The Four Pillars of Research Software Engineering

Jeremy Cohen, Imperial College London

Daniel S. Katz, University of Illinois at Urbana-Champaign

Michelle Barker, Research Software Alliance

Neil Chue Hong, University of Edinburgh

Robert Haines and Caroline Jay, University of Manchester

// We present four elements we believe are key to providing a comprehensive and sustainable support for research software engineering: software development, community, training, and policy. We also show how the wider developer community can learn from, and engage with, these activities. //

Digital Object Identifier 10.1109/MS.2020.2973362
Date of current version: 22 December 2020

ALTHOUGH SOFTWARE HAS been a part of research for many decades, the people who write, maintain, and manage this research software are increasingly seen as critically important members of research teams, rather than just "the people who write code." The past few years have seen the emergence and rapid growth of the concept of research software engineering (RSE). Beginning in the U.K. research community, circa 2013 through discussions initiated by the Software Sustainability Institute (https://software.ac.uk), groups focusing on developing sustainable, maintainable, robust research software have been set up at many institutions.

A history of this process can be found in the "2017 Research Software Engineers: State of the Nation" report.[1] Since the report was written, the number of research software groups within the United Kingdom and internationally has increased. Behind this growth is the need for ever-increasing amounts of high-quality research software development. If you are a software development practitioner, addressing this ultimately relies on people like you seeing research institutions as potential employers. For researchers building software, it relies on seeing software development as a realistic focus for a research career.

Although the majority of RSEs currently come from the research community and there is not yet a well-established path for professional developers moving into the field, a small number of RSEs already come from a professional development background. RSE-specific roles and job descriptions are making this a realistic career option, and anecdotal evidence suggests that the number of professional developers moving into RSE is growing. Although there can be similarities between RSE roles and those in major tech companies or the startup community, Cannam et al.'s "Ten Reasons to Be a Research Software Engineer"[2] provides some great examples of why a professional developer might choose an RSE career over alternative options.

Ensuring a sustainable approach to research software development is not just about how you build the software but also how you support the community of people building it and how you attract the most talented individuals, both professional developers and researchers, to join this community. In this article, we introduce four pillars of RSE that we see as providing the necessary elements to offer comprehensive, ongoing support for the development of quality research software. The structure aims to demonstrate how professional software development practices can be brought together with approaches used in the research community to provide a more sustainable environment for developers of research software.

The basis for and structure of the four pillars were initially set out in our earlier short paper[3] as a request for comments and input from the community. This article supports the previously proposed structure and provides a range of evidence to show how the areas represented by the four pillars are being realized. We show that these areas include extensive professional software development practitioner knowledge that is now more likely to be recognized as important and necessary in the development of robust research software. With the adoption of elements of this structure, research is rapidly becoming a viable environment for a professional software engineering career, with the benefits that come from working in a flexible environment characterized by interesting research challenges.

This article aims to provide insights in three core areas:

- understanding the importance of RSE to the research community and to the wider developer community and why the field has grown so rapidly in recent years
- setting a structure that defines the core elements of RSE and can support an organization in ensuring reliable, maintainable, and sustainable research software outputs and reproducible research
- providing examples of existing activities that demonstrate the areas and approaches highlighted by the four pillars.

Software increasingly underpins almost all research and, indeed, industry and much of the wider economy.

The Importance of Research Software

Software increasingly underpins almost all research and, indeed, industry and much of the wider economy. Since the earliest days of using software to support and undertake research, there have been links with industry. However, with the emergence of the data economy, enterprises increasingly need to adopt advanced computational processes and pipelines for managing and understanding the vast quantities of data available to them. The speed of development in areas such as data science, machine learning, and artificial intelligence creates a much more direct link between research software outputs, industry, and the wider world.

