Crowdsourcing scientific software documentation: a case study of the NumPy documentation project

Aleksandra Pawlik, University of Manchester, Judith Segal, Helen Sharp and Marian Petre, The Open University

Abstract

Without good documentation, even the most sophisticated and efficient scientific software is difficult to use and maintain. However, due to lack of time, resources and incentives, scientists who develop software for other scientists are not keen on writing documentation. Scientific software packages that are freely available and shared within scientific communities often suffer from poor documentation, sometimes making them unusable. This paper considers documentation crowdsourcing as a way to address the issue. Using the NumPy documentation project as a case study, we discuss how to leverage the knowledge about software that resides within the user community. We look at technical infrastructure, community engagement and motivation. We consider benefits, such as expanding the community, and challenges, including maintaining the level of commitment. The conclusion suggests preliminary guidelines for those thinking about documentation crowdsourcing.

Keywords: Computer-supported collaborative work, Software engineering for Internet projects, Knowledge sharing

1. Introduction

Research in many, if not most, scientific disciplines requires scientists to not only use but also to develop software to advance their research. Some software applications used in science are highly specialized open source packages developed and maintained by scientists themselves (for example, cross-disciplinary NumPy and SciPy[1], rOpenSci[2], or more discipline-specific, like Madagascar[3]). Ideally, documentation should be an inseparable part of software development and maintenance. Without documentation, software becomes difficult to use and to maintain. However, despite its importance, scientific software documentation is often incomplete or absent (Segal, 2007, Nguyen-Hoan, 2010).

The issue of poor documentation is not exclusive to scientific software. Professional software developers are not fond of writing documentation (Lethbridge *et al*., 2003), and scientists seem to be no different (Segal, 2005). The reason behind a lack of documentation is usually simple: priorities. Writing documentation is more boring and less satisfying than coding. What adds to the issue is that, while scientist-developers may enjoy software development *per se*, this activity remains secondary to their main goal, which is advancing their research (Segal, 2007). Documentation has even lower priority. And resources follow priorities. While in commercially-developed scientific software there is likely to be a team and budget dedicated to writing documentation, in open source projects for scientific software most likely there is neither.

How does software find its way into use despite poor documentation? Communities develop around a shared need and the software tools that address it, and the knowledge about the software resides within the community. The users and developers introduce each other to the software, and they share advice on fora, mailing lists, and internet communicators or in informal conversations over coffee (Pawlik *et al*., 2012). Scientists put a considerable trust in their peers' recommendations and opinions (Joppa *et al*., 2013).

[1] http://www.scipy.org
[2] http://ropensci.org
[3] http://www.ahay.org

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

It seems that there is potential within the communities that develop around scientific software to compensate for shortcomings documentation. The question is: how to harness this potential and channel the informal, unstructured and often volatile exchange of knowledge about the software into standardised and reusable documentation?

To answer this question, we conducted a case study of the NumPy documentation project in which the community members (consisting primarily of users) documented the software they used. We explored how the community organised and handled the process, what benefits it brought to them and what challenges they faced.

2. Background on NumPy

The NumPy library is a "fundamental library needed for scientific computing with Python"[4] providing efficient implementation of multi-dimensional arrays together with a wide range of mathematical functions to operate on them. NumPy interfaces with well developed, stable and widely used linear algebra packages: BLAS and LAPACK, as well as fast Fourier transformation package FFTPACK. Hence both the use and the development of NumPy require advanced knowledge and understanding of numerical methods and their applications.

NumPy was preceded by three other projects of related scope. The first one was Matrix Object Package for Python developed by Jim Fulton from 1994 onwards. The second one was Numeric (or Numerical Python) whose most of the initial code base was written by Jim Hugunin with contributions from the community known as the "Python Matrix-SIG"[5]. The third one was Numarray (a specialized fork of Numerical Python) whose development was led by Perry Greenfield, Rick White and Todd Miller. In 2005 Travis Oliphant started the unification work bringing together Numeric and Numarray. For a short period, this new package was called scipy.core, until it was eventually renamed NumPy, in order to distinguish it both from the previous two projects and from SciPy. It should be noted that the name 'SciPy' now also refers to " Python-based ecosystem of open-source software for mathematics, science, and engineering"[6] and to the community using it. 'SciPy' also denotes the community's conferences, workshops and meetings.

