### Good practices for research software documentation

Stephan Druskat Sorrel Harriet



### CONTEXT

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



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









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 are best practices for research software documentation?

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

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



### WHAT YOU'LL LEARN

Understand the documentation.

Learn about diffe

Make informed choices about how to approach software documentation.

Reflect on the role of software documentation within the software process.

### Understand the importance of software

Learn about different aspects of software



### 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





a productivity tool



# SOFTWARE DOCUMENTATION ...



not that tedious





#### helps build a user base



good scientific practice

### TAKE AWAY ADVICE

Think about your motivation for documenting the software.



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





When? Who for? What? How? Where?



### WHEN TO **DOCUMENT?**

Now!

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

Documentation is part of the code.



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



Consider levels of experience.

Keep in mind th over time.

Keep in mind that the target audience may change



### WHAT TO **DOCUMENT?**

#### Depends on target audience and motivation.











Collaborators













### **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

**PROJECT DOCUMENTATION** Rationale, teams, governance, community (contact, code of conduct)

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



### 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

Always human-readable

#### Machine-readable where useful or necessary

- Metadata
- Tests
- Doc strings

### Use available technology/tooling

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

• 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 docur documentation.

Ideally, all documentation can be referenced from all



### 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 <u>SSI</u> 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 selfdocumenting

- 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

<u>sdruskat.net</u> <u>stephan.druskat@dlr.de</u>

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

### Agile Documentation

A Pattern Guide to Producing Lightweight Documents for Software Projects

Andreas Rüping

#### <u>Agile Documentation:</u> <u>Andreas Rüping</u>

Home > Products > Style	Rate an	freview [	D G
Writing accessible documentation		Send fo	eedback
Table of contents V			
General dos and don'ts			
Reading ease			
Headings and titles			
Links			
Images			

General dos and don'ts

ibility need. When docur

experience for all readers

Don't use ableist language. Avoid blas and harm when discussing disability and accessibility. For more information, read Writing inclusive documentation.

entation is written with accessibility in mind, it improves the overal

- Ensure that readers can reach all parts of the document (including tabs, form-submission buttons, ar interactive elements) using only a keyboard, without a mouse or trackpad.
- Use a screen reader to test your documentation. This test can help you find accessibility issues in your
  content and is a good way to self-edit your content. To try out a screen reader, see List of screen
  readers.
- In HTML, use semantic tagging. For example, use the <em> element only to indicate emphasis, not to indicate italics.
- In HTML, prefer native elements ☑ over custom styles.

<u>Google Documentation</u> <u>Style Guide</u>



#### <u>Agile/lean Documentation:</u> <u>Scott Ambler</u>

### ISO

#### ICS > 35 > 35.080

#### ISO/IEC/IEEE 26511: Systems and software engineer for users of systems, software,

#### ABSTRACT PREVIEW

This document supports the needs of users for consistent, complete, accurate, a requirements for strategy, planning, managing, staffing, translation, production assessment for managers of information for users. It specifies processes and pr users throughout the product- or systems-development life cycle. It also include produced for managing information for users, including strategic and project pla

This document provides an overview of the information-management processe: information for users. It addresses the following activities:

developing a comprehensive strategy for information development;

- assessing user information needs

#### <u>ISO/IEC/IEEE 26511</u>

#### About this project documentation

#### 1. Documentatio

1.1. Sustainability 1.2. Tooling

1.2.1. Evaluation and

implementation Code Review

3 Periodic unreviewed code triag

People

Project proposa

- 2018ff. Hexatomic project team
- github.com/hexatomic

iexatomic software documentat

license CC-BY-SA-4.0 made with mdBook

build passing

#### Documentation

This section describes how we approach and implement documentation.

Documentation is a core aspect of our project. We argue that in order to sustain a research software project, it must be documented on different levels, including for users, developers, and maintainers. Additionally, we document decisions we make during the project runtime, in order to enable traceability of decisions, and potentially the extraction of best practices for the sustainable development and provision of research software.

Documentation within the context of the project takes four different forms, or rather, addresses four different target groups:

- Users of the Hexatomic software
- Developers of/contributors to the Hexatomic software
- Maintainers of the Hexatomic software, and of the infrastructure that is developed and implemented to develop and provide it
- The research software community as the large set of people, projects, funders and other stakeholders, etc., which is interested in research software engineering, research software development, the sustainability of research software, research software infrastructure, etc.

35

#### Project documentation <u>example</u>

#### Return of the King This is part three of a three part post detailing my RSE work on the

MetaWards project. In part one I talked about trust in software and data, and why my first step was to port the original code from C to Python.

In part two I introduced the two challenges of research software - robustness and speed, and how I overcame these by walking back to C.

In this part three I will cover a lot of topics, as the last six weeks have been very eventful. This is a very long post, so, in advance, thank you if you reach the end.

I will cover both RSE-type topics, such as adding flexibility to code, "tutorialdriven development", and "plugin-based design", before exploring more general topics such as the role of the RSE, my opinions on the Covid-Sim events, my opinion on the behaviour of some connected to government, and finally, why I believe a second wave is inevitable and what we as a culture can do to protect ourselves.

#### Part 1: RSE-type topics

#### Climbing Minas Anor

There is a big difference between normal software and research software. This is why research software looks different to normal software, and why what can be "good software" in academia can look like "bad software" to a "professional software engineer".

Research software is different to normal software for three reasons;

#### <u>Tutorial-Driven</u> <u>Development</u>