There are some excellent examples of how open source software, developed to support the research community, is being used to tackle challenges in the global world: for example, the use of Jupyter notebooks for large-scale data analysis by Netflix.[4] Research funding has also been behind some key developments in modern society, such as a research project at Stanford that lies behind the development of the Google search engine.[5] Indeed, research projects and funding have helped support the development of what could be considered some of the most fundamental and life-changing aspects of the modern world, including the Internet itself and the protocols that power it.[6]

Ultimately, good research software can make the difference between valid, sustainable, reproducible research outputs and short-lived, potentially unreliable or erroneous outputs. This is succinctly highlighted by the tagline of the U.K. Software Sustainability Institute: "Better software, better research." Good research software is difficult and time consuming to build, but it matters. However, as pointed out by Anna Nowogrodzki in "How to Support Open-Source Software and Stay Sane,"[7] funding the development of research software can be very challenging.

Four Pillars of Research Software

Our observation is that organizations' support for research software often develops in an ad hoc manner. Our four-pillar structure, summarized in this section, aims to highlight areas that we see as most important in providing comprehensive support for research software. Research software is built in a wide range of different organizations, from large universities to small independent research laboratories. The scale and variety of research-software-related activities will differ between organization types. The four-pillar structure aims to offer a complete picture; however, there will be some organizations for which a subset of the activities described is sufficient.

We are not suggesting that organizations that provide support only for the software development pillar, for example, are bad places to be a software developer. The structure offers organizations a straightforward approach to identifying where current activities could be extended. For existing research software practitioners looking to move into this space, it provides a way of gauging how advanced an organization is in this area. We hope it begins the process of helping to formalize the space within which research software is built.

When the Coding Gets Tough

When we talk about "better" software or "good quality" software, what do we actually mean in the research context, and why is it important? Many researchers who write software are, to some extent, self-taught. They may have had some basic software development training as part of an undergraduate or postgraduate course, but the first time they actually write a tool, script, or application to serve some real-world requirement is when they hit a research problem that can best be solved with some code. At this stage, getting the end result is often the most important aspect, and it is often needed quickly.

Best practices covering documentation, testing, software-design techniques, and code management are key activities. They help ensure that software is reliable and easier to maintain and that the results it outputs are reproducible. Although the RSE movement can provide training in these skills to researchers, they are a natural part of everyday software development for professional software practitioners, who can contribute significantly to this field.

When we talk about "better" software or "good quality" software, what do we actually mean in the research context, and why is it important?

Maintainability, sustainability, and robustness are core aspects of building quality software that form part of our first pillar: software development.

No Developer Is an Island

Software is an incredibly fast-changing field. New frameworks and tools can appear, gain huge traction in the community with thousands or even millions of users, and then, a couple of years later, be almost forgotten and replaced with the "next big thing." How do developers find out about such changes? More importantly, how do they know which of these changes are relevant, which should be learned/adopted, and which should be avoided? What common technical challenges do they face?

In the modern world of software, it is very difficult for developers to work entirely alone. However, someone building research software within an academic research group can frequently be the only developer on a team or might be one of a small group of such people. Life as a research developer can sometimes be lonely. RSEs in central RSE teams may have peers to communicate with, but interacting with other developers from different fields and technical backgrounds can always be beneficial. Professional software practitioners can also bring much to this space in the form of technical advice and guidance.

The need for communication, learning from the experiences of others, and keeping up with the latest developments provides the basis for our second pillar: community.

New Code on the Block

As a developer, there is always something new to learn, partly because of the speed of change in the developer community (and computing in general) and partly because it is such a large area with so many different languages, techniques, and tools. In addition to keeping coding skills up to date, developers also benefit from guidance on tools and approaches for making software open, citable, robust, readable, verifiable, and easier to reuse and contribute to.

The concept of continuing professional development is now commonplace in many industries but is less common in the research community, perhaps because the whole process of research is based around learning/developing new skills and techniques. In our experience, research software developers are enthusiastic about training opportunities.

The importance of ensuring initial and ongoing skills development for people building research software underpins our third pillar: training.

Change the World

Ensuring good software development practices, community, and training for developers is important, but in our opinion, this does not cover the full picture required by a comprehensive supporting environment for research software. The missing element is the need to develop and provide strong institutional processes that recognize the importance of developing quality software as a key to strengthening research outputs. These processes should also include support for research software careers and cover both researchers with a focus on developing software and software practitioners whose main task is supporting researchers. There is also a need for higher-level support provided through government policy and funding approaches.

