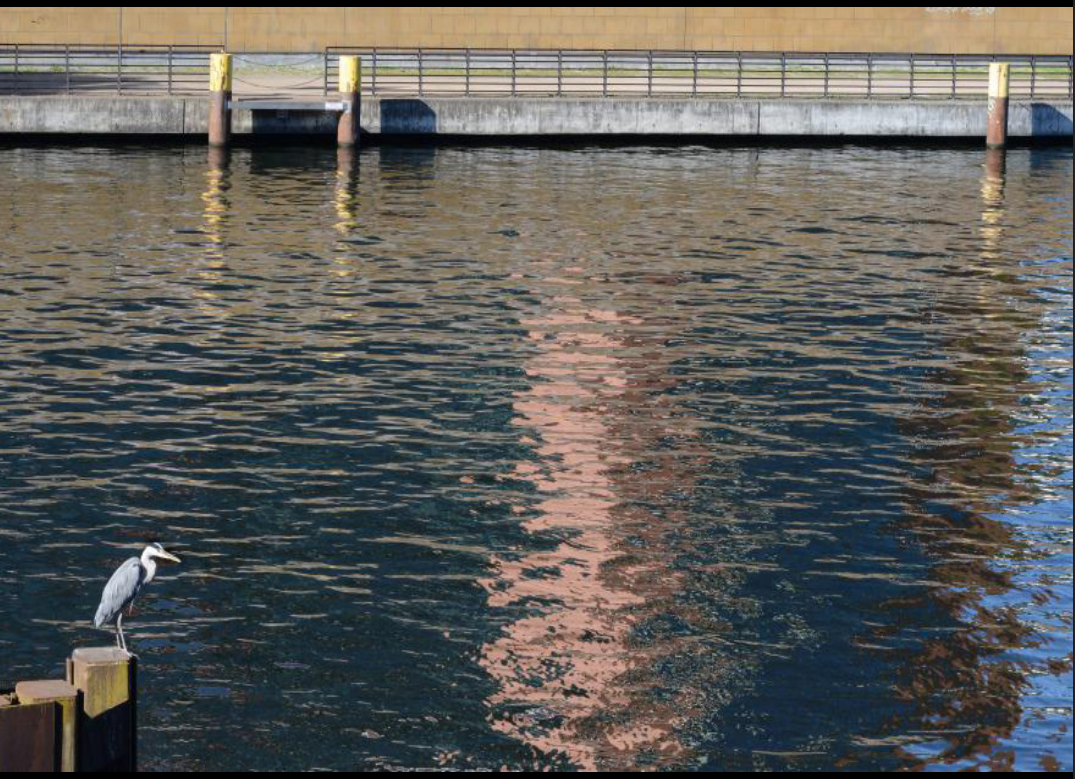
OPEN SOURCE STRATEGIES IN RESEARCH

Thierry de Crespigny
Alex Harrigan
Amelia Nishimura
Matthew St Jean



An Interactive Qualifying Project submitted to the faculty of Worcester Polytechnic Institute in partial fulfillment of the requirements for the Degree of Bachelor of Science. **D Term, Spring 2020.**

Advisors

Katherine Foo
Sarah Stanlick

WPI

Sponsor

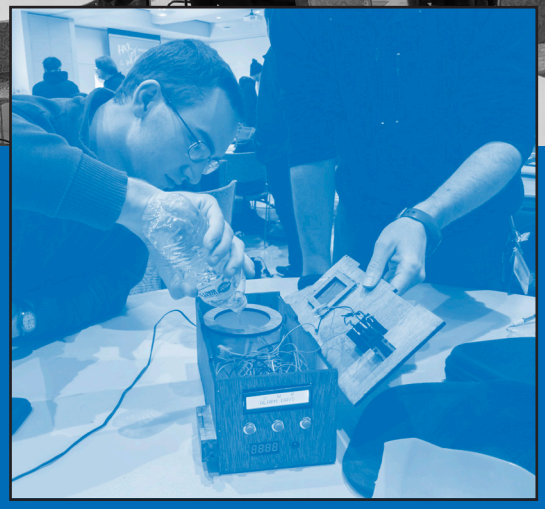
Leibniz-Institute of Freshwater Ecology and Inland Fisheries

IGB

Cover & Left: Image credit J. Freyhof/IGB

Modern research institutions use software both to produce and organize their research, making software skills incredibly useful to researchers. However, many researchers are self-taught, lacking a professional background in software development. Because of this, researchers are often unaware of or lack the time to learn the best practices for publishing work in a well-documented, open source format. This leads to difficult-to-use code being published alongside research, if it is published at all. By helping researchers improve in these areas, software and hardware that accompany research will become easier to reuse and develop. The objective of this project was twofold. First we aimed to gather information from IGB staff relevant to open source methods and software, and raise awareness of best practices for open source at the IGB. This was done through a series of surveys which provided general statistics about the IGB staff relevant to open source methodologies, and interviews with several individuals providing in depth personal experience. Secondly, we researched best practices as well as available software, and based on this along with the data collected from the IGB staff we made policy suggestions to the IGB that we thought would lead to wider adoption of open source methodology and high quality software standards. By raising awareness of and providing a set of guidelines for open source publishing platforms and ideas, we hope to assist researchers to publish high quality work that promotes cooperation and innovation.

Hackathons are an excellent example of open source in action. Often hosted by colleges or universities, hackathons are events where teams of people develop programs over the course of a 24-48-hour period for fun. At their core, they are massive collaborative projects. Teams share their work with each other in order to produce the best product they can at the end of the event, usually with the code freely available.



Pictured here are photos from a hackathon conducted by the WPI Association for Computing Machinery in February 2020. Image credit Isaac Donkoh-Halm, provided by Sarah Akbar.

Encouraging Good Software Practices in Scientific Research

Open access and open source are important components of collaborative research practices. The methods through which research is documented and published are integral to how it is reproduced and advanced over time. Open access publication of a given work ensures free and ungated access to readers, with the purpose of encouraging collaboration in the scientific community by making research more accessible. Open source follows similar principles to open access, but it applies more specifically to hardware and software projects. The philosophy behind open source is that a project is created from the ground up with public collaboration in mind, leaving the moving parts beneath a project, such as the source code of software, accessible for free for any user to modify or use. Usage of open source publication in a research environment could greatly expand the field's collaborative capabilities by enabling easier and faster usage of published code.

Despite the potential of open source, which we have already seen in the world of software development, it remains an underused framework in the field of research. Several obstacles stand in the way of effective implementation. Many researchers lack both software development training and the knowledge of tools and resources that could help improve their coding processes, which manifests in their software projects in the form of frequent fragmentation, insufficient documentation, and heavy technical debt, or flaws accrued from layers of quick, messy fixes. There is also a dearth of incentives on an institutional level for researchers to publish their software. Researchers are not awarded credit for writing and publishing software like they are for papers. Institutions also rarely have a central repository specifically for software, which would facilitate easy access and collaboration (Koglin, Personal Communication, 2020). As a result, what software that is shared is difficult for anyone other than the original author to use, modify, or interpret, taking away from the intentions of its parent work's open access nature. Policy changes at the institutional level can create incentives to publish and share programs and spread awareness of programming tools and techniques.

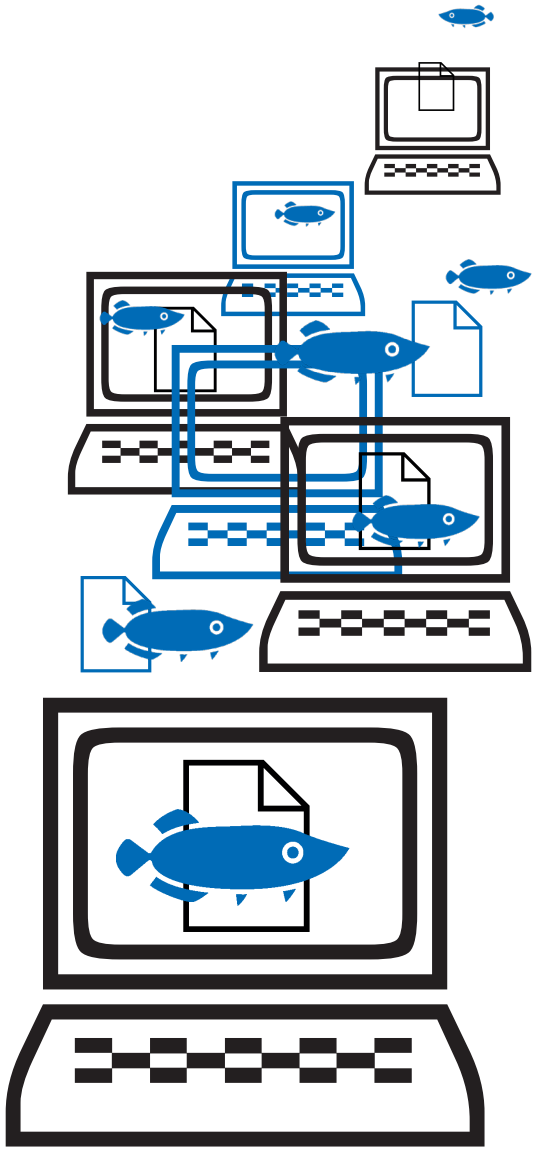
The sponsor of our project, The Leibniz Institute of Freshwater Ecology and Inland Fisheries (IGB) frequently deals with the software development and publication obstacles outlined above. Researchers working at the IGB are required to publish their work in open access journals, but often do not (or cannot) provide the same level of accessibility to the software that they produce in the process. The IGB has proactively reached out to our team in order to gather data to begin to build policies that encourage and reward open source publication, putting themselves in a unique position, potentially at the forefront of open source research.

