Documentation

Presented by

COLABS: Collaboration for Better Software for Science

In collaboration with

With prior support from

Anshu Dubey (she/her)
Argonne National Laboratory

Better Scientific Software tutorial @ ISC24

Contributors: Anshu Dubey (ANL), David Bernholdt (ORNL), Miranda Mundt (SNL)

License, Citation and Acknowledgements

License and Citation

- **The requested citation the overall tutorial is:** Anshu Dubey, Better Scientific Software tutorial, in ISC High Performance (ISC24), Hamburg, Germany, and online, 2024. DOI: 10.6084/m9.figshare.25686426.
- Individual modules may be cited as *Speaker, Module Title*, in *Tutorial Title*, …

Acknowledgements

- This work was supported by the U.S. Department of Energy Office of Science, Office of Advanced Scientific Computing Research (ASCR), and by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.
- This work was supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Next-Generation Scientific Software Technologies (NGSST) program.
- This work was performed in part at the Argonne National Laboratory, which is managed by UChicago Argonne, LLC for the U.S. Department of Energy under Contract No. DE-AC02-06CH11357.
- This work was performed in part at the Lawrence Livermore National Laboratory, which is managed by Lawrence Livermore National Security, LLC for the U.S. Department of Energy under Contract No. DE-AC52-07NA27344.
- This work was performed in part at the Los Alamos National Laboratory, which is managed by Triad National Security, LLC for the U.S. Department of Energy under Contract No.89233218CNA000001
- This work was performed in part at the Oak Ridge National Laboratory, which is managed by UT-Battelle, LLC for the U.S. Department of Energy under Contract No. DE-AC05-00OR22725.
- This work was performed in part at Sandia National Laboratories. Sandia National Laboratories is a multi-mission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC., a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Documentation Overview

Software documentation is any artifact made as part of the software development process that is intended to communicate information about the software system about which it was written.

Most people are familiar with this concept and know good documentation when they see it. More difficult, however, is how to *write* good documentation.

Benefits of Good Documentation

- *Better Maintainability*
  - Undocumented or incorrectly documented code can do more harm than good
  - It is difficult to maintain code that does not have sufficient and accurate documentation
  - Good documentation clarifies what the code is doing in each part and makes it easier to change
- *Improved Team Productivity*
  - Especially for new team members, sufficiently good documentation can help get everyone on the same page and new members up to speed
- *Increased Code Quality*
  - Documenting what you think your code does helps to clear up inconsistencies and can lead developers to refactor something that is needlessly complicated
  - Overall, documentation has a positive effect on overall quality

The Challenges to Making Good Documentation

- *Time*: Writing good documentation can take time, and especially for projects with unreliable or limited funding, it can become an afterthought. It also introduces potential technical debt if interfaces or functionality change.
- *Skill*: Writing good documentation is *hard*. It must be practiced and practiced and practiced. People can become jaded by how much practice it takes to become truly skilled at documenting well.
- *Process*: Does writing documentation feel unnatural or “clunky” to you? Without proper processes, writing documentation can feel like it’s wasting your precious time.

Some Documentation Types by their Primary Audience

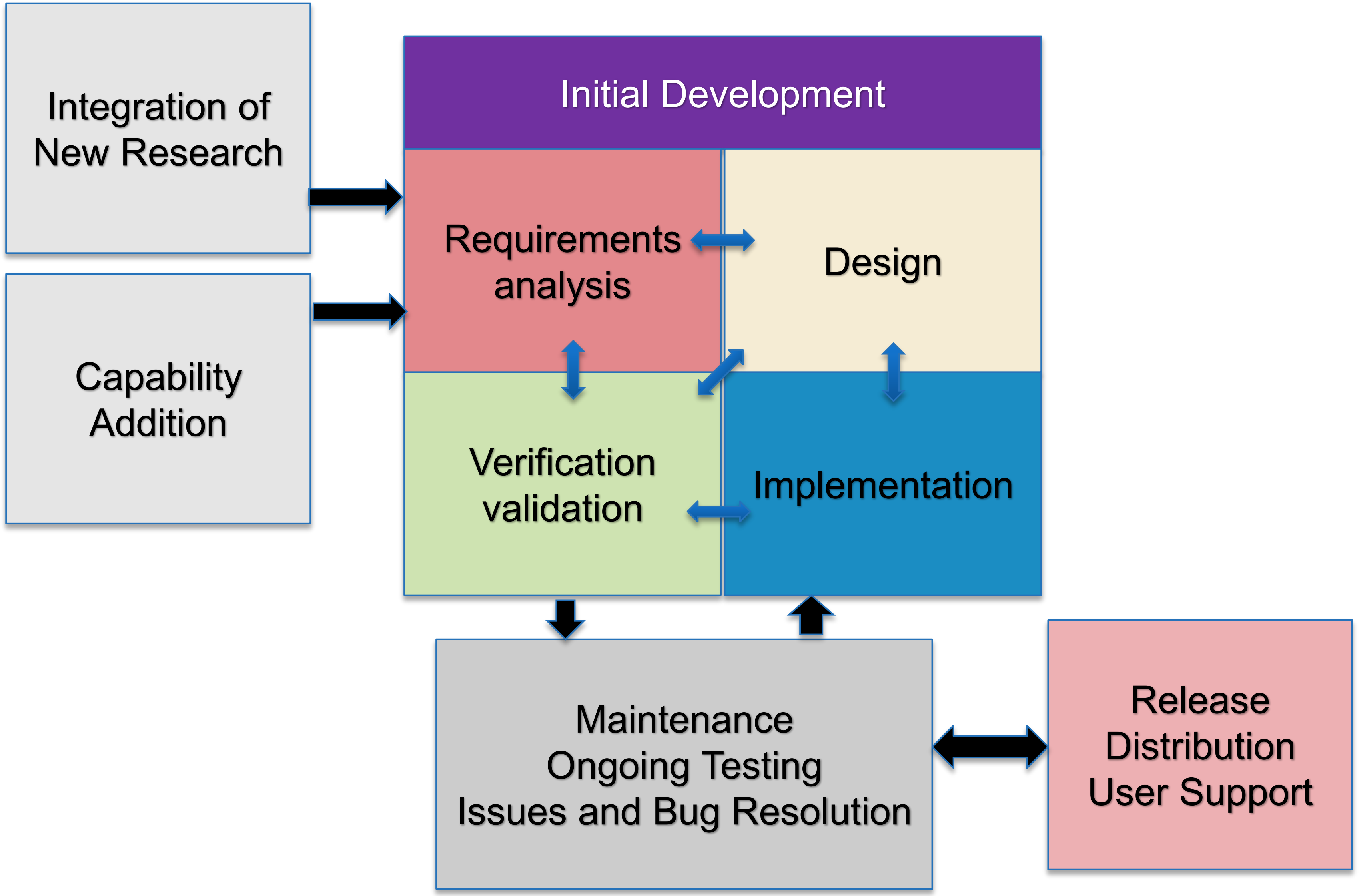
Maintainers, Developers, Outside Contributors

- Requirements analysis
- Design and architecture
- Contributor's guide
  - Information to help a new developer or outside contributor do things the way the maintainers want them done
- Coding standards
  - Automate what can be checked/enforced mechanically
  - Document all the standards that are important (noting what is automatically enforced)
- In-code documentation (*why*, not *what*)
- APIs
  - Automated tools can provide *skeletons* for API documentation, but humans need to add important content
- Reference manual

