags);
- General information about the code file or files themselves (name, description).

4.2 Functional Requirements

1.0 "Dave, a developer at ASRC Federal, wishes to be able to have the code coverage metrics tools run whenever he pushes his code to the git repository."

1.1 Write a Git Hook that will execute a shell script when a user pushes code to the git repository.
- **1.1.1** Change Git hook to execute pre-push.
- **1.1.2** Gather commit/push info in hook.
- **1.1.3** Gather author information in hook.
- **1.1.4** Gather Commit Tags.
  - **1.1.4.1** Insert commit tags into the database.
- **1.1.5** Add CPPCheck execution to the shell script.
- **1.1.6** Add UCC execution to the shell script.
  - **1.1.6.1** Make UCC Generate SLOC.
  - **1.1.6.2** Save UCC program output to a file for later parsing.
  - **1.1.6.3** Parse UCC output into only metrics.
  - **1.1.6.4** Prepare UCC metrics for database.
- **1.1.7** Parse gcov output into only metrics.
  - **1.1.7.1** Prepare gcov metrics for database.
- **1.1.8** Save CPPCheck program output to a file for later parsing.
  - **1.1.8.1** Parse CPPCheck output into only metrics.
  - **1.1.8.2** Prepare CPPCheck metrics for database.

1.2 Modify the hook to only execute given a certain tag is present on the push or the commit.

2.0 "Greg, a manager at ASRC Federal, will be able to see all of the metrics information generated by the metrics tools, the author which pushed their source code to git, the general commit info, the tags associated with the commit, and the files which were committed. Greg wishes to be able to see specific metrics from the gcov, CPPCheck, and UCC code coverage metrics tools. Greg also wishes to be able to see the build results of the Jenkins continuous integration tool."

2.1 Create a list of metrics which are generated by the code coverage tool gcov.

2.2 Create a list of metrics which are generated by the code coverage tool CPPCheck.

2.3 Create a list of metrics which are generated by the code coverage tool UCC.

2.4 Create a list of metrics which are generated by the continuous build integration tool Jenkins.

3.0 "Greg, a manager at ASRC Federal, wishes the metrics information to be stored in a MySQL database. The table layout should take full advantage of the relational nature of MySQL."

3.1 Create an EER diagram to plan the layout of the tables in the database. The EER diagram should contain the relationships between entities, the attributes for each entity, and the names of the entities.
- **3.1.1** All of the tables shall have an arbitrary, meaningless, auto-incremented primary key.
- **3.1.2** Create an Author table to store the author information.
- **3.1.3** Create a Commit_info table to store the general information about the commit.
- **3.1.4** Create gcov_metrics table to store the metrics generated by the gcov tool.
- **3.1.5** Create UCC_metrics table to store the metrics generated by the UCC tool.
- **3.1.6** Create CPPCheck_metrics table to store the metrics generated by the CPPCheck tool.
- **3.1.7** Create Jenkins metrics column to store the build result from Jenkins.
- **3.1.8** Create Programs table to store the information about the programs the files being committed belong to.
- **3.1.9** Create File table to store information about the files being committed.

4.0 "Greg, a manager at ASRC Federal, wishes that the metrics shall be sent from the tools output files to the MySQL database automatically, without input from the developers. The only input the developers should have is triggering the tools to run after they push or commit their code to Git."

4.1 The metrics for each tool shall be sent individually to the database.
- **4.1.1** Send constructed UCC metrics to the database when the tool finishes its analysis of the source code.
- **4.1.2** Send constructed CPPCheck metrics to the database when the tool finishes its analysis of the source code.
- **4.1.3** Send constructed gcov metrics to the database when the tool finishes its analysis of the source code.
- **4.1.4** Send Commit info to the database after the information is collected from Git.

4.2 Make Commit stored procedure in order to update the database with the commit hash, commit date and ticket number.

4.3 Make UCC stored procedure to update the database with physical and logical slocs, comments, and embedded comments.

4.4 Make an Author stored procedure which will update the author table with the

authors name and email.

4.5 The Author stored procedure should make sure each author is unique before inserting.

4.6 Make a file info stored procedure to insert the file information from each metrics tool.

4.7 The file stored procedure should make sure that each file is unique when associated with a Git repository URL.

5.0 "Greg, a manager at ASRC Federal, wishes to be able to see the metrics that were generated for Dave's source code. Greg should to be able to export an excel spreadsheet that will generate a report containing all of the metrics, or Greg will be able to write a custom SQL query to generate a report."

5.1 Allow the ability to select a date range when exporting metrics reports to excel from the database.

5.2 Allow the ability to select a specific commit when exporting metrics to excel.

5.3 Each report should have every single metric for every file which was selected from the specified parameters. If no metric is present, the space on the report will be blank.

6.0 "Dave, a developer at ASRC Federal, will be able to enter the number of labor hours they spent working on a commit."

6.1 Have a second program display the commits in the database which are without the number of work hours associated. The developer will be able to distinguish a commit based on the commit hash from Git and the date of the commit.

6.2 From the displayed commits, a numerical selection will allow a user to select a commit to input hours for. The hours will be able to be entered as a double.

