ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM038

TOWARDS THE ZERO CODE WASTE TO INCREASE THE IMPACT OF SCIENCE

P. P. Goryl*, L. Zytniak, S2Innovation Sp. z o.o., Kraków, Poland
A. Gotz, ESRF, Grenoble, France
V. Hardion, MAX-IV, Lund, Sweden
S. Hauf, European XFEL GmbH, Schenefeld, Germany
K. S. White, ORNL, Oak Ridge, USA

Abstract

Accelerators and other big science facilities rely heavily on internally developed technologies, including control system software. Much of it can and is shared between labs, like the Tango Controls and EPICS. Then, some of it finds broad application outside science, like the famous World Wide Web. However, there are still a lot of duplicating efforts in the labs, and a lot of software has the potential to be applied in other areas. Increasing collaboration and involving private companies can help avoid redundant work. It can decrease the overall costs of laboratory development and operation. Having private industry involved in technology development also increases the chances of new applications. This can positively impact society, which means effective spending of public funds. The talk will be based on the results of a survey looking at how much scientific institutes and companies focus on collaboration and dissemination in the field of software technologies. It will also include remarks based on the authors' experiences in building an innovative ecosystem.

INTRODUCTION

An efficient economy is key for today's world challenges related to climate and limited resources. Zero Waste can be applied to all production cycles related to material resources and non-tangible assets like software.

The primary business of large scientific infrastructures, like particle accelerators or telescopes, is to conduct research in areas other than software technology. However, software is a core tool for them. No commercial software is often available due to specific functional and operational requirements and a limited market. This means the laboratories must develop or buy software development services to satisfy their needs. As the scientific labs compete on the scientific results, not the functionalities of the software tools, there is space for collaboration. So, these institutions already share an effort to provide software tools, minimising the costs of implementing functionalities. When they do this, sharing source code, they follow the Zero-Waste idea and build an efficient economy. The above concerns are the source of the Zero Code-Waste term.

* piotr.goryl@s2innovation.com

Zero Code-Waste

The Zero Code-Waste is not about keeping maintenance and running of the legacy or obsolete software.

The Zero Code-Waste is an idea of maximizing software reuse between projects. This shall decrease duplicated effort in providing the same functionalities within different software packages. In this context, it is directly related to **collaboration**. Ultimately, it should lower development and maintenance costs, increasing the so-called development capacity of the community, which means developing more functionalities with less human effort and other resources spent.

The Zero Code-Waste can also encompass methodologies and techniques to minimize source code (re-)writing within a project. However, it is not covered by this paper.

Industry Involvement

Whereas current business models in the commercial software industry are often connected to closed-source IP rights, the scientific community, funded mainly with public money, relies heavily on open-source tools and collaborative models. It does not mean the software industry is not involved in the scientific project. Besides using standard commercial software, like operating systems or office applications, scientific institutes and collaborations often outsource or subcontract software development for their scientific needs. While the institutes do not regard this as increasing the chances of the software being reused, there are some reasons to claim it may help navigate towards Zero Code-Waste.

Disclaimer

This paper covers only a few Zero Code-Waste and collaboration aspects. Some claims come from the author's experience but are not supported with references or discussed in-depth due to publication volume. The paper is more to get focus and start a discussion on the topic, which certainly needs more research.

THE SURVEY

The survey [1] aimed to get wide feedback from software group leaders and managers about collaboration and its impact on the workload. The survey was sent to many organizations (70) all over the world. The 19 of them have answered.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM038

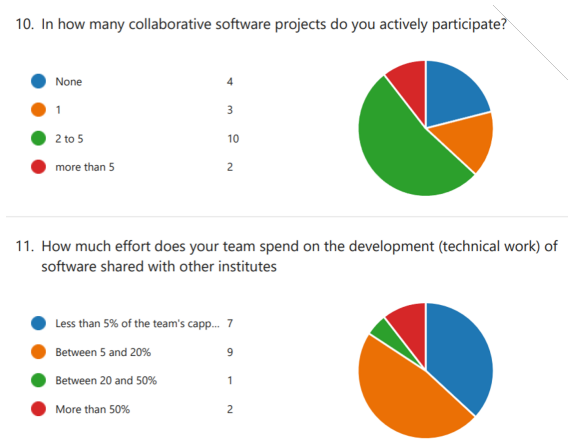

Figure 1: In how many collaborative software projects do you actively participate? How much effort does your team spend on the development (technical work) of software shared with other institutes.