Users

- Installation guide
  - If users can't build and install your software easily, they're likely to give up and try an alternative rather than figuring out problems
  - Good to provide simple tests/examples so user can verify working installation
- User guide
  - Debugging and troubleshooting help is useful
- Tutorials and examples
  - If you provide example inputs, include the corresponding outputs!
- APIs (for libraries)
- Reference manual (for libraries)

Software Lifecycle



Stages

Initial Development

Requirements analysis

- Expectations from the software
- Capabilities needed
- Solvers needed
- Constraints
- How will they be tested

Example

- Same code for different applications → configurability
- Needed capabilities
- Battery of tests

Stages

Initial Development

Design

- Software overview
- Architecture
- Interfaces
- Coding principles
- Coding standards

Example

- Design docs → snapshot of discussion
- Online example of components
- Coding standards

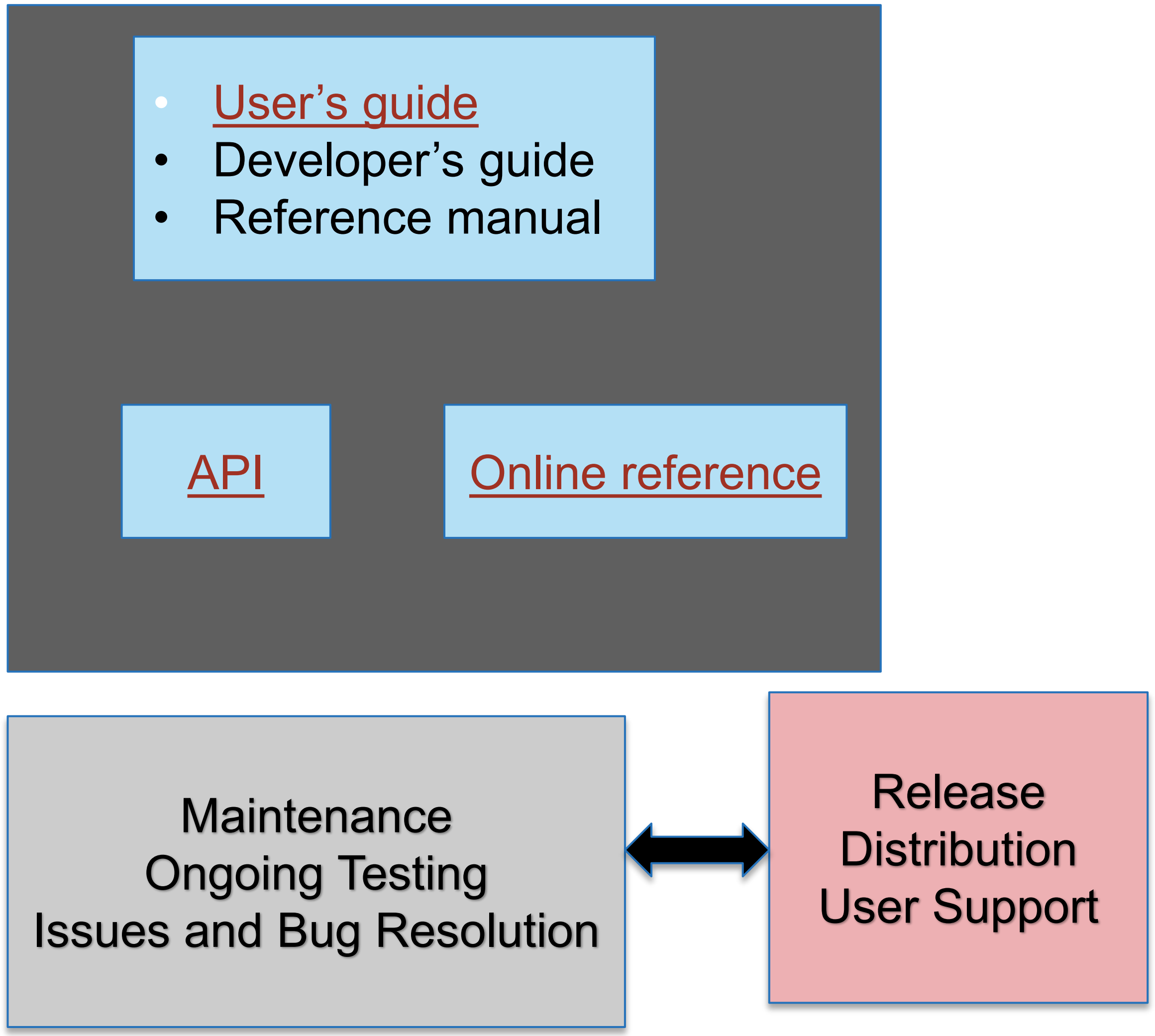
Stages

Initial Development

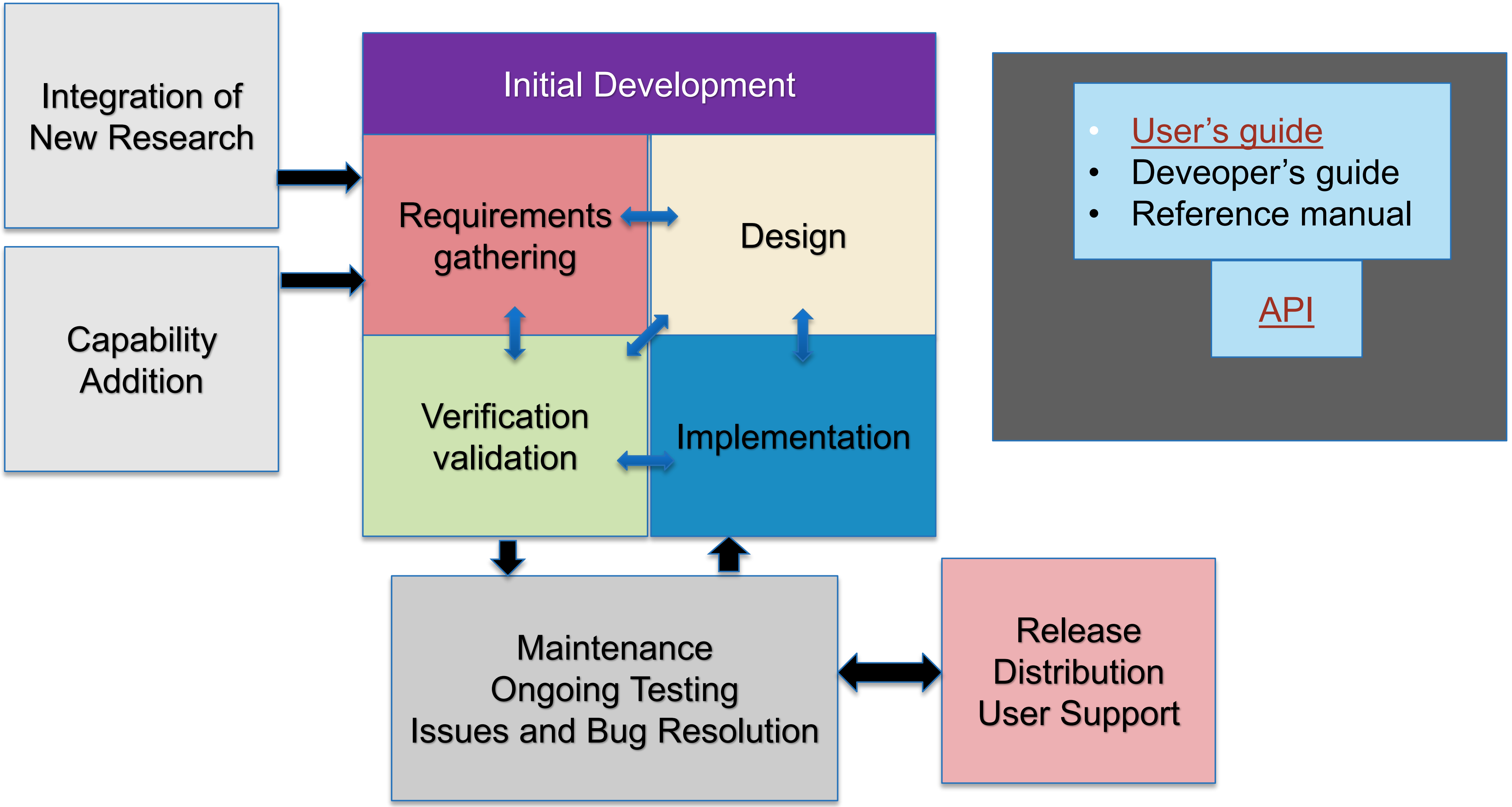
- Header – documenting functionality, inputs and outputs and outcomes
- API – tools that autogenerate documentation
  Doxygen, NDoc, Visual Expert, Javadoc, EiffelStudio, Sandcastle, ROBODoc, POD, Twin Text
- Inline documentation
  - Implementation choices

