Good practices for research software documentation

Stephan Druskat
Sorrel Harriet

CONTEXT

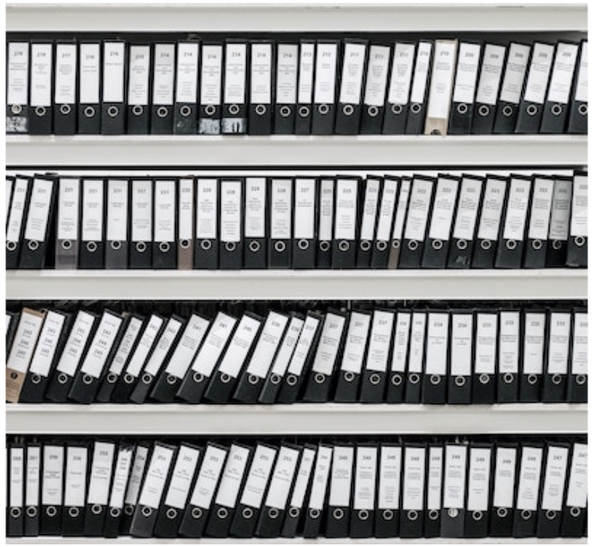
This talk is based partially on a blog post from the Software Sustainability Institute's Collaborations Workshop 2019. We would like to thank our co-authors for the blog post: Tyler Whitehouse, Alessandro Felder, and Benjamin Lee.

Software Sustainability Institute

Menu

What are best practices for research software documentation?

Posted by s.aragon on 21 June 2019 - 9:37am



By Stephan Druskat, Tyler Whitehouse, Alessandro Felder, Sorrel Harriet, Benjamin Lee

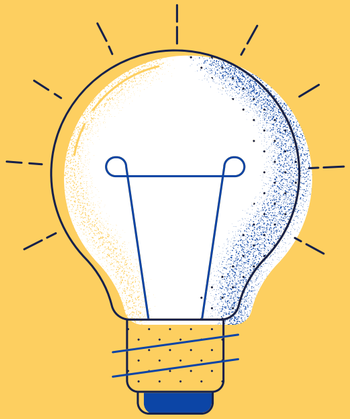
*This post is part of the **CW19 speed blog posts series**.*

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



WHAT YOU'LL LEARN

- Understand the importance of software documentation.
- Learn about different aspects of software documentation.
- Make informed choices about how to approach software documentation.
- Reflect on the role of software documentation within the software process.

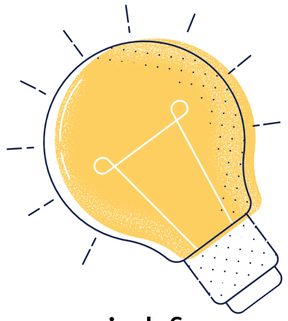
THE IMPORTANCE OF RESEARCH SOFTWARE DOCUMENTATION

Documentation is important for software.

- Ease of (re-) use
- Continued development
- Sustainability

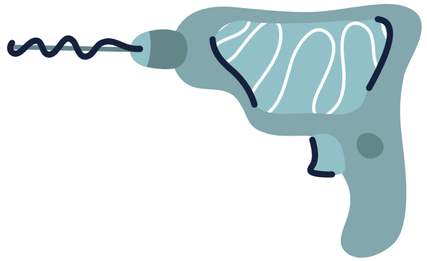
Documentation is important for research.

- Utilization of knowledge
- Expansion of knowledge
- Expansion of impact



crucial for understanding

a productivity tool

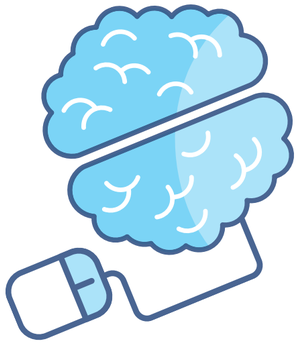


helps build a user base

SOFTWARE DOCUMENTATION ...

good for your career

not that tedious



good scientific practice

TAKE AWAY ADVICE

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

Think about your motivation for documenting the software.

