Team Project Part 1

A CLI for trustworthy module re-use

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 or sign your names: _____

Write today's date: _____

Assignment Goal

This assignment provides an opportunity for you to work as a team on a small software engineering project. It is also intended to expose you to the benefits and risks of re-using open-source software. Beyond some infrastructural constraints, your team has freedom of choice in terms of the programming language(s), design decisions, etc.

The primary constraint is this: Your project must contain at least 30% source lines of code written in one of:

- TypeScript
- Rust
- Golang
- C#/.NET[1]

We will measure this using the AlDanial/cloc tool.

Relevant Course Outcomes

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

- Outcome i:
  - Identify and follow an appropriate software engineering process for this context.
- Outcome ii:
  - Convert requirements into project specifications.
  - Design the software project, based on two UML diagrams.
  - Implement the project.
  - Validate the project.
  - Consider aspects of software re-use, including security risks.
- Outcome iii:
  - Experience social aspects of software engineering (communication, teamwork).

Resources

The following resources and links will help you understand and complete this assignment. Pick and choose what you need to read to succeed. You can read more if you are curious. You might divvy up reading and notetaking amongst your team.

- **REST APIs (if you choose to use them)**
  - Fielding's 2000 dissertation: You can start at Chapter 5 ("REST") but the whole thing is eminently readable and edifiying.

[1] The MONO project may be helpful for deploying to ECE machines. If you can't get this to work, (1) as a workaround, use a VM under your control; and (2) contact course staff.

    - 20 years later, brief commentary on what Fielding meant vs. what REST means in practice (and conjectures about why).
    - GitHub's REST API documentation.
- **GraphQL APIs (if you choose to use them)**
    - The GraphQL foundation has a tutorial.
    - GitHub introduction and API docs.
- **npm, the module ecosystem for JavaScript/Node.js**
    - The "Small World" paper from my friends Cris and Michael highlights some specific security concerns related to npm. I suggest you look at section 2 and sections 4.1 and 4.2 (especially Figures 3 and 5). Section 5.4 is provocative.
    - The "Trivial Packages" paper (pdf link) includes a brief introduction to the dynamics of the Node.js ecosystem. Hopefully it gives you a sense of why your "manager" at ACME Corp. might be concerned about npm package reuse in general.
    - If you're really enjoying papers, here is an early one on npm by my friend Erik.
- **Advanced GitHub**
    - Project management
    - Secret management
- **Software quality metrics**
    - Paper: Curating GitHub for engineered software projects.
    - Google's Scorecard project (introduction, repo).
- **Monetizing web services: selling "self-hosted" as a profit model**
    - Definition of self-hosting
    - Example: GitLab
- **Software licensing**
    - Wikipedia
    - Misc. article
- **Postmortems**
    - Postmortems at Google
    - Postmortems at Amazon
- **Messages**
    - Blog: Writing user-friendly error messages
    - Blog: Writing meaningful commit messages

Assignment

Introduction

Your team is a subcontractor for the ACME Corporation, which operates the ACME Web Service. One of their back-end components was recently ported to Node.js to facilitate re-using software packages between the (Type/JavaScript) front-end and the (now-Type/JavaScript) back-end. So far, Node.js has been working well, and even seems to be helping ACME Corporation recruit new engineers.

Based on the success so far, ACME Corporation's software architects are considering bringing up new Node.js-based services. Your team provides **infrastructure services** for ACME Corporation, and you are being asked to make it easy for the service teams to get started.

You have been looking into npm, the package manager for Node.js, and are excited to see so many modules (over 2 million!). Your team's contact at ACME Corporation, Sarah, is open to re-using these modules, but she is concerned about a few things:

- She knows open-source documentation can be sparse, and wants to make sure it is relatively easy for their engineers to learn the new module ("low ramp-up time").
- She worries that an open-source module might be held to a low standard of correctness.
- She wants to make sure that maintainers will be responsive to fix any bugs that are blocking ACME's teams.
- She is concerned that an open-source module might not have enough maintainers to continue to apply critical fixes such as a security patch. This is her highest priority.
- She said she might add some more qualities later, so your design should be able to accommodate adding new aspects.

