Collaborations Workshop 2019 (CW19) #CollabW19 2019-04-01 to 2019-04-03

What are best practices for software documentation - DSR6-CW2CC

Reporter

Stephan Druskat - stephan.druskat@dlr.de (@gmail.com in uConfly)

Participants

Stephan (stephan.druskat@dlr.de), Tyler Whitehouse, Alessandro Felder (afelder@rvc.ac.uk), Robin Long (robin.long1@hotmail.co.uk), Sharif Salah - participated in discussion but didn't contribute to the blog post, so not an author of the post (sharif.salah@gmail.com, sharifsalah@google.com), Sorrel Harriet, Benjamin Lee

Notes from the discussion

- *Two audiences (or more): teams and users and …*
- *Research software has special requirements*
    - *Users = developers*
    - *Find out what it does*
    - *Find out how it works*
    - *Find out how it interacts with other software/data*
- *Minimum (changes as software matures):*
    - *Code comments*
    - *Comprehensive README*
- *Threshold for requirements in terms of doc types*
- *Conceptual docs v how-tos v reference docs*
    - *Have a quickstart guide (with links that go deeper)*
- *Checklist*

    - *Start with what the software does*
    - *Where is the documentation*
        - *E.g., user & dev documentation external to GitHub, but README contains nothing*
    - *What features does the documentation need?*
        - *Free?*
        - *PRs?*
        - *Code review support?*
        - *Collaborative editing*
    - *Tooling?*
    - *Name the target group of the documentation*
        - *Be gentle about this, i.e., name prerequisites ("you should know some Python and understand the basics of how x works")*
    - *First elements?*
        - *Decide on the target group and select the proper type of document*
    - *How much of the documentation "tree" can be automated (e.g. with a cookie cutter)?*
        - *Templating for leafs of the tree*
- *Main point: You want to think about what and how you document before you start coding.*
    - *This can be something very lean, and shouldn't put people off*
- *Motivation for documentation:*
    - *Cite the software: How-tos, tutorials user documentation*
    - *Collaboration: Dev docs*
    - *Impact: Low-level how to uses*

Speed Blog

Please use the area below to draft the speed blog. Consult https://www.software.ac.uk/speed-blogging-and-tips-writing-one for information, tips and examples.

What are best practices for research software documentation?

Introduction

Good documentation is a fundamental aspect of research software. It influences how easy-to-use, extendable, and by extension how sustainable, a piece of software is. In this blog post, we are interested in addressing issues surrounding good documentation of research software and how they can be approached in a general sense, that may be applicable to a wide research software engineering audience.

Documentation is a broad topic and how best to approach it can depend on many factors. These can include the field the research software is used in, the needs and experience level of the user, the duration and complexity of the project, etc.

Take away advice

There are two pieces of concrete advice that you may take into account in terms of documentation for your research software:

1. **Think about the documentation of your software before you start coding.**

   This does not have to be a painfully long list of documents to write or dozens of UML diagrams, it can be a lean structure for, e.g., a README document that can grow (and extend into other formats) as you proceed with the implementation of your software. Refer to the documentation decision tree for some examples!

2. **Think about your motivation for documenting the software.**

   The motivation for your documentation may have an impact on what you want to document. If you want your software to be cited, you may want to include a how-to-cite section as well as comprehensive user documentation. JOSS, for example, sets certain standards for software that is published in the journal, which may be a good starting point. Also, check out the Software Citation Principles paper to get an overview of what you want to take into account. If you want people to collaborate with you on your software, you certainly need developer documentation and contributing guidelines. If you are after impact, beginner-level how-tos make onboarding and growing your user base a lot easier.

Things worth thinking about

Audiences

The first thing to know is that software documentation has different audiences. It may target different groups - e.g., end users, developers, maintainers, management. For research software, users and developers may be the same people. Also, research software users need to know exactly what the software does to assess suitability for their research, be able to find out how it works to make sure it works correctly, and be able to assess whether it will work with their data and other pieces of software they use. All of these groups require different forms of documentation. Knowing your audience will help you identify what types of documentation you want to create. Reference documentation, comprehensive user guides, tutorials and getting started snippets are useful for end users. Developers probably need code comments, API documentation and perhaps requirements, architecture and design documentation. Maintainers may need a bit of both user and developer documentation, but definitely need to know about development workflows, social documentation such as codes of conduct, LICENSE and CITATION files, etc.
Also, documentation may take into account the level of experience of the audience. Documentation targeted at first year students will certainly look different to documentation for RSEs or computational researchers. This aspect should be considered for your documentation as well.