When?
Who for?
What?
How?
Where?

WHEN TO DOCUMENT?

Now!

Documentation is part of the code.

A development process can determine this.
(Sorrel will talk about that later.)

WHO TO DOCUMENT FOR?

Always document for yourself.
In addition, document for your target audience.

Consider levels of experience.

Keep in mind that the target audience may change over time.

WHAT TO DOCUMENT?

Depends on target audience and motivation.

	Code	Maintenance	User docs	Developer docs	Metadata	Project docs
Yourself	✓	✓	?	?	?	
Users	?	?	✓	?	✓	?
Collaborators	✓	✓	✓	✓	✓	?
Evaluators	?	✓	✓	?	✓	✓
Funders		?	✓		✓	✓

DOCUMENTATION TYPES

CODE DOCUMENTATION

Semantic identifiers, comments, API, engineering, dependencies, requirements

MAINTENANCE DOCUMENTATION

How to build, release, review code, publish

USER DOCUMENTATION

How to get, run, use the software; parameters, data model, etc.; license

DEVELOPER DOCUMENTATION

How to contribute, contribution templates (issues, pull/merge requests)

METADATA

Software metadata (CodeMeta), Citation File (CFF), "references" (dependencies)

PROJECT DOCUMENTATION

Rationale, teams, governance, community (contact, code of conduct)

HOW TO DOCUMENT?

Conceptual documentation

Higher-level views of the software:

- Requirements, design specifications, architecture
- Project documentation

Hands-on documentation

- How-tos, getting started documents, user guides
- Templates for issues, pull/merge requests
- Contribution guidelines, codes of conduct

Reference documentation

- API documentation, build/release engineering documents, code comments
- Tests
- Metadata

HOW TO DOCUMENT IN PRACTICE?

Baseline

- README + "self-documenting code"
- Code comments where useful

Always human-readable

Machine-readable where useful or necessary

- Metadata
- Tests
- Doc strings

Use available technology/tooling, e.g.

- Simple markup languages (Markdown, RST)
- Static site / API doc generators
- Static analysis tools (style, completeness)

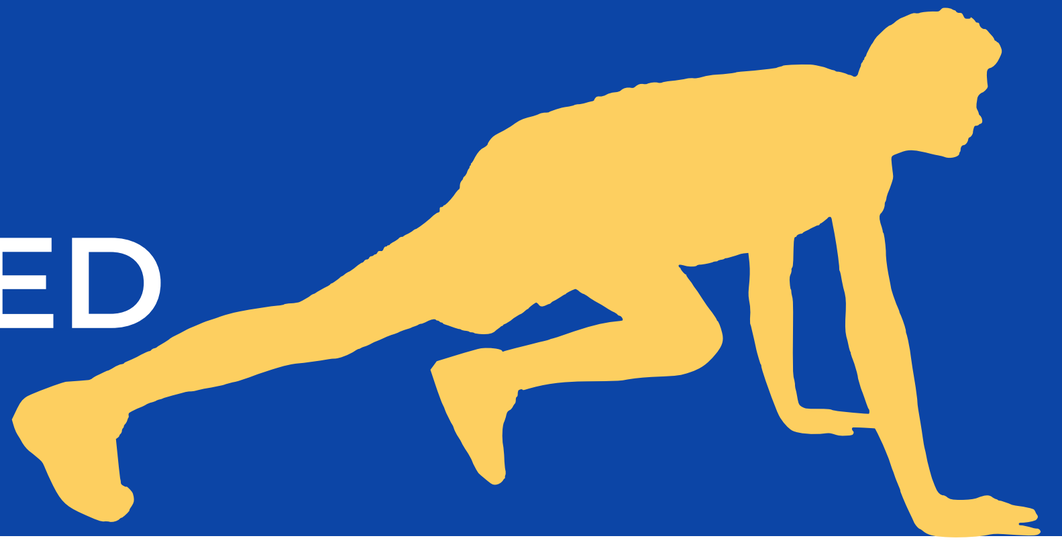
WHERE TO DOCUMENT?