Implementation

Stages


User's guide
Developer's guide
Reference manual
API
Online reference
Maintenance
Ongoing Testing
Issues and Bug Resolution
Release
Distribution
User Support

Ongoing Development and Maintenance



Better Practices for Documentation

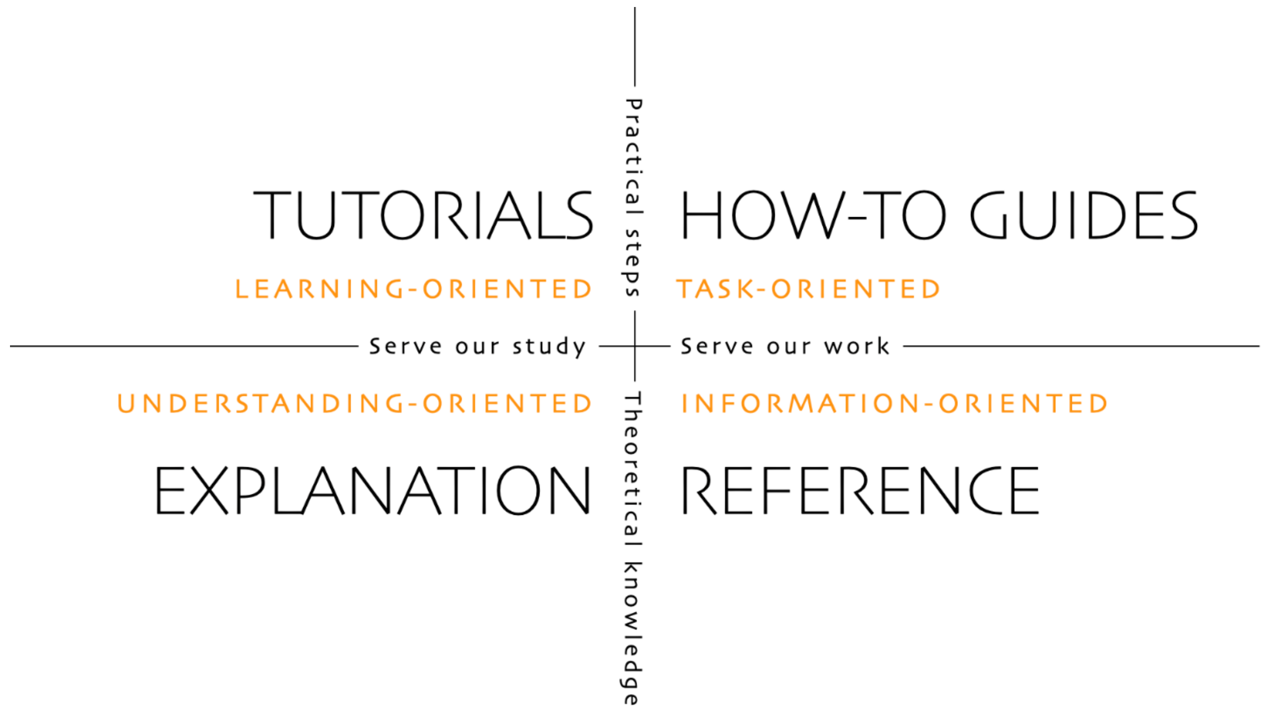
1. Version control your documentation
   – Preferably in the same repo as the code
2. Good enough is better than perfect
   – There's always room for improvement, but it is better to have something than nothing
3. Don't write more documentation than you can maintain
   – Voluminous documentation means more to maintain as the code evolves
4. Know your audience
5. Document as you go, while it is fresh in your mind
6. User test your documentation

Meet Your Readers Where They're At

- When writing documentation, put yourself in the mindset of your *reader*
- Consider…
  - Culture: you have understandings, language (e.g., slang), etc., that are a product of the culture in which you've developed. Readers from other cultural backgrounds may not share them
  - Context: different scientific domains often use technical terms differently. It can be hard to recognize when this is happening – listen carefully for indications
  - Experience: readers may have different levels and types of experience from you
  - Recall the common exercise of writing instructions for making a peanut butter and jelly sandwich
- Consider developing and utilizing a few personas for your audience(s) to help ground thinking
- Whenever possible, test documentation with exemplars from your audience(s)
  - Good opportunities include: on-boarding of new developers, tutorials for users, etc.
  - Questions you receive *may* reflect gaps in documentation

Different Types of Documentation

- The Diataxis Framework provides a way to think systematically about how to target different types of documentation
- Documents should try to be in just one quadrant! (In practice, often fuzzy)