For some questions, the Net Promoter Score (NPS) was applied. NPS is based on a single survey question asking respondents to rate they agree with it.

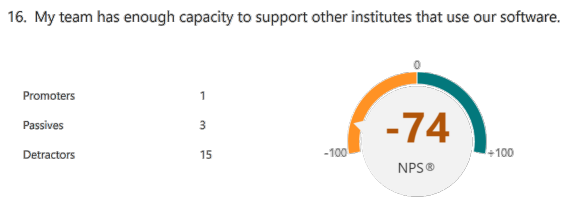

Figure 2: My team has enough capacity to support other institutes that use our software.

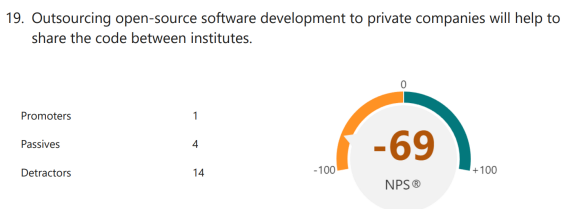

Figure 3: Outsourcing the development of shared open-source projects to private companies can speed up software development within the community.

The Results

The first conclusion of the survey is that most scientific organizations are involved in 2 to 5 collaborative software projects. On the other hand, software teams spent less than 20% time on the development of collaborative software, see Fig. 1. Most of the software group leaders confirmed that they don't have enough capacity to support other institutes that use community software, see Fig. 2. As an effect, software teams don't have enough support from other collaborating organizations as also shown by the survey results. This is a big challenge for new joiners because they mostly have problems getting support from other organizations to start using collaborative software.

An interesting outcome is that software group leaders don't believe that outsourcing open-source software development to private companies will help to share the source code between organizations, as shown in Fig. 3.

COLLABORATIONS ON SOFTWARE

Open-sourcing software is essential to making software available for others to contribute to but it is not always a guarantee to attract collaborators. In this section, we identify different models of collaboration that work. We can identify the following classes of software projects and their potential for creating collaborations:

1. **software components** - are small software projects which have a dedicated purpose e.g. a plotting widget or a device driver etc. The dedicated nature of software components makes them relatively easy to share and reuse especially if they have a small number of dependencies. In contrast to their potential to attract a large number of users, they often do not attract many contributors. This is due to them being small and often easy to use which means no strong need for additional resources. Contributions can be attracted if the software component is part of a larger ecosystem which has a large number of users e.g. a popular framework.
2. **software libraries** - are medium to large-sized software projects which implement a common set of functions e.g. a plotting library or a file I/O library. Libraries have a better chance of attracting contributions, but these still stay modest in general. This can be related to their focused scope which leaves less room for innovation.
3. **software frameworks** - are large software projects which implement a set of concepts which allow a wide range of applications to be implemented. Software frameworks have the most potential to attract contributions and collaborators.
4. **software applications** - these are small to large software projects which are standalone and can be deployed without additional software required. Software applications can attract contributions if they have been designed to accept them, but this is not often the case. Software developers often attempt to redevelop new applications, which basically do the same. This results in few developers per application, which leads to duplication.

The role of software developers and managers should be to avoid duplication unless it is really necessary for technical or strategic reasons.

Examples

To put the paper into context it is worth mentioning a few examples of successful collaborations around the software.

