Project Plan: Milestone 1

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

(On group submissions, have each team member type their name).

Type your name: Daniel Doh

Seunga Kim

So Won Kim

Write today's date: 22 Jan 2023

Project Plan

Repository URL

- https://github.com/danieldoh/software_engineering_project

Tool Selection and Preparation

- Programming Language: C#
- Toolset: VS Code, VIM, Atom, Git, Github
- Component Selection: Rest API, GraphQL API
- Communication Mechanism: Slack & KakaoTalk(Korean SNS)
- Github Token: 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

Team Contract

Member 1
Daniel Doh, hdoh@purdue.edu
Member 2
So Won Kim, kim3007@purdue.edu
Member 3
Seunga Kim, kim3558@purdue.edu

- Come to meetings prepared
- Listen to each other
- Share early and often
- Do mistakes and ask freely
- Complete work as agreed
- Don’t be a jerk

Team Synchronous Meeting Times

- Wednesday 8:00 pm - 9:00 pm
- Saturday 8:00 pm - 9:00 pm
- Sunday 7:00 pm - 8:00 pm [if needed for final submission]

Requirements

- Based on system input, the system should produce an ordered list of repositories, with the most trustworthy listed first.
- It is important to have enough maintainers to continue to apply critical fixes such as a security patch.
- Open-source module needs a high standard of correctness.

- It should be quite easy for ACME Corporation's engineers to learn the new module, "low ramp-up time".
- It is essential that maintainers are responsive to handle and fix any bugs. -> ACME's top priority
- Since there can be more qualities later, the product should be possible to accommodate adding new aspects.
- Open-source modules for ACME Corporation must be compatible with LGPL (Lesser General Public License) version 2.1 license for their later product distribution plan.

Preliminary Design

https://lucid.app/lucidchart/5e0be35f-57ab-4c34-95a9-a353d5f484b2/edit?viewport_loc=-36%2C-55%2C740%2C927%2C0_0&invitationId=inv_8e679ff3-482b-4ac7-a371-c0771c00f79e

1. **Metric Operationalizations and Net Score Formula**
    a. Bus Factor:
        i. Bus Factor is a number of members, if run over by a bus, then would make the project “die”. This indicates the information and compatibility of a project is replied to on a particular number of members, and not shared equally among all members.
        ii. Our operation will use the API to get the Bus Factor and divide the number by the total number of members (contributors), so that it will scale between 0 and 1.
        iii. The API we are going to use:
            1. https://github.com/SOM-Research/busfactor (the description for the API) https://livablesoftware.com/calculate-bus-factor-software-project/
            2. This API has 4 metrics based on the last change done on the repository, number of changes, number of distinct changes, and weighted distinct changes, to calculate the Bus Factor.
            3. We chose to use this particular API, because it takes a lot of attributes into consideration when calculating the Bus Factor. In addition, it extracts and outputs other meaningful information that could be extrapolated to other metric operationalization.
        iv. Other alternative APIs
            1. https://github.com/elek/bus-factor
            2. https://github.com/yamikuronue/BusFactor
    b. License Compatibility:
        i. Since ACME Corporation uses the GNU Lesser General Public License v2.1 for all open-source software, the licenses of given repositories should be compatible with the LGPLv2.1 license.
        ii. The API we are going to use:
            1. https://github.com/src-d/go-license-detector
            2. Find files in the root directory which may represent a license, LICENSE or license.md
            3. OR look for README files and extract the license name.
            4. Then, we should make a list of GPL-compatible licenses. Based on the list, we can check whether the extracted license is compatible.
            5. 1: if the license is compatible / 0: otherwise
    c. Low Ramp-up Time: To score the Ramp-up time of modules suitability for contexts.
        i. detect programming source code API to calculate the percentage of language written

1. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/
    a. download list of most popular programming languages in Github, then map them in a dictionary
2. https://github.com/AlDanial/cloc
    a. To calculate ramp-up time, detect programming source codes written in the module. Within that result, we will multiply each programming language by percentage of its popularity and sum them together.
    b. If the score is closed to 1, it indicates low ramp-up time. Otherwise, high ramp-up time.