The growing NumPy code base was scarcely documented. Poor documentation made the software extremely difficult to use, especially for new users. At the same time, the scientist-developers did not have time to write documentation. They were more interested in developing the code itself rather than documenting it. Several members of the community came up with the idea that the *users* could write documentation. It's not clear who exactly was the first to suggest crowdsourcing documentation. As one of the interviewees put it: "*it just happened organically*".

Python supports the docstrings, specifically-formatted string literals, similar to code comments, which can be used for displaying help information about a particular piece of code. It is a common convention to use docstrings as documentation reference in Python projects. Many docstrings for methods in NumPy library were incomplete or missing and so writing them became the main task for the documentation project. In our study we focused on the process of editing the existing and writing new docstrings.

3. Data collection and analysis

[4] http://www.scipy.org/about.html (last accessed 27th February 2014)
[5] https://mail.python.org/pipermail/matrix-sig/ (last accessed 21st June 2014)
[6] http://www.scipy.org/ (last accessed 21st June 2014)

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

We collected the data for this case study from four different sources. The first source was the publicly available archives of three mailing lists: numpy-discussion, scipy-dev and scipy-user. Using the mailing list search engine we filtered out 2,230 posts between the beginning of the archives (2000 and 2001 – depending on the list) and April 2012 including words stemming from "doc*". The filtered posts were then manually screened to select the messages which discussed the documentation process. This eventually left us with 477 posts which were then analysed in detail.

The second source of data was semi-structured interviews with nine members of the SciPy community, all of them key stakeholders involved in the documentation project. The interviews lasted between 30 and 90 minutes, most of them took just under one hour.

The third source of data was progress reports on the SciPy Documentation Project published in the SciPy Conference Proceedings (Harrington, 2008; Harrington and Goldsmith, 2009), a technical overview paper published in the SciPy Conference Proceedings (Van der Walt, 2008) and the project website (docs.scipy.org). s

The fourth source of data was logs from the server which hosted the infrastructure developed for the purpose of crowdsourcing documentation. The logs from the server included quantitative information such as the number of contributor accounts registered on the system, the total number of edited, proofed and revised words in different time periods and so on.

4. Observations

The analysis of the data revealed a number of aspects related to the organisation, challenges and benefits of documentation crowdsourcing in the scientific software context. These findings are not only a collection of observations about documentation crowdsourcing but should be informative for those considering or already implementing such a process. The outcomes of this case study discussed below help to inform guidance, and provide insights into potential caveats and benefits which may not seem obvious or expected beforehand.

4.1 How to organise documentation crowdsourcing

Organising NumPy documentation crowdsourcing included dealing with both technical and human aspects. The former was related to developing and maintaining the infrastructure, together with designing and implementing standards for writing documents. The latter included finding a project champion who both had sufficient technical knowledge and was able to engage the community, and manage the contributors' efforts. There was also a financial aspect, which had its influence, in particular at the beginning of the project.

4.1.1 Infrastructure

The key considerations for setting up the documentation crowdsourcing process were to allow a wide community to access the code base while ensuring that the source code remained safe from any potential damaging changes. The NumPy source code was kept under version control (using Subversion) at the time. The code was freely available for anyone to use, but only a selected group of core developers had write-access to the repository. This meant that any changes in the code (including docstrings) made and submitted by those who did not have write access had to be actively reviewed by the developers and then applied as patches. This obviously was a barrier for crowdsourcing documentation, as it would only add more work for the developers who were already at the limit of their capacity.

On the other hand, giving write access to the repository to anyone who wanted to write documentation was too risky. The risk of purposely-malicious injections to the source code was estimated to be rather low. However, increasing a number of people who had direct write access would also increase the probability of introducing accidental errors. In addition, the contributors who could write documentation would be limited to those who were able to use version control. Therefore, the main goal was to develop an infrastructure for

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

writing documentation which kept the code safe but was easy to use by a potentially large number of contributors.