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM038

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

TANGO The Tango Controls Collaboration [2] is a large collaboration of sites using the Tango Controls framework for controlling all or part of their installations. The first version of Tango was developed in 1999 at the ESRF. Very soon after the first release of Tango SOLEIL joined ESRF to participate and to adapt it to control the SOLEIL synchrotron. The two teams worked closely together holding monthly meetings to design and develop Tango further. In 2002 a workshop on Corba Controls was held at the ESRF [3] which gathered experts in Corba and controls to present different implementations of CORBA and control frameworks based on Corba. Following this meeting two new site, ELETTRA, ALBA and later DESY, decided to adopt Tango and join the collaboration. From this point on the collaboration continued to grow at a steady pace with the software being adopted by many sites. For the first years the development of the common libraries was done by a few facilities, led by the ESRF. In 2015 ten facilities using Tango decided to contribute financially to a collaboration contract which would allow them to sub-contract some critical maintenance developments to ensure that the common core is well maintained and continues to be developed. The collaboration contract was for 5 years and has been so successful that it has been extended for another 5 years. The software companies financed by the collaboration contract have been essential to keeping Tango maintained and new features being developed. At the same time as the contract was extended some of the core members increased the number of core developers working on Tango. This boost in developers is visible in the activity around Tango over the last 2 years [2]. The Tango collaboration is in a very healthy state thanks to the original members still being active and new major sites like the LOFAR and SKA projects adopting Tango. Even if Tango could attract more users if more promotion was done with better documentation and regular updates to the website, the best advertisement for Tango remains having good software which is well packaged. This has been achieved thanks to the collaborative efforts of the members of the collaboration.

EPICS The Experimental Physics and Industrial Control System (EPICS) toolkit was created from a collaboration started between Los Alamos (LANL) and Argonne (ANL) National Laboratories in the US Department of Energy [4]. The collaboration began by reusing code from the LANL Ground Test Accelerator Control System which was developed as a toolkit for building a control system where the communication, execution engine and user interface tools were provided as shared software which users configured to meet their control system requirements. ANL pursued this collaboration with LANL in order to efficiently build the control system for the Advanced Photon Source project. Other laboratories learned of this unproven collaboration through conference presentations and several joined the collaboration before ANL had completed building their machine using EPICS. Over 30 years later, EPICS is now used in the vast majority of DOE accelerator construction and upgrade projects and many projects outside the US, saving untold dollars and hours of work. Contributions to the code come from a small fraction of the users and the collaboration is loosely managed by volunteers within the community. Private companies now exist with staff who are trained in EPICS allowing laboratories to contribute to development or augment staff for projects without needing to hire and train employees, lending greater resource flexibility and capacity to the ecosystem.

MAX-IV - SOLARIS Collabroation Both MAX IV and Solaris adopted a multibend achromat (MBA) lattice design for their storage rings. In addition to sharing a similar design, many technical implementations are common to both facilities i.e. power supplies, vacuum equipment, plc, software language and IT infrastructure, etc. Both facilities use the Tango Controls system as their primary control system software. Given that Tango is open-source, it facilitates collaboration, allowing facilities to share solutions, best practices, and even specific implementations or modules of the system. Due to their collaborative history, MAX IV and Solaris have worked together on software components, especially those that relate to common systems or hardware. The mutual benefit of the collaboration was very strong, especially since very little overhead was introduced for developing the abstraction layer. While the two facilities have similarities due to their collaborative efforts, it's essential to note that they have distinct characteristics and goals. For instance, they both have different beamlines where the software might differ due to the instrumentation, techniques, and specific use cases. This lead to the development of different software, but it allows to explore different paths and share the lessons learned. The essential is to keep the synergy with regular communication.