Our project addressed some of the problems that result from poor publication and development practices by producing a policy brief for the IGB's administration detailing ways to better implement open source practices. We also produced a set of guidelines for researchers on tools and techniques that could help them improve the quality and accessibility of their software. To gather the data that informed these documents, we reviewed existing literature and sent surveys out to the staff at the IGB. We also conducted interviews with both IGB researchers and staff and individuals at other organizations to better understand academic publishing and research software development on a broader scale and to compare processes between institutions. Finally, we researched version control and documentation generating tools and software in order to make the best and most appropriate possible suggestions in our two deliverables. We hope this project helps further a culture of sharing, collaboration, and documentation processes for the IGB and beyond.

Image credit David Bierbach/IGB

Open Source Concepts In A Research Environment

The rapid growth of the internet has facilitated an unprecedented increase in the speed and quality of global communications. Collaboration between geographically distant groups has never been easier, and the application of open source methodologies to research programming can take advantage of this to improve the quality and accessibility of published scientific work. Open source software publication, if implemented well, would allow a scientist in Worcester to easily access a script written two years earlier by a researcher in Melbourne, and to use or modify it as needed to fit their own research. This level of collaboration and accessibility has the potential to greatly increase the rate of scientific discovery by enabling researchers to spend less time reproducing work that another individual has already done. As open access has already been a disruptive force to the world of scientific publishing, so too can open source be to the process of research itself. Understanding the current state of the former gives us an essential framework for understanding the potential future of the latter.



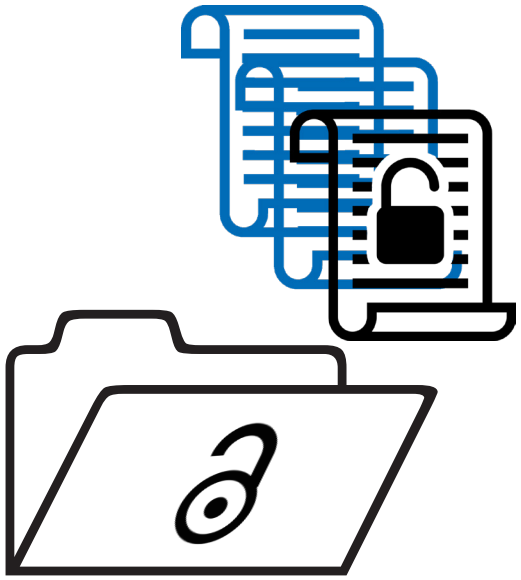
The State of Open Access

Open access philosophy provides the conceptual background for open source practices. Open access broadly applies to the free accessibility of any work and open source builds on this by encouraging not only free access but collaboration. While open access is not the focus of our work, understanding the concepts behind it is helpful in contextualizing the importance of open source practices. The fundamental goal of open access is to drive innovation by increasing accessibility to scientific knowledge; it can be defined as the "practice of providing online access to scientific information that is free of charge to the end-user and reusable" (European Commission, 2017). There are two commonly used types of open access: Gold open access and Green open access. Gold open access ensures that an article will be free and available for anyone to read whether through publication in an open access journal or being archived online. Almost all journals charge authors some editorial fee to publish their work, the difference with open access journals is that they do not also charge the reader for access. Green open access involves self-archiving an article in parallel with its publication in a paid-access journal, usually after an embargo period imposed by the publisher. Self-archiving is the term for an author saving their work in an online repository, like what is done up front under Gold open access. (European Commission, 2017). Gold open access is strongly preferred, as there are fewer legal gray areas and pitfalls, such as restrictive or overlapping copyright claims, when a work is initially published in an open access format. Green open access exists because it is unfortunately not always possible to publish using Gold open access, whether due to contractual obligations or other negotiated publication rights (Open Access Policy, 2019).

As examples of implementation of open access policy, the European Union and the United States both have recently mandated variations on open access publication. The United States under the Obama administration published a memorandum for various federal departments that outlined new requirements on the public accessibility of research published under tax-payer funding within the constraints of national security. This document outlines how free access to published work enables

	European Union	United States
Preferred Publication Standard	Gold Open Access (Open Access Journal)	Green Open Access (Stored in a public repository after 12 month embargo)
Legal Standing of Policy	Legally mandated under Grant Agreement Section 29.2	Order from Office of the President
Effectiveness	As shown by interim reports, Horizon 2020 has been very successful.	A Government Accountability Office report in 2019 showed that 11 of the 19 agencies reviewed had not fully developed mechanisms for ensuring compliance

Table 1: *European Union vs. United States Open Access Policy*



greater innovation. It recommends that research be made available in certain repositories after a twelve-month embargo period (Holdren, 2013). The publication method outlined here is Green open access, as most of the work is initially published in a paid journal and then later published in a free repository. The European Union's Horizon 2020 program also requires that publicly funded research be made publicly available; however, it goes a step further and explicitly indicates Gold open access as the preferred method of publication (European Commission, 2017). While both follow the same principles of open access, the EU's guidelines are far more comprehensive in their mandates. It should also be noted that the enforceability of the US guidelines is questionable, as a report by the Government Accountability Office in 2019 revealed that most agencies had not complied, either in whole or in part, with the 2013 memorandum (Federal Research, 2019).

Open Source Practices & Obstacles

Open source builds on the principles laid out by open access by encouraging public collaboration on software or hardware projects. While being coined as a term in the 1990's, open source has been around as a concept since the earliest computers. IBM's mainframe software, first produced in the 50's, and ARPANET, a precursor to the internet used as a means of linking certain research organizations, were developed in part using public input and independent collaboration.

As a practice, open source is simple. Source code is posted to a free public repository, allowing other users to download and run the software, edit the code freely, and submit their modifications for integration into the project, if they so wish. The quality of the code improves with each iteration of this cycle, as more eyes on the project means more feedback, more chances to catch and fix bugs, and more new and better features. This process should not be mischaracterized as a messy anarchist free-for-all, as in most cases the original author maintains final control over how and when major changes are made, and new versions are published. Other users' contributions can be likened more to professional suggestions that must be rigorously tested before they can be incorporated into the final product.

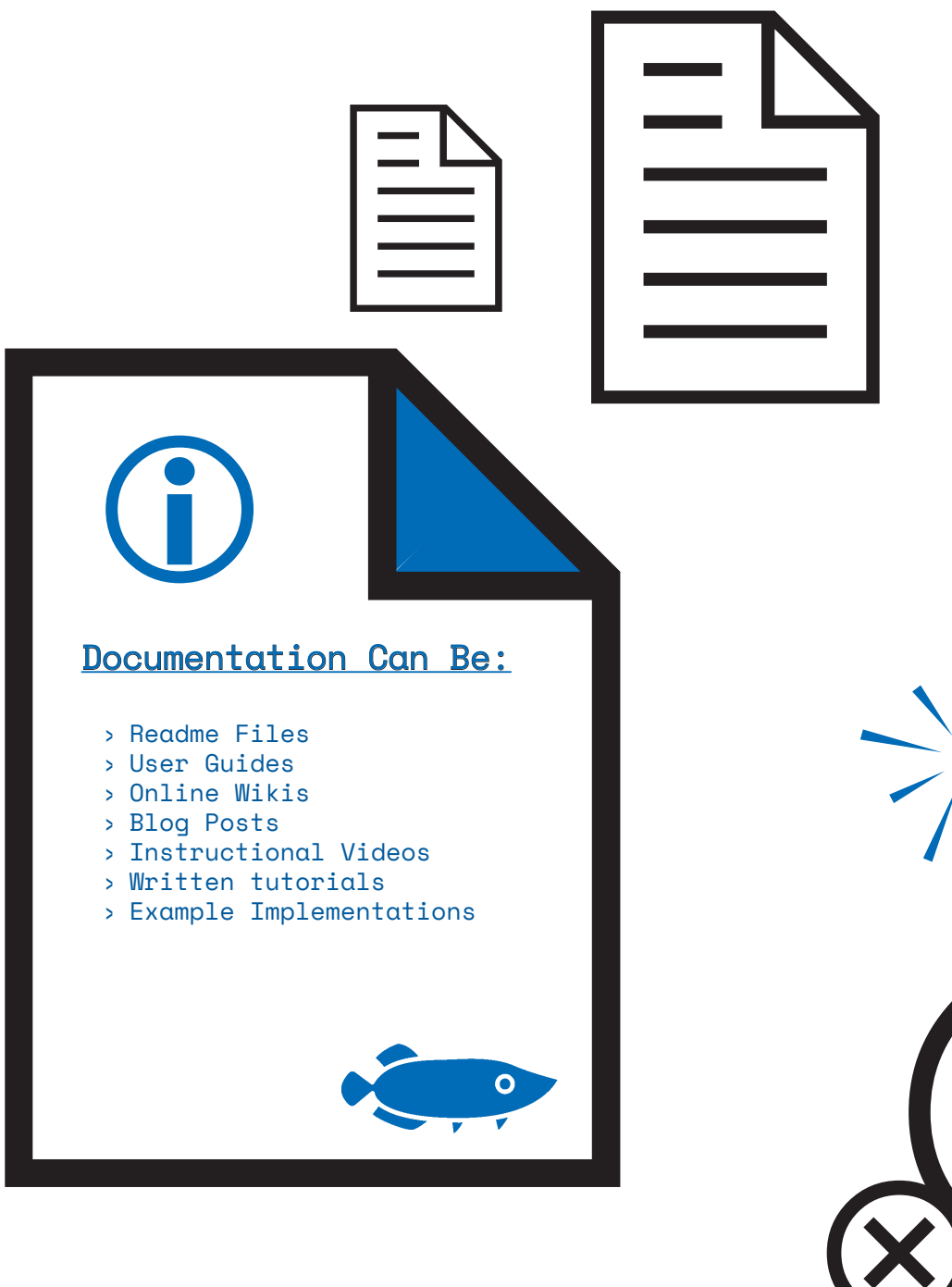
Certain practices, especially version control, are key to ensuring that this development process runs smoothly and effectively. Version control uses software like Git to track changes to each file in a project as they are made, so that specific changes can be undone later, and entire previous versions can be recalled if necessary. This organization and transparency let development processes be more flexible and gives software and hardware projects much greater adaptability.