d. Correctness:
    i. Correctness is measurement indicating how well the code has been written. Level of correctness is vital for a code, low standard of correctness might not even be able to make execution.
    ii. The API will help us get each score of CI test, CII test, and Vulnerability of the given repository in a scale up to 10. Then we will divide the sum of the score by 30 to make it into a scale between 0 and 1.
    iii. The API we are going to use:
        1. https://github.com/ossf/scorecard
        2. The API takes the repository and assesses a number of important features and output scores for each of them.
        3. We chose this API because it will help us output the scores to check the vulnerability of the code, which indicates the potential risk of security flaws. In addition, it outputs many useful scores that could be used to evaluate security level of the code.
    iv. Other alternative APIs
        1. https://github.com/EngineeringSoftware/inlinetest (inline test)
        2. https://github.com/cs50/check50 (cs50)

e. Responsiveness
    i. The API we are going to use:
        1. https://github.com/ossf/scorecard
        2. This API calculates the score for several checks. Among those, we are going to use the “Maintained” check.
        3. This check determines whether the project is actively maintained based on the number of commits in the last 90 days.
        4. This check will only succeed if a GitHub project is older than 90 days. Otherwise, it will be considered too young to assess.

Each Score [0, 1]
0: Total Failure
1: Perfection

Weight for items (Total 100% = 1)
Ramp-Up Time: 0.15

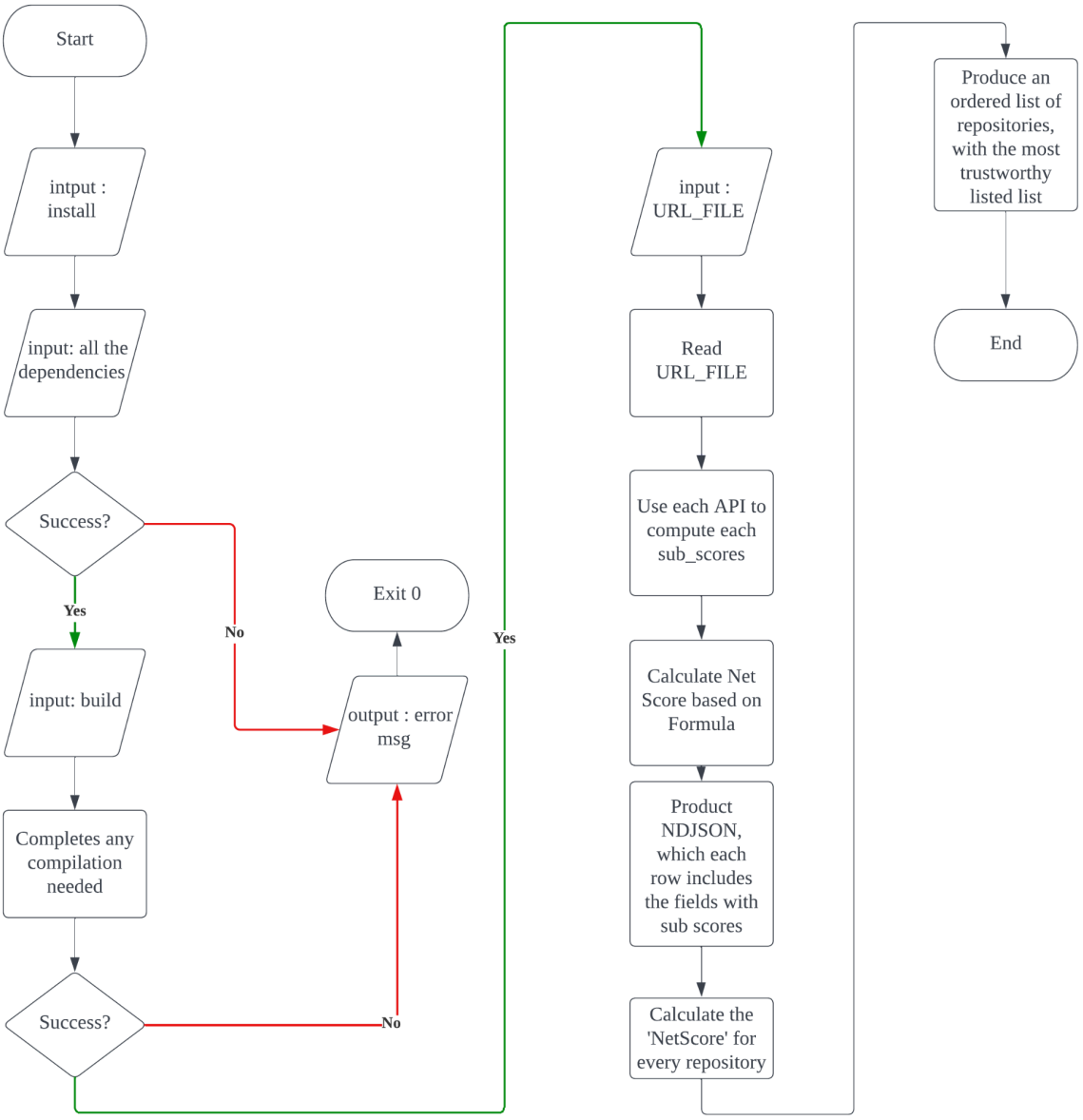
Correctness: 0.15
Bus Factor: 0.40
Responsiveness: 0.15
License Compatibility: 0.15

