Validation Plan for the Software Metrics Collection System

Produced By:	Marcus Peñate
Organization:	Rowan University
Date:	11/23/2019
Version:	1.0

I. Background Information

I.I Purpose of this Document

This document is meant to ensure that the metrics collection system operates as expected and to provide adequate documentation of such. By validating the metrics collection system, it shall be verified to be within specification of the requirements and design documents.

I.II Documents Delivered Upon Validation

To properly assess validation, this document along with the following documents must be presented:

- Test Scripts
  - The proof from testers that validation was conducted
- Requirements Document
  - The specification of what the project scope is and what is required for completion
- Design Document
  - The specification of how the requirements will be implemented as a system together.

1. Scope of Validation Plan

The metrics system contains four interacting components. One of these components is the schema of a database for the metrics to be stored. The other components individually interact with this database component. The Git hook component executes automatically on a software developer's machine once they push their code to a git repository. The build status component is a script that executes automatically once a build manager has built a specific version of the program in Jenkins. The final component is the hours entry script that is invoked by a manager. All of these components are in scope of validation as they are all responsible for the production and/or retention of software metrics. If any of the listed components were to fail, the system has risk of entering a state with incorrect results.

1.1 Depth of Validation

In general the depth of validation shall depend on the likelihood of the risk for each component and the units of those components. Following is an analysis of the depth required for each component.

1.1.1 Database Schema

The schema's tables and relationships are integral to the production of accurate results from the schema's procedures and functions. For this reason the tables and relationships do not need individual testing, but shall be validated by the successful tests of the procedures and functions that run against these parts of the schema. These procedures and functions shall be tested with unit tests. Only procedures and functions that are not components of an integration test shall be tested. That is to say, if a procedure or function exists as the sole action of another system component that is tested in an integration test, that procedure or function shall not have a unit test. The database shall be tested using a snapshot of a development database. Each test shall load the snapshot so that the previous test's side effects could not be present.

1.1.2 Git Hook

The Git Hook runs in an automated environment and has checks for the presence of the programs it needs to run successfully. To verify this component some rigorous testing against its units is required but there is little variation of the possible input. Therefore, while there shall be testing script for the units, there will likely be no more than one or two script per unit. Integration tests are required to test the hook's communication of software metric results to the database.

1.1.3 Labor Hours Script

The depth required of this component is the testing of its core units and an integration test of running the entire script as it communicates with the database.

1.1.4 Jenkins Build Event Script

The Jenkins build event script is rather small, uses constants provided at runtime, and has no user input. For this reason, only a simple integration test on the validity of the results produced by this script is needed.

2. Rationalization of Testing Strategy

A system test is never performed. The database schema does not initiate interaction with any other component. The other components are disjoint from each other and only initiate an interaction with the database schema. The integration tests of these components shall verify the state of the system.

2.1 Database Schema

As mentioned, the schema's tables and relationships are integral to the production of accurate results. If the procedures and functions of the schema were to fail it would be a result of incorrect relationships between the tables or incorrect configuration of the tables themselves. No more depth of testing is required other than the unit testing of these procedures. Once these units are verified, the likelihood of an error occurring is little to none.

2.2 Git Hook

The likelihood for one of the metrics tools that the hook uses to produce incorrect results is low as they are outsourced programs with their own development communities. The severity of the hook having an error trying to run a program or getting incorrect results is high as it would put the database in an invalid state. The hook has mitigations in place for the lack of programs required for the hook to run and shall not attempt to update information in the database because of these mitigations. Justifiably the hook is the component of the project with the most risk involved and therefore has the most testing, including an integration test and multiple unit tests.

2.3 Labor Hours Script

The labor hours script's robust handling of user input mitigates the likelihood of errors and risk occurring. For this reason only the most core parts, like the calls to the database, need to be tested as unit tests. For verification of the robust input validation, an integration test with invalid and valid user input shall be conducted.

2.4 Jenkins Build Event Script

The likelihood of this script producing incorrect results is little to none as it uses constant values to query Jenkins and then update the database with that query's results. This extremely small likelihood of error is the reason for only a single integration test of the validity of the results of this script.

3. Content of Testing Scripts

In order for the testing script to be comprehensive they shall include the following information:

- Test Environment (if applicable for each)
    - Pre-existing files, programs, shell environment
- Required skills of the tester
- Test ID Number
- Numbers for each step of the test
- Instructions of each step and explicit expected results
- User input is applicable to the step of a test
- Requirements document reference of the requirement that each step of the test verifies
- Actual results from completion of a step
- Initials of tester per completed step

- Pass of Fail of each step in the test
  - A test is passed if and only if the actual results from the test match the expected results

4. Roles of Parties Involved

Validation of the metrics collection system shall be performed by designated testers. A tester cannot conduct a test unless they meet the required skills as specified by that test's corresponding test script.