Karabo A recurring situation at light sources is that the accelerator control system is distinct from instrument controls, and possibly even uses a different control framework. For instance, at the European XFEL the super conducting linac is operated using the DESY Object Oriented Control System (DOOCS) [5], while photon beam lines and instruments use Karabo [6, 7]. In such cases, opportunities exist to reduce code waste through an efficient means of bridging data and interfaces between two control systems. Such bridging technology can frequently reduce the need to implement features or integrate hardware multiple times. At EuXFEL, optimal tuning of the accelerator, e.g., requires access to diagnostic data from photon beam imaging devices read out in Karabo. Conversely, scientists and facility users depend on machine performance parameters, available in DOOCS, to interpret experiment results. This latter case is so common that an interface to create ad-hoc DOOCS to Karabo bridges is provided to operators.

The routine and successful collaboration between DESY and EuXFEL on bridging DOOCS and Karabo is considered a template for a generic Tango to Karabo bridge currently being developed at EuXFEL on top of PyTango [8] and the Karabo Middlelayer API [9]. The project's initial aim is to

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM038

reuse all but the interfaces of existing Tango integrations for a high-speed goniometer, the RoadRunner [10], and possibly, optical laser systems at the EuXFEL's HED instrument [11]. Due to the generic approach that was chosen, future application scenarios are the integration of hardware sourced from other facilities and industrial partners familiar with Tango controls. The development of tools that bridge between different control frameworks is facilitated if involved software is open source, i.e. sources, not just interfaces, are accessible to developers, and if developers of the involved systems are open to collaborating and acknowledge specifics of each system. The observation that accessible code is an important documentation method, and thus can foster collaboration, is one of the considerations that led to the release of Karabo as free and open-source software after more than 10 years of development.

CHALLENGES

IP Rights and Licensing

When starting a collaboration, or making an existing, possibly mission-critical project available as open-source software, the licenses of project dependencies and intellectual property (IP) rights need to be considered. For instance, the Karabo control system is considered mission-critical for the operation of the European XFEL. Because Karabo was developed in-house, the facility is the single IP rights holder. When the project was released into the public domain, care was taken that contributor license agreements ensure that contributions' intellectual property remains with EuXFEL to an extend that allows the facility to e.g. relicense the software. Additionally, with MPL2.0 [12] a license that includes a so-called anti-patent-trolling clause was chosen. Releasing a large existing project with many third-party dependencies can generally lead to non-trivial license inter-dependencies. For Karabo, the core framework is currently limited to licenses which are neither in conflict with the various GPL flavors, nor the Apache 2.0 license, while Tango uses LGPL and GPL licenses. EPICS has a bespoke license, accommodating specific requirements by the U.S. Department of Energy. Determining an appropriate license for an existing large project that is to be made publicly available as part of a collaboration, can thus be a non-trivial challenge, which might require expert legal advice.

Duplications

However, the community see the benefits of re-using the software there are a couple of examples of duplication:

- Tango Controls GUI tools: Taurus, QTango, ATK,
- Web interfaces: Waltz, Taranta, Puma/Cumbia,
- Alarm tools: PyAlarm, AlarmHandler, Achtung,
- GraphQL for Tango CS: ESRF and MAX-IV solutions,
- Control systems: Tango Controls, EPICS, Karabo, Doocs

There are reasons why duplication happens. Often specific software is crucial for the laboratory, and it decides to have a solution which can be fully maintained internally i.e. matching skills of the internal team. Another reason is the schedule. Usually, new needs are connected with building a new facility or upgrading an existing one, which triggers new software development. It may be adding new features to existing tools or writing a new tool from scratch. The first approach can be time-consuming for projects with a large community and complicated in terms of management. Then, technological progress starts with the prototyping of new solutions. It is good if there are prototypes built on different technology stacks. This allows to find the best solution. However, the prototypes quickly became production solutions. If it happens, it is hard for a facility to resign from what it developed in favour of a tool provided by another team. Also, the information flow is an essential factor. If one institute does not know that another is addressing the same need, they duplicate the effort. Today, we have very effective means of information exchange, but the challenge still exists due to the volume of software used at institutes and its paramount role. The duplication can easily happen to "person-month" projects, like equipment drivers, as there are hundreds of kinds of equipment commonly used by laboratories. However, the same happens to larger tools, too.