Documentation lives where the source code lives!
(This is *never* in an email, chat, or similar!)

All documentation should be accessible to anyone who can access the software.

Ideally, all documentation can be referenced from all documentation.

PRELIMINARY CONCLUSION:
HOW TO GET STARTED



Answer before coding:

- **Who do you document for?**
- **What is your motivation for documentation?**

This determines

- what to document
- how to document it.

Always document for yourself!

Start documenting when you start coding. Add, change, remove documentation when you add, change, remove code.

The documentation lives where the code lives.

For each documentation type or format you use, answer these questions:

- Who is it for?
- What do they want and need?
- How else could I communicate with them?
- (When do they need it?)

How do you feel about adopting new processes?

Why do it?

Acknowledge your motivations for following/not following Stephan's advice!

Process and documentation are connected

If you have not defined your process, you may make poor decisions about your documentation

I'm interested in people and processes

As an SSI Fellow, I have been exploring how people manage and collaborate on research software projects.

Read my latest blog post for SSI

RESEARCH SOFTWARE PROJECTS ARE SUCCESSFUL BY DEFAULT

Securing funding requires a lot of upfront planning

Success is typically defined at the project level, not at the developmental level

Quality control mechanisms often weak or lacking

What is missing from our definition of success?

Capacity of the software to support future work

'Soft' critical success factors

What might we learn by changing the definition of success?

management and process deserve our attention

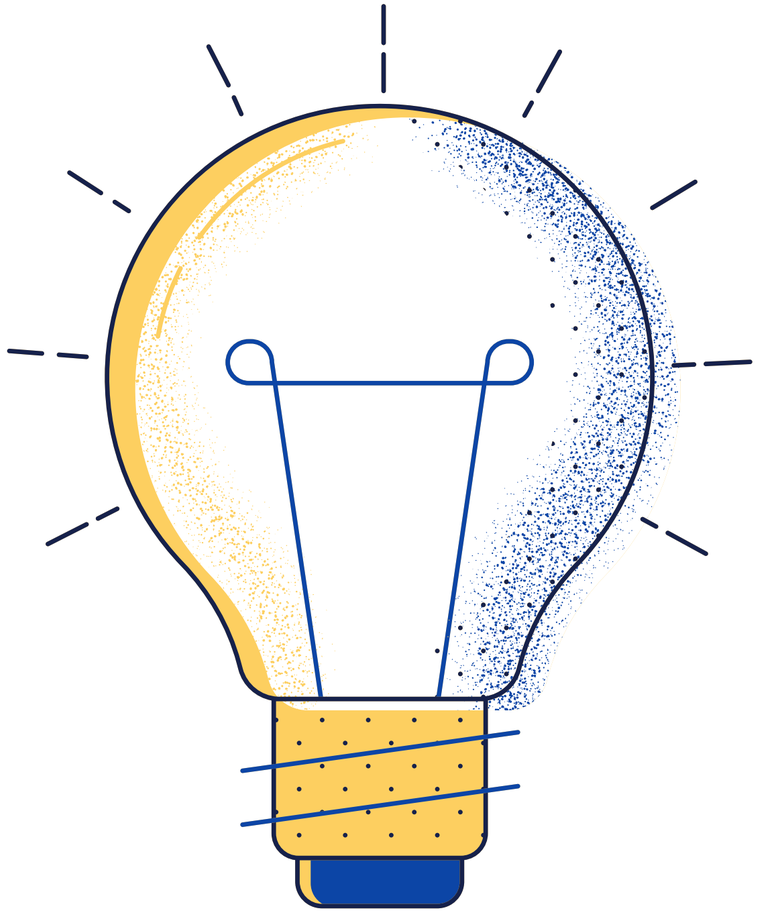
including the documentation!

Why?

Try asking yourself...

- What is your development process?
- What does your project timeline look like?
- How have you broken the project down?
- How are resources allocated?
- Who is working on this project?
- Who might work on the project in the future?
- How are you working together on the project?