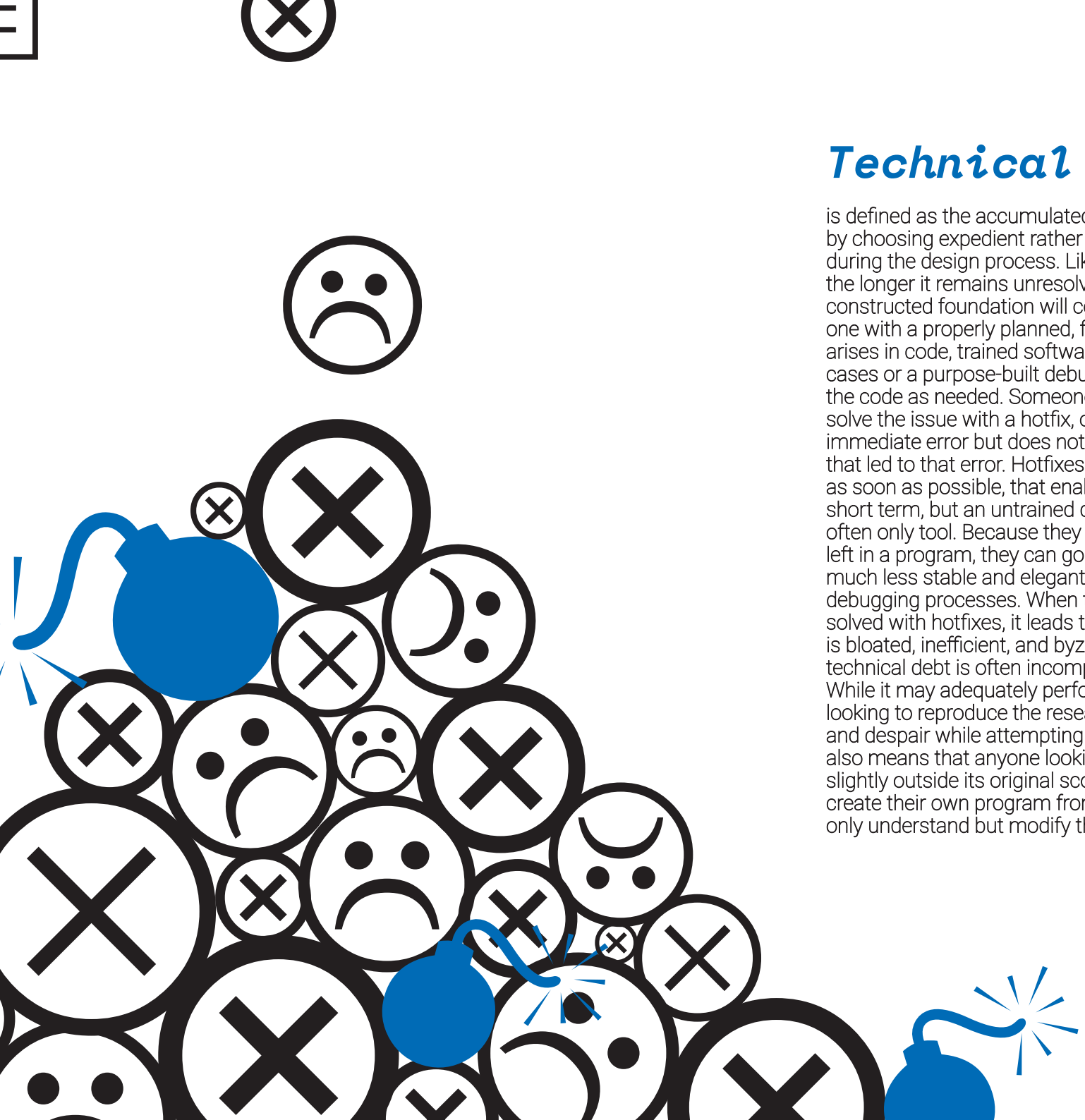
Open source brings the same ideas of free accessibility and collaborative science to code developed by researchers that open access does to their papers. The ability to easily access and modify the works of others afforded by open source publication enables faster workflows and more open and productive collaboration between both individuals and teams

Open access policies are not difficult to implement, as evidenced by the European Union's Horizon 2020 program. The program aimed to ensure that publicly funded research within the EU was published in an open access format and was drafted and ratified by a large majority of member nations in 2013 (European Commission, 2020). Trouble emerges when moving from policy to implementation of open source practices. Despite the collaborative and productivity benefits of open source practices, there are a few common obstacles to their realization in the research environment. One issue is the lack of training in software development that many researchers have. Despite 84% of a surveyed body of researchers viewing creating software as either important or very important to their work, only 34% of that same body of researchers believed that formal education or training in software development carried the same level of importance (Hannay et al., 2009). As a result of this lack of more formal training, three major symptoms present themselves and impede open source software collaboration and publishing: insufficient documentation, technical debt, and fragmentation..

Documentation

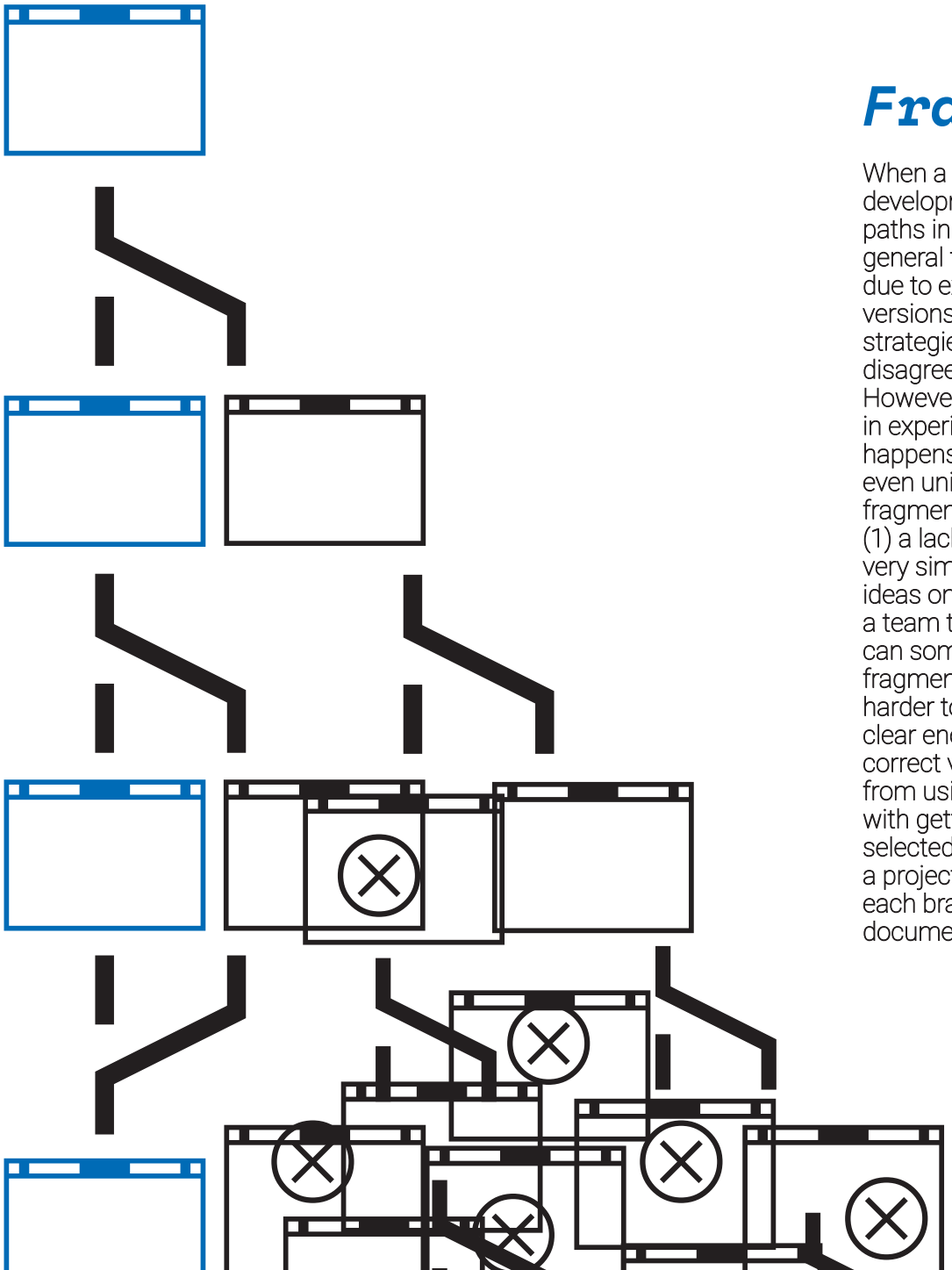
is the supporting descriptions, instructions, or explanatory documents that enable a user to understand a piece of software. In many cases, research software is published alongside its paper in a ZIP file without any supporting documentation, relying on that paper to explain its usage and implementation (Dämpfling, Personal Communication, 2020). Most users of open source research software (about 71%) feel that the published paper alone is not sufficient documentation to enable proper usage of the software, and believe that user manuals or similar paperwork would be "very useful" (Pianosi et al., 2019). Unfortunately, there is no overarching consensus on what constitutes "sufficient" documentation within the scientific community. Even if the documentation provided includes sample workflows or tutorials, as many commercial pieces of software do, less intuitive features can be overlooked by the user if explicit instructions on their use and purpose are not given. Even if a piece of scientific software comes with a comprehensive usage guide, the results it generates are meaningless if the original distributors do not give detailed instructions for analysis and interpretation of data. Formal software development training does teach about documentation and explanation of software, and the lack of these skills in many researchers means that even if they produce an incredible piece of software and publish it in an open source repository, it may still be unusable as no one can figure out how it works.