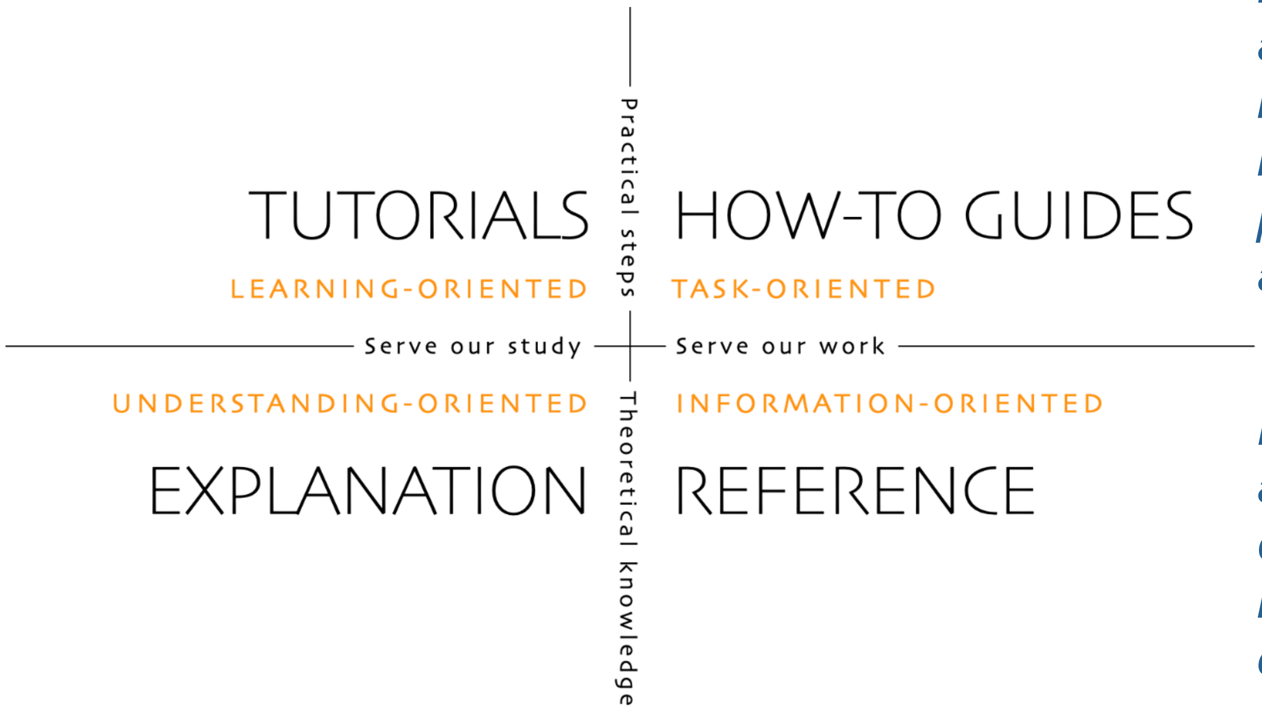


Different Types of Documentation

- The Diataxis Framework provides a way to think systematically about how to target different types of documentation
- Documents should try to be in just one quadrant! (In practice, often fuzzy)

*Tutorials are **lessons** that take the reader by the hand through a series of steps to complete a project of some kind. Tutorials are **learning-oriented**.*

*How-to guides are **directions** that take the reader through the steps required to solve a real-world problem. How-to guides are **goal-oriented**.*



*Explanation is **discussion** that clarifies and illuminates a particular topic. Explanation is **understanding-oriented**.*

*Reference guides are **technical descriptions** of the machinery and how to operate it. Reference material is **information-oriented**.*

Characteristics of Documentation in the Diataxis Framework

	Tutorials	How-to guides	Reference	Explanation
What they do	introduce, educate, lead	guide, demonstrate	state, describe, inform	explain, clarify, discuss
Answers the question	“Can you teach me to…?”	“How do I…?”	“What is…?”	“Why…?”
Oriented to	learning	tasks	information	understanding
Purpose	to allow the newcomer to get started	to show how to solve a specific problem	to describe the machinery	to explain
Form	a lesson	a series of steps	dry description	discursive explanation
Food/cooking analogy	teaching a child how to cook	a recipe in a cookery book	a reference encyclopedia article	an article on culinary social history

Much more discussion at https://diataxis.fr

Tools Can Help with Documentation

- Tools can’t write your documentation for you…
- But they can help make documentation easier to create and maintain
  - Templates for content
  - Extraction, formatting, publication

Documenting Code

- Coding standards help make code itself more uniform, more readable, and understandable
- Many large organization have (and publish) their coding standards for various languages
  - Can be used as examples, or adopted wholesale (reduces style-related arguments on your team)
- Tools can check and even enforce some standards (often called “linters”)
  - Other aspects of coding standards aren’t easily automated (e.g., meaningful variable names)
  - Python: flake8, ruff, black, etc.
  - C/C++: clang-tidy, clang-format
  - Fortran: see https://fortranwiki.org/fortran/show/Tools

In-Code Documentation

- In-code documentation should focus on the *why* not the *what*
  - The *what* is right there in the code, and should be self-evident
- APIs should be thoughtfully documented in the code
  - Many tools can help extract API and documentation into standalone documents
  - The more widely the API is meant to be exposed/used, the better the documentation needed
  - Thoughtful? Yes, the documentation should be more than a regurgitation of the parameter names used in the API. Document pre-conditions, post-conditions, error conditions, actions, side-effects, show examples of usage, etc.
  - Tools: Doxygen, NDoc, Visual Expert, Javadoc, EiffelStudio, Sandcastle, ROBODoc, POD, TwinText, and many more…
  - Typically produce HTML output that can be (automatically) published to a website
- Many IDEs offer plugins to help with inline documentation (e.g. templating) for many languages
  - E.g., VS Code, NetBeans, PyCharm, Eclipse, etc.

Tools for General Documentation and Publishing

- Markup languages
  - E.g., Markdown (MD), ReStructured Text (RST)
  - Simple text-based markup plus appropriate tools provides reasonable formatting capabilities
  - Work well with version control
  - Many free/open source tools available
- Formatting/generation tools
  - Sphinx: started in Python community, but not limited to it. RST or MD inputs. Can output many formats (e.g., HTML, ePUB, LaTeX, …)
  - Jekyll: static (web)site generation tool. MD input, HTML output. Started as a blogging tool, but much more capable
- Publishing (hosting sites)
  - ReadtheDocs: hosting of Sphinx-based documentation. Free for open-source projects, commercial hosting available. Supports multiple versions.
  - GitHub Pages: *.github.io URLs, associated with GitHub organizations and repositories. Good support for HTML and Jekyll, others via Actions
- Automate where it makes sense