The infrastructure developed for the documentation project was eventually inspired by two tools, which were developed independently: one was a wiki, and the second was "LiveDocs". The advantage of the wiki was that everyone could register and start editing it. The disadvantage was that it was not connected with the code and hence the changes made on the wiki did not have any effect on the docstrings. In addition, the wiki was not automatically updated with any changes to the source code (including the docstrings). LiveDocs, setup by the original creator of NumPy, ran on a server and provided hierarchically-indexed documentation using docstrings. The advantage of this system was that it showed the most up-to-date docstrings. The main disadvantage was that it was 'read-only'.

The solution was to combine wiki and LiveDocs functionalities allowing as many contributors as possible to edit docstrings and at the same time providing full control over the write access to the code repository. This documentation system was developed by one of the core NumPy developers with help from a few others. The documentation editor[7] allowed two-way documentation updating: the changes made by the contributors in the editor were turned into a patch and applied to the code back in the Subversion repository and the documentation in the editor itself was updated with any changes.

The documentation editor supported the workflow, which assumed that a docstring may have more than one status: *needs editing, being written, needs review, needs work (reviewed), reviewed (needs proof), proofed* and *unimportant*. The docstrings that went through a number of iterative changes and that were classified as correct made it into a patch. This removed the burden from the developers of checking the docstrings before applying them to the code.

4.1.2 Stylistic guidelines

Even though some bits of NumPy code were documented using docstrings, there was no comprehensive or recommended documentation standard or style. However, for crowdsourcing such guidelines were essential and needed to be clearly laid out and explained to the contributors. The discussion about the standards and guidelines for contributors took place mainly on the mailing list. All subscribers were openly encouraged to express their opinions.

Initially, one of the NumPy core developers proposed that the guidelines follow the style in which he had written docstrings for several years. According to him the docstrings should include: function inputs, outputs, algorithm, authors (of the given piece of code), examples, tables, references and additional notes. The standard was then in a plain text file "HOW_TO_DOCUMENT.txt" which was made publicly available online[8].

The standards were also reflected in the infrastructure of the documentation system. First, following a suggestion of one of the community members, a professional technical writer, all of the documentation that needed writing was split into small bits. As the technical writer argued on the mailing list: *"A short document has less opportunity to be poorly structured than a long one. A collection of short documents can be assembled into a helpful organization as it evolves."* The documentation system was designed to allow the contributors to edit a small section of documentation, and the sections were also divided according to the standards proposed earlier: parameters, returned values, examples and so on. It is interesting that, according to one of the developers interviewed, this standard was then picked up in other scientific Python packages.

4.1.3 Funding

One of the factors that sped up the start of the documentation project was the funding which one of the NumPy users secured for employing a full-time documentation writer, who was also supposed to lead and

[7] Still available at http://docs.scipy.org/numpy/Front%20Page/ (last accessed on 27th Feb 2014)
[8] The updated version is available at https://github.com/numpy/numpy/blob/master/doc/HOWTO_DOCUMENT.rst.txt (last accessed on 27th Feb 2014)

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

coordinate the efforts of other contributors. The user, a professor at one of the US universities that provided the funding, had wanted his students to use NumPy during one of the courses. Due to poor documentation, the students struggled and could not complete the assignments. Frustrated with this situation, the professor managed to allocate some of the funding he had available to employ one of the NumPy developers for a year to write documentation. It quickly became clear that the documentation task was much too large for one person to complete. The documentation writer's role then expanded to encompass managing and leading the work done by the others.

Eventually two writer-leaders were employed, one after the other, with about a couple of months gap between them, for the total time of about two and a half years. The money came from a grant that did not specifically mention documentation writing. The flexibility of the grant made it possible to spend some of the funds on the documentation. However, it also meant that the funding was limited and that the documentation project needed a sustainability model.

4.1.4 Project leaders

Both project leaders were employed full-time to write and coordinate the documentation. Both were experienced developers. However, while the first was one of the core NumPy developers, the second did not contribute code to this particular library. The in-depth knowledge of implementation details of NumPy was useful but not essential for documenting the code.

The task of documenting NumPy was too big for one person. Writing documentation every day all day was also described by one of the leaders as incredibly mundane and simply boring at some stages. Crowdsourcing the documentation lifted some of that burden from the full-time documentation writers but also added other responsibilities. They monitored and reported back to the community on the progress with documentation writing; motivated the contributors to carry on with their work; and proposed changes and improvements for the process or the infrastructure.

