An Empirical Study of Open-Source Development Practices for Safety Certified Software

Benjamin Steenhoek

***Abstract*—Open-source software can improve the quality of a software project and allow greater openness and user engagement. However, safety-critical software has special requirements to maintain certification which makes it difficult to develop as open-source. Certain projects use special development practices such as automated testing and code style checkers inorder to develop their software as open-source while maintaining safety standards. In order to illuminate these development practices, we expand on a recently compiled database of 42 open-source and safety-critical projects. Our analysis (1) describes the level to which open-source projects seek to meet safety standards and (2) describes the projects' development practices used to integrate community contributions while maintaining safety standards**

I. Introduction

Safety-critical software is software used in a system in which failure may cause great harm to people or the environment. Safety-critical software often carries special requirements for documentation and development processes to ensure that the software is sufficient quality. Sometimes, formal standards are used to make sure the software meets the requirements. Standards often require submission of artifacts to the standard certification board. These include artifacts such as FMEA and safety manuals, as well as processes such as requirements elicitation and safety culture. As well, the software should comply with more stringent requirements such as high test coverage and coding style guidelines.

Open-source software is software that is released with a license that allows users to read and modify the source code. The software's source code is distributed publicly for the purpose of modification or reuse. Part of the motivation for open-source software is that it can lead to higher-quality code. A principle coined as Linus's Law states that "given enough eyeballs, all bugs are shallow". When many users examine the source code, bugs are found relatively quickly, even before they show up in practice.

Where the two classes overlap, some open-source projects try to meet a safety standard. The extra artifacts required by the safety standard can slow down the development process ad can be difficult for community contributors to fulfill. Some projects employ automation and development practices in order to meet their goals of being open-source and maintaining their safety standard. We examine the prevalence of safety certification in open-source development, and the development practices used to maintain it.

This report is based on a previous dataset compiled by Rafaila et al. [1]. We extend their dataset by (1) examining the safety standards targeted by these projects and (2) tallying the quality assurance methods used to vet open-source contributions [1].

II. Background

We define a safety standard as a set of requirements set to ensure the safety of a system. In order to be meaningful, the standard should be recognized by some government or industry and improve the confidence that a safety-critical system is indeed safe. In order to show that it meets a standard, a project must show evidence that its system is safe (according to the criteria defined by the standard), and if the project changes, the evidence must be updated.

A pull request (PR) is a request to integrate a set of code changes into a version control repository. PRs are the main way that projects hosted on GitHub and GitLab integrate contributions from the open-source community. The contributor makes changes in a fork (a private copy) of a repository, then submits a PR requesting to integrate the changes into the repository, in the form of a set of changes to the files and (usually) a message explaining the reasoning or methods of the PR.

Continuous integration (CI) is a set of processes for automatically testing and integrating code changes. It is used in open-source projects to reduce the manual effort of integrating code changes. Often, open-source projects use CI to automatically build the projects, check style rules, or run tests, especially when integrating a PR.

III. Research Questions

In order to study the development practices employed in open-source safety-critical software development, we developed 2 research questions.

a) RQ1: Is it common for open-source projects to meet a formal safety standard?: To answer this question, we categorized the projects by (1) commercial backing and (2) safety standard. We hypothesized that the backing category will be needed because projects with commercial or government backing will be more likely to target a safety standard.

b) RQ2. What quality assurance processes are used to integrate community contributions while continuing to meet their chosen safety standard?: To answer this question, we tallied the quality assurance practices detailed in the contribution guidelines of each project.

[1]The original dataset's table, extended with the data we collected, are shown here: https://docs.google.com/spreadsheets/d/1HlQkQLUdyf8l_mIy6QRjxTY3iDrK724wXQCHH8ucAbM/edit?usp=sharing.

IV. Results

A. RQ1: Is it common for open-source projects to meet a formal safety standard?

In order to examine this question, we gathered extra information about the projects in the dataset.

We categorized each project by their commercial backing based on 4 categories:

- **Volunteer** projects are governed by a single maintainer or board of maintainers primarily for community benefit rather than commercial profit.
- **Government** projects are maintained by a government organization.
- **Commercial** projects are maintained as open-source by a company.
- **Research** projects are maintained for the sake of prototyping and research, but are not yet employed in a real application.