ADDRESSING THE CHALLENGES

Enabling Collaborations

Communication Prior to the start of large software collaborations like Tango and EPICS, accelerator control systems were developed as completely custom efforts internal to a single project or laboratory despite sharing a lot of common requirements. This could be attributed to little sharing of information between laboratories about their controls systems. In 1985, a small group of accelerator controls professionals met in Los Alamos to discuss sharing control system strategies. From this meeting and a second meeting in Villars came the idea and creation of a biennial series of conferences, eventually called the International Conference on Accelerator and Large Experimental Physics Control Systems (ICALEPCS). Over 30 years later, this conference now attracts hundreds of participants from around the world giving accelerator controls engineers an opportunity to learn what other laboratories are doing and a forum for discussion and collaboration. In fact, the majority of the talks at the conference refeence so use of shared software. Another factor fueling the push towards collaborative control systems was the reality of cost and schedule. Prior to 1990, it was not uncommon for a machine to be ready for commissioning while the control system needed more time. Collaborative solutions have also served to reduce the cost of providing a system. Proposals for controls systems for new machines are substantially lower (sometime as low as 5% of total project cost) due to widespread reuse of communication layers, execution engines, common tools for operations and on-line models and data analysis tools. The downside of collaborative software is the lack of funding for support. New projects get money to built a machine but when collaborative software is used, there is often no clear path to how maintenance

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM038

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI

will be funded. Collaborative software requires people to work more closely with more people to succeed. When using open source software developed by other facilities try to join them in developing the software.

Good Practices

Avoid Duplicating Software! A common pattern amongst software developers is to first develop a new software withut looking at what already exists out there. The more developers a facility has the more common this is and the larger the software package or framework which can be developed. The large the software being developed the more difficult it is too share. A first step when deciding to develop a new software package is to look around to see what is out there and seriously try to join an existing development rather than starting a new development. When using software developed by others try to contribute to it instead of extending it on a private fork or working around it by developing new features outside. Vice versa this requires the original developers being open to contributions.

If It's Not Needed (Anymore), Remove It! As projects grow over time, certain functionality becomes outdated. Code paths that implement this functionality may first be deprecated, and then become non-reachable at all, e.g. by hard-coding a flag that was previously configurable, or adding a preprocessor directive that avoids compilation of sections of code. At this point the latest the deprecated code should be deleted, rather than leaving developers new to the project to figure out what *actually* is in use or relevant. Version control systems make sure the code is not wasted, as it can be recovered from the version should it ever be needed again.

Write Self-Documenting Code Code can be a good documentation of itself and help developers reuse it in different settings. Good self-descriptive code avoids constructs that obfuscate intention: extremely condensed short-hands can be ingenious, but also incomprehensible by others. An appropriate level of comments should be added where aspects of the design might not be straight-forward. One should keep in mind though, that comments, as any documentation, are more likely to become outdated than the code itself. Obviously, especially the public interface of the code should be sufficiently documented, importantly including input and output data types, assumptions on the data, as well as exceptions that might occur. Finally, unit tests, and integration tests, can be useful for documenting intended use cases, and best practices of using the code they test.

Avoiding Unnecessary Assumptions Code can be reused more easily if it doesn't make unnecessary assumptions on the usage scenario. A library for a motor driver, e.g., ideally does make assumptions on the control system it is to be used in and limits itself to data types native to the programming language it is implemented in or composite types thereof. Similarly, a scan engine is more reusable if it implements an abstraction layer for interactions with hardware or the control system. In the case of Bluesky, Ophyd for instance has this role [13].

Cultivate Code Review Processes A good code review culture is an efficient means to address the aforementioned points as part of normal workflows, especially if new contributors join from time to time. Reviewers should be encouraged to ask also seemingly trivial questions, point out and ask for clarification on contributions that are not straightforward for them to understand, and insist on an appropriate level of testing and documentation to be provided *before* a contribution is accepted.