In addition to Sarah's concerns, ACME Corporation currently offers its web service product directly via a REST API. However, she told you that in the three-year roadmap, they are exploring a licensed version of the web service that its customers can deploy internally: **self-hosted ACME**. Some considerations:

- 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.
- ACME Corporation uses the GNU Lesser General Public License v2.1 for all open-source software.
- Any modules that ACME Corporation relies on could then be distributed as part of this product. Therefore, any open-source module's licenses that ACME Corporation's service engineers use must be compatible with the LGPLv2.1 license. You may suppose that the license description is given in the project README, like this example: https://github.com/nodejs/node#license (you might employ a regex here).

Sarah has asked your contracting team to prepare a tool to help the ACME service engineering teams choose modules wisely. She suggested that you start with a command-line interface. She says it would be nice if your tool is "not super slow". In addition, she prepared an initial specification. See the following for details.

Sarah's initial project specification

System input

- Should support input from command line arguments.

System output

- Should produce an ordered list of repositories, with the most trustworthy listed first.
- Each repository should be accompanied by its overall score, as well as its sub-scores for "ramp-up time, "correctness", "bus factor", "responsiveness", and "license compatibility".
- Should print all output to stdout (though this output mode might change in the future).

Internal Requirements (To make the course staff's lives easier!)

Auto-grader API

We will auto-grade part of the project. To this end:

- There should be an executable file in the root directory of your project called "run".
- It should have the following CLI when executed on a Linux machine (note that you can use whatever CLI you want; you can then wrap that CLI within this auto-grader-friendly CLI):
- "./run install"
    - Installs any dependencies in userland
    - Should exit 0 on success
- "./run build"
    - Completes any compilation needed
    - Should exit 0 on success
- "./run URL_FILE", where URL_FILE is the absolute location of a file consisting of an ASCII-encoded newline-delimited set of URLs.
    - These URLs may be in the npmjs.com domain (e.g. https://www.npmjs.com/package/even) or come directly from GitHub (e.g. https://github.com/jonschlinkert/even).
    - This invocation should produce NDJSON output. Each row should include the fields: "URL", "NetScore", "RampUp", "Correctness", "BusFactor", "ResponsiveMaintainer", and "License".
    - Each score should be in the range [0,1] where 0 indicates total failure and 1 indicates perfection. The specific operationalizations are up to you, but you must provide rationales as part of your documentation.
    - The "NetScore" should be calculated as [0,1] as well, as a weighted sum. You should choose the weights based on Sarah's priorities, and explain your choice.
    - Should exit 0 on success.

- "./run test", which runs a test suite and exits 0 if everything is working.
    - The minimum requirement for this test suite is that it contain at least 20 distinct test cases and achieve at least 80% code coverage as measured by line coverage.
    - The output from this invocation should be a line written to stdout of the form: "X/Y test cases passed. Z% line coverage achieved."[2]
    - Should exit 0 on success.
- In the event of an error, your program should exit with return code 1, and print a useful error message to the console. Look at the resource on error message design for guidance.

Your software must produce a log file stored in the location named in the environment variable $LOG_FILE and using the verbosity level indicated in the environment variable $LOG_LEVEL (0 means silent, 1 means informational messages, 2 means debug messages). Default log verbosity is 0.

Before submitting, ensure you run your software on the ECEPROG server[3] and confirm it runs successfully according to the "auto-grader" interface. This is to ensure your software compiles and runs successfully when we try testing or auto grading your software.

The course staff will publish input/output examples for you to test with. These will be available by January 30.

Metric calculations

At least two of your metrics must use data from the GitHub API (e.g. examining the contributors, issues, pull requests, etc.).

- Many npm modules are stored on GitHub. Your software need only support metric calculations on modules that are hosted on GitHub (although your software should behave "appropriately" in other hosting circumstances).
- You must create GitHub tokens to programmatically access the GitHub API (FYI there is a rate limit).
- At least one metric must use the REST API.
- At least one metric must use the GraphQL API.
- You must not conduct "web scraping" of GitHub, where you hit the web service with raw URLs and parse the resulting HTML. Oh, the inefficiency…
- You should not upload your tokens to a publicly-visible location. GitHub tokens should always be specified by the environment variable $GITHUB_TOKEN. (That way we can provide our own token for grading).