To search each project's backing, we navigated to their "About" page. If the page noted that it was **Commercial**, **Government**, or **Research**, we noted it; otherwise, we assumed the project was **Volunteer**.

To search each project's certification status, we used a two-pronged approach. (1) we searched on the project's home page, about page, and README page for keywords "safety" and "certification". (2) we searched *<project name> safety* and *<project name> certification* in Google and browsed the first page of results for material directly related to the project and a formal safety standard.

Government and **Commercial** projects usually have a team of developers who are employed by the company. Because of the organizational similarities, these categories are often grouped together in our analysis. The development team reserves decision power for the architecture of the project and makes most of the core contributions, with additional support and usage by the community. The project is usually maintained as part of a product or initiative. This differs from the "full bazaar" open-source model used by **Volunteer** projects, where the project is maintained for the sake of novelty or community benefit rather than commercial profit.

Several **Volunteer** projects were originally community-driven but receive most of support or are now maintained by a commercial entity, presenting difficulty for our classification. These projects are still listed as **Volunteer** unless they were directly sold as a product.

These categorizations are driven by the material published by the developers of the projects and were collected manually.

We also categorized each project based on whether it attempts to meet a safety standard. The shared property of safety standards is that they have stringent requirements of evidence which asserts the safety of the product. This evidence is sometimes based on the quality of the development team, which poses challenges for open-source projects which may gain contributions from the global development community. The evidence must also be updated when the software is updated and usually requires special expertise to write it, which adds effort to already short-staffed **Volunteer** projects.

The bottom row of Table I shows the total counts of projects by organization type. We see that most of the projects are **Volunteer**, followed by **Commercial** and **Research**.

Table I shows the proportion of projects which target a safety standard for each motivation category. We see that only 8 projects (19% of total dataset) total seek to meet a formal safety standard, and only 4 projects (9.5% of total) fully meet a standard.

The 1 commercial project meeting a standard is the Apollo self-driving system. Baidu Apollo is an open-source autonomous vehicle architecture which has been deployed for testing in robotaxi, minibus, and valet parking applications. It has obtained IATF 16949, ISO 26262, and A-SPICE certifications, as well as T4 road test licenses in China for driverless road testing.

The 2 volunteer projects meeting safety standards are FreeRTOS and SoftRF. FreeRTOS is a Real-Time Operating System (RTOS). A RTOS is guaranteed to be compliant with real-time requirements, such as predictability and determinism. FreeRTOS has a variant named SafeRTOS which is pre-certified for functional safety. The two projects share a high-level kernel model, but SafeRTOS diverges significantly from FreeRTOS in that it is a from-scratch rewrite following IEC 61508-3 SIL 3 process. SafeRTOS is pre-certified to IEC 61508 SIL 3 and ISO 26262 ASILD, and is designed to easily transition from FreeRTOS to SafeRTOS for safety-certified applications. SoftRF is a DIY general aviation proximity awareness system with open-source hardware. SoftRF is approved for compliance by the FCC (in the US) and CE (parallel in European Union), meaning it is approved for land use and with genuine factory firmware. This is less of a safety certification but does have ramifications for the compliance and safety of the product.

The 3 projects with certifications do not openly display the documentation they submitted to gain the compliance. It will be an interesting future work to discover the practices of other software projects which make their documentation more open.

8 projects are attempting to gain safety certification but have not yet gained it. We examine 3 projects in closer depth. Nightscout is an open-source Continuous Glucose Monitor (CGM), which monitors the glucose level in blood. This is an important device for diabetics to maintain the healthy blood sugar level. The developers are seeking formal clearance from the FDA and have met with the FDA several times, but do not have formal clearance, partly due to their open-source model. "Nightscout represents a novel application of FDA rulemaking in that it is "open-sourced" having a distributed model of ownership and/or responsibility for compliance with various medical device, treatment, and research overseers." [2] ChibiOS is an free and open-source RTOS. It has sought to be compatible with safety certification and provides example