4.1.5 Engaging the community

Engaging many members of the user community was a high priority and was essential to make the project work. Calls for contributors were sent to the mailing list regularly, encouraging everyone to write documentation. All community members were welcomed to contribute documentation: from advanced users who might have been developers themselves, to beginners who only recently started using the library and had limited experience as developers. In one of the early emails, one of the developers explicitly encouraged a user who started writing documentation: "*Being a 'newbie' is maybe the best time to write documentation – while you still know what new users need to know, and how to explain it to them simply. Don't feel shy to contribute.*"

The community members were not only encouraged to write documentation but also to provide feedback on the whole project. The mailing lists were the place to discuss the infrastructure as well as the standards and formats. The emphasis on community spirit and a "shared ownership" of the documentation was evident from the beginning of the project.

The custom-made infrastructure for collaborative documentation writing was a low-level entry point. It was easy to set up an account and then to write bits of documentation. The registration (setting up an account) did not automatically give permission to edit the contents; first, the registered users had to send an email to the developers' mailing list providing their login, and they would automatically be given permission by one of the administrators of the documentation system. There was no background check, and everyone who registered was given write-access to the docstrings in the system. The only reason behind the extra step was to prevent potential spammers from setting up accounts. Figure 1 shows the cumulative number of accounts registered in the system over the period of the case study. It should be noted that these numbers do not equal the number of actual contributors, as some accounts were never activated, and some people registered and activated their accounts but never actually contributed to the project.

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

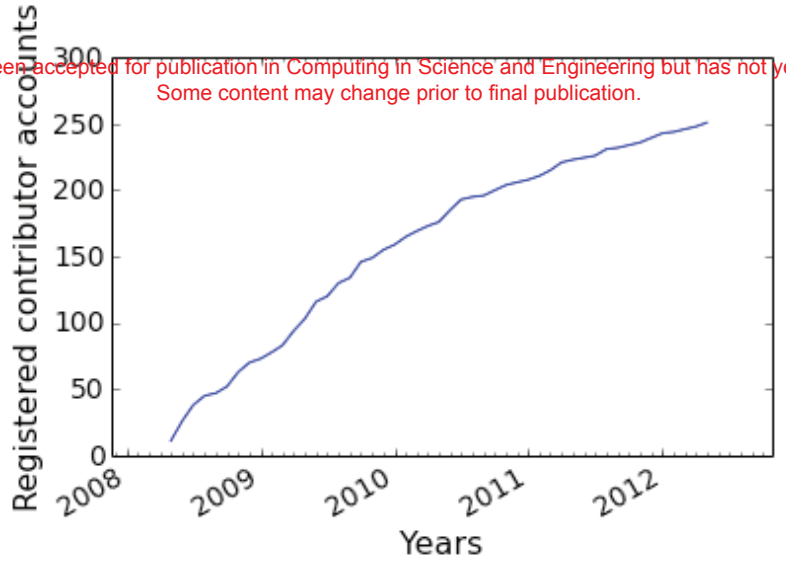

Figure 1 The cumulative number of registered contributor accounts in the documentation system.

In the interview, the professor who secured the funding for employing the full-time documentation writer admitted that he was rather sceptical about whether the community would engage in the project. He actually made bets about the number of people who would sign up: "*I think the threshold was about 30 people and I told them that I don't think that we are going to have that many signups for the first 2 months. I think the bet was on ice-cream. I lost that bet. Lots of people signed up. The response was very good. Now we've just got hundreds of people out there.* "

4.2 Challenges with documentation crowdsourcing

Inevitably, the process of documentation crowdsourcing did not proceed without some challenges. largely to do with review, expectations and momentum.

4.2.1 Endorsing stylistic guidelines and workflow

The discussions on the mailing list about the standards for documentation were lengthy but eventually led to a consensus and guidelines for contributors. In order to ensure that the contributed documentation actually met the standards the docstrings were checked by *reviewers.* Reviewers were registered users of the documentation system whose main role was to revise edited docstrings before they were approved to be applied on the source code in the repository.

It turned out that the reviewing and proofing steps in documentation crowdsourcing were difficult to complete. Our interviewees were not sure why this was so: perhaps the role and responsibilities of reviewers were not clearly defined, or the reviewing itself was not an interesting thing to do.

