Project Phase 1

A CLI for Trustworthy Module Re-Use

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.

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. Some additional resources and clarifications are sprinkled throughout as footnotes.

- **UML diagrams**
  - IEEE 1016-2009: IEEE Standard for Information Technology--Systems Design--Software Design Descriptions
  - UML per Wikipedia (many helpful links)
- **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.
  - 20 years later, brief commentary on what Fielding meant vs. what REST means in practice (and conjectures about why).
  - GitHub's REST API documentation.
  - I wrote a paper with 2 Purdue undergrads involving REST APIs, you might enjoy: https://docs.lib.purdue.edu/cgi/viewcontent.cgi?article=1179&context=ecepubs

- **GraphQL APIs (if you choose to use them)**
  - The GraphQL foundation has a tutorial.
  - GitHub introduction and API docs.
  - Davis wrote some papers with IBM involving GraphQL, search the website for "GraphQL": https://davisjam.github.io/publications/
- **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.
  - I wrote a paper that measured other cybersecurity aspects of npm: https://davisjam.github.io//files/publications/DavisCoghlanServantLee-EcosystemREDOS-ESECFSE18.pdf
- **Project management using GitHub Projects**
  - 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
  - This article is mostly about variations in *commercial* licensing. I thought it was interesting. It will not help you in the project but it will make you a better engineer.
- **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 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,[1] 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.[2]
- 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 wants to make sure that maintainers will be responsive to fix any bugs that are blocking ACME's teams.
- 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 one of two places: either in the project README, under a Markdown heading "License", as in this example: https://github.com/nodejs/node#license (you might use a regex to parse); or as a separate file in the root directory named LICENSE.{extension}.

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 (Project Phase 1), get feedback, and then move to a web service (Project Phase 2). She says it would be nice if your tool is "not super slow", and would appreciate some performance measurements. In addition, she prepared an initial specification. See the following for details.

[1] ACME Corp. maintains an in-house engineering staff that establishes requirements and architectures for the software infrastructure on which they rely, but the actual design and implementation of this software is contracted out.

[2] This term ("standard of correctness"), like many others in this document, is left deliberately vague. Operationalizing it is part of your task.

Sarah's initial project specification

System input

- Should support input from command line arguments.

System implementation

- Should be majority TypeScript

System output

- Should print all output to the stdout (though this output mode might change in the future, so design accordingly).
- 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".

Sarah's other requirements

Sarah wants to know how long it will take your system to respond. Please select and justify some representative input projects to show the cost as the projects vary, e.g. in size.

For better latency, Sarah wants each metric to be calculated in parallel, although she acknowledges that you may wish to consider the number of available cores when deciding on the level of parallelism.

Auto-grader API (To make the course staff's lives easier!)

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

- There should be an executable file in the root directory of your project. This file should be named "run".
  - *NB: When I say that this file should be executable, I mean that means its permissions are set to executable. That does not mean it must be a compiled program. I guess it could be. But you can write it in whatever programming language you want. Just run "chmod +x run" on it to set its permissions to executable.*
- The file "run" 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 (e.g. *pip install --user*).
    - Should exit 0 on success, non-zero on failure
  - "./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", "NetScore_Latency", "RampUp",

“RampUp_Latency”, “Correctness”, “Corectness_Latency”, “BusFactor”, “BusFactor_Latency”, “ResponsiveMaintainer”, “ResponsiveMaintainer_Latency”, “License”, and “License_Latency”.

    - 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; you must design and justify them in your report.
    - Latency values should reflect the time to calculate that component of the net score. Report values in seconds and round results to three decimal places (i.e., to the nearest millisecond).
    - 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, non-zero on failure
  - "./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.”
    - Should exit 0 on success, non-zero on failure

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[3] $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.[4]

Before submitting, ensure you run your software on the ECEPROG server[5] 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 on Brightspace by Friday August 30.

[3] In documents like this one, environment variables are written in ALL_CAPS, use snake_case, and sometimes have a $ in front. Sorry about the $, this is an old habit of mine. When I write “the environment variable $GITHUB_TOKEN” I mean that there is a variable defined in the environment whose name is GITHUB_TOKEN, which you should access using (in TypeScript) something like: `const githubToken = process.env.GITHUB_TOKEN`.

[4] There are many views on what and how much to log. Some good resources are (1) this blog; and (2) this SO post.

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

Metric calculations

At least one 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, eg returning a suitable error message).
- You must create GitHub tokens to programmatically access a GitHub API. This API is rate-limited and you should design your interactions with the API appropriately.
  - You may use either the REST API or the GraphQL API.
  - You must not conduct excessive "web scraping" of GitHub, where you hit the web service using raw URLs and parse the resulting HTML.
- You should not upload your tokens to a publicly-visible location. This would allow other people to impersonate you, with unpleasant consequences.
- GitHub tokens should be provided specified by the environment variable $GITHUB_TOKEN.