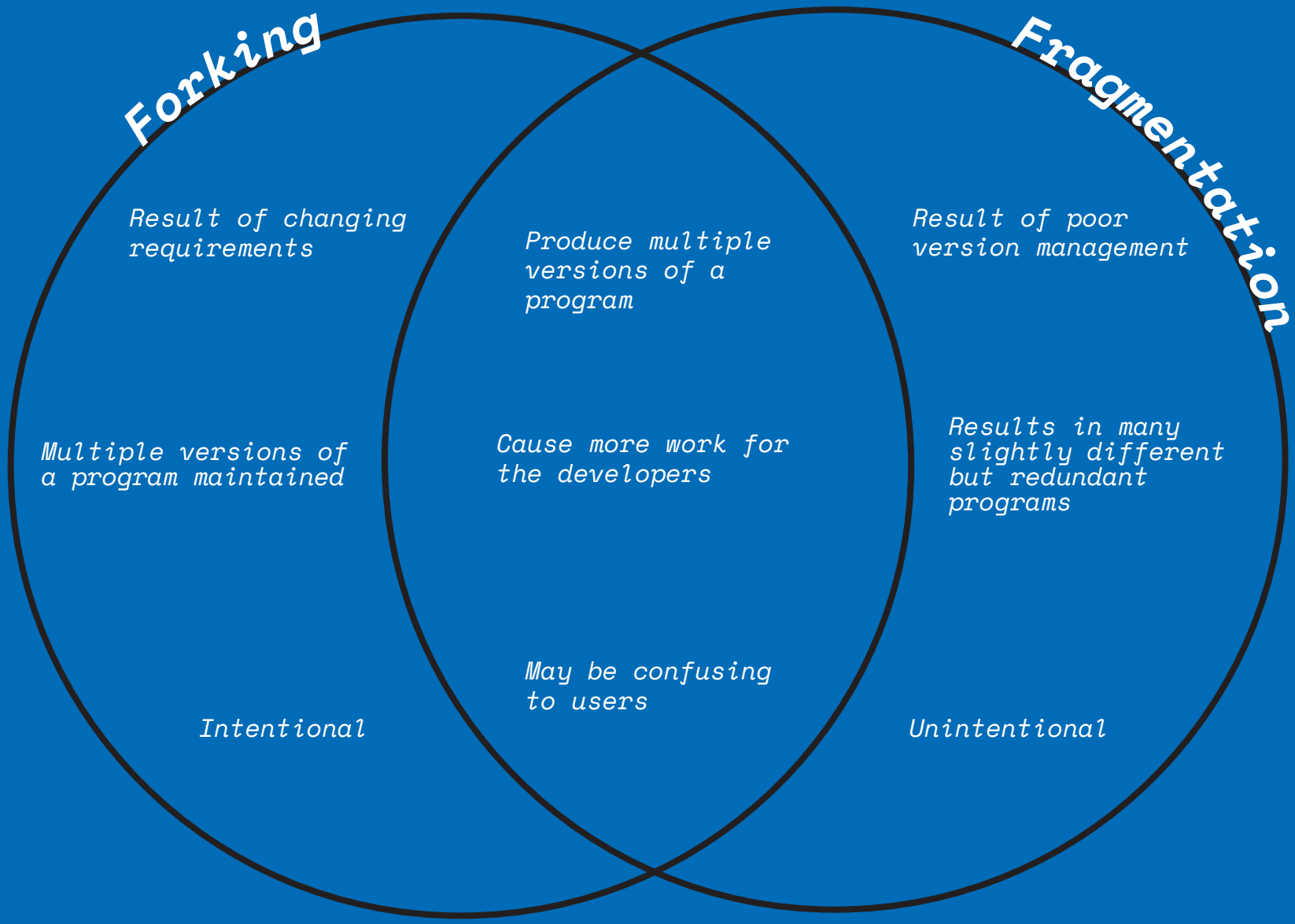
Technical Debt

is defined as the accumulated cost of potential future reworks caused by choosing expedient rather than comprehensive solutions to problems during the design process. Like financial debt, it accrues greater weight the longer it remains unresolved; a house with an unstable, shoddily constructed foundation will cost more to fix and maintain over time than one with a properly planned, funded, and built core. When a problem arises in code, trained software developers and engineers will use test cases or a purpose-built debugging program, find the source, and rewrite the code as needed. Someone with less experience would likely instead solve the issue with a hotfix, or a quick solution that deals with the immediate error but does not address the underlying problem or problems that led to that error. Hotfixes are meant to be interim solutions, replaced as soon as possible, that enable a program to continue to function in the short term, but an untrained developer will use them as their primary, and often only tool. Because they are supposed to be temporary, if hotfixes are left in a program, they can go on to cause other errors later on; they are much less stable and elegant solutions than those generated by thorough debugging processes. When the errors caused by hotfixes are themselves solved with hotfixes, it leads to a cascade resulting in a final program that is bloated, inefficient, and byzantine in structure. A program with high technical debt is often incomprehensible to anyone but its original author. While it may adequately perform its intended function, most anyone looking to reproduce the researcher's work would spiral into madness and despair while attempting to unravel how the software functions. It also means that anyone looking to repurpose this software for a use even slightly outside its original scope would be better off spending the time to create their own program from scratch as opposed to attempting to not only understand but modify the obtuse original.



Fragmentation

When a project's versions are not properly managed, the development cycle of a piece of software forks into multiple paths in a process known as fragmentation. Forking is the more general term and is not in itself a bad thing. It can occur either due to extenuating circumstances necessitating that multiple versions be maintained, despite otherwise solid development strategies, or intentionally, as the result of a higher-level disagreement (Dämpfling, Personal Communication, 2020). However, this deliberate division of a project is far more likely in experienced, well-trained groups - when something similar happens in a less skilled team, it is often poorly managed or even unintentional, in which case it is more specifically called fragmentation. Fragmentation results from two main causes: (1) a lack of proper version control leads to multiple pieces of very similar software existing simultaneously, or (2) diverging ideas on functionality or implementation, where members of a team take the software in different directions. While forking can sometimes be beneficial or necessary, as outlined above, fragmentation is explicitly negative, and makes software harder to use. If multiple versions of a program exist without clear enough distinction, it can cause confusion as to what the correct version to use is. This ambiguity may turn someone off from using the software or could cause significant problems with getting the program to run if the incorrect version is selected. Managing and maintaining multiple active versions of a project is also a huge amount of work for programmers, since each branch requires its own distinct, detailed, up to date set of documentation.


Forking
Result of changing requirements
Multiple versions of a program maintained
Intentional
Produce multiple versions of a program
Cause more work for the developers
May be confusing to users
Fragmentation
Result of poor version management
Results in many slightly different but redundant programs
Unintentional

These three factors - lack of documentation, accruement of technical debt, and fragmentation - come together to create software that cannot be easily interpreted by anyone other than the original author. Open source is predicated on the accessibility of a given work, and obtuse software hinders this. Better understanding of these issues through education and openly available resources would facilitate the adoption of open source practices that in turn would boost the ability of researchers to collaborate on their software.

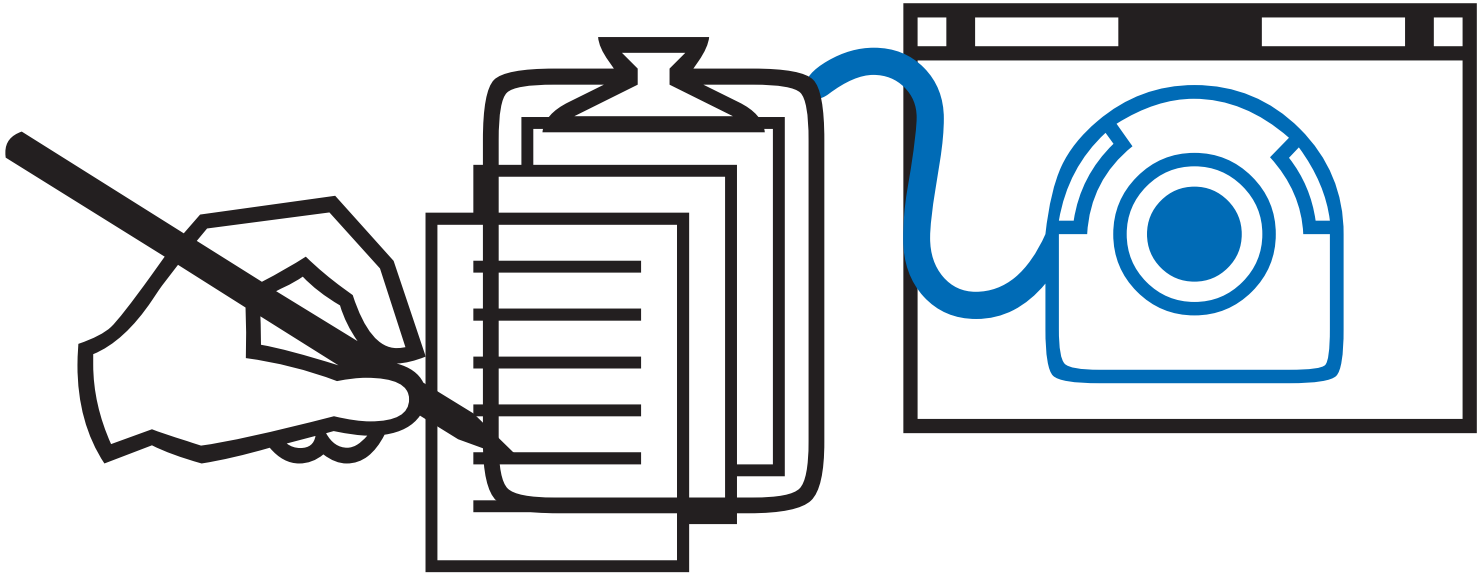
Institutional Incentives (or Lack Thereof)

The other major obstacle to effective implementation of open source practices is the lack of incentive for researchers to do so. There is no reason, beyond personal motivation, for a researcher to go above and beyond to make the software they are publishing clean, documented, and accessible (Dämpfling, Personal Communication, 2020). While it is true that there are many individuals who would, given the time and ability to do so, produce more accessible software, that this effort is not rewarded or even recognized at an institutional level ensures that it is not common practice. Nowhere in the European Union's open access policies for papers are standards on how accompanying software should be published. Researchers also do not receive specific credit for the software they produce for a project, with publishers and organizations viewing the software as an extension of the research, rather than its own entity. Because many pieces of software are large, complex, and able to be used in other avenues of research, the fact that an individual is not given credit by their parent organization or other groups for the software they create disincentivizes them from ensuring that it is not just usable but sustainable (Dämpfling, Personal Communication, 2020).