6.3 With the metrics of the commit already in the database, push hours to associate with commit.

6.4 A developer will only be able to see the commits which they were working on.

4.3 Non-Functional Requirements

0.1 Program runs on Linux.

0.2 Program uses metrics from gcov.

0.3 Program uses metrics from CPPCheck.

0.4 Program uses metrics from UCC.

0.5 Program uses metrics from Jenkins continuous integration tool.

0.6 Database uses MySQL v5.0 or greater.

0.7 Git Hook runs in a bash script.

0.8 The Git server must be able to connect to the MySQL database.

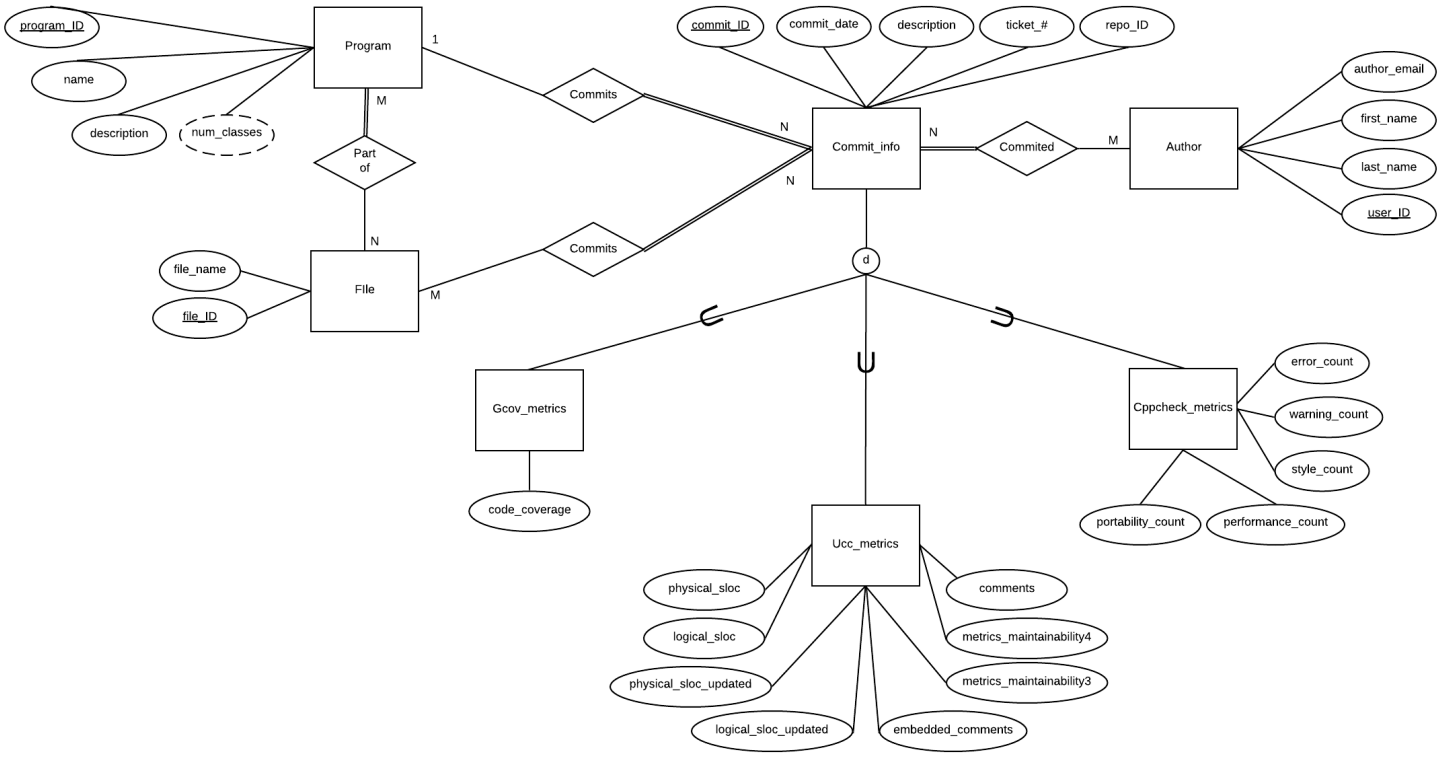
4.4 Operational and Performance Requirements

0.9 The system will be available for developers and managers 24/7. The users that can access the database will be able to access it at any time that they wish.

5. Appendix

5.1 Enhanced Entity Relationship Diagram (EER)

Below is the EER diagram for the Tortoise Metrics Database. It is a tool used for modelling the entities in a database in addition to the relationships between the entities.



6. Index

7. Bibliography

[1] CPPCheck, "Home," [Online]. Available: http://CPPCheck.sourceforge.net/.

[2] GNU, "10.1 Introduction to gcov," [Online]. Available: https://gcc.gnu.org/onlinedocs/gcc/Gcov-Intro.html#gcov-Intro.

[3] University of Southern California, "UCC," [Online]. Available: https://csse.usc.edu/ucc_new/wordpress/.