Knowing what to document means asking these kinds of questions.



4 examples...

1. THE INFRASTRUCTURE PROJECT

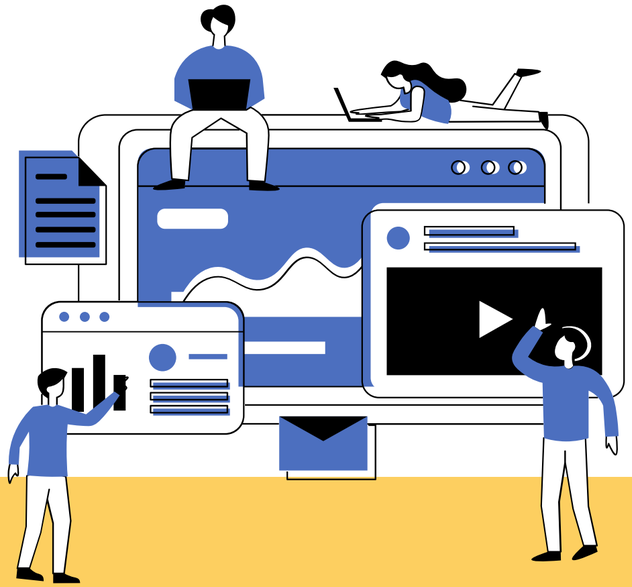
You are working on a critical infrastructure project that is likely to secure further funding. You are part of a large and distributed team of researchers and RSEs, most of whom are on fixed-term contracts.

2. THE OPEN-SOURCE PROJECT

You are working on an open-source toolkit. You have funding for the next year, but you know that securing further funding will be hard and you will be reliant on the open-source community to keep the project alive.

3. THE USER-FACING APPLICATION

You are a working in a small team on a user-facing application whose main users are other researchers. The project is funded for 3 years but may struggle to receive further funding. All members of the team are on fixed-term contracts/PhD students and likely to move on.



4. THE PHD STUDENT

You are a PhD student writing scripts to perform simulations. You are mainly working in isolation. Your PhD project may form the basis of future work.

One size does not fit all

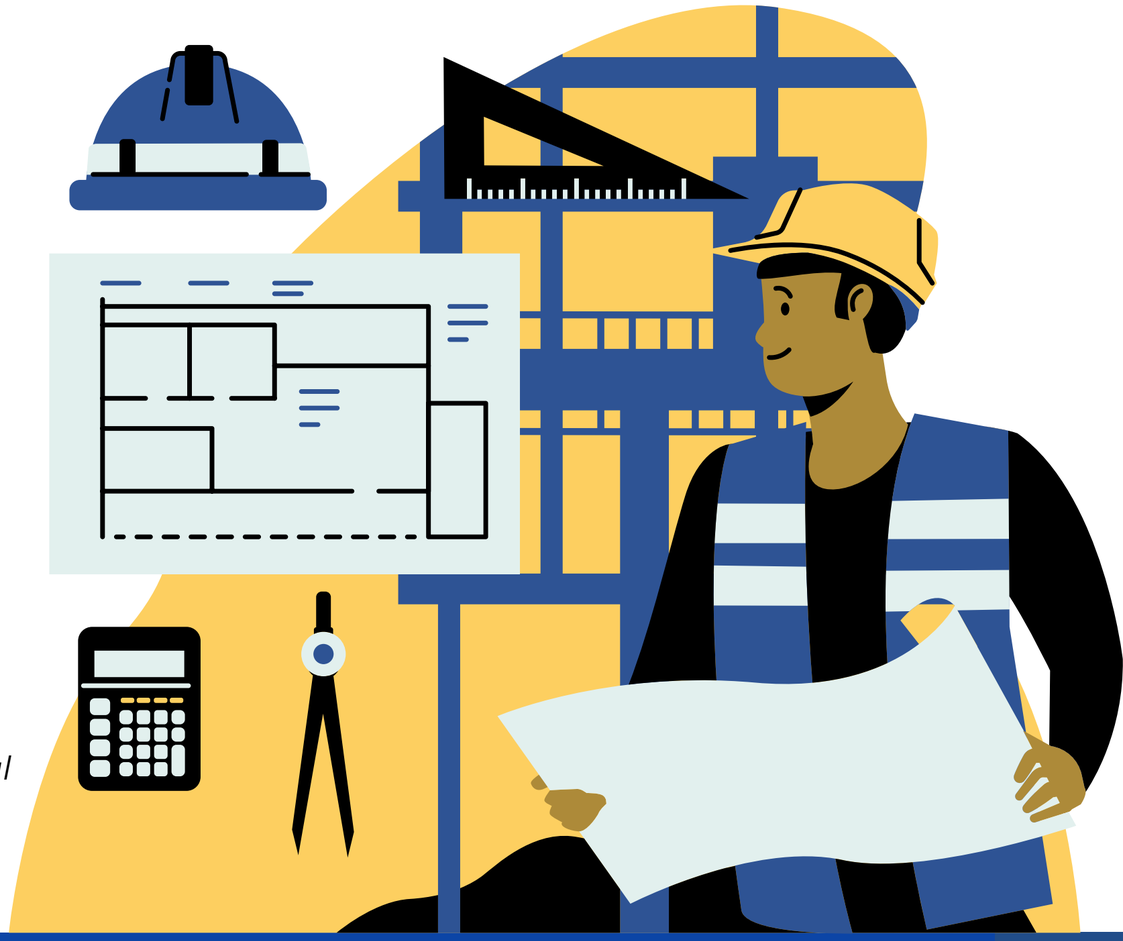
In each of the examples, your documentation decisions will have been different.