When these two major obstacles, lack of training and lack of proper incentives, are combined, they result in published research software being unnecessarily difficult for anyone other than the original author to interpret or implement. This inaccessibility runs contradictory to the open access principles along which many papers are published. While the idea of setting up every team of research staff with the years of formal software engineering training that developers receive is equal parts beautifully idealistic and utterly unrealistic, education on formalized and tacit best practices, the importance of those techniques, and tools that can enable researchers to produce better, more accessible software would go a great distance in creating a more free, efficient, and collaborative research environment.

Experimental Methods For Data Collection

The objectives of this study were twofold: to develop a set of policy guidelines for the administration at the IGB and to articulate the importance of proper documentation and methodical code writing to researchers through a written guide. We used two primary methods for data collection. The first was a survey sent out to determine the availability of IGB staff for interviews and collect preliminary information. We also sent a second survey to a series of digital humanities researchers in order to broaden the pool of individuals sampled and gain insight into the practices of other fields. The second means of data acquisition was a series of interviews that we conducted with a variety of employees at the IGB, including researchers, technical support staff, and students, as well as a broad selection of individuals from the digital humanities. These interviews were used to develop our policy recommendations and determine how in depth our guidelines needed to be.



Survey & Interview Details

The survey and interview methods were chosen as they offered the ability to determine what specific practices are used by the surveyed and interviewed individuals. This was necessary because no two people produce software and hardware in the same way, despite widespread availability of information on general best practices for open source. The survey was conducted first, in order to determine which members of our sample pool were available and willing to be interviewed. It should be noted that interviews are a large time investment, as they need to be conducted on an individual basis and can often take a significant period of time each. Despite this fact, there was no other means of data collection that offered the same insight into both the exact practices and knowledge of the individual and the current state of affairs within the IGB and the field as a whole. While a survey was used, its main purpose was not to gather specific knowledge but rather to give the data that drove how subsequent interviews were focused. The importance of reaching out to individuals and groups outside of the IGB should not be overlooked. By gathering data from people outside of our sponsor organization, we were able to implement practices and techniques that other fields, primarily digital humanities, used.

The survey consisted of an eight question Google Form that was sent out to the IGB as a whole in coordination with our sponsor and a modified version that was sent out to a group of digital humanities researchers. The bulk of the questions were intended to determine surface-level information on our sample groups, including the software they used, the average time they spent coding, and their familiarity with open source. The primary objective was to determine availability for interviews, which meant that there were questions on both willingness to be interviewed and what times would be best for the interviewee. The survey questions used are listed to the right. We hoped that we would receive a response rate of 20% from the IGB and a similar rate from the digital humanities pool. This gave us 46 individuals to gather data from at the IGB, and eight or nine from the digital humanities group. This sample size allowed us to account for imbalances in response rate between departments.

We chose interviews as our primary means of data acquisition, in order to collect individualized qualitative data on opportunities and barriers for open source. We set a goal of a minimum of 10 people to be interviewed. This was assuming those individuals were spread proportionally to the staffing ratios at the IGB (i.e. most of the interviewees would be research students and staff). An ideal number for interviews was around 15, or more if scheduling allowed. This gave us a large enough sample size to evaluate trends and general themes within IGB staff while also remaining realistic regarding the amount of time that the interviews would take to conduct. These interviews were conducted in teams of two with one team member directing the interview while the other took notes, and done over Zoom calls in order to bridge proximity (interviewees spanned locations from California to Germany) and allow for "face-to-face"

- What is your name/position/organization?
- What languages do you program in (if any)?
- What CAD software or other engineering software do you use (if any)?
- How many hours per day do you spend on average developing scientific software?
- How many hours per day do you spend using scientific software developed by others?
- How would you rate your familiarity with open source?
- How would you rate your familiarity with version control/documentation software?
- Have you received any formal Software Engineering training?
- Would you be willing to be interviewed as part of a study on open source in research?

interaction in a time of stay-at-home orders due to the COVID-19 pandemic. An audio recording of each interview was taken as well to act as backup for the written notes. The primary group interviewed was the research staff and students from different research groups. This was because in addition to making up the bulk of the staff at the IGB, the students have the most direct experience with scientific software development. The objectives of the interviews with researchers were as follows:

- To evaluate the development cycle at the IGB.
- To determine the level of experience/ knowledge with/of open source.
- To determine the level of experience with software/hardware development.
- To determine how much time is spent by individuals on software development.
- To gauge willingness to accept policy change.

For interviews conducted with IGB administrative staff, we used the same questions as in the researcher interviews for consistency, with the addition of questions regarding how the institution's administration perceives open source policy as well as inquiry into the administrative perspective on the usage of open hardware projects. When it came to the technical support staff, such as the electrical and mechanical engineering staff, we created additional questions that focus on the open hardware side of the issue. These included general inquiry into experience with open hardware, how often open hardware projects are used within the IGB, as well as what, if any, cost savings are experienced by using open hardware projects

Outside of the IGB, we also surveyed a selection of digital humanities researchers. In gathering these external perspectives, we hoped to find similarities or differences we could use to "calibrate" our IGB data, through identifying trends across countries and fields of study and themes exclusive to one group or another. This range of data would help us to not only get a better understanding of our core topic on a broader scale, but also to see the institutional view at the IGB more clearly.

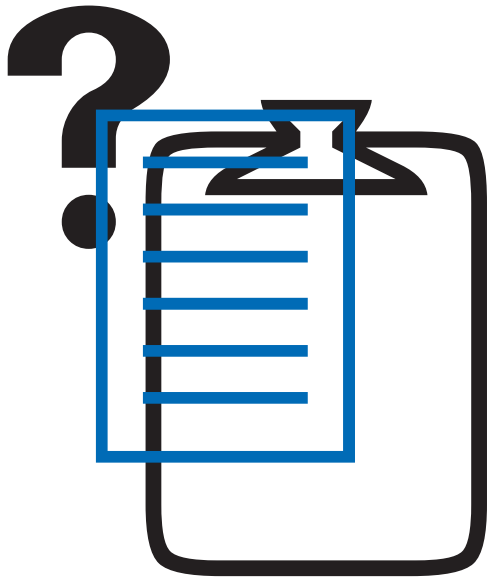
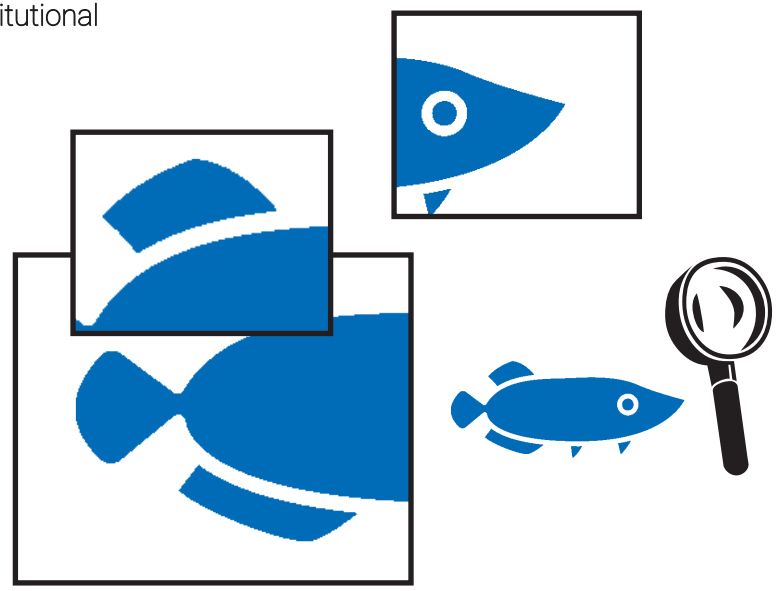


Image credit Philipp Freudenberg/IGB

Results and Analysis

During our research, we used generalized surveys and remote interviews in order to understand the role of open access software in research both within the IGB and across other contexts. The survey responses we received gave us an initial understanding of each group's experience with programming and open source concepts. Our primary focal group of the IGB staff, and secondary group of digital humanities researchers had several distinct trends warranting separate surveys and analysis. The IGB staff contributed most of the survey data with 41 total respondents, while the humanities survey produced 6 responses. To flesh out and build on the data from our initial survey, we conducted interviews with individuals from the digital humanities group and a variety of positions at the IGB, including grad students, post doc researchers, and technicians. By interviewing people in diverse research-related roles, we were able to get a wide range of opinions and experiences. This allowed us to put together a more comprehensive picture of the strengths and weaknesses of software development and publishing practices at the IGB. The data we gathered confirmed how important programming is to the research process and showed that both coding experience on a personal level and institutional support would be key to successfully implementing open source practices.



