A Dataset of Open-Source Safety-Critical Software

Rafaila Galanopoulou[1], Diomidis Spinellis[1]

[1]Department of Management Science and Technology Athens University of Economics and Business
Patission 76, Athens, 10434, Greece

Abstract

We describe the method used to create a dataset of open-source safety-critical software, such as that used for autonomous cars, healthcare, and autonomous aviation, through a systematic and rigorous selection process. The dataset can be used for empirical studies regarding the quality assessment of safety-critical software, its dependencies, and its development process, as well as comparative studies considering software from other domains.

Keywords

open-source, safety-critical, dataset

1. Introduction

Safety-critical systems (SCS) are those whose failure could result in loss of life, significant property damage, or damage to the environment [1]. Over the past decades ever more software is developed and released as open-source software (OSS) — with licenses that allow its free use, study, change, and distribution [2]. The increasing adoption of open-source software in safety-critical systems [3], such as those used in the medical, aerospace, and automotive industries, poses an interesting challenge. On the one hand, it shortens time to delivery and lowers development costs [4]. On the other hand, it introduces questions regarding the system's quality. For a piece of software to be part of a safety-critical application it requires quality assurance, because quality is a crucial factor of an SCS's software [5]. This assurance demands that evidence of OSS quality is supplied, and an analysis is needed to assess if the certification cost is worthwhile.

Despite the increased interest by both industry and the research community for the adoption of OSS in SCS, there seems to be little evidence regarding the quality of such systems. An earlier study [3] explored the field through a systematic literature survey. It identified 22 relevant studies and isolated 31 software systems. Given that many OSS projects lack academic publications associated with them, we approach the question of finding OSS associated with SCS by looking directly for relevant software systems.

First workshop on trustworthy software and open source, March 23-25, 2021, Virtual Conference

t8160018@aueb.gr (R. Galanopoulou); dds@aueb.gr (D. Spinellis)
https://www.dmst.aueb.gr/dds/ (D. Spinellis)
0000-0002-5318-9017 (R. Galanopoulou); 0000-0003-4231-1897 (D. Spinellis)

CEUR Workshop Proceedings (CEUR-WS.org)

2. Software Selection Method

Our search is based on the methodology guidelines proposed by Kitchenham and Charters [6]. The software selection method consists of three steps:

- candidate system identification based on Google queries, GitHub tags queries, and results of other studies,
- project repository identification and exclusion based on selection criteria, and
- filtering based on software purpose and characteristics.

To retrieve relevant SCS open-source projects, we run by hand the following search queries. The provided search queries are notional; the disjunctions were performed by combining by hand the results of individual queries.

OSS: *("open source system" OR "github project" OR "git repository" OR "open source project" OR "open source hardware")* We used this query in combination with the ones detailed in the following paragraphs, which target specific SCS domains, applications, and standards. By using "github" and "git repository" as search keywords, the *OSS* query finds projects that are likely to have a Git repository, which can then be queried through an API to further narrow down the selected projects. We added the "open source hardware" term based on the assumption that an open-source hardware system might use OSS as well.

SCS Domains: OSS AND *("Infrastructure" OR "Medicine" OR "Nuclear engineering" OR "Recreation" OR "Transport" OR "Railway" OR "Automotive" OR "Aviation" OR "Space flight")* This query associated with broad SCS domains identified 15 projects in total (through a Google search and a GitHub tag search). We complemented these results for the domain of health applications by adding five projects derived from the data associated with a recent related study [7].

SCS Automotive Applications: OSS AND *("Airbag systems" OR "Braking systems" OR "Seat belts" OR "Power Steering systems" OR "Advanced driver-assistance systems" OR "Electronic throttle control" OR "Battery management system for hybrids and electric vehicles" OR "Electric park brake" OR "Shift by wire systems" OR "Drive by wire systems" OR "Park by wire")* We derived the terms of this query from Pimentel's book chapter [8].

SCS Medical Applications: OSS AND *("Heart-lung machines " OR "Ventilators" OR "Insulin pumps" OR "Life critical monitors" OR "Infusion pumps" OR "Robotic surgery")* The terms of this query were derived from Hamilton's [9] and Alemzadeh *et al.* [10] studies.

SCS Infrastructure Applications: OSS AND *("Circuit breaker" OR "fire systems" OR "electrical and hydraulic systems" OR "buildings infrastructure" OR "burner control systems")* The query keywords were derived from studies conducted for civil infrastructure and systems used for emergency in buildings (e.g. fire) [11, 12].