Types

There are different types of documentation, which can be broadly split into conceptual documentation, hands-on documentation, and reference documentation. Conceptual documentation usually contains higher-level views of the software, e.g., its requirements, design specifications, architecture. Hands-on documentation features how-tos and getting started documents, user guides, etc. Reference documentation includes, e.g., API documentation, build and release engineering documents, and code comments. Different software projects may want to use different documentation types. While there is probably no real benefit in including design specifications for a short Python script, you may want to include them for a multi-component system.

Why is software documentation important to research?

With the context introduced above, we can now state the documentation problem we address in more detail: The primary purpose of research is to convey knowledge, in most cases software is a tool to utilise or extract that knowledge. Without user documentation, the software cannot be used and thus the knowledge cannot be utilised. Without developer documentation the tools and thus the

scope of knowledge cannot be expanded. Without tutorials, the exploitation of the knowledge cannot be realised. This overlays and covers some of the political demands on research such as citations, collaboration and impact.

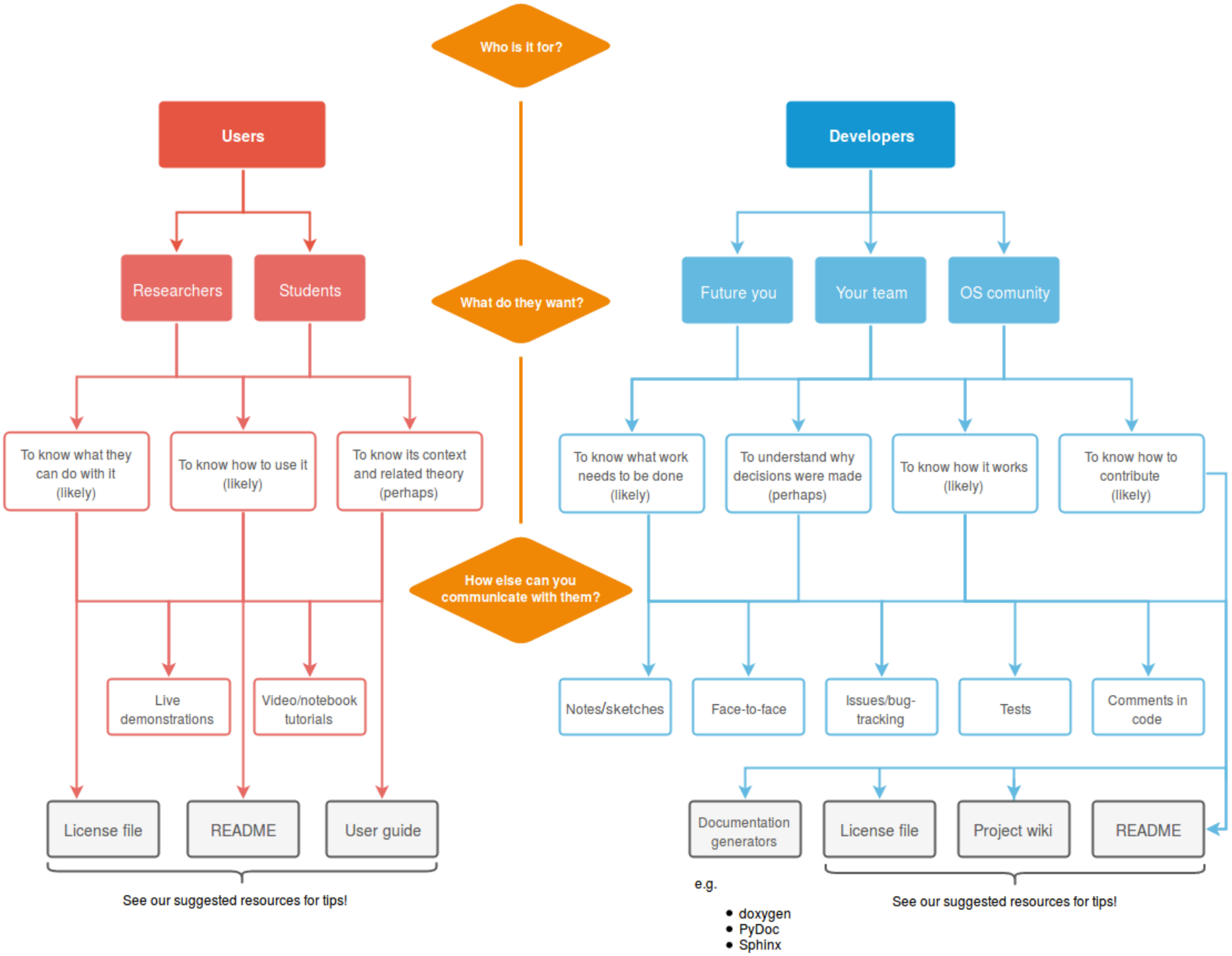
Good user documentation allows collaborators to use the software, this can then lead to citations of the software and recognition of work. Tutorials and how-to's enhance this, but also allow the expansion of impact. Whilst researchers in a similar field may know the software, and can run it with user documentation, a tutorial gets them up-to-speed quicker. At the same time, it allows non-researchers, or non-experts in the area, to use the software (and thus research and knowledge). The final type of documentation is developer documentation, which allows potential collaborators or bug fixers to know how to modify the software. At a fundamental level it can better allow users to understand and use the software, but also makes it clear to those that would like to expand the software how to do that. This can result in a higher impact for the software as more features are added, or more collaboration and citations.