Each project had different aims and priorities, as well as different people and processes.

How to plan for documentation

- Have a documentation plan (however lightweight)
- Try to be realistic in your time and budget estimations
- Eliminate waste (keep it 'lean' where appropriate)
- In some cases, it may help to refer to ISO standards

ISO/IEC 15910-2002
ISO/IEC/IEEE 26512:2018
Mikhail Ostrogorskij (2018) *Approach to Term Time Estimation in Technical Documentation Development*



Start with the basics

Whatever your project, here are **3 golden rules** that can serve as a starting point.

1. Make code self-documenting

- Semantic identifiers
- Documentation comments / docstrings
- Clear comments on any code that is not self-documenting
- If working on a large codebase in a dynamically typed language, consider using a static type checker

2. Document mindfully

- Apply tools and best practices mindfully
  - Is this necessary?
  - What are the benefits?
  - What are the risks?
  - How else could I communicate this?

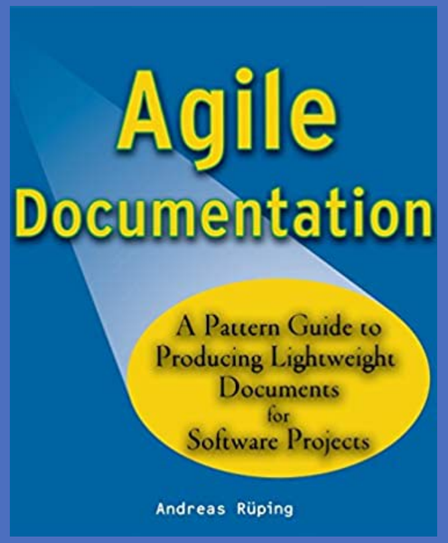
3. Document consistently

- Who else needs to be on board with this?
- Make decisions with the support of your team
- Use code review to ensure consistency

