ECE 46100 Team 3

Parker Bushey, Kaitlyn Roberts, Ashwin Senthilkumar

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

Signed: Parker Bushey, Kaitlyn Roberts, and Ashwin Senthilkumar
Date: September 11, 2021

Table of Contents

1 - Tool Selection and Preparation **3**
- 1.1 - Programming language, toolset, component selection 3
- 1.2 - Communication mechanism 3
- 1.3 - GitHub token statement 3

2 - Team Contract **3**
- 2.1 - Commitments 3
- 2.2 - Meetings 4
- 2.3 - Formatting 4

3 - Team Synchronous Meeting Times **4**

4 - Requirements **4**
- 4.1 - Internal 4
- 4.2 - Client 5

5 - Preliminary Design **6**
- 5.1 - Metric calculations 6
  - 5.1.1 - Ramp-up time (Git API) 6
  - 5.1.2 - Correctness (JGit) 7
  - 5.1.3 - Bus factor (Git API) 7
  - 5.1.4 - Responsiveness (Git API) 7
  - 5.1.5 - License compatibility (Git API) 8
  - 5.1.6 - Overall Score 8
- 5.2 - Diagrams 9

6 - Planned Milestones **10**
- 6.1 - Week 2 milestone 10
- 6.2 - Week 3 milestone 10
- 6.3 - Week 4 milestone 10
- 6.4 - Week 5 milestone 10

7 - Validation and Assessment Plan **10**

1 - Tool Selection and Preparation

1.1 - Programming language, toolset, component selection

We will use Java for this project's code, which will be stored on Github. We will use Java to interface with Github REST API and Jgit.

Resources:

- https://docs.github.com/en/rest/reference
- https://itsallbinary.com/github-rest-api-search-files-content-pull-requests-commits-programmatically-using-java-without-cloning/
- https://download.eclipse.org/jgit/site/5.12.0.202106070339-r/apidocs/index.html
- https://www.codeaffine.com/2015/12/15/getting-started-with-jgit/

1.2 - Communication mechanism

For setting up meetings and general communication, we will use GroupMe. If meetings do not take place in person, Webex will be used to allow for distanced synchronous work. General project information will be posted under the projects tab of our team's GitHub repository.

1.3 - GitHub token statement

We will utilize the REST API to generate tokens for our project, which will get stored in the $GITHUB_TOKEN environment variable.

2 - Team Contract

2.1 - Commitments

- Each team member will complete the work that is expected.
- Team members will complete the work given to them to the best of their ability.
- Everyone is expected to keep each other informed of their progress and struggles.

2.2 - Meetings

- Team members are expected to attend all meetings unless other members are alerted ahead of time or in the case of an emergency.
    - If someone cannot make it to a meeting, they will alert other members at least a day in advance.
    - In the event a meeting is missed, work shall be made up on one's own time.
- Each member will come prepared to meetings.
- Team members will be respectful to other members during meetings.
- Weekend meetings will be dedicated to completing the milestone documents and recording progress made / roadblocks that week

2.3 - Formatting

- Format code in an organized manner.
    - Class and variable names should be written in camel-case.
    - Use descriptive variable names everywhere that is appropriate.
- Include comments on important parts/changes.
- Avoid putting too much on a single line.

3 - Team Synchronous Meeting Times

Meetings will take place over Webex and we will plan on at least twice weekly. Additional meetings will be scheduled as needed. The meeting times are as follows:

- Weekly Meeting 1: Saturday, 1:30 PM
- Weekly Meeting 2: Wednesday, 6:00 PM

4 - Requirements

4.1 - Internal

- Executable file "run" in root directory
    - ./run install: installs dependencies needed, exit 0

    - ./run URL_FILE: URL_FILE is a text file with a list of urls
        - Output: “URL_1 NET_SCORE RAMP_UP_SCORE CORRECTNESS_SCORE BUS_FACTOR_SCORE RESPONSIVE_MAINTAINER_SCORE LICENSE_SCORE” … exit 0
    - ./run test: runs a test with at least 20 test cases and has at least 80% line coverage
        - Output: “X/Y test cases passed. Z% line coverage achieved.” … exit 0
- Scores are between 0 and 1, i.e. a percentage in decimal form
- **At least one score should be obtained using metrics from Github API**
- NET_SCORE is a weighted sum
- Use the auto-grader-friendly CLI when executed on a Linux machine
- **Program must produce a log file** stored in the location pointed to by the environment variable $LOG_FILE. Set a logging level with $LOG_LEVEL.
- **npm modules on Github need tokens in Github API. Use REST API to access them** instead of scraping html code off the web. Store these tokens in the environment variable **$GITHUB_TOKEN** in your local run environment.
- **At least one score should be obtained using metrics from source code repository data**, not using the GitHub API.
    - To do this, clone the repository and interact with the metadata using Jgit or run a static analysis using a javascript parsing tool.
- Any published code used from external sources should not be copy and pasted, instead import it directly so that the code is updated automatically.

