Good Practices for Research Software Documentation
(the slides are available under “Presentation Materials” in the above URL)
Date: February 10, 2021
Presented by: Stephan Druskat (German Aerospace Center (DLR)) and Sorrel Harriet (Leeds Trinity University)

Q. Can you detail some impact metrics you touched upon as it relates to funding? Documentation websites publish the number of downloads, number of unique IP addresses per month, or rank among packages in an ecosystem to indicate their popularity.
What are the (1) ideal and (2) current practises for evaluating the impact of research software packages? Often a proxy for software impact is finding a paper that accompanies the software and then using their metrics, citations, or impact. How does one minimize bias in evaluating research software popularity for funders?

A. Ideally, document how people can cite your software (this can be included in the README, for example, or in a citation file in the Citation File Format). Daniel Katz discusses software citation principles in a previous webinar in this series (17, 2018). The publisher usually counts your citation references. Write about how your software has been used. GitHub metrics such as stars, downloads, forks etc. should be taken with a pinch of salt: not only can they mean different things to different people, but they can also be obtained fraudulently (this thread on Stack Exchange is quite insightful - notice that ‘quality of the documentation’ is mentioned as a more reliable indicator of sustained popularity.) Might have to show your software has been cited as software and not as the scientific paper.

Q. Would you version your documentation like you version code?

A. Depending on the type of documentation and the tools you are using, it may naturally be under version control alongside your code (for example, if you are using an ‘in code’ documentation framework such as Doxygen/JavaDoc, you would be able to recreate earlier versions of the documentation from earlier revisions of the source code).
You can also keep user guides and developer/maintenance documentation in the same version control system if it is useful for users and developers. This way, you cannot only recreate earlier versions of the documentation when you retrieve an earlier revision of the code, it may also make it easier to ship documentation with your software.
Alternatively, in the case of user guides and installation guides you can have your documentation under a separate system of version control. You will then need a system for synchronising your documentation and software version numbers.
There are lots of different tools available for creating and managing software documentation (e.g. Read the Docs, GitHub pages, etc.) Whatever tools you choose, the most important thing is to make sure your documentation is well organised so that people can find and update it when they need to.

Q. How to get others on your team to prioritize documentation?

A. I believe that decisions about documentation need to be taken collectively, as it is important that the approach is consistent. I would suggest that documentation needs to be discussed early on in the project process and 'budgeted for', but if it is something you are trying to do retrospectively, maybe the first thing to do is initiate a discussion with your team about why you feel it is important. You could highlight some of the benefits of well documented software for them and for their research. If you sense reluctance or there is a lack of time, agree some achievable documentation goals with your team.

Q. There could be two types of documentation for a research project. One could be for the software itself (if it's complicated) and one for using it and more research-oriented. How to address both in a project effectively?

A. There isn't a straightforward answer to this...it depends on the situation. For documenting the software itself, you can start with the basics (i.e. README file, self-documenting code.) For more complex software, you could use an 'in code' documentation generator such as Doxygen or JavaDoc (there are various tools available for whatever language you're using.) For other types of documentation (user guides, installation guides, project documentation etc.) you really need to consider the intended audience and purpose of the documentation. I would tend to treat these as 2 separate things. Again, there are many different tools you can use to create and manage user documentation. Often it comes down to personal preference. For example, do you and your collaborators prefer markdown or Latex? Does your team use SVN or git? etc.

Q. A lot of these points about documentation also apply to testing (e.g. it becomes part of the code). In my experience, it is hard enough for most people to write tests when writing software. Is there a risk of putting too much burden on developers not trained in software engineering, like domain researchers?

A. The short answer is yes, I do believe it is possible to over-burden inexperienced developers and I have even experienced this myself. For example, Test Driven Development (TDD) has many great benefits but it is hard for inexperienced developers to practice consistently without support. The same could be said of more advanced documentation tools. I think you need to make the decisions that are right for your team, taking into account the complexity of the software you are writing, the relative experience-levels of developers in your team, project requirements, time and budget etc. You may need to start with more modest ambitions in order for them to be applied consistently. If more advanced approaches are deemed important, consider how you will support less experienced developers. For example, could you encourage frequent paired programming within the team?

Q. Some projects use the network to relay usage metrics: how reasonable is this and what metrics are respectful of privacy?

A. I'm not entirely sure what the question is asking or how it relates to documentation. If the question is about usage metrics that are relayed back to the service provider (for example, error logging etc.), I think you probably need to find out what information is collected/not collected by the service provider. It should be possible to find this out. I would expect, in most cases, that the metrics they collect are respectful of privacy, but if you are at all concerned it would seem prudent to check. Unless there is a genuine cause for concern, I would probably not recommend disabling these features as they serve a legitimate purpose, i.e., usage metrics can be very valuable to developers to pivot future developments in the direction of actual usage.

Q. When it comes to complex modular software, the hierarchical documentation system helps. Any recommendation on hierarchical documentation best practices?

A. I'm not sure what exactly "hierarchical documentation system" refers to here. In cases where documentation needs to follow ISO 9001, teams should be embedded in an institution which also employs experts on these standards, or has dedicated QA personnel. If it means to refer to documentation produced in a standardized software engineering process (e.g. as outlined in SWEBOK), again there should be professional software engineers available to teams working with such a process, who should guide developers through it. I believe that in practice, few research software projects have large enough team structures to follow these procedures (or even some agile procedures which work with a large number of roles). In any case, complex modular software should (and will usually) be developed together with software engineering professionals or Research Software Engineers.

Q. What do you suggest to measure software impact with respect to the actual *usage* of your software?

A. If it's research software, citations are the only "hard currency" impact metric (I know of) which shows that software has actually had impact (on research). Practice is still suboptimal, but picking up (see also FORCE11 Software Citation Implementation Working Group, and the Software Citation Principles paper).

Q. Documentation is a bit like writing. Sometimes when you change one thing in one place, you might need to read the whole thing to see if the change affects something else because somewhere else might refer to the one thing that you change. Is it possible to make documentation modular so that the ripple effect does not cause us a huge amount of work to maintain?

A. There are many documentation frameworks which can help with this. You may wish to investigate documentation tooling for whatever language you are working in. On a more abstract level, try and follow the DRY principle ("Don't Repeat Yourself") in documentation as well. In short: don't spread descriptions across, e.g., the README and the user guide (Instead, link from one to the other). For example, there should be exactly one place where usage of a feature

is described for users, and exactly one place which explains how new modules can be developed to work with existing modules, etc.

Q. How do I write documentation so I don't need to change it too often?

A. Think about the requirements/needs of the documentation and where in the software development life-cycle you are doing it. For example, it may not be necessary to document in a lot of detail early on in a project. Using a framework may also help keep your documentation modular and easier to maintain. Another way to reduce effort is to collaborate with others. Documentation tooling can also make it easier for others to contribute to your documentation. If time is lacking, you might also consider a 'You Ain't Gonna Need It' approach, by waiting for users or contributors to raise issues before addressing them through your documentation. Another very different way of doing this is to follow a "tutorial-driven development" process, where only what's documented (in a tutorial for users) gets implemented.