Companies Involvement

Whereas the survey results show that institutes are sceptical about the positive role of involvement of the companies there are some examples, where those provide significant input into the community software i.e. Tango Controls Collaboration subcontracted development and maintenance of the framework and ecosystem to a couple of private companies. Moreover, these companies are treated as and indeed are part of the community. The both-sided collaborative approach is an important factor. If it is implemented the involvement of the private software industry allows to address resource and support-related challenges. The companies can also address information flow within the community, as they are in regular contact with their (potential) customers.

CONCLUSION

Open source software and collaboration thereon is also a form of acknowledgement for the developers contributing to a project [14]. This can be especially important in a scientific setting, where the accepted form of recognition is publication, and engineers and scientists with a focus on software development, and technical support, frequently publish less by virtue of their (choice of) role in the organization. Platforms like GitHub.com and Gitlab.com facilitate such recognition through individual contribution statistics, and frequent contributors earn reputation over time. Such statistics nowadays are e.g. known to potentially being looked at as part of a recruitment process [15] and thus can impact careers.

ACKNOWLEDGEMENTS

The authors want to thank all who answered the survey [1] as well as all who participated in the discussions. I want also to thank Reynald Bourtembourg, for the footnote he places with his posts on the Tango Controls forum, which is a trigger for this paper.

REFERENCES

[1] Results of the Towards Zero Code-Waste survey, doi:10.5281/zenodo.8411372

ISBN: 978-3-95450-238-7 ISSN: 2226-0358 doi:10.18429/JACoW-ICALEPCS2023-TUMBCM038

[2] T. Jurges *et al.*, "The Tango Controls Collaboration Status in 2023", presented at ICALEPCS'23, Cape Town, South Africa, paper TH1BCO03, this conference.

[3] Corba Controls workshop, https://www.esrf.fr/conferences/Corba_Controls/

[4] Leo R. Dalesio *et al.*, "The experimental physics and industrial control system architecture: past, present, and future", *Nucl. Instrum. Methods Phys. Res., Sect. A*, vol. 352, no. 1–2, 1994. doi:10.1016/0168-9002(94)91493-1

[5] DESY Object Oriented Control System (DOOCS), https://doocs.desy.de

[6] D. Goeries *et al.*, "The Karabo SCADA System at the European XFEL,2023", *Synchrotron Radiat. News*, to be published.

[7] S. Hauf *et al.*, "The Karabo distributed control system", *J. Synchrotron Radiat.*, vol. 26, no. 5, pp. 1448–1461, 2019. doi:10.1107/S1600577519006696

[8] PyTango, 2023, https://pytango.readthedocs.io

[9] How To Middlelayer, https://howtomiddlelayer.readthedocs.io/en/latest/

[10] P. Roedig *et al.*, "High-speed fixed-target serial virus crystallography", *Nat. Methods*, vol. 14, no. 8, pp. 805–810. 2017. doi:10.1038/nmeth.4335

[11] U. Zastrau *et al.*, "The high energy density scientific instrument at the European XFEL", *J. Synchrotron Radiat.*, vol. 28, no. 5, pp. 1393–1416, 2021. doi:10.1107/S1600577521007335

[12] Mozilla Public License Version 2.0, https://www.mozilla.org/en-US/MPL/2.0/

[13] Bluesky, https://blueskyproject.io

[14] Y. Ye and K. Kishida, "Toward an understanding of the motivation of open source software developers", in *Proc. 25th Int. Conf. Software Eng.*, Portland, OR, USA, 2003. doi:10.1109/ICSE.2003.1201220

[15] Standing Out in a Competitive Market: The Impact of an Impressive GitHub Profile, https://code.likeagirl.io/standing-out-in-a-competitive-market-the-impact-of-an-impressive-github-profile-771cf2e084d4

Content from this work may be used under the terms of the CC BY 4.0 licence (© 2023). Any distribution of this work must maintain attribution to the author(s), title of the work, publisher, and DOI