4.2 - Client

- The system should support input from command line arguments.
- The system will print all output to stdout.
- The system will produce an **ordered list of repositories starting with the most trustworthy.**
- In the list of repositories, each entry will include a set of scores including:
    - **Overall score**
    - **Ramp-up time (i.e. ease-of-use and well-defined documentation)**
    - **Correctness**
    - **Bus factor (i.e. number of active collaborators)**
    - **Responsiveness (i.e. issue response time)**
    - **License compatibility**

- In initial conversations, their prospective customers say that it will be important for **self-hosted ACME to be open-source** so that they can tailor it to their needs.
- Any open-source module's licenses that ACME Corporation's service engineers use must be **compatible with the LGPLv2.1 license.**
- **Program should take no longer than 30 minutes to run.**

5 - Preliminary Design

5.1 - Metric calculations

5.1.1 - Ramp-up time (Git API)

Ideas/notes:

- Longer readme, includes links or pictures: GET /repos/{owner}/{repo}/readme
- Less files (small tree size): GET /repos/{owner}/{repo}/git/trees/{tree_sha}
- Number of views/downloads/forks/clones:
    - GET /repos/{owner}/{repo}/traffic/views
    - GET /repos/{owner}/{repo}/releases/assets/{asset_id} (includes download count)
    - GET /repos/{owner}/{repo}/forks
    - GET /repos/{owner}/{repo}/traffic/clones

RAMP_UP_SCORE = X * (README_SIZE / FILE_COUNT + Y * DOWNLOAD_COUNT), where X and Y are some constant

Justification:

It is assumed that the larger the readme, the more detailed and helpful the instructions will be. It is also assumed that a larger file count implies a more complex program to learn. Lastly, it is assumed that the more downloads a repository has, the more there are knowledgeable users willing to help on external sites.

5.1.2 - Correctness (JGit)

Ideas/notes:

- Compiles and runs (True/False)
- Code coverage %
- Self tests are passing %

CORRECTNESS_SCORE = COMPILE_RUN * ((COVERAGE > 0.85 ? (COVERAGE - 0.85) / 0.15 : 0) + (SELF_TESTS > 0.7 ? (SELF_TESTS - 0.7) / 0.3 : 0)) / 2

Justification:

If the program does not compile, it gets a correctness score of 0, since no other tests can be run on it. The code coverage test is then run, with the result assumed to be greater than 85% for trustworthy programs. Common self-tests, or those given in the repository, are run as well, with the result assumed to be greater than 70% of tests passed for trustworthy programs. Lastly, the average is taken between the code coverage and self test coverage percentages.

5.1.3 - Bus factor (Git API)

Ideas/notes:

- 2-6 collaborators normally
- Some have single collaborator
- Some have many collaborators
    - GET /repos/:owner/:repo/stats/contributors

BUS_FACTOR_SCORE = ((NUM_CONTRIBUTORS) >= 6) ? 1.0 : (NUM_CONTRIBUTORS) / 6

Justification:

Most repositories on GitHub have between 1 and 6 collaborators, however some may be company built and therefore have more contributors. Those with more than 6 contributors are assumed to do well if the leader/author stops contributing, whereas those with less may have greater trouble without a leader.

5.1.4 - Responsiveness (Git API)

Ideas/notes:

- Number of open and closed issues
- Issue Date opened compared to date closed
- Date of most recent closed issue

    - GET /orgs/{org}/issues
- Commit activity:
    - GET /repos/{owner}/{repo}/stats/commit_activity
    - GET /repos/{owner}/{repo}/commits/{ref}/status
    - GET /repos/{owner}/{repo}/stats/code_frequency

RESPONSIVE_MAINTAINER_SCORE = (CLOSED_ISSUES / (OPEN_ISSUES + CLOSED_ISSUES) + (MOST_RECENT > 90 ? 0 : 90 - MOST_RECENT) / 90) / 2

Justification:

It is assumed that if commit activity is frequent and recent, contributors are more likely to respond to issue requests and questions about their program. It is also assumed that the number of closed issues implies that contributors respond to and solve problems quickly. Furthermore, if the most recent issue has not been closed within the last 3 months, the contributors are assumed to be unresponsive. Lastly, an average is taken between issue coverage and recent issue date.