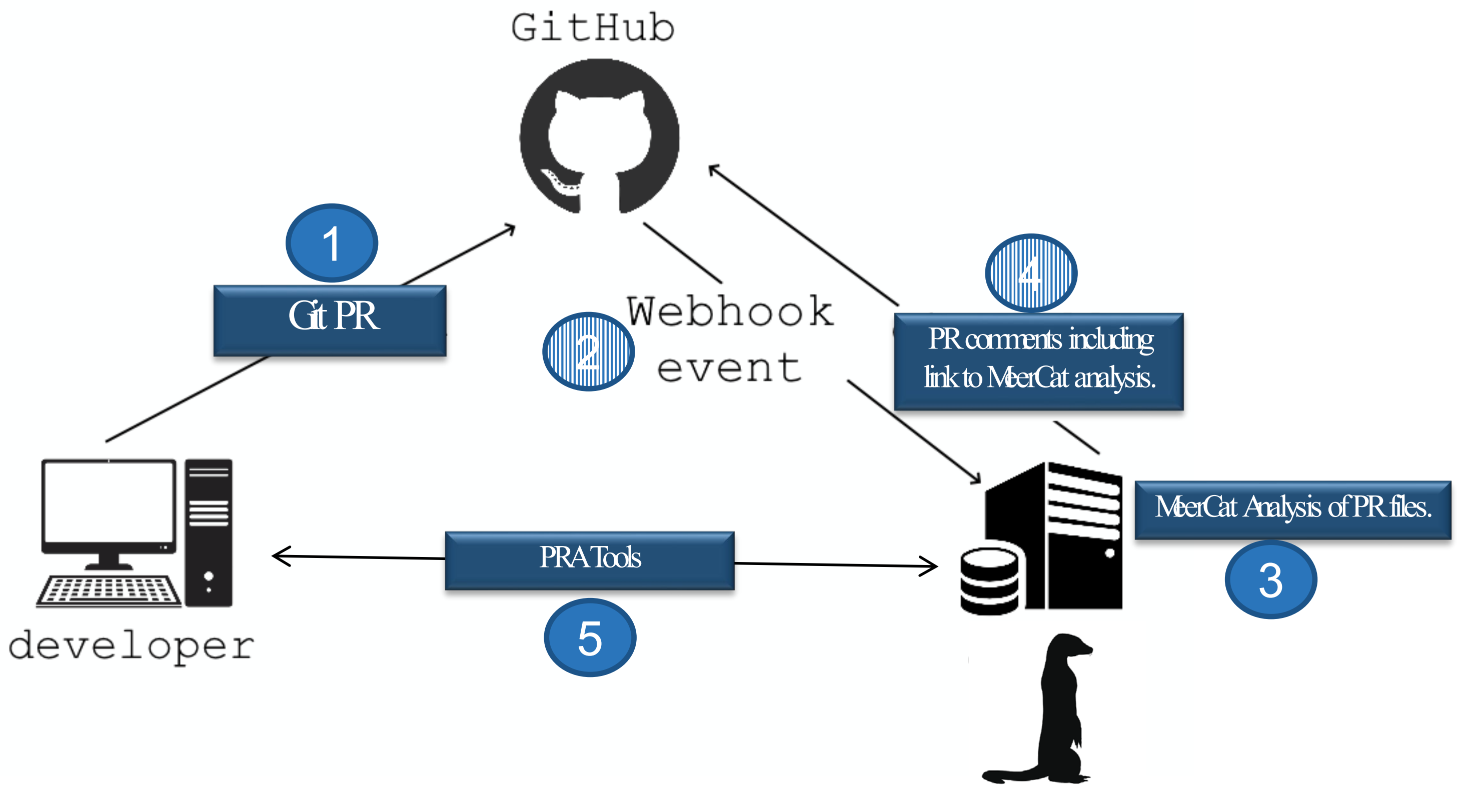
A Look at the Future – MeerCat

MeerCat is a system that supplies tools to support people involved through a ***Pull Request Assistant*** **(**PRA).

- The idea of the PRA, and its requirements to be a useful tool, came by working with PR reviewers.
- MeerCat attempts to work with the PR author
  - To diagnose issues with documentation and/or test coverage
  - To aid in fixing those issues during the PR process.
- The goal is a clean, well-thought-out PR that moves its way to formal review.

Context

- ❑ Before showing MeerCat in action, it helps to know the way it integrates with: (a) GitHub, and (b) the PR process.
- ❑ Installation of a Webhook (step 2) is optional. Step 3 can be invoked directly from the MeerCat server.
- ❑ Once step 5 is completed, the PR is ready for formal review.


GitHub
1
Git PR
2
Webhook event
4
PR comments including link to MeerCat analysis.
MeerCat Analysis of PR files.
3
PRA Tools
5
developer

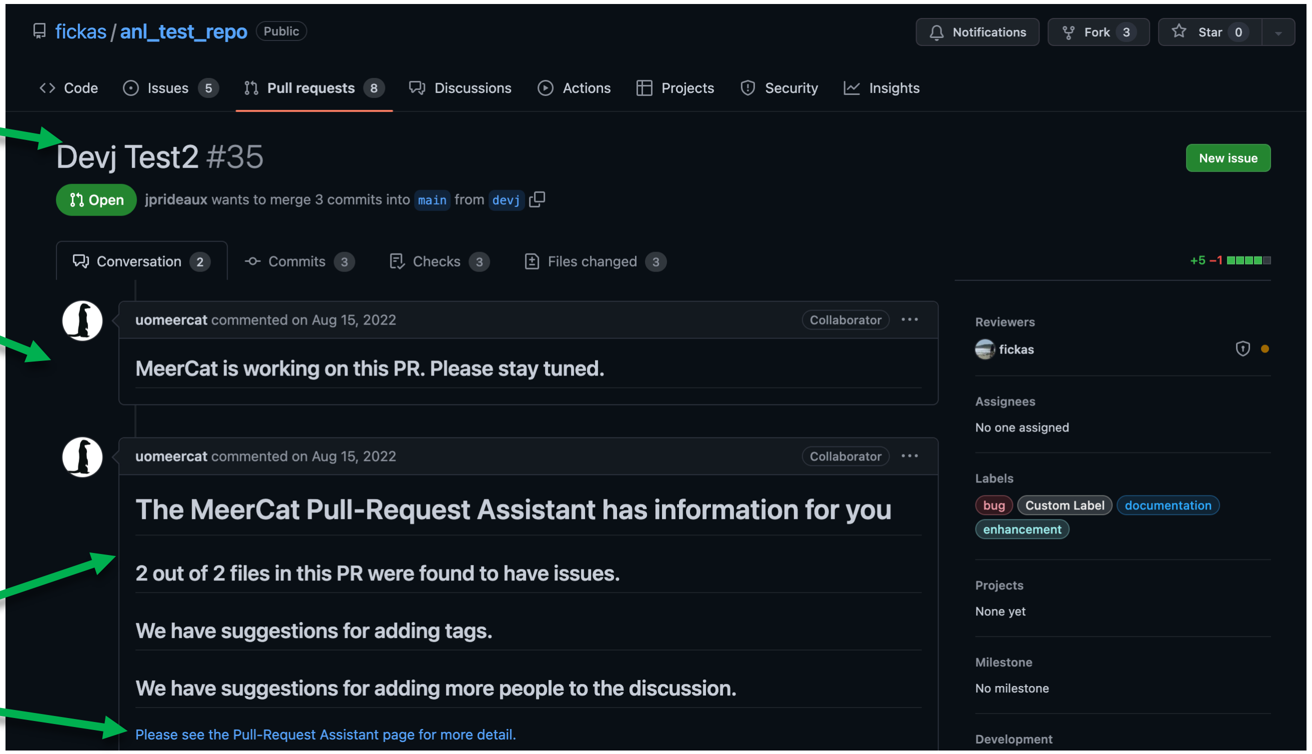
Steps 1 - 4

1 PR created.

2 MeerCat triggered by Webhook.

3 Analysis carried out on MeerCat server.

4 Brief description of analysis and link to full analysis on MeerCat web site.



Devj Test2 #35

Open jprideaux wants to merge 3 commits into `main` from `devj`

uomeercat commented on Aug 15, 2022 Collaborator

MeerCat is working on this PR. Please stay tuned.

uomeercat commented on Aug 15, 2022 Collaborator

The MeerCat Pull-Request Assistant has information for you

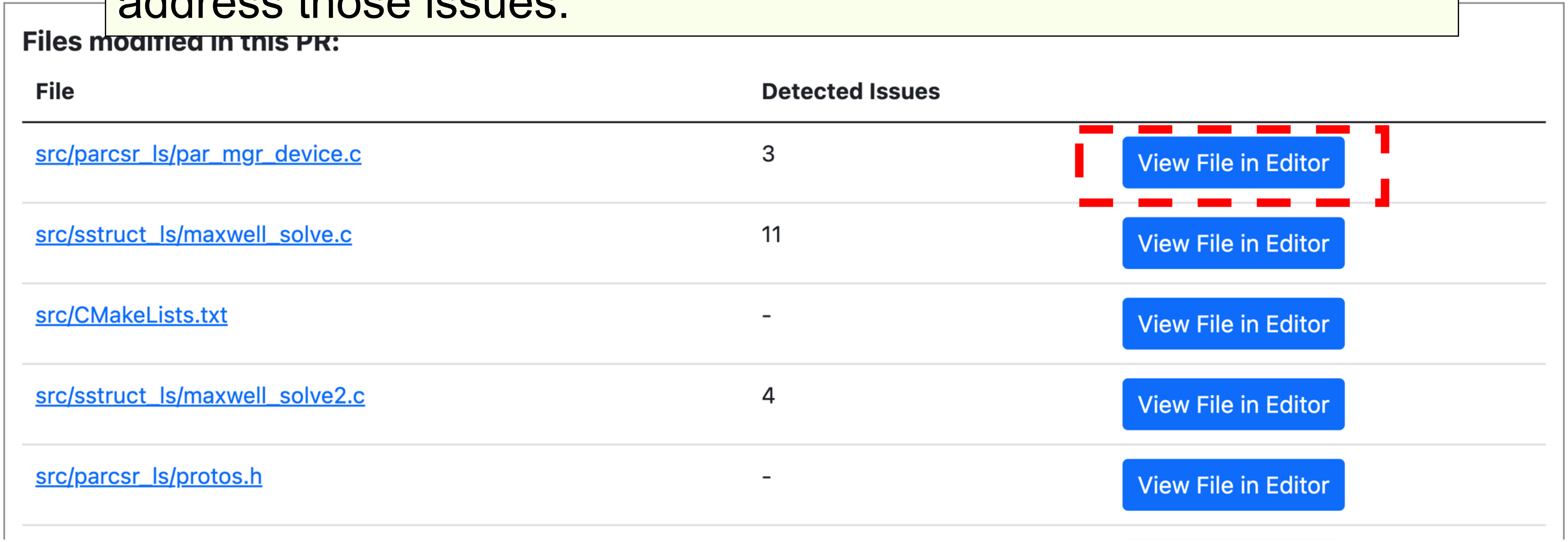
2 out of 2 files in this PR were found to have issues.

