Proposal to adopt C4 for SG45 repositories
Collective Code Construction Contract (C4)

https://rfc.zeromq.org/spec:42/C4/

http://zguide.zeromq.org/page:chapter6

Morgan C. White

June 26, 2019

NNSA
National Nuclear Security Administration
Managed by Triad National Security, LLC for the U.S. Department of Energy's NNSA

Observations regarding public and controlled information and the implications on our current project

- There is a considerable body of laws and regulations governing the exchange and use of nuclear technology
- From the start, the US Atomic Energy Act helped define clear boundaries regarding public, export controlled and classified information
  - *The intent was to support the Atoms for Peace initiative*
  - Nuclear theory, nuclear structure, decay and reaction data (*fundamental data*)
  - Benchmark experiments, e.g. criticality and shielding (*set the line here*)
  - Detailed design information regarding nuclear fuel cycle or weapons (*controlled*)
- This is clearly evident in our historical records
  - Fundamental data are available in public domain databases
  - Benchmark data have been consistently, publicly shared over the decades Nuclear Science and Engineering articles (1950s), Los Alamos reports LA-2415-MS, LA-3067-MS, LA-4208, CSEWG BNL-202 (“crit handbooks”)

US Open Data Policy

- In recent times, there has been a consistent movement within the US Government towards an Open Data Policy both (1) to make more transparent the process and basis for such decisions, and (2) to make these data available to the public and commercial sectors to fuel growth of new entrepreneurship, innovation and scientific discovery. To the extent permissible by US Government law and executive order, data and methods used to arrive at safety and security decisions should be publicly provided to enable peer review of these important decisions. Within nuclear technology we quote herein the following goals to which this effort will contribute.

OMB Memorandum M-13-13, Open Data Policy - Managing Information as an Asset (May 9, 2013).

https://digital.gov/open-data-policy-m-13-13/

Getting down to the “nitty gritty” details

- Organize and categorize information intended for public access and ensure it is searchable across agencies. The procedures to cost-effectively fulfill this requirement are outlined in two categories below. Increasingly sophisticated Internet search functions (including their crawl and index mechanisms) greatly assist agencies in this area.
  - When disseminating information to the public-at-large, publish your information directly to the Internet. This procedure exposes information to freely available and other search functions and adequately organizes and categorizes your information. [*In this case, commitment to NEA GitLab as community resource.*]
  - When interchanging data among specific identifiable groups or disseminating significant information dissemination products, advance preparation, such as using formal information models, may be necessary to ensure effective interchange or dissemination. This procedure is needed when freely available and other search functions do not adequately organize and categorize your information. [*Agreement to C4 to organize and control the process.*]

Thoughts on building constructive communities

- Manners, rules, contracts and laws exist to reduce friction so that we may live and work together in constructive exchanges
  - Most conflict derives from misunderstandings regarding expectations
  - We need to be clear up front regarding our shared intent

- The best communities continuously consider these issues, define the expectations clearly, and revise them as needed
  - Open source software has existed long enough to have tried many variants

- The ZeroMQ project has one of the most mature, well-described community philosophy, backed by legally enforcable contracts
  - *The proposal herein is to adopt these practices for our work*

The Collective Code Construction Contract (C4) is a mature, well-constructed open-source framework

C4 is meant to provide a reusable optimal collaboration model for open source software projects. It has these specific goals:

- To maximize the scale and diversity of the community around a project, by reducing the friction for new Contributors and creating a scaled participation model with strong positive feedbacks;
- To relieve dependencies on key individuals by separating different skill sets so that there is a larger pool of competence in any required domain;
- To allow the project to develop faster and more accurately, by increasing the diversity of the decision making process;
- To support the natural life cycle of project versions from experimental through to stable, by allowing safe experimentation, rapid failure, and isolation of stable code;
- To reduce the internal complexity of project repositories, thus making it easier for Contributors to participate and reducing the scope for error;
- To enforce collective ownership of the project, which increases economic incentive to Contributors and reduces the risk of hijack by hostile entities.

Clearly define Roles and Responsibilities

- The project SHALL use the git distributed revision control system.
- The project SHALL be hosted on NEA GitLab or equivalent, herein called the "Platform".
- The project SHALL use the Platform issue tracker.
- The project SHOULD have clearly documented guidelines for code style.
- A "Contributor" is a person who wishes to provide a patch, being a set of commits that solve some clearly identified problem. [*New benchmark, revision or typo.*]
- A "Maintainer" is a person who merges patches to the project. Maintainers are not developers; their job is to enforce process.
- Contributors SHALL NOT have commit access to the repository unless they are also Maintainers.
- Maintainers SHALL have commit access to the repository.
- Everyone, without distinction or discrimination, SHALL have an equal right to become a Contributor under the terms of this contract.

Licensing and Ownership

- The project SHALL use the *GNU Free Documentation License*.
- All contributions to the project documents SHALL use the same license as the project.
- All documents are owned by their authors. There SHALL NOT be any copyright assignment process.
- The copyrights in the project SHALL be owned collectively by all its Contributors.
- Each Contributor SHALL be responsible for identifying themselves in the project Contributor list.

