Project 1 Milestone 4 (“Release Notes”)

As a Boilermaker pursuing academic excellence, we pledge to be honest and true in all that we do. Accountable together – We are Purdue.

Type or sign your names: Mohammed Fashola, Owen Prince, Ryan Villarreal
Write today’s date: 10/04/2021

Assignment Goal

In this assignment, you will deliver your Project 1 implementation (*software*) and communicate the final status of your Project 1 (*release notes*).

This document provides a template for the release notes. This is a form of documentation that your customer can use to decide (a) whether you've met the contract, and (b) re-negotiate the contract based on deviations therefrom.

Relevant Course Outcomes

A student who successfully completes this assignment will have demonstrated the ability to

- *Outcome ii*: the ability to conduct key elements of the software engineering process, including…deployment
- *Outcome iii*: Develop an understanding of the social aspects of software engineering… including…communication [and] teamwork.

Resources

This document is a form of "release notes", albeit on the more technical end of the spectrum.

Definitions:

- Definition 1
- Definition 2

Examples:

- Apache Maven
- Microsoft Windows

Assignment

Fill out each of the following sections.

Location of project

Provide the URL to the GitHub repository containing your team's project.

https://github.com/Purdue-ECE-461/project-1-project-1-4

Succinct description

In a 5-7 sentence paragraph, describe the system that you have implemented.

We have implemented a system that analyzes a repository and generates a series of 5 metrics that analyze the ramp-up time, program correctness, bus-factor, maintainer responsiveness, and license. From these 5 metrics, we generate a net score reflecting the customer's priorities. We make all requests through the Github API, and clone the repository locally to do some analysis. This program can handle a list of URLs of arbitrary length, as well as the arguments "test" and "install." Test runs a testing suite on the program, whereas "install" runs an installation script that installs all the dependencies necessary for this program. There are logging features with three levels of verbosity: silent, info and debug.

Fitness for purpose

In Milestone 1, you identified the requirements for the system.

Fill in this table of the customer's requirements and the degree to which you've met each of them. (If on Milestone 1 you lost points for inadequate requirements, then you should provide a more detailed set in this table).

Requirement	**Is the requirement met? (yes/no)**	**Explanation (2-4 sentences). If met, how did you measure it? If unmet, discuss.**
Documentation requirements	Yes.	Each function and class has a corresponding docstring. There is a readme that explains how to install the program, run the program, and test the program.
Program requirements	Yes	1. Tool is open source 2. In milestone 3, we found that it would take longer than 1 minute to run 10 URLs. Our updated time limit was 3 minutes per URL (to account for poor network bandwidth and huge repositories). We meet this new metric. 3. We can run using an executable in the root directories 4. "./run install" installs dependencies 5. "./run test" tests the tool for correctness 6. "./run URL_FILE" where URL_FILE is the absolute location of a file consisting of an ASCII-encoded newline-delimited set of URLs runs the file at the specified absolute path

		7. We can handle a variable size list of URLs that are ASCII-encoded newline-delimited sets 8. We can analyze repositories both under npmjs and github 9. We calculate all 5 metrics and each score is in the interval [0, 1] 10. We clone the repository to calculate the ramp-up score metric 11. We print the scores in the order URL_1 net_score ramp_up_score correctness_score bus_factor_score 12. We sort the scores from highest to lowest and then print it to console 13. Program does not expose token to a public location
Testing Requirements	Yes	● Test cases covering a good portion of our implementation is evidently shown in our test_coverage.py file ● ./run test successfully runs on ECE grid. ● A log file is generated during every run. This Log file takes the name specified in the .env file and produces our log statements from our implementations. We confirmed this by looking in the Log file after every run to check for any errors. ● "X/Y test cases passed. Z% line coverage achieved." This is achieved as evidently shown in stdout after running "./run test" ● We achieve 77% of coverage that does not meet the 80% requirement. This is because we wrapped the majority of our functions in "try, except" blocks to ensure an error message gets printed to our log_file if that function does not execute properly for any reason. Our test cases test all our try blocks but do not enter some of the except blocks causing the coverage report to be short of the requirement by a very small amount. These except blocks not covered in our coverage are for extreme errors that should not normally occur. An example of one of these blocks is a check to make sure a valid GitHub Personal Access Token is being used. Since this token is an environment variable, during coverage testing, we can not test this block of code. The coverage report also ignores the function definitions and class definitions. Due to the large number of functions we used, this takes a huge toll on our coverage report percentage.
Ramp-up score requirements	Yes	● We count the number of lines of comments within the repository