The three domain-specific queries identified 96 projects in total.

SCS Standards: OSS AND *("DO-178C" OR "MISRA" OR "MISRA-C" OR "IEC 61580" OR "IEC 880" OR "ISO 9000")* The standard names were derived from conference presentations associated with the examined SCS domains derived from the *Conference on Digital Avionics Systems* [13]. This query identified 4 projects in total.

For practical reasons in each step we excluded projects lacking a description or having a non-English description (e.g. Japanese, Portuguese).

The second step of our selection method was based on a project's repository characteristics. Here we excluded projects without a Git repository (Github or GitLab), inactive projects (lacking a commit in the last eight months) and unpopular projects (having fewer than 70 GitHub stars). Through these criteria we rejected 57 projects leaving us with 63.

In our last step we applied an exclusion criterion based on whether the project served a safety-critical role. For example, in the medical category, we excluded projects associated with a hospital's enterprise resource planning (ERP) services or the keeping of patient records. We did this by running the following Google search query for each project and studying the results to identify the project's purpose. *(project-name AND ("applications" OR "in open source hardware" OR "safety critical systems").* Through this criterion we rejected 21 projects leaving us with 42.

3. Results

The selected projects are listed in Table 1. For each project we list its name, application field, popularity (in awarded stars), size (in thousands of lines of code), implementation language(s), and repository location. The whole dataset and replication package are available online.[1]

Most projects are in the automotive (AM) sector (12 out of 42), followed by ten projects in avionics (AV), six in medicine (MED), three in spaceflight (SP), two in nuclear engineering (NE), and one in recreation (RE).

Additionally, we found that 34 unique programming languages are used by the selected 42 projects. The most popular are C++ (used in 29 projects), C (25), Python (24), and the Unix shell (15).

4. Dataset Limitations

The presented dataset suffers from some limitations, which could be lifted in the future. First, the threshold and inclusion criterion of 70 stars we used is arbitrary. Ideally it should be replaced by objective criteria, based e.g. on attributes that characterize engineered projects [14]. Second, the study has excluded projects hosted on platforms other than Github and Gitlab. This was done, because we are aiming to evaluate the dataset against community metrics that can be gathered from these platforms, such as the number of forked repositories, open and closed issues, and contributed pull requests. Nevertheless, if one does not care about these metrics, then more repository hosting platforms should be considered. Third, the dataset does not incorporate SCS testing software and components, which can be vital for satisfying safety requirements [15].