Developing both institutional and national policies that recognize the importance of research software provides the basis for the fourth pillar: policy.

Although all four pillars contribute to the cultural change required, changes to policy frameworks can provide significant top-down impact to support the other three pillars. Positioning policy within the framework, therefore, is challenging because the other pillars can be seen as relying on, and building on, policy. However, the organic growth of research software activities at institutions means that we observe many cases where other activities are already in place before policy aspects are considered or addressed. Therefore, we include policy within our framework as a pillar supporting the overall RSE space, alongside the other pillars.

Figure 1 summarizes the four pillars, which are similar to the elements of social change models that focus on the individual, organization, community, and policy levels. One example is Nosek's strategy for cultural change[8] (utilized by the Center for Open Science), which identifies five levels: infrastructure, user interface/experience, communities, incentives, and policy.

RSE in the Wild

We now look at some examples of how aspects of our four-pillar structure are being realized "in the wild" by different organizations around the world.

Software Development

It is probably fair to say that software development methodologies familiar to the professional software development community have traditionally not been applied in an environment

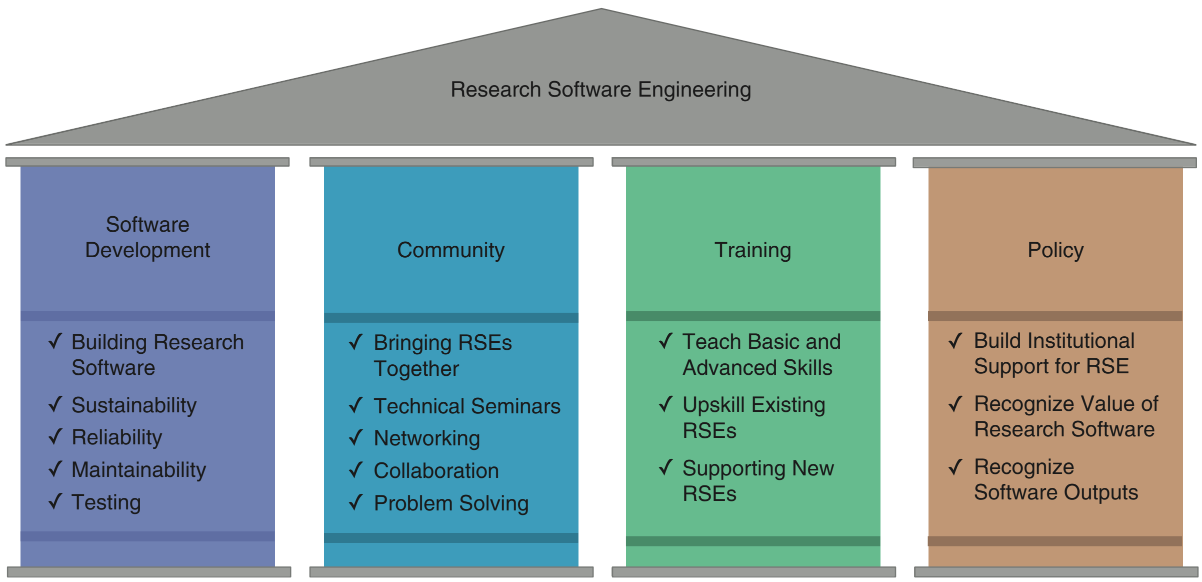


FIGURE 1. The four pillars of RSE.

where one or two self-taught research developers are building some software to solve a research challenge. Having said that, the approaches these developers often end up using actually have similarities to modern agile or rapid-application development methodologies. The process of building a quick prototype, getting feedback, and then iteratively updating and reviewing the code is very useful in a research environment.

This is similar to the "Release Early, Release Often" principle described by Eric S. Raymond[9] in the context of the development of the Linux kernel. Although perhaps not intuitively conducive to ensuring the release of quality, reliable software, this is now recognized as a valuable approach in many scenarios. For open source software, putting code where everyone can see and comment on it can help with gaining valuable feedback and even developing new collaborations that can lead to better-tested, better-quality outputs.