4.2.2 Expectations concerning the contributors

Even though the main assumption of crowdsourcing documentation was to harness the community's potential and knowledge, there were some concerns about opening up to contributions from anyone. These concerns were raised on the mailing list during the discussions about the standards, the format and the infrastructure for documentation crowdsourcing.

The main argument was that there should be some minimum set of skills and knowledge that should be expected of the potential contributors. Some community members were questioning on the mailing list whether contributors with, for example, unable to use some tools commonly thought of as "basic" for scientists (such as LaTeX) could be trusted with documenting complex libraries like NumPy. However, the

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

prevailing view was that the knowledge of the tools for word processing should not be the criterion for selecting the contributors. The decision was made that anyone could make a contribution, and that further down the line these contributions would be checked for quality before they were included.

4.3.2 Keeping up the momentum

The project kicked off with considerable enthusiasm from the community. The contributors (also referred to as "editors") started registering and writing documentation in the online editor. The challenge was to keeping up the momentum, ensuring that the initial enthusiasm did not wear out quickly.

An early idea was the Documentation Marathon[9] announced on the mailing list in May 2008. The name 'Marathon' referred to 'coding sprints' (short, few-day meetings when developers get together to write code, usually to develop a particular feature or functionality). 'Marathon'also clearly indicated that it would be an ongoing effort lasting months rather than days (unlike typical coding sprints). One of the goals of the Documentation Marathon was: "*Produce complete docstrings for all numpy functions and as much of scipy as possible*," and "*Check everything into the sources by 1 August 2008 so that the Packaging Team can cut a release and have it available in time for Fall 2008 classes*."

Another idea was to award all contributors who edited at least 1,000 words with a T-shirt. The names of contributors were also included in a progress report which was published and presented at the annual SciPy conference (Harrington, 2008). Both were tokens of appreciation and recognition for the work done for the community. The progress report presented a comprehensive summary of how much had been achieved.

The threshold for the T-shirt award was set arbitrarily, but it turned out to be relatively good approximation of the cut-off point for the most engaged contributors. Almost 70% of contributors wrote or edited less than 1,000 words. The biggest contributions, over 10,000 words were actually made by only 7% of editors (which included the 2 full-time documentation writers). Figure 2 shows the amounts of words either written or edited in the documentation system by the registered contributors.

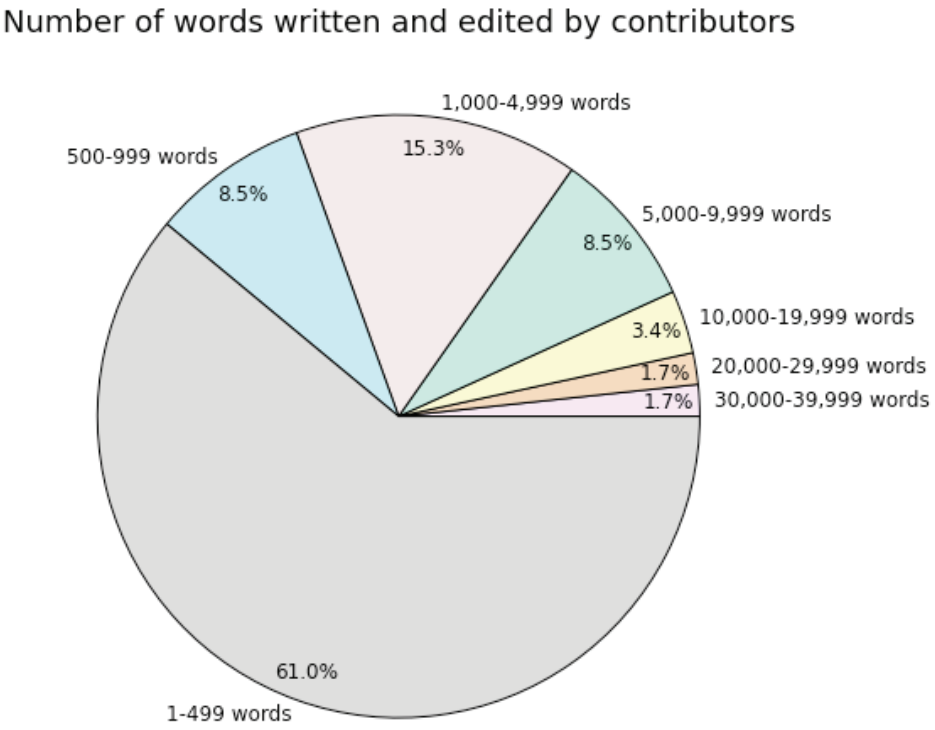

Figure 2 Number of words edited by the contributors in the documentation system.