		• We rate the repository on the descriptiveness of a readme These two indicators will make it easier for new developers to get started on a project.
Correctness score requirements	Yes	• We use the 1- open issue / total issue ratio as a metric, more open issues mean lower score • We use the number of people that watch, download, and rely on the repository to judge how many people indirectly vet the program • We check for the existence of a testing framework The combination of these should judge how "correct" the repository is.
Bus-factor score requirements	Yes	• We look at the total number of contributors on the project • We look at the number of contributors to the last 10 releases • We look at the # of contributions of the person who contributed the most compared to the total number of contributions These 3 should make sure that a project can still function with team members missing.
Responsive maintainer score requirements	Yes	• We check the time between releases to make sure that there is a low turnaround time. • We check to make sure that issues are not left open for too long. • We base the final metric on the number of total contributors in the project.
License compatibility score requirements	Yes	We check to make sure that the license that the repository has is compatible with the LGPLv2.1 license by comparing the repository license to a comprehensive list of licenses that are compatible with the LGPLv2.1 license.

Final metric designs

In Milestone 1, you described your initial design of the metrics. Please compare your initial design to your final implementation. (This will help us figure out if our auto-grader is not working well for your approach.)

Metric	Initial design	Final implementation ("Same" or whatever the new def. is)	Explain any difference

Ramp up	(Readme includes install, usage, run section) + (# lines comments / # slocs). Sum is divided by 2	0.4 * (Readme includes install, usage, license) + 0.6 * (# lines of comments / slocs).	Slight difference in weighting, now the readme is weighted at 40% and the comments / slocs is 60% instead of 50/50. Look for install, usage and license in readme instead of install, usage and run. These are more commonly present in a good readme.
Correctness	1. #outstanding issues / total # issues 2. Date of latest release is within the last year 3. # clones this week 4. Module has been checked for correctness by a third party Average of all four scores	1. #outstanding issues / total # issues 2. age, # stars, # subscribers, releases, # repository forks 3. existence of a test folder/test files 30% issue ratio, 20% age, stars, subs, releases, repo forks, 50% test existence	Instead of looking at clones this week and date of latest release, we look at a combination of repository age, # of stars, # of subscribers, # of releases, and repository forks. We compare them against baseline standard to establish some sort of "reputation" score. Weighted lightly because this is pretty subjective. We also look for the existence of a testing infrastructure within the repository because this is far more common and a much better indicator of correctness.
Bus factor score	(1 - (1 / # contributors)) + (1 - (1 / # contributors in most recent release)) + ((1 - top contributor commits / # all commits))	(1 - (1 / # contributors)) + (# authors in last 10% of commits / Total # of authors) + ((1 - top contributor commits / # all commits)) / 3	We modified the second equation to check the total number of authors in the last 10% of commits against all the authors in the repository. This will tell us the number of authors that have recently been actively contributing and will contribute greatly to the bus factor score calculations.
Responsive maintainer score	1. 2 or more releases per year for the last 5 years 2. Issues should be resolved within a week 3. High number of contributors to the project (> 10) Average between the 3	1. 2 or more releases per year for the last 5 years 2. Issues should be resolved within a week 3. High number of contributors to the project (> 10)	We added a fourth equation that will check if any issue has been opened for more than 100 days. This contributes greatly to the maintenance score because it tells us that there is a possibility that certain issues may not be resolved no matter the number of mainteners

		4. 0 if any issue has been opened for more than 100 days. 1 otherwise Average of the 4	
License score	1 if compatible, 0 if not	Same	We refined the list of compatible licenses as shown in milestone 3
Net score	license_score * (0.4 * maintenance + 0.25 * bus_factor + 0.25 * correctness + 0.1 * ramp_up)	Same	No difference

Notes for the auto-grader

If your submission cannot be automatically parsed by the auto-grader described in the project specification, provide explanatory notes that the course staff can consider while scoring your submission. Be specific. Since this spec was provided well in advance, accommodating any deviations is at the discretion of the staff.

Deviation	Details
Maybe an extra line of text?	We have an extra line of text (URL_1 net_score ramp_up_score correctness_score bus_factor_score) that is printed before the start of the scores. We saw this line of text printed in the sample output, so we included it in our output.
The message “Installing…” prints upon execution of the install command, and prints “X Dependencies were installed”	This was done to increase the readability and user friendliness of the tool. This will also alert the user if any critical packages failed to install. This does not have any effect on the installation process or its outputs.
The message “Testing…” prints upon execution of the test command	This was done to increase the readability and user friendliness of the tool. This will make sure the user is aware that their request is being processed. It does not have any effect on calculations or other outputs.