[1] https://dx.doi.org/10.5281%2Fzenodo.4568977

Table 1: Selected Projects

Project Name	Field	Stars	KLoC	Programming Languages	Repository
3D Slicer	MED	248	805	C++, Python, CMake, C, Shell	github.com/Slicer/Slicer
AirSim	AM	11 329	290	C++, C#, Python, C, CMake	github.com/microsoft/AirSim
Apollo	AM	17 923	1 800	C++, Python, Starlark, JavaScript	github.com/ApolloAuto/apollo
ArduPilot Project	AM	5 656	1 200	C++, C, Python, Lua, MATLAB	github.com/ArduPilot/ardupilot
ARMI	NE	104	148	Python	github.com/terrapower/armi
ASTPP	IN	112	91	PHP, TSQL, SQLPL, JavaScript	github.com/iNextrix/ASTPP
AutoAS	AM	566	550	C, C++, Python, Assembly	github.com/autoas/as
AutoRally	AM	593	40	C++, Cuda, Python, CMake	github.com/AutoRally/autorally
Autoware Auto	AM	153	43	C++, CMakefile, Python, Dockerfile	gitlab.com/autowarefoundation/autoware.auto/AutowareAuto
Carla Simulator	AM	5 645	552	C++, Python, Batchfile, Shell	github.com/carla-simulator/carla
CARMAPlatform	AM	191	299	C++, CMake, Shell, Java, Python	github.com/usdot-fhwa-stol/CARMAPlatform
ChibiOS	IN	448	4 100	C, Shell	github.com/ChibiOS/ChibiOS
contiki	AM	3 309	877	C, Java	github.com/contiki-os/contiki
diyBMS	IN	340	38	C++, C	github.com/stuartpittaway/diyBMS
FreeCAD	IN	8 829	6 400	C++, Ruby, Python, Shell	github.com/FreeCAD/FreeCAD
FreeRTOS	IN	1 461	103	C	github.com/FreeRTOS/FreeRTOS
GAAS	AV	1 365	3 100	C++, JavaScript, Python, CMake, C	github.com/generalized-intelligence/GAAS
Garmin International	AV	235	23	C++, Go, Python, C, Java	github.com/garmin/LIDARLite_Arduino_Library
Inspire-OpenLung	MED	195	301	C++	github.com/Inspire-Poli-USP/Inspire-OpenLung
littlenavmap	AV	551	470	C++, Shell. QMake	github.com/albar965/littlenavmap
NASA Flight System	SP	219	1	CMake	github.com/nasa/cFS
NASA cFE	SP	212	125	C, C++, CMake	github.com/nasa/cFE
NASA OSAL	SP	288	90	C, CMake	github.com/nasa/osal
nightscout	MED	1 428	118	JavaScript	github.com/nightscout/cgm-remote-monitor
Oem	IN	207	244	C++, C, Python, Shell	github.com/openenergymonitor/emonpi
OpenLung Ventilator	MED	227	4	C++, OpenSCAD	gitlab.com/open-source-ventilator/ventilator/OpenLung
Openpilot	AM	22 701	457	C++, C, Python, Roff	github.com/commaai/openpilot
openScope	AV	353	475	JavaScript, Python	github.com/openscope/openscope
Paparazzi UAS	AV	1 171	1 100	C, Rust, C++, Python, Java	github.com/paparazzi/paparazzi
PX4 Drone Autopilot	AV	4 229	3 600	C++, C, Python, Shell	github.com/PX4/PX4-Autopilot
PyNE	NE	168	798	C++, Python, Fortran, MATLAB	github.com/pyne/pyne
RTLSDR-Airband	AV	376	36	C++, C, Makefile, Shell, Assembly	github.com/microtony/RTLSDR-Airband
SortRF	AV	316	97	C, C++, Shell, Python, Objective-C	github.com/lyusupov/SoftRF
Spinnaker	IN	7 667	8	Java, Groovy, Go, Kotlin, Python	github.com/spinnaker/spinnaker
startux	AV	767	54	C, Go, JavaScript, Shell	github.com/cyoung/stratux
Stethoscope	MED	613	394	Ruby, OpenSCAD, MATLAB	github.com/GliaX/Stethoscope
Subsurface	RE	148	893	C++, C, XSLT, QML, Shell, CMake	github.com/torvalds/subsurface-for-dirk
trampoline	AM	315	3 800	C, C++	github.com/TrampolineRTOS/trampoline
UAVCAN	AV	211	23	C, C++, JavaScript, Assembly	github.com/UAVCAN/libuavcan
Self-driving-car	AM	5 512	307	C++, Python, CMake, Shell	github.com/udacity/self-driving-car
Ventilator	MED	1 660	171	C++, C, OpenSCAD, Python	github.com/jcl5m1/ventilator
Zephuropen	IN	4 274	2 000	C, Python, Shell, Perl, Dockerfile	github.com/zephyrproject-rtos/zephyr

5. Discussion and Future Work

The development of SCS as OSS did not prove to be a widespread industrial practice, especially for avionics. We noticed that some of the field's leaders (BAE System Avionics, Furuno, Airbus Helicopters, GE Aviation, Telefunken) maintain GitHub organizations, but evidently prefer to keep their repositories private.

The low number of SCS OSS projects we found may be associated with onerous requirements and diminished incentives. Many SCS application domains, in common with accessibility OSS [16], may require either specialized peripherals (such as LIDAR or ECG sensors) or powerful hardware. In addition, SCS software may be based on specialized, inflexible, and non-free development tools [17]. These requirements, can hinder OSS developer participation. Furthermore, due to its specialized nature and regulatory requirements, SCS deployment may require strong organizational backing, thus further limiting and discouraging OSS volunteer participation.

Notably, none of the systems identified in the earlier study [3] passed our inclusion criteria. Most were excluded due to lack of activity or popularity in terms of GitHub stars. This is a concern regarding the long-term viability and maintenance of OSS SCSs.

The presented dataset can be used to evaluate the quality of OSS SCSs. This can be done e.g. based on the defect prediction model proposed by Foyzur and Premkumar [18], which outlines relevant process and code metrics. It is important to study both facets, because the most successful OSS projects are those featuring not only a high-quality code base, but also a thriving user community [4]. Results can then be compared with other OSS endeavors that have similar engineered project characteristics [14]. Furthermore, results can be qualitatively evaluated based on practices advocated by Wilson *et al.* [19, 20].

Acknowledgments

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 825328.

References

[1] J. C. Knight, Safety critical systems: challenges and directions, in: Proceedings of the 24th International Conference on Software Engineering. ICSE 2002, 2002, pp. 547–550.