[9] http://wiki.scipy.org/Developer_Zone/DocMarathon2008 (last accessed on 27th February 2014)

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

4.3 Benefits of documentation crowdsourcing

It may seem that the primary benefit of documentation crowdsourcing is obvious: more complete documentation. However, our case study showed that documentation crowdsourcing brought other benefits: expanding the community, and setting up and promoting standards. These benefits were perceived as being at least as important as improving the documentation.

4.3.1 Documentation improvement

When the documentation project started around mid-2008, the total number of words in the NumPy reference pages was 8,658. By July 2012, this number reached 140,000 words. Although more does not necessarily equal better, the increase in documentation did not happen in a random process. The documentation was written to guidelines and standards, and the new documentation, or at least a majority of it, was discussed and reviewed.

All interviewed participants of the study said that the documentation improved immensely. As evidence, they cited the growing number of NumPy users. These new users started using the library on their own, and did not have to rely on someone guiding them through the code and explaining how to use it. Such guidance was almost a necessity before the documentation was crowdsourced.

However, evaluating whether documentation crowdsourcing actually improved NumPy documentation is not straightforward. It should be noted that the increased uptake of NumPy could be also related to the increasing popularity of Python in the scientific community in general. It is possible that, as more scientists write their software in Python, NumPy is used more often. Evaluating the quality of the documentation was not the goal of our study, but the insights into documentation crowdsourcing presented here can help develop measures for such a study.

4.3.2 Expanding the community

Documentation crowdsourcing helped to expand the SciPy community. The improved documentation meant that the library was easier to access, which in turn enabled more new users to use NumPy in the software they develop. However, it was not only the new users who made the addition to the community.

Writing documentation empowered those who, before the documentation project was launched, felt they were unable to contribute. They did not have sufficient skills and experience in software development that would allow them to write and review the code. Despite the fact that they were willing to help, they were unable to do so. As one of the interviewees put it: "*There is a human dynamic in that which is very appealing. You want to get engaged. Obviously there are people who simply want to get things for themselves but most decent human beings when they are treated with generosity, there is a natural response to at least return a little bit. But the problem is, if in order to return in kind, you have to cross very high barriers, you may end up giving up. I think the point of creating a system like this was 'Let's give anyone who wants to give back in a way that is important which is helping to improve the documentation, let's make it very easy for them to give back. Let's lower the unnecessary barriers so that they can easily contribute back*"

Interestingly one of the interviewees said that writing documentation gave her enough confidence to start developing the source code. She was a part of the community before the documentation project, and she used the library, but she felt that her skills were not sufficient to develop and commit software. The work she did whilst documenting, and the feedback she received, encouraged her to join one of the software projects related to NumPy as a developer.

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

5. Discussion: Is documentation crowdsourcing a good fit for scientific software?

In scientific software, documentation typically needs to cover two things: information about the implementation and the science it represents. The software users need to have the relevant knowledge about both of these areas. The case of NumPy showed that the user community represented different levels of knowledge about the code which they documented, in a continuum from advanced user/developers, to naïve users with very limited knowledge of implementation. For the former, the library was a 'white box' with mechanisms they understood well, and for the latter it was more of a 'black box', as what concerned them was the science underlying the methods they used. The case study demonstrated that users across the continuum were capable of making useful contributions to the project.

The documentation editor based on the wiki system lowered the entry barrier, allowing contributors to write documentation almost immediately without needing to use complex tools for access and editing. Other examples, such as public version-control hosting-services such as GitHub[10] (where NumPy is currently hosted) or BitBucket[11], provide additional evidence that, by removing the overheads of learning *how* to make contributions, software projects may benefit from the knowledge of wider communities**.** Even small "drive-by" contributions add up to bigger improvements and may potentially lead to new collaborations (Ram, 2013).