[2]https://t1pal.com/legal/faq_8_18_2020_13_38

Certified	Commercial/Government	Volunteer	Research	Total
Yes	1	2	0	3
WIP	3	5	0	8
No	9	15	7	31
Total	13	22	7	42

TABLE I
NUMBER OF PROJECTS MEETING STANDARD BY ORGANIZATION TYPE

codes to this effect [3], but does not have safety certification of its own [4]. Comma.AI OpenPilot is an open-source driver assistance system. It is designed to be deployed in a module to drive individual cars and is compatible with a wide range of vehicles. It observes ISO 26262 safety guidelines. However, it is not formally certified; the Comma.AI team has claimed that they are in the process of gaining formal certification. Notably, the Baidu company focuses on deploying to a fleet of vehicles managed by themselves, while Comma.AI focuses on users deploying and maintaining the self-driving module in their own car.

No projects from the **Research** category target a formal safety standard. This is probably because these projects are developed as tools used to investigate novel research, but are not put directly into production. Since they will not be commercialized or put people at risk, they do not need a safety certification. Most of these projects come with an informal disclaimer that warns that the project should be used "at your own risk".

The main result of RQ1 is that *it is not common for open-source projects to meet a safety standard* as only 3 projects are formally certified. However, several projects maintain a certain level of safety even if not formally certified. RQ3 will examine the reasons that many projects do not meet standards. Surprisingly, the **Volunteer** certified projects actually outnumbered the **Commercial** projects, though both were slim.

B. RQ2: What quality assurance processes are used to integrate community contributions while continuing to meet their chosen safety standard?

Many projects which welcome open-source contribution include a contribution guidelines (such as a file `CONTRIBUTING.md`) readily available in their software repository. In order to investigate the usage of quality assurance processes, we examined the contribution guidelines of each project's repository. This is the entry point for new contributors, so intuitively, the . We noted whether each project employed some common quality assurance processes:

1) Coding style guide: some projects conform to a coding style guide/standard in order to get rid of common code smells. To find these, we searched for the keywords "style" and "convention" in the repository index and contributor's guide.
2) Unit/integration tests: many projects use unit tests to test individual units of software, or integration tests to test the integration between multiple modules (often in an operational environment). Some projects also require contributors to provide tests for all new code. To find these, we searched the keyword "test" in contributor's guide and manually extracted the information.
3) Automatic PR test: many projects automatically test all significant changes to the repository. To find these, we searched for configuration files which configured continuous integration tools such as GitHub actions and Travis CI, and badges on the README.md file.

33/42 projects (79%) have contributor guides on their repository. These guides are the entry point for those interested in contributing to the project. If a project is open to contributions, it is essential that the project provide a contribution guide to reduce the resistance to submitting contributions. Many contribution guides also guide first-time contributors by suggesting that they start by adding to the project's documentation or unit tests (as opposed to implementing features).

25/42 projects (60%) require contributions to conform to style guides. The style guides control various stylistic elements such as whitespace and indentation, as well as semantic elements such as usage of macros and memory allocation. 8 out of the 25 (32%) use a standard style guide [5], while the remaining 17 projects (68%) use a custom style guide. Several projects cite the fact that there are many competing coding style standards as motivation for forming a custom standard. 20 out of the 22 (91%) cite tools to reformat and check the code automatically. An automatic tool significantly reduces the effort of code review [2]. Most projects use a standard formatting tool such as `clang-format`, rather than implementing the tools from scratch.

3 projects (ChibiOS, FreeRTOS, and Zephyr) conform to most of the MISRA-C guidelines [3], which are more stringent than stylistic issues such as whitespace. These guidelines specify best practices for safety-critical code and can be thought of as a subset of the C language which restricts developers from using error-prone constructs, libraries, and features.

19/42 projects (45%) recommend or require contributors to add tests for new functionality which is added. Most projects enforce this recommendation/requirement during code review, but several projects also measure this automatically by maintaining a high level of line or branch coverage. For example, the Autoware project's maintainers "do not accept changes that reduce the coverage value [below 100%]" [4]. This automatic enforcement reduces the strain on the developers during code review.