GNU Free Documentation License

- The purpose of this License is to make a manual, textbook, or other functional and useful document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.
- This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.
- We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

https://www.gnu.org/licenses/fdl-1.3.html

Patch Requirements

- Maintainers and Contributors MUST have a Platform account and MUST use their real names. [*Because of the content, we must require use of verifiable identities.*]
- A patch SHOULD be a minimal and accurate answer to exactly one identified and agreed problem. [*again, new, revised or fixed contribution*]
- A patch MUST adhere to the code style guidelines of the project if these are defined.
- A patch MUST adhere to the "Evolution of Public Contracts" guidelines defined below.
- A patch SHALL NOT include non-trivial code from other projects unless the Contributor is the original author of that code. [*Don’t contribute others work.*]
- A patch MUST compile cleanly and pass project self-tests on at least the principal target platform. [*the input deck must run*]
- A patch commit message MUST consist of a single short (less than 50 characters) line stating the problem ("Problem: ...") being solved, followed by a blank line and then the proposed solution ("Solution: ...").
- A "Correct Patch" is one that satisfies the above requirements.

Development Process [1/2]

- Change on the project SHALL be governed by the pattern of accurately identifying problems and applying minimal, accurate solutions to these problems.
- To request changes, a user SHOULD log an issue on the project Platform issue tracker.
- The user or Contributor SHOULD write the issue by describing the problem they face or observe.
- The user or Contributor SHOULD seek consensus on the accuracy of their observation, and the value of solving the problem.
- Users SHALL NOT log feature requests, ideas, suggestions, or any solutions to problems that are not explicitly documented and provable.
- Thus, the release history of the project SHALL be a list of meaningful issues logged and solved.
- To work on an issue, a Contributor SHALL fork the project repository and then work on their forked repository.
- To submit a patch, a Contributor SHALL create a Platform pull request back to the project.
- A Contributor SHALL NOT commit changes directly to the project.
- If the Platform implements pull requests as issues, a Contributor MAY directly send a pull request without logging a separate issue.
- To discuss a patch, people MAY comment on the Platform pull request, on the commit, or elsewhere.

Development Process [2/2]

- To accept or reject a patch, a Maintainer SHALL use the Platform interface.
- Maintainers SHOULD NOT merge their own patches except in exceptional cases, such as non-responsiveness from other Maintainers for an extended period (more than 1-2 days).
- Maintainers SHALL NOT make value judgments on correct patches.
- Maintainers SHALL merge correct patches from other Contributors rapidly.
- Maintainers MAY merge incorrect patches from other Contributors with the goals of (a) ending fruitless discussions, (b) capturing toxic patches in the historical record, (c) engaging with the Contributor on improving their patch quality.
- The user who created an issue SHOULD close the issue after checking the patch is successful.
- Any Contributor who has value judgments on a patch SHOULD express these via their own patches.
- Maintainers SHOULD close user issues that are left open without action for an uncomfortable period of time.

Branches and Releases

- The project SHALL have one branch ("master") that always holds the latest in-progress version and SHOULD always build.
- The project SHALL NOT use topic branches for any reason. Personal forks MAY use topic branches.
- To make a stable release a Maintainer shall tag the repository. Stable releases SHALL always be released from the repository master.

- *While the GIT (or a logical successor) revision control system is mandated, the usage model requires a bare minimum of knowledge of the tool and follows the Keep It Simple Stupid (KISS) philosophy*
  – *This is commonly referred to the as the “fork+pull” model*

Evolution of Public Contracts
For example, management of format specifications.

- All Public Contracts (APIs or protocols) SHALL be documented.
- All Public Contracts SHOULD have space for extensibility and experimentation.
- A patch that modifies a stable Public Contract SHOULD not break existing applications unless there is overriding consensus on the value of doing this.
- A patch that introduces new features SHOULD do so using new names (a new contract).
- New contracts SHOULD be marked as "draft" until they are stable and used by real users.
- Old contracts SHOULD be deprecated in a systematic fashion by marking them as "deprecated" and replacing them with new contracts as needed.
- When sufficient time has passed, old deprecated contracts SHOULD be removed.
- Old names SHALL NOT be reused by new contracts.

Project Administration

- The project founders SHALL act as Administrators to manage the set of project Maintainers.
- The Administrators SHALL ensure their own succession over time by promoting the most effective Maintainers.
- A new Contributor who makes correct patches, who clearly understands the project goals, and the process SHOULD be invited to become a Maintainer.
- Administrators SHOULD remove Maintainers who are inactive for an extended period of time, or who repeatedly fail to apply this process accurately.
- Administrators SHOULD block or ban "bad actors" who cause stress and pain to others in the project. This should be done after public discussion, with a chance for all parties to speak. A bad actor is someone who repeatedly ignores the rules and culture of the project, who is needlessly argumentative or hostile, or who is offensive, and who is unable to self-correct their behavior when asked to do so by others.