The users who contributed to documentation were scientists, mostly working in an academic environment. Their everyday jobs required explaining and instructing others. These academic skills could potentially help them write documentation. On the other hand, academics' writing training focuses mainly on writing academic papers in which they construct clear arguments, set up hypotheses and draw conclusions. Software documentation is a different form a writing, regimented in terms of style, format and content; it does not allow for individual approaches or solutions.

The main issue with software documentation writing done by scientists in academia is the lack of recognition. Academic reputation and careers are built on publications (Howison & Herbsleb, 2011). Developing scientific software, even software that is widely popular and useful for the community, brings scientists few career benefits apart from making them more employable outside of academia (Parr, 2013). Recognition for developing scientific software tends to be low in academia, let alone recognition for documentation writing. Documenting does not add anything to an academic resume, and it is not a skill valued in research institutions. Therefore, the rewards to contributors arise from the community to which they contribute, or from their intrinsic satisfaction with contribution.

6. Guidelines: What to consider when planning documentation crowdsourcing?

The outcomes of our study provide a basis for forming guidelines that are useful to take into account when considering crowdsourcing documentation.

Technical infrastructure should support both the community members who want to contribute and the community champions who manage the collaborative effort. The infrastructure may reinforce the inclusive or exclusive nature of the crowdsourcing project. The simpler the infrastructure is to use, the more people are likely to use it and get involved. On the other hand, if use of the infrastructure requires knowledge of some specific tools and skills, it may help to filter the community for the desired contributors who have the selected skill set.

[10] http://github.com/
[11] http://bitbucket.org/

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

Stylistic guidelines help with writing consistent documentation enhancing its usability. The guidelines should clearly state what to do and how to do it. For software documentation, the guidelines should focus specifically on the information that needs to be captured, the level of detail, the language, the order of information, the format and so on. The stylistic guidelines should be readily available for anyone wanting to contribute.

Clear and sustainable workflow turns crowdsourcing from a chaotic process into a process resembling a system of gears that move together with precision. The workflow can help arrange the tasks that are involved in documentation into a sequence that prevents clashes and deadlocks. The workflow help new contributors joining the project to 'get up to speed'.

The project leader coordinates the process of crowdsourcing. The leader's role is not limited to executing the workflow. The coordinator may need to use a 'divide and conquer' approach, breaking larger and more complex goals into self-contained tasks, which then are assigned to the community members. The leader may need to set up and revise the milestones. One of the most challenging responsibilities may be keeping up the momentum, ensuring that, when the novelty and enthusiasm wears out, the collaborative effort does not die down.

Motivation and recognition is much needed to keep the project running. Showing the contributors the outcomes of their effort helps them believe that what they do makes sense and has value. It is about making the return of investment visible. Regular updates on the progress and tokens of recognition show the contributors that their investment is not wasted and does not go unnoticed.

7. Conclusions

Documentation crowdsourcing may in some cases be a solution to the challenge of providing documentation for open source scientific software. However, it is a solution that requires investment in its implementation. While it may lessen at least some of the burden of documenting for the scientist-developers, it adds the burden of planning, organising and running crowdsourcing for those driving the process. It also has its own challenges, especially in terms of maintaining community momentum long enough to establish an effective body of documentation. Seeing the effects of their work, the contributors are likely to be motivated to continue their efforts. But intrinsic motivation must be complemented with explicit recognition. Recognition for writing good documentation – not just by the community but also by academia and employers – is as much needed as recognition for developing good scientific software.

Crowdsourcing documentation can have significant advantages *in addition* to the generation of necessary documentation. Removing technical barriers from the process of writing documentation helps build the user community and can help users grow into developers.