30/42 projects (71%) automatically build and test contributions to their project. Most often, this is done on the level of a pull request (PR). Once the PR is submitted, the automatic tests are run by a continuous integration (CI) worker. If the tests succeed, the PR is annotated with a flag, and if they fail, the worker usually reports the degree and reasons for failure. This is a common method to make sure that the unit test suite still passes after every contribution.

[3]https://forum.chibios.org/viewtopic.php?f=3&t=5269

[4]https://forum.chibios.org/viewtopic.php?t=1736

[5]e.g. ROS, ISO C99, Google

17/42 projects (40%) have extra integration tests or coverage. For example, PX4 requires contributors to test their changes in a real flight and prefers that contributors upload their log file so the maintainers can review it. Testing on real hardware provides extra verification that the entire system is safe (by running in the real environment) and allows integration issues to be spotted which may not arise in software alone (such as those caused by non-determinism). For another example, ArduPilot publishes a Software In The Loop (SITL) tesing tool, which allows the build to pilot a plane, copter, or rover in a simulator. It is difficult to test the software on real hardware, especially automatically, and this option allows testing the integration with the hardware model without the difficulty.

The main result for RQ2 is that *testing and style guides are commonly used to maintain safety standards in open-source projects*. Even though the projects do not formally meet a safety certification, they seek to uphold a certain level of quality; to this end, 60% of projects required contributions to meet a style guide, and 71% of projects required contributions to pass an automated test suite.

V. Threats to Validity

The original dataset did not come from a comprehensive search, so it may be missing safety-critical open-source projects or may not be representative of the typical safety-critical open-source project.

Additionally, our extensions to the dataset were not comprehensive, but employed keyword searches and manual review. This may have missed some results where projects were certified to a safety standard, or included some quality assurance practice, but we did not find it. Several of these projects may have been forked (copied and used as a basis for further development) by projects which went on to gain safety certification; we did not investigate these "secondary-type" projects.

VI. Future Work

Interesting future work includes exploring problems and solutions used by safety-critical open-source projects. Our current study extends the original dataset with indication of the potential difficulties and with information about the development practices used, but does not deeply investigate the perspectives of the projects themselves and other projects which have spoken out about the difficulties of maintaining safety certification materials as open-source. Some standards preclude open-source development because the development process must be under watch (explained here by a member of the PX4 forum [5]). As well, the effort involved with maintaining safety certification often outweighs the benefits of the certification. Several new safety standards have been proposed [6**?** , 7] which are compatible with open-source development and the usage of open-source software.

Another interesting future work would examine the statistics of the contributions to safety-critical open-source projects. Intuitively, a safety-critical project may not get as many PRs because it has more stringent code quality requirements. As well, some safety-critical applications remain in use without updates for a long time, because the updates would require too much effort or risk to integrate into an existing safety-critical system.

The data added to the dataset can be used to examine the relationship between code quality and quality assurance practices. The projects which integrate quality assurance practice such as testing and code style guides should have a lower rate of defects, and it would be interesting to investigate this further.

References

[1] R. Galanopoulou and D. Spinellis, "A Dataset of Open-Source Safety-Critical Software," p. 7.

[2] G. J. Holzmann, "Right Code," *IEEE Software*, vol. 38, no. 1, pp. 13–15, Jan. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9305891/

[3] "MISRA C," Apr. 2022, page Version ID: 1084818860. [Online]. Available: https://en.wikipedia.org/w/index.php?title=MISRA_C&oldid=1084818860

[4] "Guidelines and Best-Practices," Nov. 2020. [Online]. Available: http://web.archive.org/web/20201125220251/https://autowarefoundation.gitlab.io/autoware.auto/AutowareAuto/contributor-guidelines.html

[5] "V1.9 Software "Release notes preview" - #4 by auturgy - Discussion Forum, for PX4, Pixhawk, QGroundControl, MAVSDK, MAVLink," Apr. 2022. [Online]. Available: http://web.archive.org/web/20220427041239/https://discuss.px4.io/t/v1-9-software-release-notes-preview/9379/4

[6] "Pixhawk | The hardware standard for open-source autopilots." [Online]. Available: https://pixhawk.org/

[7] "open-DO | Toward a cooperative and open framework for the development of certifiable software." [Online]. Available: https://www.open-do.org/