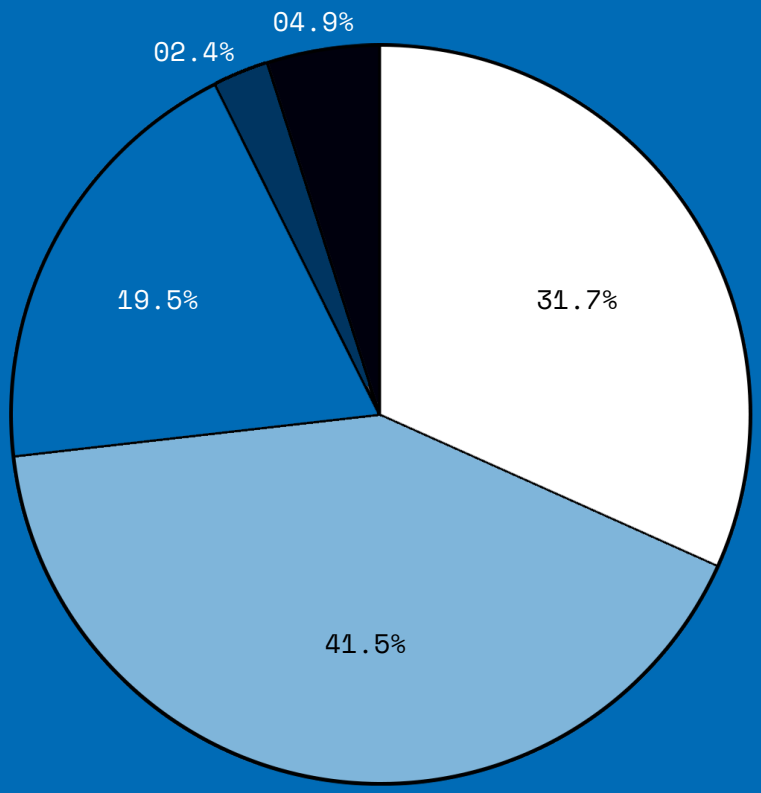


Figure 1: *Hours per day spent using software.*

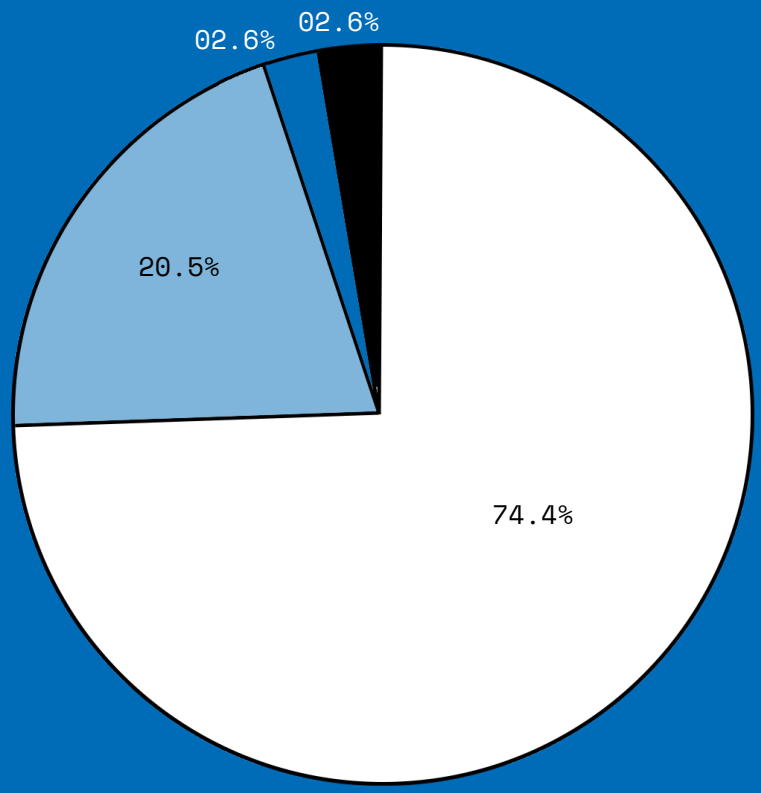


Figure 2: *Hours per day spent developing software.*

Software in Research

Our surveys and interviews overwhelmingly reaffirmed the important role that software development plays in research at the IGB. Of the responses to the survey question, "How many hours per day do you spend using scientific software developed by others," shown in Figure (1), 68.3% spent at least two hours per day on average using scientific software, and of those, 26.8% spent four or more. This survey data is reinforced by the interviews conducted with the IGB staff, all of whom used software for applications including modeling, data processing and analysis, and data visualization. For example, Subject 1 (a doctoral student) said that although they do not create their own software, they regularly use software created by others.

However, Subject 1 is the exception, not the rule at the IGB. The majority of the subjects interviewed regularly wrote scripts for data analysis, visualization, and other automation, and a few developed larger software packages for more sophisticated tasks. This may seem to contradict the results of our survey, which suggest that researchers spend very little of their time programming. As shown in Figure (2), 74.4% of those surveyed spent less than 2 hours a day developing software. The small percentage (4.9%) that spent more than 4 hours a day developing software were not researchers but modelers or software developers. This apparent contradiction is likely because many of those who responded with less than two hours interpreted the question to be referring to larger software packages and not the small scripts that they wrote. Many of our interviewees commented on how they used a patchwork approach to software development, using portions of code taken from other works to put into theirs. It is possible that some survey respondents did not interpret this type of programming to qualify as software development and as such did not respond with an accurate value to the question posed in Figure 2.. Also, spending less than two hours per day writing software is not necessarily indicative of insignificant programming. Researchers are not professional software developers and programming, while very important to their work, is not their primary focus. As such, they may only write scripts when analyzing their data but what they write is still a significant aspect of their work. Subject 2 of our interviews described this well, stating "the process of developing code and the methods behind it are research in itself."

Subject #	Subject Position	Group	Interview Date
01	Postdoc Scientist	IGB	April 07, 2020
02	Doctoral Student	IGB	April 17, 2020
03	Doctoral Student	IGB	April 17, 2020
04	Doctoral Student	IGB	April 20, 2020
05	Technician	IGB	April 23, 2020
06	Modeler	IGB	April 24, 2020
07	Engineer	IGB	April 27, 2020
08	Research Assistant	IGB	April 27, 2020
09	Head of Student Programs	HUA	April 30, 2020
10	Professor	N/A	April 29, 2020
11	Assistant Professor	HUA	May 01, 2020
12	Research Librarian	IGB	April 15, 2020

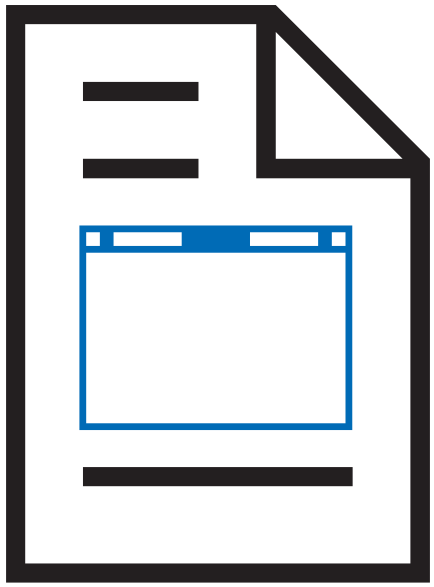
Table 2: *Interviews by subject number.*

Personal Programming Experience

Both creation and usage of software are crucial aspects of work at the IGB. However, our data highlighted a lack of significant software development training amongst staff, though this is of no fault of the researchers themselves. Our survey of the IGB staff showed that most respondents (53.6%) were primarily self-taught, with some individuals having attended workshops to strengthen their skills, as shown by Figure (4). This is supported by our interviews, in which most stated they had learned their software skills on their own, using online resources, talking with coworkers, and independent trial and error. Many of our interviews showed that student researchers had learned the basics of software development while at university, but from then on were completely self-taught. Some stated that they had taken a couple introductory courses in college, but only a small number directly involved with software development had received a comprehensive computer science education, as evidenced by the 14.6% of survey respondents who answered with formal courses in Figure 4. It should be noted that 24.4% of respondents did not have any software engineering experience. We attributed this to respondents who did not work with software in their research or simply utilized the skills of their coworkers to meet their programming requirements. This too was reflected in our interviews, where subject 2 stated that they did not program themselves but did use preexisting code.

The specific programming languages employed by staff also reveal the level of experience within the IGB. As shown by Figure 3 and corroborated by our interviews, the overwhelming majority of respondents programmed in R and Python, at 80.5% and 34.1% respectively. Both languages are free and easily attainable, both of which ease access for anyone to download or create software written in them. They are, however, not the most robust and fast executing free languages on the market. These languages are capable of doing just about anything you need them to, but will not do it as fast as a language with a higher barrier for entry like C#. However, their high accessibility makes R or Python easier to teach yourself, and a community of coworkers who know R and Python greatly increases the likelihood that a researcher will also learn those languages (Subject 6, Subject 7).

The level of software development training within IGB staff is also shown in individuals' level of exposure to version control and documentation. As shown in Figure (5), 46.3% of those surveyed had no experience at all with version control and documentation software, and only 24.5% were familiar to extremely familiar with these tools. Although many researchers at the IGB may not have direct experience with these tools, they are aware they exist and lack the time or resources to learn how to use them. For example, subject 1 says they often come back to previous projects wishing they had used better documentation and version control, but at the time had no reason to do so. However, many of those interviewed recognized the value of these tools and expressed a desire to learn how to use them. It is important to note that researchers should not be expected to be professional software developers. That is not their job or where their skills lie, and inexperience does not in any way detract from their professional abilities.