Katz et al.[10] looked at case studies of developing research software at three institutions in the United States and the United Kingdom, focusing on how their staff are organized and the models and processes that they use. In some cases, research software groups undertake activities that go beyond the software development pillar, such as training or running communities, whereas in others, they focus purely on software development.

Over the last couple of years, the U.K.'s Software Sustainability Institute, in collaboration with partners in several other countries, has undertaken an international survey of RSEs. The most recent survey, from 2018,[11] covers RSEs in eight countries and demonstrates how the use of software best practices in areas such as testing is improving over time.

Community

Research software communities can exist at four different levels: local, regional, national, and international.

Local communities generally exist within an institution and support researchers and RSEs with the software development aspects of their roles and networking with peers within their institution. These communities aim to raise the profile of software and of best practices for developing it. Examples of local communities include Imperial College London's research software community[12] and the software engineering community at DLR, the German Aerospace Center.[13] The example of DLR also highlights another general class of RSE communities that are discipline specific. In addition to bringing developers and researchers together, community members can become aware of and engage more effectively with open source software being developed across a community, driven by the release-early-and-often paradigm highlighted previously in the "Software Development" section.

National communities serve a more strategic role in coordinating national research software activities and advocating for better recognition and support for research software professionals. Communities, such as the DevLOG network (http://devlog.cnrs.fr/) in France, have existed for some time, providing support and skills development opportunities to individuals undertaking software-related work.

Following the coining of the RSE name and the development of the U.K. RSE community, national communities were set up in other countries and regions. These include Germany (deRSE: https://www.de-rse.org), The Netherlands (NL-RSE: https://nl-rse.org/), the Nordic region (Nordic-RSE: http://nordic-rse.org/), the United States (US-RSE: https://us-rse.org/), and Australia/New Zealand (RSE-AUNZ: https://rse-aunz.github.io/). Although we acknowledge that this is a small number of countries, we see scope and interest in other countries and believe this is a movement that will continue to expand. The most recent development has been the creation of a professional society for RSEs: the Society of Research Software Engineering (https://society-rse.org/) in the United Kingdom.

We are beginning to see the emergence of regional research software communities that bridge the gap between local and national communities. An example is the Research Software London community (https://rslondon.ac.uk) for London and the South East of England. Regional communities provide opportunities to learn from the different approaches taken at geographically local institutions, share the organization of community activities, and support and engage smaller institutions that may not have a critical mass of people to set up and sustain their own local community.

An international community level is also emerging with the formation of Research Software Engineers International in 2018, after leaders of national RSE associations, groups, and related initiatives from around the world came together in London for the first International RSE Leaders Workshop, and with the launch of the Research Software Alliance (https://www.researchsoft.org/). Other international communities, such as Working Towards Sustainable Software for Science: Practice and Experiences (http://wssspe.researchcomputing.org.uk), bridge the core research software community and more research-focused aspects of software engineering and computing, covering areas such as empirical software engineering.

Training

Research and technology are on a never-ending progression of advancement and change. Software is part of this. Training, therefore, is vital to ensure that the value of research software is understood. It should focus not only on the purely technical skills of writing code but also on how to apply these skills in a manner that results in sustainable, reusable, and maintainable software.

Research and technology are on a never-ending progression of advancement and change. Software is part of this.

There is a diverse range of materials and approaches for training. The Carpentries (https://carpentries.org/) provide an important set of base-level training material covering software, data, library/information tools, and high-performance computing. CodeRefinery provide a series of carpentry-style lessons[14] that include some more advanced technical material but also cover best practices regarding how software can support research through aspects such as reproducibility. Many institutions also provide their own in-house training courses, which are sometimes delivered by members of a central RSE team. For example, RSE groups in U.S., U.K., and German universities and research organizations are increasingly becoming the hub for digital training at their institutions. Such groups are helping to play a key role in enabling sustainable and scalable delivery of the Carpentries (https://carpentries.org/) to the research community. There are also now an increasingly large number of meetup-style groups that offer a more informal approach to training, through technical seminars, for example. These groups are often not tied to a specific organization, and their ability to attract people from a wide range of academic institutions and industry can provide a great opportunity to encourage and support diversity. An example is the international R-Ladies community (https://rladies.org).