5.1.5 - License compatibility (Git API)

Ideas/notes:

- License scan passed: GET /repos/{owner}/{repo}/community/profile

LICENSE_SCORE = LGPL21_COMP

Justification:

This will be a dictionary containing True/False compatibility values for each license usable on GitHub. LGPLv2.1 license is compatible with many licenses, documented at https://www.gnu.org/licenses/license-list.en.html and https://www.gnu.org/licenses/gpl-faq.html#AllCompatibility. Licenses that are commonly used on Github can be found in the following article: https://www.fastcompany.com/3014553/what-coders-should-know-about-copyright-licensing

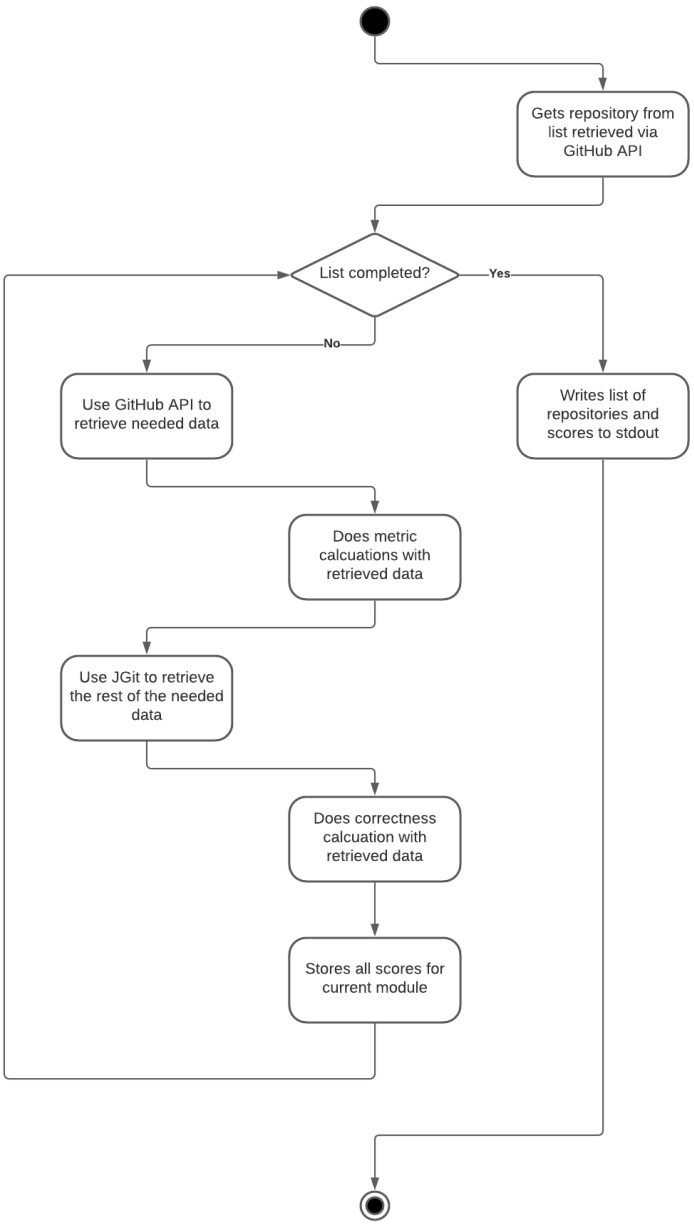
5.1.6 - Overall Score

NET_SCORE = (0.4 * CORRECTNESS_SCORE + 0.3 * RAMP_UP_SCORE + 0.2 * RESPONSIVE_MAINTAINER_SCORE + 0.1 * BUS_FACTOR_SCORE) * LICENSE_SCORE

Justification:

License score is an all or nothing requirement of the program the client will be using. Correctness is most important because if the program does not work then using it will be difficult until it is fixed, which may take too much time for the client. If the program is correct then the other metrics have less usefulness since a working program takes less effort to maintain and therefore has less need for many contributors. Ramp up score is still moderately important since the client and their employees will still need to learn how to use the program.

5.2 - Diagrams



6 - Planned Milestones

6.1 - Week 2 milestone

- Get repos list function (Katie / all)
- License score API code (Parker)
- Bus Factor API code (Ashwin)

6.2 - Week 3 milestone

- Main function
- Responsiveness API Code
- Ramp up time API code

6.3 - Week 4 milestone

- JGit Correctness Code
- Logging functionality
- Make file
- Deliver completed software solution
- Create/submit brief report on your current solution and provide one example of a module that you think your approach scores well on

6.4 - Week 5 milestone

- Post-mortem report

7 - Validation and Assessment Plan

We will use the requirements described earlier in this document along with running the auto-grader tests to validate the final product. Part of this process will be to check if the right API calls are being made and we are getting a correct, reasonable metric score. Further, a code coverage test will be run on our program since our correctness metric is justified with a code coverage greater than 85% for trustworthy programs. The program will also output a log file that (among other data) will output runtime, which is required to be under 30 minutes as an extreme maximum.