We have suggestions for adding tags.

We have suggestions for adding more people to the discussion.

Please see the Pull-Request Assistant page for more detail.

Once on MeerCat, the PRA lists the files in the PR, along with issues. The developer can then invoke the PRA editor to address those issues.



Files modified in this PR:

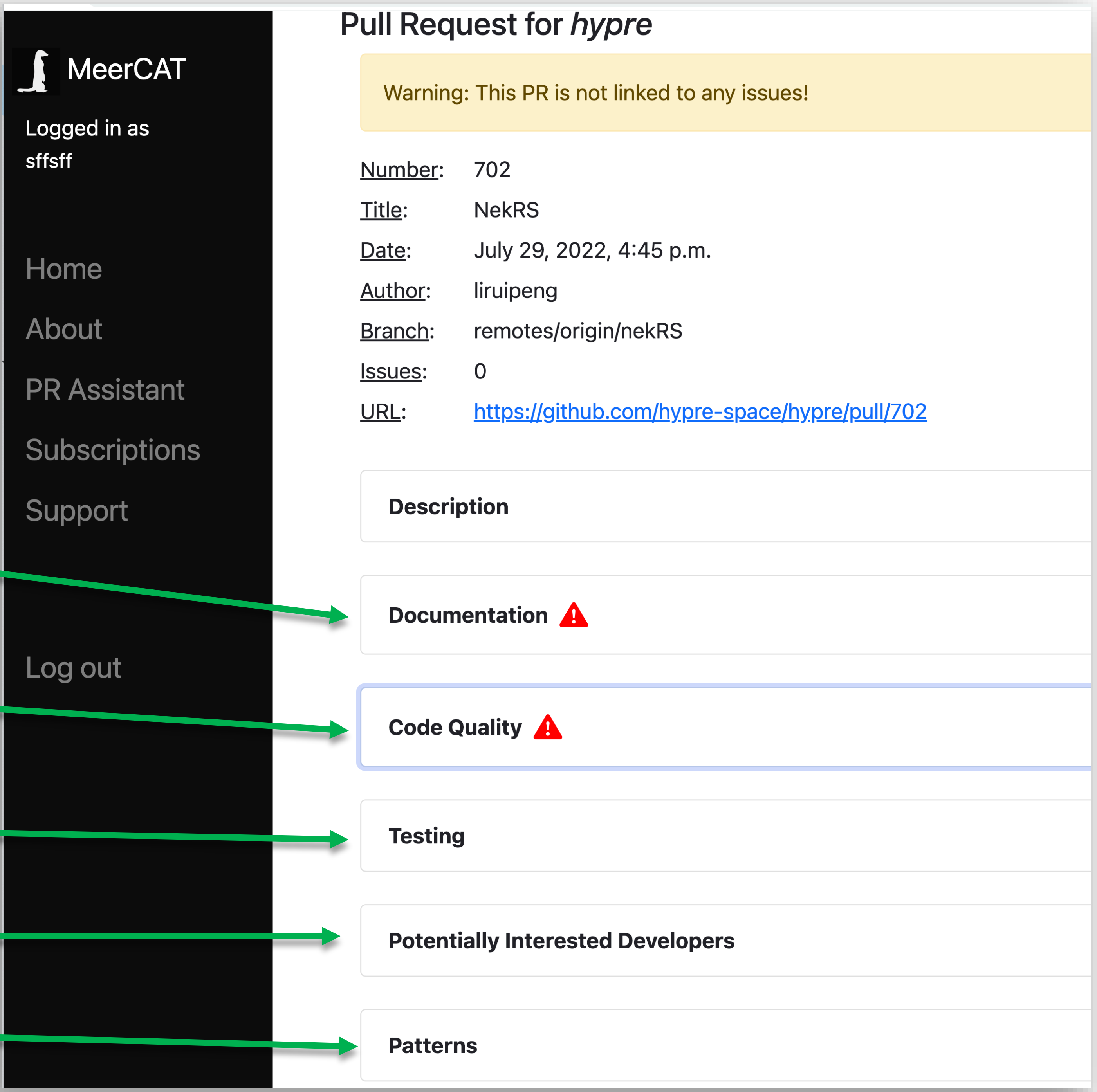
File	Detected Issues	
src/parcsr_ls/par_mgr_device.c	3	View File in Editor
src/sstruct_ls/maxwell_solve.c	11	View File in Editor
src/CMakeLists.txt	-	View File in Editor
src/sstruct_ls/maxwell_solve2.c	4	View File in Editor
src/parcsr_ls/protos.h	-	View File in Editor

5

Step 5: PRA working with PR author to improve quality before reaching PR reviewer(s).

Currently there is analysis, for each file in the PR, on the following **project** quality requirements.

- Quality of documentation, e.g., correct Doxygen formatting.
- Code quality, e.g., linting.
- Testing quality, e.g., missing/erroneous unit tests.
- Collaboration.
- Dangerous development patterns, e.g., churn.



Pull Request for *hypre*

Warning: This PR is not linked to any issues!

Number:	702
Title:	NekRS
Date:	July 29, 2022, 4:45 p.m.
Author:	liruipeng
Branch:	remotes/origin/nekRS
Issues:	0
URL:	https://github.com/hypre-space/hypre/pull/702

Description

Documentation ⚠

Code Quality ⚠

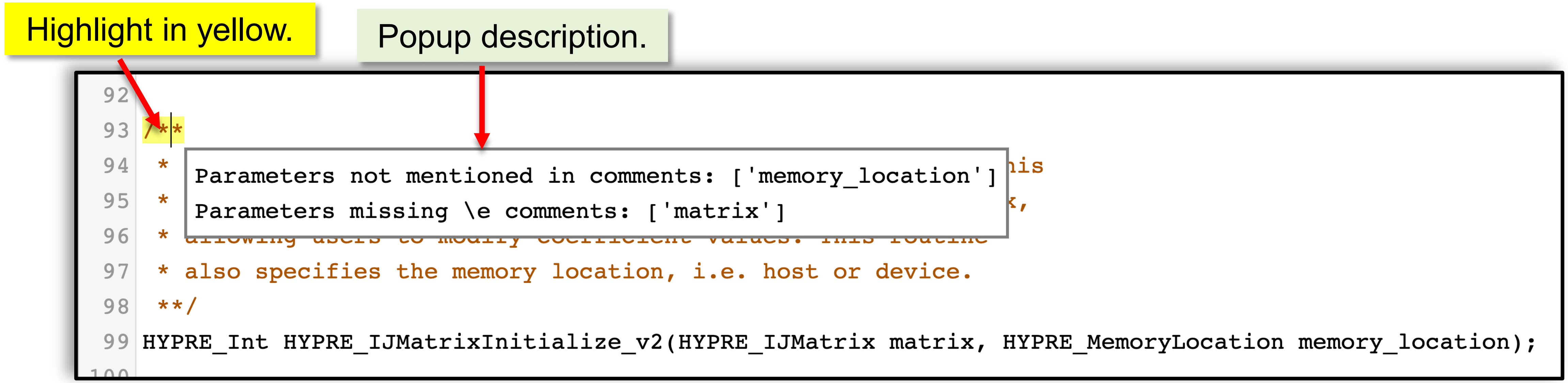
Testing

Potentially Interested Developers

Patterns

Assistance for: Documentation

Example: the Hypre project requires that developers place Doxygen comments preceding all C++ function declarations. These comments are expected to provide a description of each parameter. MeerCat checks this requirement by parsing the Doxygen comments and comparing against parameter list.