Policy

There are now many active groups helping develop, support, and influence policy to aid the research software community. An issue of particular importance and something that is frequently raised at "grassroots" gatherings of RSEs is careers. This is important not only for supporting individuals already in a research software role but also for attracting software practitioners from other fields into research.

Providing roles and career paths tailored to individuals supporting research is, of course, not new. The "research engineer" role that is common in the French research community is one example; it encompasses individuals undertaking a wide range of research-related activities, of which software development is one key area. In countries where there has not been a direct counterpart to this role for individuals focusing on research software, the RSE movement has provided a way to recognize, support, and offer career paths to developers of research software. As their numbers have grown, the challenge RSEs face in finding a long-term, sustainable career path has also become more obvious.

A recent paper[15] on RSEs in the United Kingdom provides a good summary of some of the types of policy change needed, detailing the role that employers and funding bodies can play in career progression. This paper also points out the crucial role of measuring the impact of RSE groups in improving the efficiency of research projects and, in turn, saving time and money and improving the quality of research outputs. These kinds of data can influence government policy makers to recognize the importance of both software and those who develop it. More of this evidence is needed.

Research software teams greatly benefit from having a combination of software practitioners from both industry and research backgrounds. However, research credit is heavily biased toward traditional research outputs, such as published papers. This puts off many researchers from considering a research software career. Although it has been possible to obtain unique digital object identifiers (DOIs), for papers for quite some time, obtaining DOIs for software and data, and having accepted and agreed upon means for citing these research assets is comparatively recent.

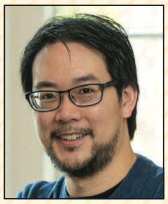
It is important that research software be recognized as a first-order element of research. Policy makers have a very strong role to play here, and things are now changing in this area. The FORCE11 Software Citation Working Group created a set of software citation principles.[16] Set up in 2013, Zenodo (https://zenodo.org) provides a way to obtain DOIs for software and

ABOUT THE AUTHORS

JEREMY COHEN is a research fellow in the Department of Computing at Imperial College London, London, U.K., and holds an Engineering and Physical Sciences Research Council research software engineering fellowship. Cohen received a Ph.D. in computing from Imperial College London. His research interests include high-performance computing and cluster middleware, software libraries and tools, and research software communities. He leads the London and South East of England regional research software community (RSLondon) and is a member of the Society of Research Software Engineering. Further information about him can be found at https://www.imperial.ac.uk/people/jeremy.cohen. Contact him at jeremy.cohen@imperial.ac.uk.

DANIEL S. KATZ is the assistant director for scientific software and applications at the National Center for Supercomputing Applications, and a research associate professor in computer science, electrical and computer engineering, and the School of Information Sciences at the University of Illinois at Urbana-Champaign, Urbana, Illinois, USA. Katz received a Ph.D. in electrical engineering from Northwestern University. His research interests include software applications; algorithms; fault tolerance; programming in parallel and distributed computing; and policy issues, including citation and credit mechanisms and practices associated with software and data, organization and community practices for collaboration, and career paths for computing researchers. He is a Senior Member of IEEE. Further information about him can be found at https://danielskatz.org/. Contact him at d.katz@ieee.org.

MICHELLE BARKER is the director of the Research Software Alliance, Cairns, Queensland, Australia, the international body for research software that brings communities together to collaborate on the advancement of research software. She is the former chair of the Organisation for Economic Cooperation and Development Global Science Forum Expert Group on Building Digital Workforce Capacity and Skills for Data Intensive Science, which makes policy recommendations to member states. Barker received her Ph.D. in sociology on social change from the University of Queensland and an M.B.A. in organizational change from James Cook University. Contact her at michelle@researchsoft.org.