[2] This data should be truthful. Don't hardcode things please.

[3] ECEGRID has retired. See https://engineering.purdue.edu/ECN/Support/KB/Docs/ECEThinlinc for the new machines.

At least one of your metrics must use data from the source code repository, not using the GitHub API. To conduct this analysis, you should:

- Clone the repository locally
- Write the analysis using a programming language of your choice.
- You might want to interact with the Git metadata programmatically.
    - Use a Git library, such as: Java-Git (Jgit), Python (GitPython)
    - You *cannot* implement analysis by "shelling out" to the git bash CLI
- You might also choose to analyze the software itself. **Do not execute the software nor its test suite in an unsandboxed environment.** If the software is malicious, you expose yourself to substantial risk. But you could consider a static analysis, e.g. attempting to parse and walk the AST using a JavaScript parsing tool to do something like measure comment lines.

Source code hosting

Your team's repository should be shared publicly on GitHub.

- This will promote good practices with respect to tokens and keys. (See Resource "Secret management".)
- It will allow you to share it with future employers if you are so inclined.
- Per the academic honesty requirements of this course, and unless otherwise indicated, you are not permitted to work with other teams, compare or copy software implementations, and so on.

Project management

I recommend but do not require that you use GitHub Project Boards for progress tracking.

Software re-use

You are allowed to re-use existing software to support your implementation, either as tools (e.g. VSCode; git; GitHub; TravisCI) or as components in your implementation (e.g. a module to help you parse command-line arguments). **Your documentation must include a justification of any components you choose to re-use** – how will you decide whether they are trustworthy? (discuss in the Project Plan), and how did that assessment work out in practice? (discuss in the Project Postmortem)

You are allowed to re-use code snippets from software engineering resources such as Stack Overflow. You must provide a citation (web URL is fine) to the relevant post. Also, please review Prof. Davis's general perspective on Stack Overflow. Technically, Stack Overflow snippets are themselves governed by a software license, but you are not trying to distribute your project code and I do not think anyone would seriously pursue litigation along these lines.

You are **not** allowed to copy-paste code snippets out of an open-source project – this is a great way to expose ACME Corporation (and your future employer in the real-world) to lawsuits. Any re-use of this nature must use existing module APIs and/or extend those APIs so that you can access the logic you want.

Timeline and Deliverables

The project will be completed over a 5-week period:

- Week 1: Initial Planning and Design
- Weeks 2-4: (Inevitable re-designs, and) Implementation, Validation, Delivery
- Week 5: Postmortem

Week 1: Design and Planning

One member of your teams should submit a Project Plan Document (Word Doc or PDF) including the following:

- Repository URL
- Tool selection and preparation
    - Programming language, toolset, component selection [linter? Git-hooks? CI? Testing framework? Logging library?]
    - Communication mechanism [Slack? Teams? Email?]
    - Statement that GitHub tokens are obtained
- Team contract
    - For example, your team might agree: to do the work they take on, to document their code, to follow Greg Wilson's meeting guidance, to have a TDD style (or not), to follow a style guide, to use code review, …
- Team synchronous meeting times
    - I recommend at least one (short) mid-week sync to discuss issues, and one end-of-week sync to put together your weekly reports.
- Requirements
    - A refined and organized list of requirements, based on Sarah's description and specification.
- Preliminary design
    - Metric operationalizations and net score formula
    - Diagrams to support planning
        - UML Activity Diagram to depict the activities performed by your system.
        - (Simplified) UML Class Diagram to depict the critical entities in the system.
            - (Photos of a whiteboard are fine, or online UML tools like those provided by LucidChart or draw.io or diagrams.net)
    - Explanation of the design of the "metrics" feature so that you can accommodate Sarah's projected need to add new metrics later. What logical flow and what entity structure (e.g. classes) did you select to improve the modularity of this portion?
    - Explanation of the design of the "handle URLs" feature so that you can accommodate URLs from either npm or GitHub. What logical flow and what

entity structure (e.g. classes) did you select to improve the modularity of this portion?