Getting started

To help make navigating these waters a little smoother, we suggest starting with 3 simple questions:

- Who is it for?
- What do they want?
- How else could I communicate with them?

We have visualised this process as a ‘documentation decision tree’ which can serve as a starting point when considering what to document. Since there is no ‘one size fits all’ for documentation, the tree won’t provide you with a definitive answer, but we hope it may prompt you to think more carefully about what you choose to document and how.


Who is it for?
Users
Researchers
Students
What do they want?
To know what they can do with it (likely)
To know how to use it (likely)
To know its context and related theory (perhaps)
How else can you communicate with them?
Live demonstrations
Video/notebook tutorials
License file
README
User guide
See our suggested resources for tips!
Developers
Future you
Your team
OS comunity
To know what work needs to be done (likely)
To understand why decisions were made (perhaps)
To know how it works (likely)
To know how to contribute (likely)
Notes/sketches
Face-to-face
Issues/bug-tracking
Tests
Comments in code
Documentation generators
License file
Project wiki
README
e.g.
doxygen
PyDoc
Sphinx
See our suggested resources for tips!

About the Tree

The tree tells us the first question to ask when deciding what to document is: *"Who is this for?"* This will be closely followed by: *“what do they want from this?”* For example, if the documentation is aimed at other developers, ask yourself what they need to know about your software. If you can't imagine other people using or contributing to your software at this point in time, then consider the documentation is directed at 'future you' e.g. "what will *I* need to remember about this project in 6 months time?"

When considering what others might want from your documentation, you're also going to have to factor in how much time you have to deliver it, and how long you expect the project to be maintained. There's no point in spending a week writing detailed design specifications if you've only got 2 weeks to deliver the project! Try to gauge the relative importance of any documentation requirements you come up with.

The third thing to consider is, *'could I communicate this to my audience some other way?'* Rarely is a page of text going to be more effective then sitting down with a team member and discussing the problem face-to-face. If it's possible to use other forms of communication, consider doing so. Depending on the scale and complexity of the project, it might be possible to communicate a lot of the details via the code base itself. For example, if you're adopting a TDD approach, the tests can serve as a functional specification without need for additional documentation. Comments and docstrings can also negate the need for excessive documentation - if done well. Not only that, docstrings can be leveraged by auto documentation generation packages.

Once you've identified the 'must haves', ask yourself, 'must they have it *now*?' You can avoid wasting effort by documenting incrementally. For example, if you're producing a user guide or project wiki, why not keep it lean to begin with, but build on it as the needs of the audience become clearer. If possible, allow others to contribute to it. In agile, the general approach is to keep documentation minimal throughout the planning and implementation stages, ramping it up a bit towards the end when the requirements have stabilised. While the ‘document late’ approach might not work for all projects, it’s still worth asking these questions before getting started.

Then, once you get behind the keyboard, there are a raft of tips and recommendations for how to write effective documentation. We've suggested a few below to help you get started. Feel free to suggest others! [not sure if readers can comment?!]

Suggested resources

- README file template: https://gist.github.com/PurpleBooth/109311bb0361f32d87a2 (Billie Thompson)
- General tips: https://www.writethedocs.org/guide/writing/beginners-guide-to-docs/ (Eric Holscher)
- Technical writing tips: https://medium.com/@Archanachowty/a-quick-writing-checklist-for-technical-writers-eadc1b25e04e (Arcana Chowty)
- Hemingway Editor (plain writing assistant): http://www.hemingwayapp.com/ (Adam and Ben Long)
- Agile documentation: http://www.agilemodeling.com/essays/agileDocumentation.htm (Scott Ambler)