At least one of your metrics must perform an analysis of the package without using the GitHub API. To conduct this analysis, you should clone the repository locally

- You might then want to interact with the Git metadata programmatically. If you do so:
  - You *cannot* implement analysis by "shelling out" to the git bash CLI
  - Instead, use a Git library, such as isomorphic-git (https://github.com/isomorphic-git/isomorphic-git).
- You might also choose to analyze the software itself. You might employ a static analysis, e.g. attempting to parse and walk the AST using a JavaScript parsing tool. You might also apply a dynamic analysis, e.g. running its test suite and doing something with the resulting information. If you use dynamic analysis, you may make reasonable assumptions about the test frameworks being used, e.g. supporting the most popular 1-2 frameworks (bring evidence to justify this).

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

You must use GitHub Project Boards for progress tracking. You should refer to your board during your weekly milestone updates.

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). You should 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.

Your team must use one or more large language models. Here are some suggestions:

- **To accelerate your implementation**: Within your IDEs, there are many pluggable implementation-type LLMs that can help you write code. Two examples are GitHub's CoPilot and Meta's Code Llama. As a student, you can access GitHub's tools through the GitHub Education program.
- **To support your other software engineering activities**: At time of writing, many companies offer free chatbot-style LLMs, including Claude (Anthropic) and ChatGPT 3 (OpenAI). Through Purdue, you have access to Office 365, which includes Microsoft's commercial tool Microsoft CoPilot (not to be mistaken for GitHub CoPilot). These tools can support many other phases of the software development lifecycle, such as:
    - Requirements analysis ("*ChatGPT, what does the following confusing passage from Prof. Davis's giant Word Doc mean?*")
    - Software design ("*How might these components communicate?*")
    - Validation ("*Any recommendations for automated testing of a component that does X?*")
    - Project management ("*Here are our skills. How might we divide up work?*")
    - Learning new skills and technologies ("*Can you summarize the different AWS Free Tier options for persistent data storage?*")

Your Project Plan should include a description of how you used the LLM in a responsible way. Remember that the course has a "Bring Your Own Brain" policy. If the LLM makes a mistake then it is you who will bear the consequences. Here is some general advice from Prof. Davis on the proper use of tools and brains.

Timeline and Deliverables

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

- Week 1: 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. This document should be organized with headings, such that there is a usable “Table of contents” to make it easier for people to read. (In Word, this shows up under “Navigation Pane”):

- Tool selection and preparation
    - Toolset, component selection [linter? Git-hooks? CI? Testing framework? Logging library?]
    - Communication mechanism(s) [Slack? Teams? Email?]
    - Statement that GitHub tokens are obtained and a repo created
- Team contract
    - For example, your team might agree to: do the work they take on, document their code, follow testing rules, follow style guide, and to communicate in advance if they cannot deliver on schedule
- Team synchronous meeting tempo and 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. These should be drawn using LucidChart or similar. I expect at least two, imitating the purpose (if not the exact style) of these UML diagram types:
        - UML Activity Diagram to depict the activities performed by your system.
        - Simplified UML Class Diagram to depict the critical entities in the system and how they will relate to each other.
    - 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. class hierarchy?) 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, hierarchies?) did you select to improve the modularity of this portion?
- Planned milestones for weeks 2-4
    - Each milestone should list the necessary tasks, the expected owners of those tasks, the estimated time to complete it,[6] and how success will be measured.

[6] Bad news: You are bad at estimating how long things will take. Good news: With careful practice, you can get better. As a simple rule of thumb, add 50% to your team’s best guess.

  - 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 internal 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.[7]
- 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?"

See the resources on "Postmortems" at the beginning of this document.

Grading rubric

Points breakdown:

- 30% Design & Planning document + Milestone documents.
- 60% Working delivery, broken down as:
  - 30% "It runs and follows the auto-grader interface above"

For more reading, this blog post is chock full o' wisdom.

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

  - 10% "It has a reasonable-looking test suite that achieves the required coverage"
  - 10% "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".[8]
- 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**[9], 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 **<u>revised</u>** plan as part of your Week 3 update. That way Sarah can keep management abreast of progress and aware of any changes in the functionality that will be delivered.
- 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.
  - Recall the aircraft requirements document from the Requirements Engineering unit – one of the final chapters designated useful subcomponents that the vendor could deliver.

[8] Specific elements assessed in the grading rubric include (1) the presence of a main README and in any sub-directories to document the design – no project should leave home without them! – and (2) suitable module-level (top of file) and function-level comments documenting the implementation

[9] "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.