NEIL CHUE HONG is the director of the Software Sustainability Institute and a senior research fellow at Edinburgh Parallel Computing Centre, University of Edinburgh, Edinburgh, U.K. Chue Hong received an M.Phys. in computational physics from the University of Edinburgh. His research interests include communities of practice and policy for research software. He is the editor in chief of *Journal of Open Research Software*, coeditor of *Software Engineering for Science*, and coauthor of *Best Practices for Scientific Computing*. He is a fellow of the British Computing Society and member of the Association of Computing Machinery and the Society of Research Software Engineering. Further information about him can be found at https://www.ed.ac.uk/profile/neil-chue-hong. Contact him at n.chuehong@epcc.ed.ac.uk.

ABOUT THE AUTHORS

ROBERT HAINES is the head of research information technology and an honorary lecturer at the University of Manchester, Manchester, U.K. A computer scientist by training, Haines received a B.Sc. from the University of Manchester. His research interests include software engineering, software sustainability, software use in open and reproducible research, software citation and credit, and career paths for software engineers and data scientists. He is a fellow of the Software Sustainability Institute and was a founding trustee of the Society of Research Software Engineering. He is also a member of the Association for Computing Machinery and the British Computer Society. Further information about him can be found at https://www.research.manchester.ac.uk/portal/robert.haines.html. Contact him at robert.haines@manchester.ac.uk.

CAROLINE JAY is a reader in computer science at the University of Manchester, Manchester, U.K. Her research interests cross the domains of psychology and computer science, exploring the relationship between human behavior, data science, and software engineering. Jay received her C.Psychol. in psychology and her Ph.D. in computer science from the University of Manchester. She is the research director of the UK Research and Innovation Software Sustainability Institute and a fellow of the Alan Turing Institute. Further information about her can be found at https://www.research.manchester.ac.uk/portal/caroline.jay.html. Contact her at caroline.jay@manchester.ac.uk.

data assets. In collaboration with GitHub, details are provided on how to obtain a DOI to cite a GitHub repository.[17] It is not just the need to be able to cite software that is important. Di Cosmo and Zacchiroli, in their description of the Software Heritage initiative,[18] provide a strong case for the need to preserve the information represented by, and within, source code and their work to address this.

In this article, we have presented four pillars representing a set of areas and activities that we consider to be vital in offering coordinated and comprehensive support for research software and the people who build it. In turn, we hope this will demonstrate to professional developers and researchers alike that research is a viable, and interesting, environment for a software development career.

The wide-ranging need for large-scale data processing and computation in industry means that many companies now have technical roles that are similar to RSE roles. These can bring with them salaries and benefits with which research institutions may be unable to compete directly, but there are still reasons to choose an RSE career. The results of the previously mentioned 2018 international RSE survey[11] include "Desire to advance research" and "Freedom to choose own working practices" as some of the most highly ranked responses to a question about reasons for choosing your current job. The career choices that professional developers have, and the demand for and value of their skills, makes it even more important for organizations to invest in the activities represented by the four pillars.

Although we are currently unaware of any research institution/organization that comprehensively implements all of the aspects of the four-pillar structure, some institutions are close to achieving this and are already realizing benefits from their support of research software activities. However, as highlighted in the "Four Pillars of Research Software" section, we recognize that full realization of this structure within an organization, where some form of support is provided for each of the four pillars, may not always be practical, realistic, or even necessary.

We have tried to present examples of existing activities under the different pillars to give an idea of how the various elements of our structure may be realized and to demonstrate how they might be able to help support the development of more effective, sustainable, reliable research software. Providing this support is important to ensure the quality and longevity of research outputs and attract professional software practitioners to the research community.

Acknowledgments

We would like to thank the large number of individuals who have initiated and supported the development of RSE over several years, leading to the activities used as examples in this article. Although it is impractical to name specific people, we would particularly like to acknowledge the individuals who have been at the forefront of developing RSE groups, communities, and supporting infrastructure in the United Kingdom and beyond. Jeremy Cohen acknowledges the UK Research and Innovation's Engineering and Physical Sciences Research Council (EPSRC) for support from grant EP/R025460/1. Daniel S. Katz acknowledges support from NSF 1743188. Caroline Jay and Neil Chue Hong acknowledge support from EPSRC EP/S021779/1, and Neil Chue Hong acknowledges EP/N006410/1.