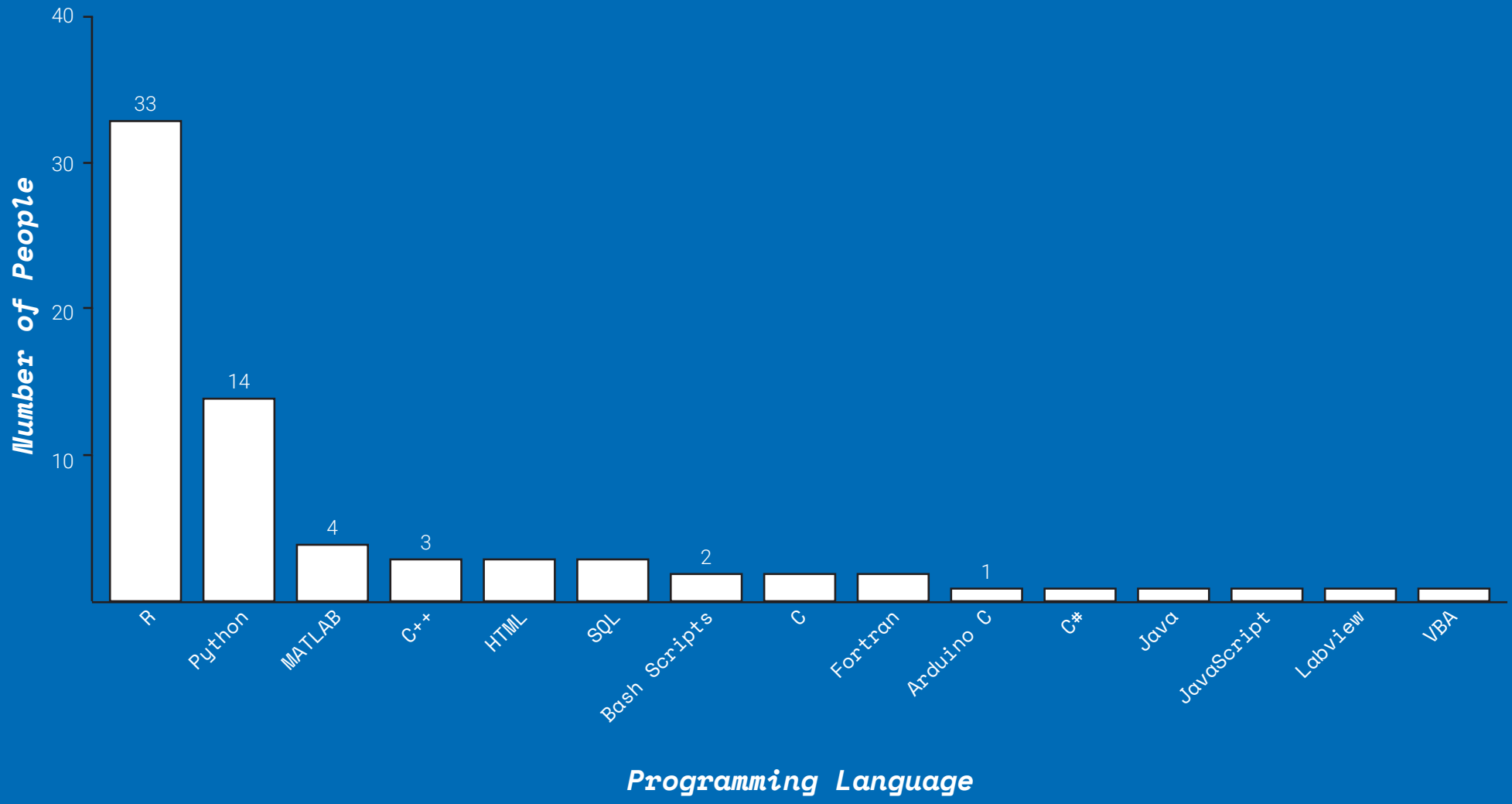


Figure 3: *Programming languages used by survey participants.*

Institutional Support for Open Source

Institutional support for software publication and sharing of code was one of the aspects we had identified as important to the success of open source implementation during our initial research. [Something like:] The influence of institutional level support underpinned multiple survey responses and interviews. For the effective dissemination/ proliferation of open source practices, institutions must provide the resources for effective sharing and open source publication. Individuals may learn how to program and use software, as discussed above; however, institutional policy/leadership plays a major role in providing greater access to code and software created by others and better support for open source publication.

It is crucial to have access to tools that facilitate sharing and collaboration. Several interview subjects stated that having access to code used in published work would be helpful in their own work. As described by subject 1, seeing different people's code helps to develop one's own programming skills even if it is in a different language. Looking at and using code produced by others can show other means of approaching a problem or functions within a programming language that were otherwise unknown. Subject 1 also felt that access to coworkers with more software development experience was helpful for learning and producing better code, and that having a workgroup with a mix of researchers and modelers was beneficial. This sentiment was reflected throughout our interviews. This access also lets a researcher pick out specific bits of code to repurpose into their own work. Researchers at the IGB often build on existing research, data, and most importantly the code used to reach said results. Without access to these programs, they are often forced to recreate the unpublished work of another scientist. Better access to the code produced by their colleagues would both enable them to create better code themselves and would cut down on work that has already been done. A couple services for code sharing and publication do exist (a bitbucket server and a GitHub service), but our interviews showed that a very limited number of people knew about these services. Subject 6 was the only individual who had any significant knowledge of the Bitbucket service, and they did not use it. Both services exist independent of each other and fill similar roles by creating a space where code and programs can be stored for access by other IGB staff or for personal reference at a later date. These services are the responsibility of the IGB to make available to their staff. This availability does not only mean creating and maintaining the services but also advertising and incentivizing their use. The wide-scale adoption of these services is important to their use. Interviewees consistently commented that a community of staff using these services would be one of the biggest forces driving them to also use these services; moreover, if properly supported and used, our interviewees would be very willing to use these services.

Institutional support for open source publication is crucial to their successful implementation. Our interviews described how the current system for crediting publications does not include software or hardware projects that may have been published outside of formal journals. The current system credits researchers based upon the impact factor of their work on a .5-5 scale, with publications in more significant journals like Nature being awarded greater credit. If a program is published to a service like GitHub, it does not qualify for credit under this system. This is despite the very large user base of such services and a potentially very high impact. Publication in GitHub also comes with certain implicit community requirements in regard to documentation that could be used to increase the quality of documentation produced. Many of those we interviewed, such as Subject 4, did not feel their code was up perceived standards for public sharing. Normalizing publication in these informal repositories helps make public sharing of code more accepted. Subsequently, it can then be used to slowly make sharing of less comprehensive software and other code more accepted too.

Institutional support can also be used to help overcome some of the obstacles of personal programming experience previously discussed. As stated previously, researchers are not professional software developers; however, resources can be provided on an institutional level to train researchers on tools and techniques that would be beneficial to their work. This could include workshops on the most regularly used languages R and Python as well as workshops on what constitutes effective documentation. Our interviews showed that if these services were provided, they would be very well received.