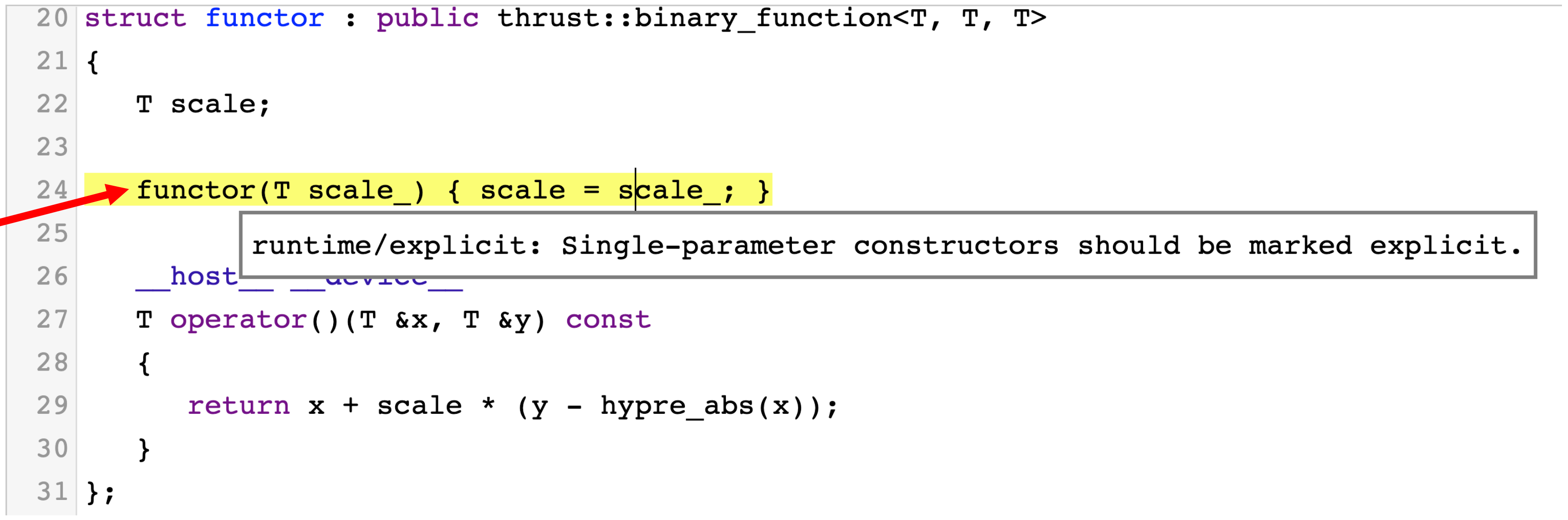
```

/**
 *                                                             his
 *                                                             x,
 * allowing users to modify coefficient values. This routine
 * also specifies the memory location, i.e. host or device.
 **/
HYPRE_Int HYPRE_IJMatrixInitialize_v2(HYPRE_IJMatrix matrix, HYPRE_MemoryLocation memory_location);
```

```
Parameters not mentioned in comments: ['memory_location']
Parameters missing \e comments: ['matrix']
```

Assistance for: Code Quality

Similar to Documentation, code quality warnings shown in yellow and click to see descriptions.



```
struct functor : public thrust::binary_function<T, T, T>
{
   T scale;

   functor(T scale_) { scale = scale_; }

   __host__ __device__
   T operator()(T &x, T &y) const
   {
      return x + scale * (y - hypre_abs(x));
   }
};
```

```
runtime/explicit: Single-parameter constructors should be marked explicit.
```

Assistance for: test maintenance

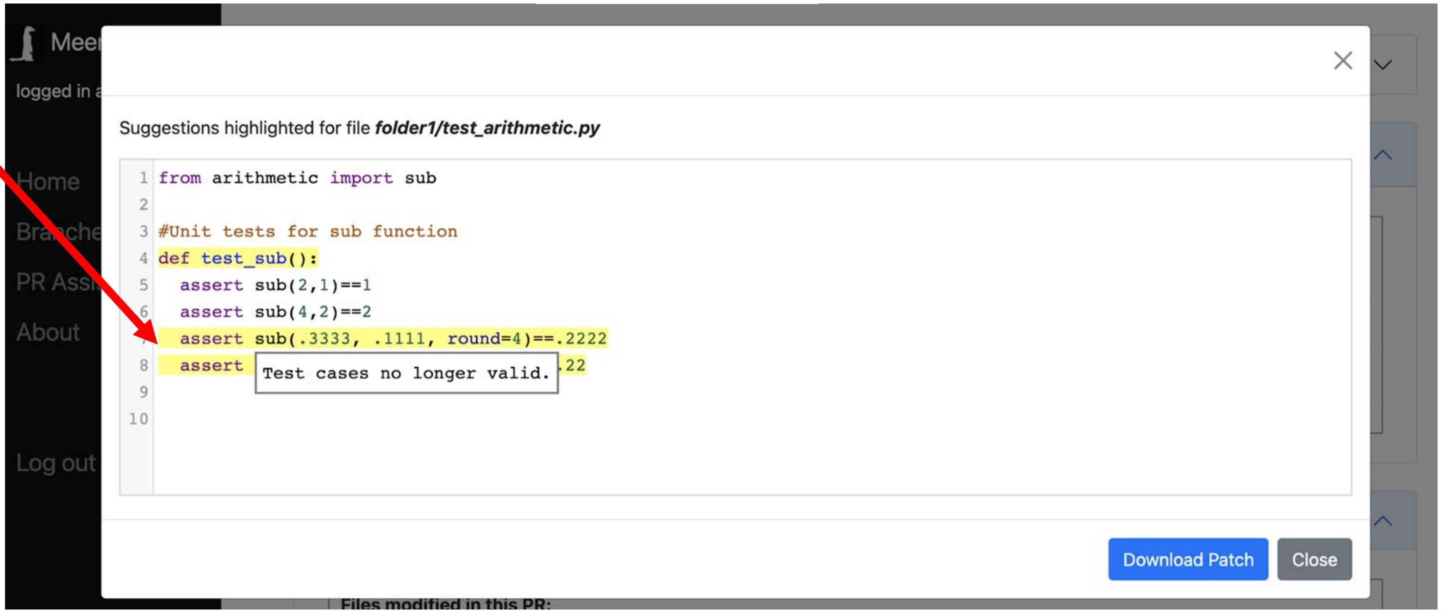
Given a change to a function **`sub`** in a code file F, MeerCat searches for test files associated with **`sub`**.

Here it finds a test that is no longer valid given the change to **`sub`** in the PR.