We have decided to give the Bus factor sub-score a weight of 0.40. The main reason is that this was the client's highest priority. For the rest of the sub-scores, we have decided to weigh them equally. However, these numbers may change based on the client's request.

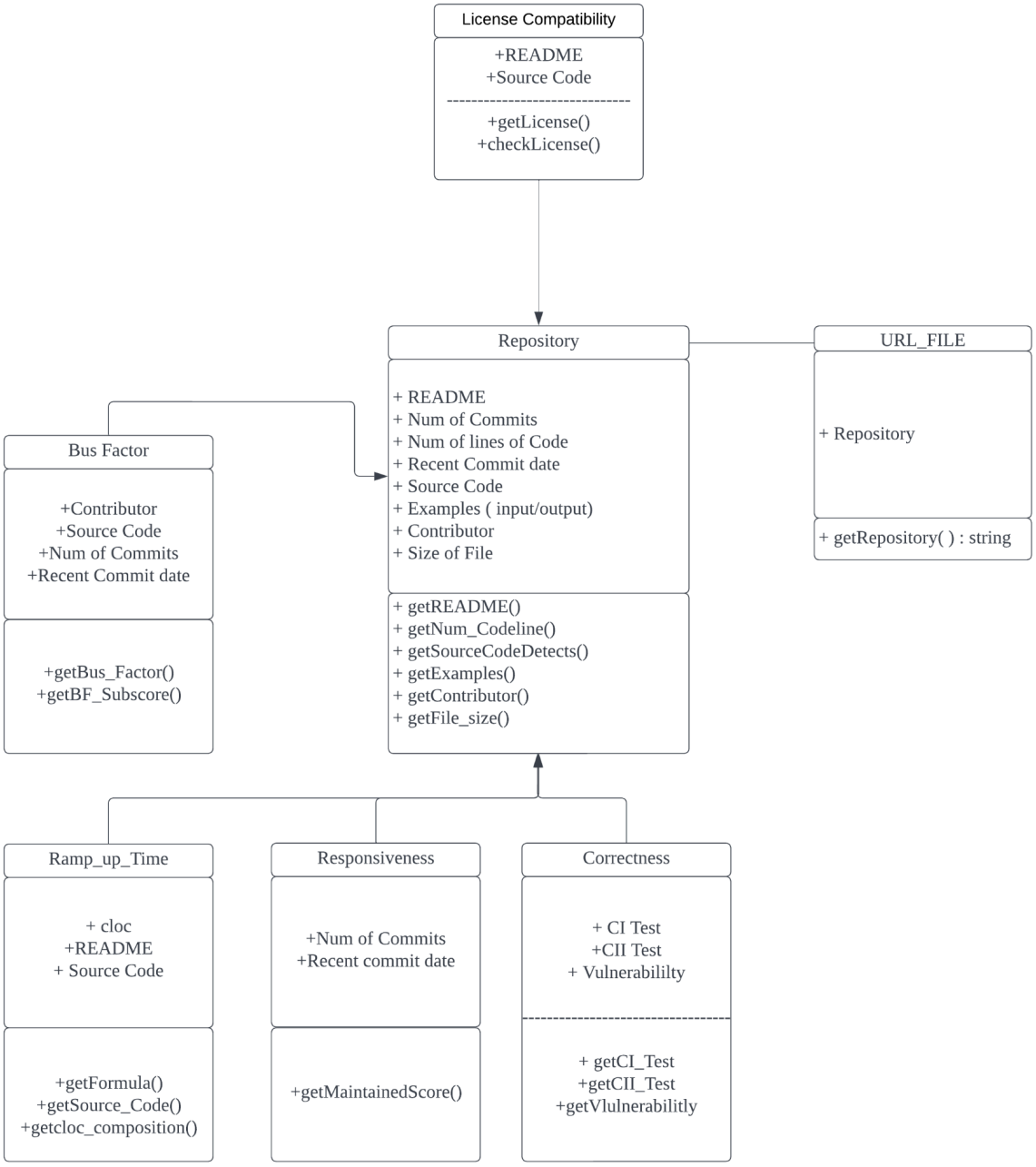
Net Score Formula [0, 1]
= (R_T sub-score) * 0.15 + (C sub-score) * 0.15+ (B sub-score) * 0.40 + (R sub-score) * 0.15 + (L_C sub-score) * 0.15