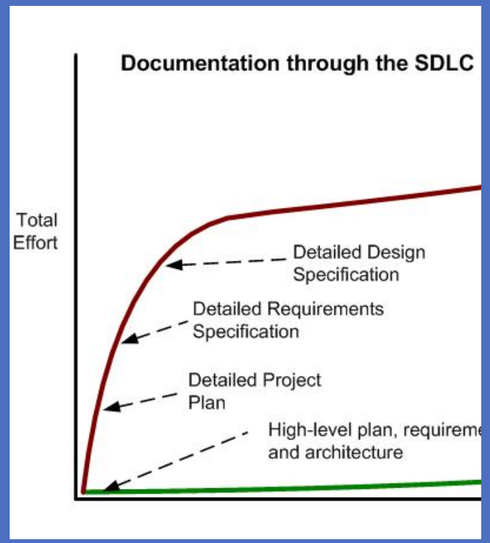
Thank you

sdruskat.net
stephan.druskat@dlr.de

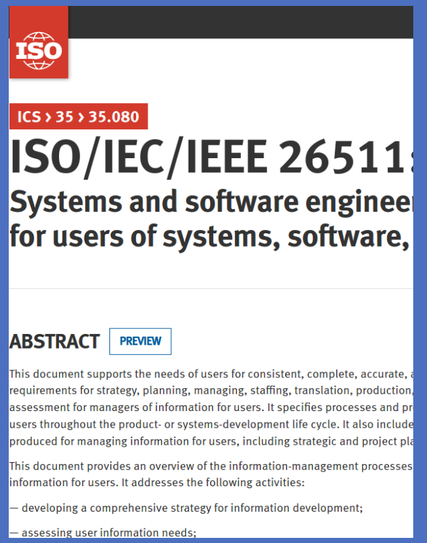
linkedin.com/in/sorrelharriet
sorrelharriet.medium.com



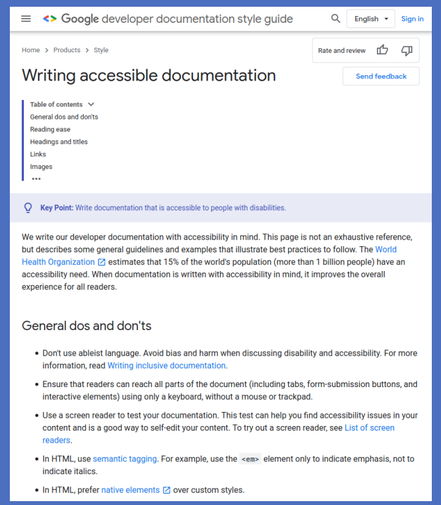
Agile Documentation: Andreas Rüping



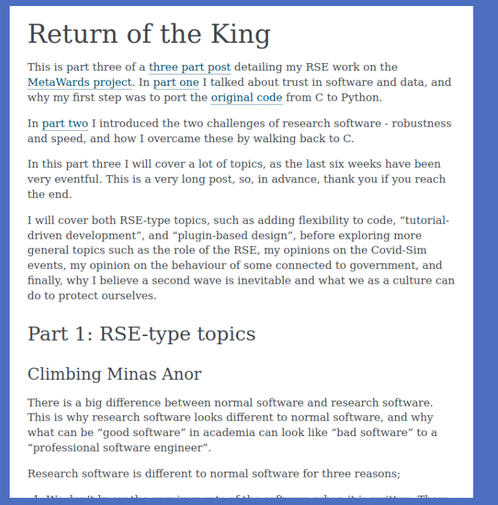
Agile/lean Documentation: Scott Ambler



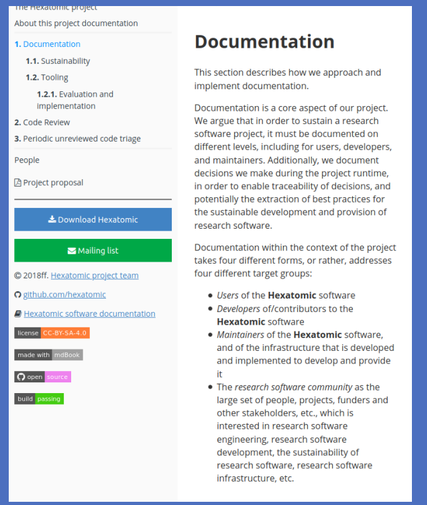
ISO/IEC/IEEE 26511



Google Documentation Style Guide



Tutorial-Driven Development



Project documentation example