References

1. A. Brett et al., "Research software engineers: state of the nation report 2017," Apr 2017. [Online]. Available: https://zenodo.org/record/495360#.XkR28W5Fx5A
2. C. Cannam, D. Gorissen, J. Hetherington, C. Johnston, S. Hettrick, and M. Woodbridge, "Ten reasons to be a research software engineer," Aug. 23, 2013. [Online]. Available: https://www.software.ac.uk/blog/2013-08-23-ten-reasons-be-research-software-engineer
3. J. Cohen, D. S. Katz, M. Barker, R. Haines, and N. C Hong, "Building a sustainable structure for research software engineering activities," in *Proc. 2018 IEEE 14th Int. Conf. e-Science*, Oct. 2018, pp. 31–32. doi: 10.1109/eScience.2018.00015.
4. M. Ufford, M. Pacer, M. Seal, and K. Kelley, "Beyond interactive: Notebook innovation at Netflix," Aug. 16, 2018. [Online]. Available: https://medium.com/netflix-techblog/notebook-innovation-591ee3221233
5. D. Hart, "On the origins of Google," Aug. 17, 2004. [Online]. Available: https://www.nsf.gov/discoveries/disc_summ.jsp?cntn_id=100660
6. V. Cerf, "A brief history of the Internet and related networks." Accessed on: Jan. 7, 2020. [Online]. Available: https://www.internetsociety.org/internet/history-internet/brief-history-internet-related-networks
7. A. Nowogrodzki, "How to support open-source software and stay sane," *Nature*, vol. 571, no. 7763, pp. 133–134, July 2019. doi: 10.1038/d41586-019-02046-0.
8. B. Nosek, "Strategy for culture change," June 11, 2019. [Online]. Available: https://cos.io/blog/strategy-culture-change/
9. E. S. Raymond, "Release early, release often." Accessed on: Oct. 29, 2019. [Online]. Available: http://www.catb.org/esr/writings/cathedral-bazaar/cathedral-bazaar/ar01s04.html
10. D. S. Katz, K. McHenry, C. Reinking, and R. Haines, "Research software development and management in universities: Case studies from Manchester's RSDS Group, Illinois' NCSA, and Notre Dame's CRC," in *Proc. IEEE/ACM 14th Intl. Workshop Software Engineering for Science (SE4Science)*, pp. 17–24, 2019. doi: 10.1109/SE4Science.2019.00009.
11. O. Philippe et al., "Softwaresaved/international-survey: Public release for 2018 results (version 2018-v.1.0.2)," March 6, 2019. [Online]. Available: https://zenodo.org/record/2585783#.XkR7I25Fx5A
12. Imperial College London, "Research software community." Accessed on: July 5, 2019. [Online]. Available: http://www.imperial.ac.uk/computational-methods/rse/
13. C. Haupt and T. Schlauch, "Track 1 Paper: The Software Engineering Community at DLR—How we got where we are," April 9, 2017. [Online]. Available: https://figshare.com/articles/The_Software_Engineering_Community_at_DLR_How_we_got_where_we_are/5331703/2
14. Nordic e-Infrastructure Collaboration (NeIC), "CodeRefinery: Lessons" Accessed on: July 8, 2019. [Online]. Available: https://coderefinery.org/lessons/
15. J. Switters and D. Osimo, "Recognising the importance of software in research–Research software engineers (RSEs), a U.K. example. Open science monitor case study," Jan. 2019. [Online]. Available: https://ec.europa.eu/info/sites/info/files/research_and_innovation/importance_of_software_in_research.pdf
16. A. M. Smith, D. S. Katz, K. E. Niemeyer, and The FORCE11 Software Citation Working Group, "Software citation principles," *PeerJ Comput. Sci.*, vol. 2, no. e86, Sept. 2016. doi: 10.7717/peerj-cs.86.
17. GitHub Guides, "Making your code citable." Accessed on: Oct. 29, 2019. [Online]. Available: https://guides.github.com/activities/citable-code/
18. R. Di Cosmo and S. Zacchiroli, "Software heritage: Why and how to preserve software source code," in *Proc. 14th Int. Conf. Digital Preservation (iPRES2017)*, Kyoto, Japan, Sept. 2017, pp. 1–10. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01590958