2. Diagrams to support planning

a. UML Activity Diagram



b. (Simplified) UML Class Diagram



3. **Explanations of the design of**
    a. Matrics
        i. Use each API to compute each sub-score of entity (“RampUp”, “Correctness”, “BusFactor”, “ResponsiveMaintainer”, and “License”)
        ii. find the proper API for the new entity
        iii. weight = 1 / number of entity, make a list of priority,
    b. handle URLs
        i. In order to handle non-Github-based URL, we need to implement the same method as how we handle attributes in Github-based URL
        ii. If that is not capable to handle in the same way, we have to find another method for the future

4. **Intermediate Milestones for Weeks 2, 3, and 4**
    a. Milestone 1
        i. necessary tasks & owners of the tasks
            - Goal: Implement the ‘install’ and ‘build’ parts in CLI
            - Owners of the tasks
                - Programming: all together
                - Daniel: choose proper dependencies & search how to implement the ‘build’ part using a script
                - SoWon: search how to install dependencies
                - Seunga: search how to implement the ‘build’ part using the script
        ii. estimated time to complete (Total 14 - 18 hours)
            - Implementing “install dependencies”: about 4 - 5 hours
            - Implementing “build dependencies”: about 4 - 5 hours
            - Search how to install dependencies and implement the building part using a script: about 4 - 5 hours
            - Testing: run the code and fix the bugs: about 1 - 2 hours
            - Reflection: about 1 hour
        iii. any communication required
            - Daniel and Sowon should discuss the dependencies they are going to use.
            - Daniel and Seunga should discuss the ‘build’ part using the script and the way they are going to implement it.

    b. Milestone 2
        i. necessary tasks & owners
            - Goal: Implement getting sub-score for each entity by using API
            - Owners of the tasks
                - Programming: all together
                - Daniel: search how to apply API & think about how to apply the net score formula into the software
                - SoWon: search how to apply API & think about how to manage sub scores
                - Seunga: think about how to read the URL_FILE & search how to apply API

ii. estimated time (Total 15 - 21 hours)
- Search how to apply API: about 4-6 hours
- Think about the algorithm: about 4-6 hours
- Programming: about 5-7 hours
- Testing: about 1-2 hours
- Reflection: about 1 hour

iii. any communication required
- We should discuss and decide which specific API will be applied at which location.
- Daniel and Sowon should discuss how the sub-score and net score formulas will be used.
- Seunga should tell Daniel and Sowon how the software will read URL_FILE.

c. Milestone 3

i. necessary tasks & owners
- Goal: Submit the software and write a brief report describing the status of the software in regarding requirements and specifications & optimize the code
- Owners of the tasks
  - Programming: all together (optimize)
  - Daniel: Submit & write a report & check the status of the software regarding the requirements
  - SoWon: write a report & check the status of the software regarding the specification
  - Seunga: write a report & check the status of the software regarding the specification

ii. estimated time (Total 11 hours)
- summit: about 5 minutes
- Writing report: about 9 - 10 hours
- Checking status(=Testing): about 30 minutes

iii. any communication required
- We all should discuss and check the status of the software.

5. Validation and Assessment Plan

a. Plan to assess

i. Successfully able to execute all the aspect requirements
ii. Successfully able to execute CLI
iii. Successfully able to demonstrate final output on the stdout with high standard of correctness

b. Behaviors check

i. Check coverage of code with extreme test cases
ii. When we have invalid cases, we will exit 1

c. Performance Metrics

i. Check runtime
1. using a python script to measure runtime
   a. time.time()

ii. Check the difference between the given output and our software output
    1. Compare two lists
    2. Calculate the similarity of the two lists