[2] S. Androutsellis-Theotokis, D. Spinellis, M. Kechagia, G. Gousios, Open source software: A survey from 10,000 feet, Foundations and Trends in Technology, Information and Operations Management 4 (2011) 187–347. doi:10.1561/0200000026.

[3] S. M. Sulaman, A. Oručević-Alagić, M. Borg, K. Wnuk, M. Höst, J. L. de la Vara, Development of safety-critical software systems using open source software – a systematic map, in: 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, 2014, pp. 17–24. doi:10.1109/SEAA.2014.25.

[4] D. Margan, S. Čandrlić, The success of open source software: A review, in: 2015 38th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), 2015, pp. 1463–1468. doi:10.1109/MIPRO.2015.7160503.

[5] E. P. Jharko, The methodology of software quality assurance for safety-critical systems, in: 2015 International Siberian Conference on Control and Communications (SIBCON), 2015, pp. 1–5. doi:10.1109/SIBCON.2015.7147057.

[6] B. Kitchenham, S. Charters, Guidelines for Performing Systematic Literature Reviews in Software Engineering, Technical Report EBSE-2007-01, School of Computer Science and Mathematics, Keele University, 2007.

[7] G. Minohara, C. Rocha, J. Costa, P. Meirelles, Benefits and challenges of open-source health informatics systems: A systematic search of projects and literature review., JMIR Preprints (2020). doi:10.2196/preprints.18489.

[8] J. Pimentel, SECTION 1: INTRODUCTION TO SAFETY-CRITICAL AUTOMOTIVE SYSTEMS, 2006, pp. 1–1. doi:10.4271/PT-103.

[9] D. K. Hamilton, Chapter 7 - design for critical care, in: A. Sethumadhavan, F. Sasangohar (Eds.), Design for Health, Academic Press, 2020, pp. 129–145. URL: https://www.sciencedirect.com/science/article/pii/B9780128164273000075. doi:https://doi.org/10.1016/B978-0-12-816427-3.00007-5.

[10] H. Alemzadeh, R. K. Iyer, Z. Kalbarczyk, J. Raman, Analysis of safety-critical computer failures in medical devices, IEEE Security & Privacy 11 (2013) 14–26.

[11] R. G. Little, Toward more robust infrastructure: observations on improving the resilience and reliability of critical systems, in: 36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the, IEEE, 2003, pp. 9–pp.

[12] J. Bosch, P. Molin, Software architecture design: evaluation and transformation, in: Proceedings ECBS'99. IEEE Conference and Workshop on Engineering of Computer-Based Systems, IEEE, 1999, pp. 4–10.

[13] N. Metayer, A. Paz, G. El Boussaidi, Modelling do-178c assurance needs: A design assurance level-sensitive dsl, in: 2019 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW), 2019, pp. 338–345. doi:10.1109/ISSREW.2019.00094.

[14] N. Munaiah, S. Kroh, C. Cabrey, M. Nagappan, Curating GitHub for engineered software projects, Empirical Software Engineering 22 (2017) 3219–3253. doi:10.1007/s10664-017-9512-6.

[15] K. Amarendra, A. V. Rao, Safety critical systems analysis, Global journal of computer science and technology (2011).

[16] M. Heron, V. Hanson, I. Ricketts, Open source and accessibility: Advantages and limitations, Journal of Interaction Science 1 (2013). doi:10.1186/2194-0827-1-2.

[17] S. Suomalainen, Kunnanmäki, Open-source components in safety critical systems, 2004.

[18] F. Rahman, P. Devanbu, How, and why, process metrics are better, in: Proceedings of the 2013 International Conference on Software Engineering, ICSE '13, IEEE Press, 2013, p. 432–441.

[19] G. Wilson, D. A. Aruliah, C. T. Brown, N. P. Chue Hong, M. Davis, R. T. Guy, S. H. D. Haddock, K. D. Huff, I. M. Mitchell, M. D. Plumbley, B. Waugh, E. P. White, P. Wilson, Best practices for scientific computing, PLOS Biology 12 (2014) 1–7. URL: https://doi.org/10.1371/journal.pbio.1001745. doi:10.1371/journal.pbio.1001745.

[20] G. Wilson, J. Bryan, K. Cranston, J. Kitzes, L. Nederbragt, T. K. Teal, Good enough practices in scientific computing, PLOS Computational Biology 13 (2017) 1–20. doi:10.1371/journal.pcbi.1005510.