The development team and project owner are responsible for the creation of the metrics collection system and its compliance with the requirements and design documents. It follows that should a test fail it remains the responsibility of the development team and project owner to devise a strategy to resolve the failure of the test.

5. Traceability Matrix

This matrix serves to link a requirement to its specification in the design document as well as the test script that validates the requirement.

Requirements Reference	Requirement Name	Design Document References	Test Script Reference	Comment
0.4.1	Programs runs on Linux	1.3, 1.4	2.4.1, 3.2.1	
0.4.2	Program uses metrics from GCOV	1.2, 3, 3.6, 6		Not yet implemented
0.4.3	Program uses metrics from CPPCheck	1.2, 3, 3.1, 6	2.4.4	
0.4.4	Program uses metrics from UCC	1.2, 3, 3.1, 6	2.4.5	
0.4.5	Database uses MySQL	1.3	3.1.1	
0.4.6	Hook runs in a bash script	1.3, 1.4	2.4.1	
0.4.7	Post-push hour entry script is written in bash	1.2, 2, 5.1.1, 5.2.1	3.2.1	
1.1.1	Git hook executes on push	4, 5.1.1	2.4.3	
1.1.3	Hooks runs metrics when tag "metrics" is present	4	2.4.3	
1.1.5	Gather commit info in hook	6	2.3	
1.1.6	Gather author information in hook		2.3.1,2.3.2	Not referenced in design doc
1.1.7	Gather Commit Tags	4	2.3.6, 2.3.7	

1.1.8	Insert tags into database		2.4.6	Not referenced in design doc
1.4.2.2	Create an Author table	6	2.4.3	
1.4.2.3	Create a Commit_into table	6	2.4.3	
1.4.2.4	Create GCOV_metrics table	6		Not yet implemented
1.4.2.5	Create UCC_metrics table	6	2.4.5	
1.4.2.6	Create CPPCheck_metrics table	6	2.4.4	
1.4.2.7	Create Jenkins metrics column	6	4.1.3	
2.1.2	CPPCheck execution in the hook script	3, 3.1	2.4.4	
2.1.3	UCC execution in the shell script	3, 3.1	2.4.5	
2.1.3.1	UCC Generates SLOC		2.4.5	Not referenced in design doc
2.2.1.1	Parse GCOV output into only metrics	3, 3.1		Not yet implemented
2.2.1.2	Prepare GCOV metrics for database	3, 3.1		Not yet implemented
2.2.2	Save CPPCheck program output to a file for later parsing	3, 3.1	2.2.2	
2.2.2.1	Parse CPPCheck output into only metrics	3, 3.1	2.2.3	
2.2.3	Save UCC program output to a file for later parsing	3, 3.1	2.1.2	
2.2.3.1	Parse UCC output into only metrics	3, 3.1	2.1.3-2.1.4	
2.2.3.2	Prepare UCC metrics for database	3, 3.1	2.1.5-2.1.6	
2.3.1	Send constructed UCC metrics to the database	3, 3.1, 4, 5.1.1	2.4.5	
2.3.2	Send constructed CPPCheck metrics to the database	3, 3.1, 4, 5.1.1	2.4.4	
2.3.3	Send constructed GCOV metrics to the database	3, 3.1, 4, 5.1.1		Not yet implemented
2.3.5	Send Commit info to the database	3, 3.1, 4, 5.1.1	2.4.3, 1.3.2, 1.4.4	
2.3.6	Send Jenkins Build Status to Database	3, 3.1, 4, 5.1.1	4.1.3	

🟥	Files are stored uniquely in database	🟨	1.5.3	Thought of after creation of req. and design doc
2.4.1	Make Commit stored procedure	3, 3.1	1.2.2	
2.4.2	UCC stored procedure	3, 3.1	2.4.5	
3.1.1	Have a second program display the commits in the database without the work hours associated	🟨	1.1.3, 3.2.3	Not referenced in design doc
3.1.2	From numerical selection of displayed commits allow a user to select a commit for hours input	🟨	3.2.5, 3.2.7	Not referenced in design doc
3.2.1	With the metrics of the commit already in the database, push hours to associate with commit	🟨	3.2.8	Not referenced in design doc
X.1.2	Allow date range or pushed commit range when exporting metrics to excel	5.1.2	▨	Not yet implemented
X.1.3	Pull reports from the database the bounds previously specified	5.1.2	▨	Not yet implemented