- Planned intermediate milestones for weeks 2 and 3, plus the final milestones for week 4
    - Each milestone should list the necessary tasks, the owners of those tasks, the estimated time to complete it, and how success will be measured.
    - Any communication requirements between tasks should be noted, e.g. "Jason and Tahani need to discuss the interface involved between task A and task B."
- Validation and Assessment plan
    - What is your plan to assess whether the delivered software satisfies Sarah's requirements? What behaviors will you check? What performance metrics (if any) will you apply?

Weeks 2-3: Complete your milestones

Each week, submit a report with your updated list of milestones, tasks, etc. representing completion and the actual time spent by each team member on the project.

This report should be self-contained, e.g. including the relevant information from the original plan.

If you *deviate substantially* from your timeline, consider attending one of the course staff office hours to discuss the deviation.

Week 4: Deliver the software

Submit the software itself, along with brief report describing the status of the software in relation to Sarah's requirements and specification.

- If your submission will not survive the auto-grader described above, provide explanatory notes so the course staff can score you fairly.[4]
- Provide one example of a module that you think your approach scores well.
- No automated measurement is perfect. Provide one example of a module that you think your approach scores poorly in some regard. How could you modify your design to improve the outcome for this module?
- Provide the URL to your project repository in your report

Week 5: Postmortem

Deliver a project postmortem report. This report should reflect on each aspect of your Plan (from week 1) compared to your Execution. What went well? What went poorly? Where did your time estimates fail? When and why did you deviate from your Plan? For all of these questions, try to answer the question "Why?"

[4] Pro-tip: After you have a design, build an end-to-end skeleton to get the interfaces working.

Grading rubric

Points breakdown:

- 30% Design & Planning document + Milestone documents.
- 60% Working delivery, broken down as:
    - 35% "It runs and follows the auto-grader interface above"
    - 10% "It has a reasonable-looking test suite that achieves the required coverage"
    - 5% "Per our manual inspection, the software follows reasonable-looking engineering practices, e.g. good file/class/variable names, consistent style, choice of data structures, use of patterns to isolate what is changing, clean commit messages, pull requests, code review"
- 10% Post-mortem.

(The project handoff will be graded as its own entity).

Provided that the teammates complete the tasks they were assigned as part of the project plan, all team members will receive the same grades. If there is an issue with teamwork, please raise it with Prof. Davis as early as possible.

- Your team's milestones should allow you to observe problems with forward progress.
- For personality clashes etc., use your judgment to determine if you want to speak with Prof. Davis.

ACME Corporation's Budget is not Bottomless

Sarah reminds you that your team members are from an independent contracting firm. She says the company is **willing to pay your team for up to 40 hours per person for this project**[5], and would rather see ***something that works – at least partially! – by the deadline.***

- Your project plan and your weekly progress updates should reflect an appropriate amount of time for the project, e.g. 6-9 hours per team member per week. If you wait until the last minute, Sarah will be nervous, pull the plug on the project, and might break off future contracts with your company.
- If you begin to deviate from your planned timeline, you should submit a **revised** plan as part of a weekly update. That way Sarah can keep management abreast of progress and aware of any changes in the functionality that will be delivered.[6]
- You should plan your project in such a way that you can deliver incremental value to Sarah even if you cannot complete all of her requirements.

[5] "40 hours per person for this project" – Since the project will run for 5 weeks, that will average out to ~8 hours per teammate per week. The postmortem week should be lighter and the earlier weeks a little heavier.

[6] Deviations might include "We cannot do this part that we planned, for (good) reason X", accompanied by a reduced project scope. ***Provided that your team is following a reasonable process and the trouble is their cost estimates, reducing scope will not negatively impact your grade. However, you should not report major problems at the last minute – does that sound like a reasonable engineering process to you?***

- Recall the aircraft requirements document from the Requirements Engineering unit – one of the final chapters designated useful subcomponents that the vendor could deliver.