Caveat: this only works with pytest at the moment, which has a specific structure that can be exploited. GoogleTest is similar.

Many other projects have custom testing structure that require purpose-built search engines, a time-consuming process. Remains a research area for MeerCat.

pytest



Suggestions highlighted for file *folder1/test_arithmetic.py*

```
from arithmetic import sub

#Unit tests for sub function
def test_sub():
  assert sub(2,1)==1
  assert sub(4,2)==2
  assert sub(.3333, .1111, round=4)==.2222
  assert  Test cases no longer valid. .22
```

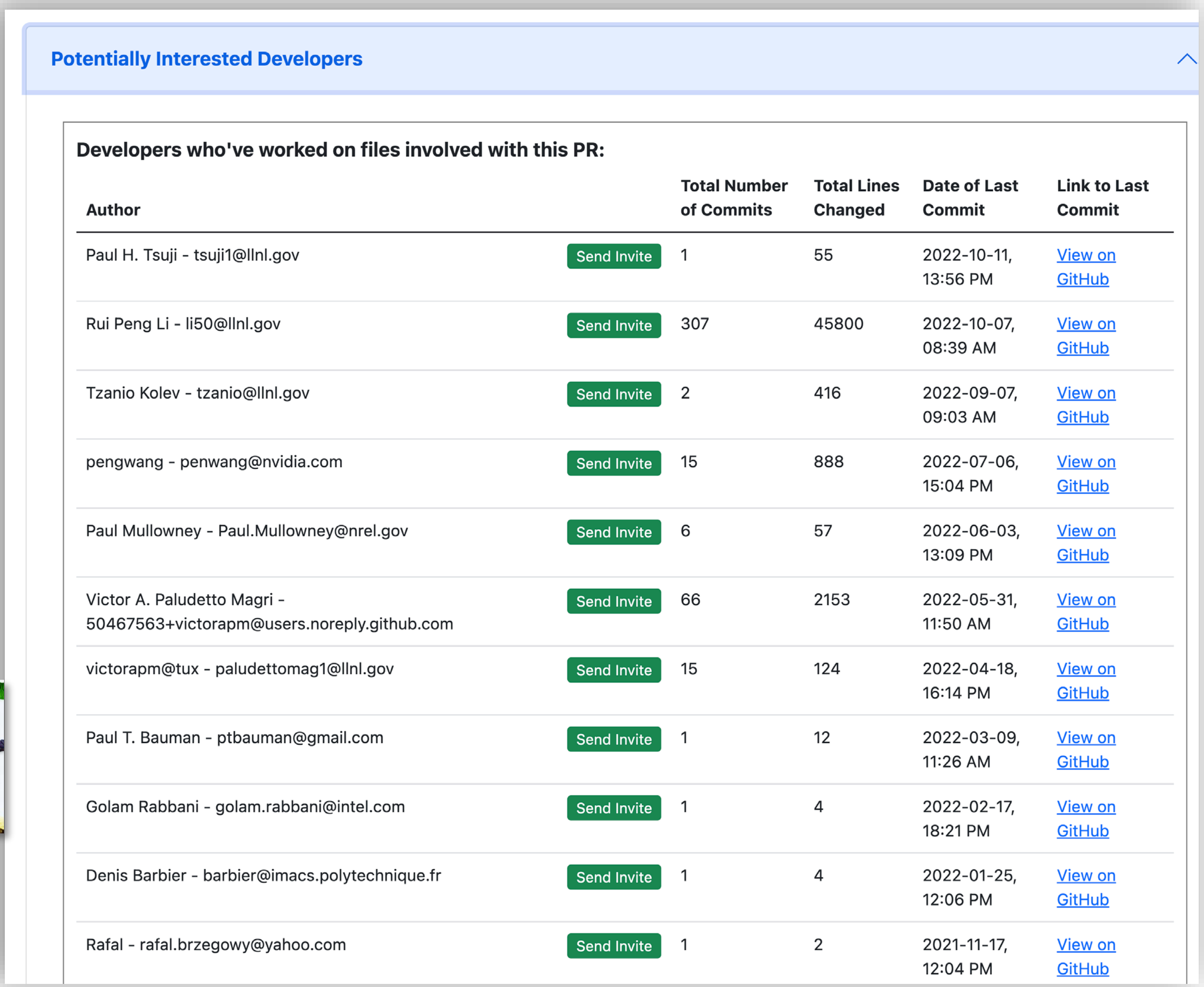
Assistance for: Collaboration

MeerCat finds developers who might have an interest in the PR discussion and provides a means to invite them in.

They can be sent an email-invite to join, which they can accept or ignore.

This is an active area of research for the MeerCat team. We are researching ways to find the “best” candidates. Currently only using past commits to one of PR files.

But now looking at indirect ways a developer might have an interest, e.g., **uses** a subroutine/function that has been changed, been part of a discussion related to a commit.



Potentially Interested Developers

Developers who've worked on files involved with this PR:

Author		Total Number of Commits	Total Lines Changed	Date of Last Commit	Link to Last Commit
Paul H. Tsuji - tsuji1@llnl.gov	Send Invite	1	55	2022-10-11, 13:56 PM	View on GitHub
Rui Peng Li - li50@llnl.gov	Send Invite	307	45800	2022-10-07, 08:39 AM	View on GitHub
Tzanio Kolev - tzanio@llnl.gov	Send Invite	2	416	2022-09-07, 09:03 AM	View on GitHub
pengwang - penwang@nvidia.com	Send Invite	15	888	2022-07-06, 15:04 PM	View on GitHub
Paul Mullowney - Paul.Mullowney@nrel.gov	Send Invite	6	57	2022-06-03, 13:09 PM	View on GitHub
Victor A. Paludetto Magri - 50467563+victorapm@users.noreply.github.com	Send Invite	66	2153	2022-05-31, 11:50 AM	View on GitHub
victorapm@tux - paludettomag1@llnl.gov	Send Invite	15	124	2022-04-18, 16:14 PM	View on GitHub
Paul T. Bauman - ptbauman@gmail.com	Send Invite	1	12	2022-03-09, 11:26 AM	View on GitHub
Golam Rabbani - golam.rabbani@intel.com	Send Invite	1	4	2022-02-17, 18:21 PM	View on GitHub
Denis Barbier - barbier@imacs.polytechnique.fr	Send Invite	1	4	2022-01-25, 12:06 PM	View on GitHub
Rafal - rafal.brzegowy@yahoo.com	Send Invite	1	2	2021-11-17, 12:04 PM	View on GitHub

Assistance for: File-level patterns

PLURALSIGHT
20 patterns to watch for in your engineering team
A field guide to help you recognize achievement, spot bottlenecks, and debug your development process with data

Many of these 20 patterns are based on repo-wide analysis. However, several are at the file level and can be checked against the files in the PR.

We describe one in particular: **Churn**.

Churn symptoms: a single developer making (much) above average changes to the same file, often without mooring to an **Open Issue**.

Churn causes:

- **Perfectionism**: a single developer constantly tweaking code to make it perfect.
- **Struggling**: a single developer rewriting large pieces of code on a routine basis.

To learn more about MeerCat, and/or using it in your project

Contact fickas@cs.uoregon.edu

Summary

- Documentation is valuable to both developers and users (different kinds)
  - Some of the most valuable documentation for developers has to do with why things are being done a certain way – especially requirements and design decisions that may have been made long ago
- It takes work, but should be maintained together with code changes rather than as an afterthought
- Think about your target audience when writing documentation
- Follow best practices and use appropriate tools to make it easier and more systematic to maintain your documentation