# Project 1: Delivery rubric

50%

* 30%: "It runs and follows the auto-grader interface above"
* 10%: "Reasonable-looking test suite"
* 10%: "Manual inspection for engineering practice"

Create the following test files for the simple auto-grader checks:

* "One-URL file": A single npmjs URL
* "Two-URL file": One npmjs URL, one GitHub URL

There are a variety of toy repos you should create to test against. They are described below. One of them is time sensitive (responsiveness).

|  |  |  |  |
| --- | --- | --- | --- |
| Component | Points | | Rubric |
| Auto-grader: install | | 3 | * + 1 point for rc (return code) 0   + 1 point if a *subsequent* call to "./run test" yields rc 0   + 1 point if a *subsequent* call to "./run URL\_FILE" with 1 URL yields rc 0 |
| Auto-grader: test | | 3 | * + 1 point for printing the correct output string: "X/Y test cases passed. Z% line coverage achieved".   + 2 point if the suite achieves at least 80% coverage |
| Auto-grader: URL\_FILE interface | | 5 | * + 1 point: The one-URL file, LOG\_FILE=/tmp/log, LOG\_LEVEL= and GITHUB\_TOKEN=<token> set produces parse-able stdout and rc 0.     - In such tests, you should always delete LOG\_FILE before you run their program. The spec is silent on what to do when LOG\_FILE exists already, so let's just not check that.   + 0.5 point: All the scores are in the range [0,1] or have the value -1   + 0.5 point: The NET\_SCORE value is no smaller than the smallest non-negative score, and no larger than the largest non-negative score   + 1 point: Can handle the two-URL file -- parse-able stdout, rc 0, and the URLs are in the correct order   + 1 point: If LOG\_FILE is not defined, exits with rc 1   + 1 point: If GITHUB\_TOKEN is not defined, exits with rc 1 |
| Auto-grader: Logging | | 2 | * + 1 point: LOG\_LEVEL 0 results in an empty LOG\_FILE   + 1 point: LOG\_LEVEL 1 on the Two-URL file yields some logging, but less than LOG\_LEVEL 2 |
| Auto-grader: GitHub token | | 2 | * + 1: Using a valid GitHub token works   + 1: Using an invalid GitHub token (e.g. the string "INVALIDTOKEN") yields rc 1 and a reasonable error message |
| Auto-grader: RAMP\_UP\_SCORE | | 2 | * + 1: Gives a score >= 0.5 to jQuery indicated via an npmjs URL: <https://www.npmjs.com/package/jquery>   + 1: Gives a score of < 0.5 to a GitHub-based JS project under our control, which has a few lines of code, no comments, and an empty README. (e.g. make a copy of <https://github.com/dcousens/is-sorted> and strip the comments) |
| Auto-grader: CORRECTNESS\_SCORE | | 2 | * + 1: Gives a score >= 0.5 to a project with a test suite -- e.g. has a directory somewhere in the tree named "test" and an entry in package.json for the "test" command     - Confirm that "npm run test" actually works on this project, since they may execute it.   + 1: Gives a score < 0.5 to a project that does NOT meet the preceding condition |
| Auto-grader: BUS\_FACTOR\_SCORE | | 2 | * + 1: Gives a score >= 0.5 to jQuery   + 1: Gives a score < 0.5 to a clone of <https://github.com/substack/safe-regex> (1 maintainer, last committed years ago) |
| Auto-grader: RESPONSIVE\_MAINTAINER\_SCORE | | 2 | * + 1: Gives a score >= 0.5 to a toy repo with 3 issues, all resolved within 1 day   + 1: Gives a score < 0.5 to a toy repo with 3 issues, all open and never resolved     - NB: You should make these repos **this week** so the open issues have time to get old |
| Auto-grader: LICENSE\_SCORE | | 3 | * + 1 point: A project that uses the same license gets a 1   + 1 point: A project that uses a compatible license gets a 1   + 1 point: A project that uses an incompatible license gets a 0 |
| Auto-grader: General principles | | 4 | * + 1 point: If we point them at an empty repo (github URL; no files, no history, etc.), then their program should assign scores of 0 to every field, with a net score of 0.   + 1 point: If we point them at a meaningful repo (use jQuery), then their program should assign scores of either -1 or "> 0, <= 1" to each metric.   + 1 point: Given an npmjs URL and it’s associated GitHub URL, their program should calculate the same scores for both URLs.   + 1 point: If we run a file with the same NPM module repeated multiple times, student’s program completes successfully without throwing any error. |

Reasonable-looking test suite: 10% (This one is graded via a brief manual inspection)

* 3 points: Contains at least 20 test cases
* 3 points: Each metric has at least 1 test case
* 1 points: Contains at least 2 end-to-end tests
* 2 points: Contains at least 10 unit tests ("small scope")
* 1 points: Has a test case for at least 1 error condition ("should throw" or similar)

Fair partial credits to be awarded where the project does not completely satisfy the testing requirements.

Manual inspection for engineering practice: 10%

|  |  |  |
| --- | --- | --- |
| Documentation | 3 | * + 1 point: Has a README that explains the purpose of the project, how to configure it, and how to invoke it   + 1 point: README describes the format of valid input and valid output, including a copy-pasteable example (in one block, or with interleaved explanatory text)   + 1 point: Three randomly chosen files all have introductory remarks at the top and some semblance of JavaDoc/PyDoc/etc. throughout |
| Effective communication | 2 | * + Look at the "run" program, some file for metrics, and some file associated with talking to github (e.g. as enforced by a linter     - 1 point: Are the names of variables and functions “pretty good”     - 1 point: Consistent style across these files |
| Anticipating natural extensions | 2 | * + 1 point: No hardcoded bound on the number of URLs they can process   + 1 point: Metrics are designed to let us add a new metric easily, e.g. there is a consistent interface for each metric, and we would need to add one entity (like an object or another header file) and update one entity (like a list of metrics) in order to add the metric |
| Teamwork | 3 | * Each teammate contributed at least 20% of the code according to a simple “sum the lines/authors via git blame” check. There is probably some script for this or you can roll your own. |
| Planning ahead and dropping features | ? | * **If the team notified us of a dropped feature in advance** (i.e. in a milestone), you may return up to all of the lost points for that feature. Use your judgment. The awards should be based on:   + Criticality of the feature (recall that the spec says the customer cares about some features more than others)   + How far in advance they notified us   + How well they justified dropping support (e.g. a breakdown of the hours vs. just “we cannot do it”)   + The degree of “paper trail” justifying their effort. If their milestones have hours worked, and those hours are 8-10/person, then dropping a feature is entirely reasonable. *This is the only case where returning more than half of the points is reasonable.* |

**Caveat: If at any point during inspection you detect evidence of hardcoding to address any of these cases, give the team a 0/50 for their deliverable.**