References

[1] L. Nguyen-Hoan; S. Flint & R. Sankaranarayana. A survey of scientific software development *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement,* **2010**, 1-10

[2] J. Segal. Some problems of professional end user developers *Visual Languages and Human-Centric Computing, 2007. VL/HCC 2007. IEEE Symposium on,* **2007**, 111-118

[3] A. Pawlik; J. Segal; H. Sharp & M. Petre. Documentation practices in scientific software development *Cooperative and Human Aspects of Software Engineering (CHASE), 2012 5th International Workshop on,* **2012**, 113-119

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

[4] T. Lethbridge; J. Singer & A. Forward. How software engineers use documentation: the state of the practice *Software, IEEE,* **2003**, *20*, 35 - 39

[5] J. Segal. When Software Engineers Met Research Scientists: A Case Study *Empirical Software Engineering,* **2005**, *10*, 517-536

[6] L. N. Joppa, G. McInerny, R. Harper, L. Salido, K. Takeda, K. O'Hara, D. Gavaghan, and S. Emmott. Troubling trends in scientific software use. *Science*, 340(6134):814–815, **2013**.

[7] J. Harrington. The SciPy documentation project. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 7th Python in Science Conference*, pages 33 – 35, Pasadena, CA USA, August 19-24 **2008**.

[8] J. Harrington and D. Goldsmith. Progress report: NumPy and SciPy documentation in 2009. In G. Varoquaux, T. Vaught, and J. Millman, editors, *Proceedings of the 8th Python in Science Conference*, pages 84–87, Pasadena, CA, August 18-23 **2009**.

[9] S. J. Van der Walt. The SciPy documentation project (technical overview). In G. Varoquaux, T. Vaught, and J. Millman, editors, Proceedings of the 7th Python in Science Conference, pages 27 – 28, Pasadena, CA USA, **2008**

[10] K. Ram. git can facilitate greater reproducibility and increased transparency in science, *Source Code for Biology and Medicine,* **2013**, *8*, 7

[11] J. Howison & J. D. Herbsleb. Scientific software production: incentives and collaboration *Proceedings of the ACM 2011 conference on Computer supported cooperative work, ACM,* **2011**, 513-522

[12] C. Parr. Save your work give software engineers a career track. *Times Higher Education*, 15 August **2013**.

About the authors:

Aleksandra Pawlik works at the University of Manchester, UK, in the Software Sustainability Institute where she leads the training activities and supports community development. Her research interests focus around the practices of scientific software development. She promotes and teaches computing skills essential for researchers from different domains helping them be more innovative and productive in their work
Contact details:
Room 1.17 Kilburn Building, University of Manchester
Oxford Road, M13 9PL, Manchester, United Kingdom
e-mail: aleksandra.pawlik@manchester.ac.uk
phone: + 44 (0) 161 275 0218

Judith Segal is a visiting research fellow at the Open University, UK. Her research interests focus on the organisational and contextual aspects of scientific software development.
Contact details:
Computing and Communications Department
The Open University
Walton Hall, Milton Keynes,
MK7 6AA, United Kingdom

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE

e-mail: judith.segal@open.ac.uk
fax: + 44 (0) 1908 65 21 40

Helen Sharp is Professor of Software Engineering at the Open University, UK. Her research focuses on the study of professional software practice with a particular focus on human and social aspects of software development. She has been conducting qualitative studies of software practice since the early 1990s and is very active in both the software engineering and interaction design (HCI) communities. Helen is joint author of one of the leading textbooks on Interaction Design (id-book.com) now in its fourth edition. She is associate editor for Transactions on Software Engineering, a member of the Advisory Board for IEEE Software, and reviews for many journals and conferences. She is a member of ACM, IEEE Computer Society and the British Computer Society, and Fellow of the HEA.
Contact details:
Computing and Communications Department
The Open University
Walton Hall, Milton Keynes,
MK7 6AA, United Kingdom
e-mail: h.c.sharp@open.ac.uk
fax: + 44 (0) 1908 65 21 40

Marian Petre is a Professor of Computing at the Open University in the UK. She held a Royal Society Wolfson Research Merit Award, in recognition of her research on the nature of expertise in software design, and her empirical studies on reasoning and representation in software development. She has a PhD in Computer Science from University College London and a BA in Psycholinguistics from Swarthmore College.
Contact details:
Computing and Communications Department
The Open University
Walton Hall, Milton Keynes,
MK7 6AA, United Kingdom
e-mail: m.petre@open.ac.uk
fax: + 44 (0) 1908 65 21 40

Digital Object Indentifier 10.1109/MCSE.2014.93 1070-9924/$26.00 2014 IEEE