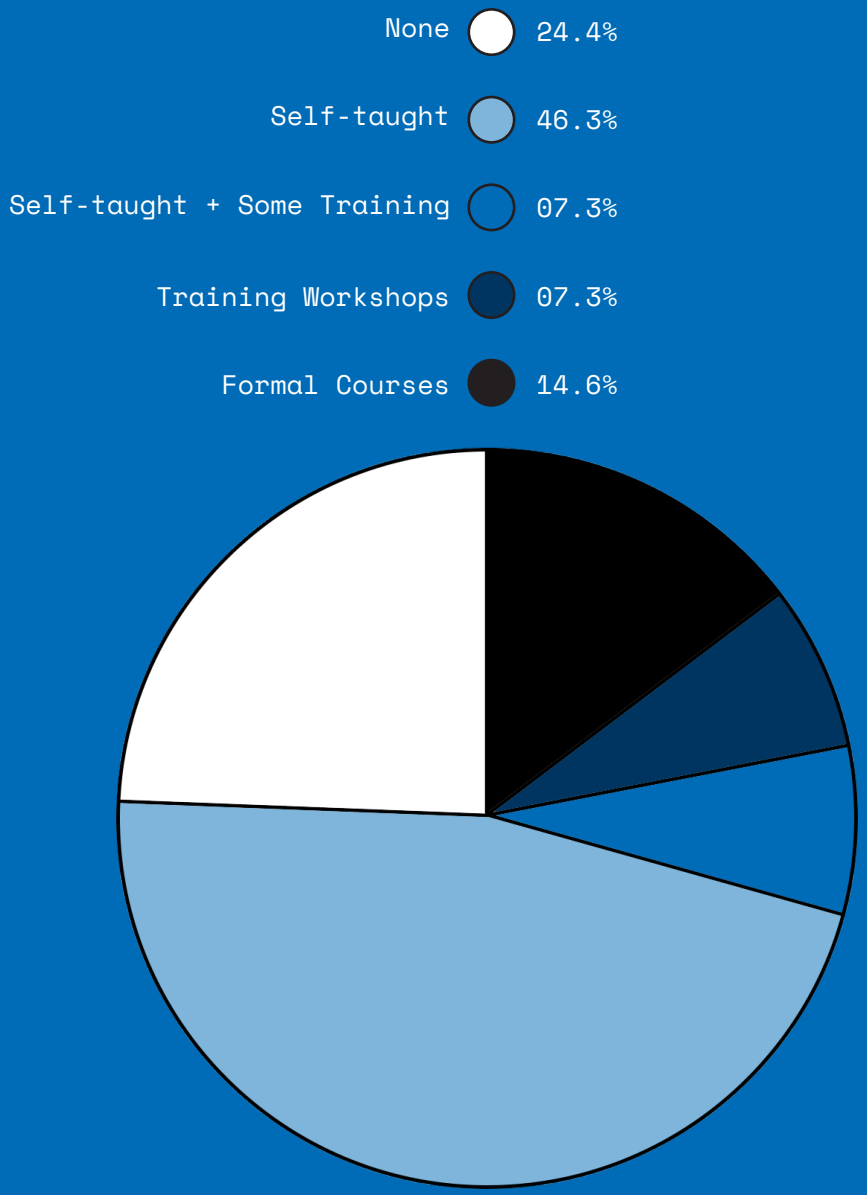


Figure 4: *Participant software training.*

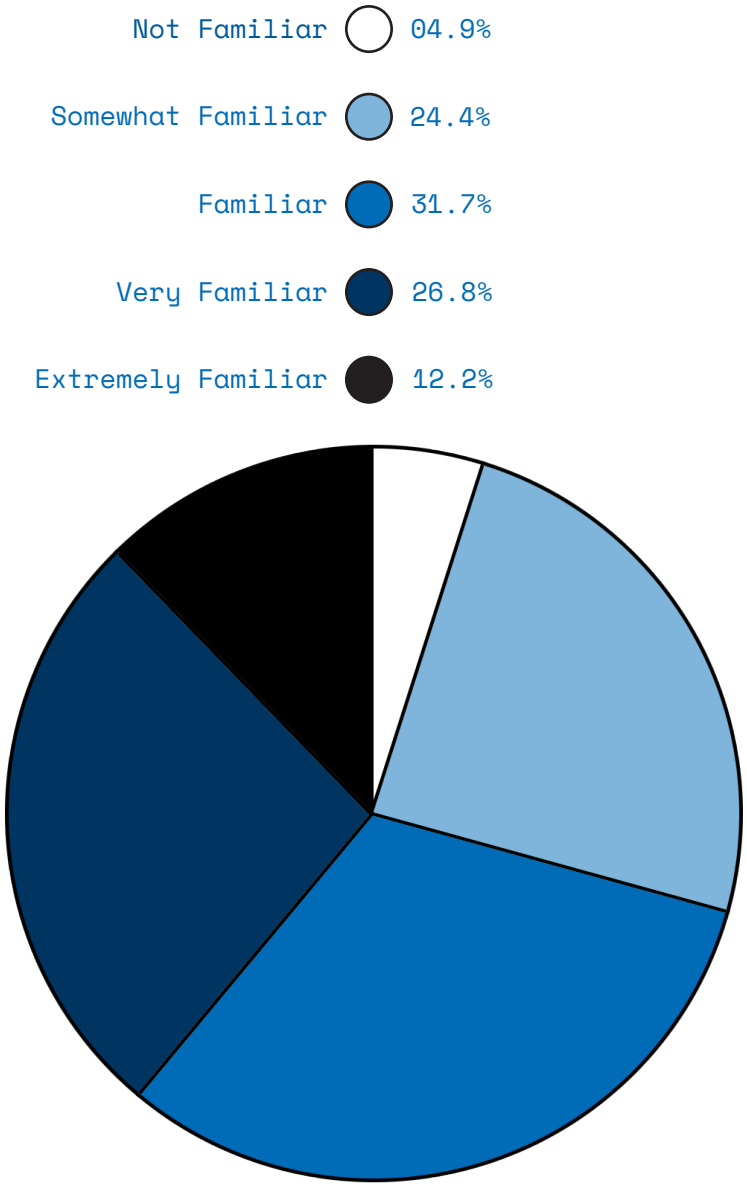
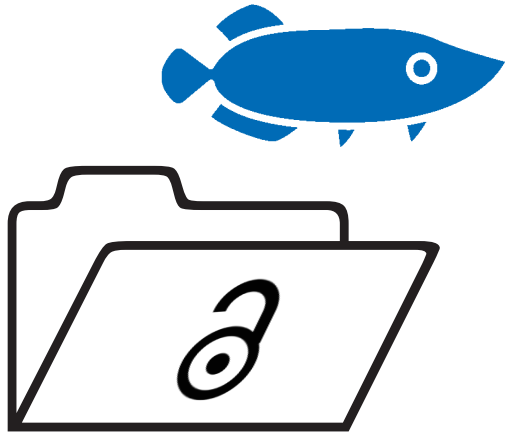


Figure 5: *Participant experience with version control and documentation software.*

Open Source Can Work

Effective implementation of open source publication is possible, as shown by the survey and interviews we conducted outside of the IGB with digital humanities researchers. In that field, open source is the standard practice, with all survey respondents stating something to that effect. One of our survey responses stated, "I think open is, perhaps to a fault, an understood characteristic of DH [Digital Humanities] work and anyone not making their materials open looks a little out of place." Our interviews reflected this as well, with one of our interviewees describing how he teaches his students to make a blog post at least twice a semester in order to both maintain the open source nature of their work and generate interest in it. This interviewee also commented on how publication and sharing in non-journal areas results in a larger audience as more people would read a blog or see a program on GitHub than would read a scientific journal. Generating interest in one's work was a key factor noted by our interviews as it helps with circulation and can give immediate feedback. Our interviewees noted how the success of these open source practices were in part the results of how they were normalized within the field. Success within the IGB hinges on normalizing these practices in order to create a larger enough user base to maintain them. As noted by the digital humanities researchers, assumptions about what constitutes open source can be detrimental to the success of work. They noted how there is a perceived lack of quality from open source work, regardless of the actual quality of what has been created. This perceived, although false, lack of quality can in some cases turn people away from a work. Countering this perception, as noted by our digital humanities interviews, is important to the acceptance of open source publication. Public collaboration is an important aspect of the digital humanities and this public interaction is not exclusive to this field, with the fields the IGB works in also being able to leverage it and successfully implement these open source practices.



Concluding Remarks and Recommendations

Overall, our data showed the importance of institutional support for open source, the need for coding resources on a personal level, and how open source has been effectively implemented in another field. With these takeaways in mind, we produced a policy brief for the IGB administration on policy recommendations (Supplementary Materials, Appendix 1). The objective of these policy recommendations was threefold: to better support those looking to publish their software in open source formats, to encourage greater internal sharing of produced code, and to highlight potential programs that would boost the level of programming skill at the IGB. In addition to this policy brief, we also produced a flyer for distribution within the IGB that both highlighted the significance of open source publication and gave brief details on tools and techniques that were available for use in software development (Supplementary Materials, Appendix 2).

The administrative policy brief focused on workshops that could be provided such as programming in R/Python, usage of version control software, and documentation. This brief also aimed to address how to better use the existing infrastructure to support open source publication by recommending revamping the Bitbucket and GitHub services as well as extending the crediting system to non-conventional publication methods based on the work's analytics.

The flyer that was created for distribution at the IGB explained the potential benefits of open source software publication and outlined some tools that were available to help with publication, documentation, and version control. Amongst other things, it highlighted that open source publication increased the reproducibility of their work and encouraged a collaborative environment. This infographic was designed to be small and easy to read in order to ensure that people would in fact read it rather than get turned away by a large volume of text.

Implementation of the recommended policies and distribution of the infographic will increase awareness and usage of open source publication, foster a more collaborative research environment, and increase the quality of software produced as a part of research at the IGB. It will take time for these changes to occur, but they will serve as a basis for increasing the acceptance of open source in the mainstream of science.

References and Attributions

Crouch, S., Hong, N. C., Hettrick, S., Jackson, M., Pawlik, A., Sufi, S., … Parsons, M. (2013). The Software Sustainability Institute: Changing Research Software Attitudes and Practices. Computing in Science & Engineering, 15(6), 74–80. doi: 10.1109/mcse.2013.133

European Commission. What is Horizon 2020. European Commission Horizon 2020. https://ec.europa.eu/programmes/horizon2020/en/what-horizon-2020.

European Commission. (2017). Commission Staff Working Document Interim Evaluation of Horizon 2020. http://ec.europa.eu/research/evaluations/pdf/archive/h2020_evaluations/swd(2017)221-interim_evaluation-h2020.pdf#view=fit&pagemode=none

European Commission. (2019). H2020 Programme AGA - Annotated Model Grant Agreement. https://ec.europa.eu/research/participants/data/ref/h2020/grants_manual/amga/h2020-amga_en.pdf

Hannay, J. E., Macleod, C., Singer, J., Langtangen, H. P., Pfahl, D., & Wilson, G. (2009). How do scientists develop and use scientific software? 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering. doi: 10.1109/secse.2009.5069155

Holdren, J. P. (2013). MEMORANDUM FOR THE HEADS OF EXECUTIVE DEPARTMENTS: Increasing Access to the Results of Federally Funded Scientific Research. United States Office of Science and Technology Policy. https://web.archive.org/web/20170112020320/http://www.whitehouse.gov/sites/default/files/microsites/ostp/ostp_public_access_memo_2013.pdf

Ince, D. C., Hatton, L., & Graham-Cumming, J. (2012). The case for open computer programs. Nature, 482(7386), 485–488. doi: 10.1038/nature10836

J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl and G. Wilson, "How do scientists develop and use scientific software?," 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering, Vancouver, BC, 2009, pp. 1-8.

Leibniz-Institute of Freshwater Ecology and Inland Fisheries (IGB). (2019, December). Open Access Policy of Leibniz-Institute of Freshwater Ecology and Inland Fisheries. Leibniz-Institute of Freshwater Ecology and Inland Fisheries. https://www.igb-berlin.de/sites/default/files/media-files/download-files/2020-01-13_OA-Policy_EN.pdf.

Pianosi, F., Sarrazin, F., Wagener, T. (2019). How successfully is open-source research software adopted? Results and implications of surveying the users of a sensitivity analysis toolbox. Environmental Modelling & Software, 124.

United States Government Accountability Office. 2019. Additional Actions Needed to Improve Public Access to Research Results. https://www.gao.gov/assets/710/702847.pdf

Fish by Dron Desain from the Noun Project