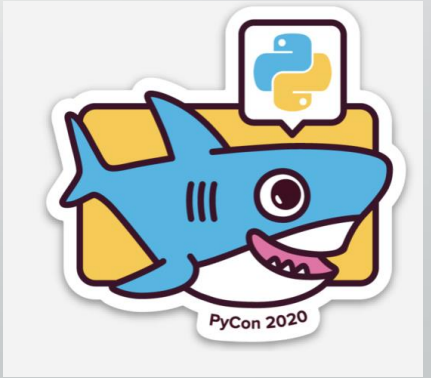
Building Docs like Code: Continuous Integration for Documentation

Presenter: Mason Egger
@masonegger
twitch.tv/zelgiuscodes

Who am I?

- Developer Advocate @ DigitalOcean
- Volunteer Educator - TEALS
- Documentation fanatic



Who is this talk for?

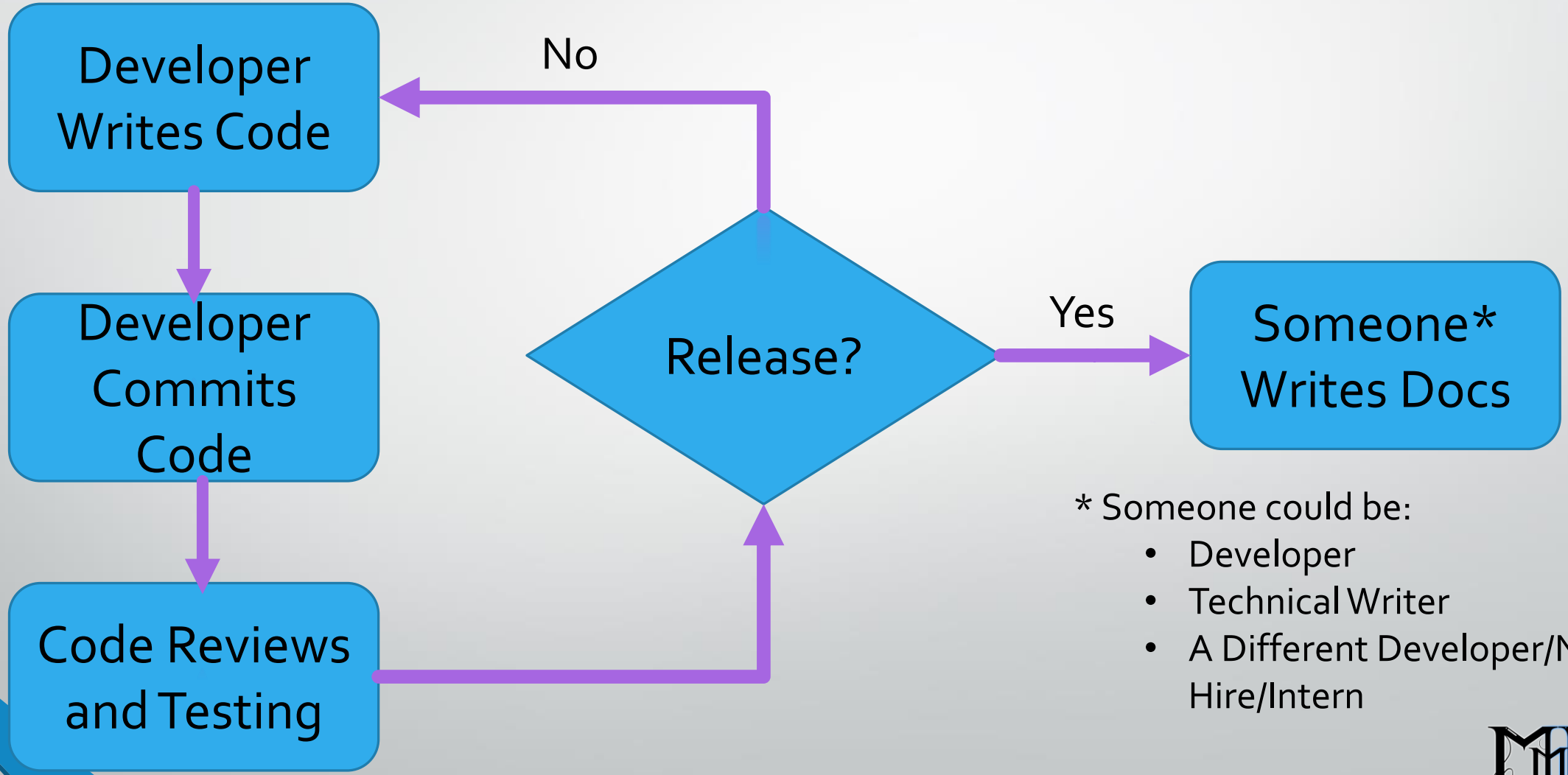
This talk is for:

- Open Source Maintainers
- Jr. Developers
- Sr. Developers
- Program/Project/Community Managers
- Dev Ops Engineers
- .....the list goes on and on

Anyone who writes, maintains, or manages a product that they intend to share with someone else

How we manage documentation

A Common Approach to Documentation



* Someone could be:
- Developer
- Technical Writer
- A Different Developer/New Hire/Intern

Issues with This Approach

- Documentation is almost an afterthought.
- Long release cycles can lead to things being forgotten.
- The more layers of separation between the implementer and the author, the more likely for inaccurate docs.
- The developer dislikes documenting.

Why Do Developers Dislike Writing Documentation?

- Developers enjoy writing code….
- Developers enjoy talking about their code….

- ….so why do developers hate writing about their code?

The Real Issue

- Most developers don't dislike writing documentation. What they dislike is the workflow.
  - A developer has to switch tools when they want to document something (outside of in-code documentation).
  - This context-switch makes them reluctant to write docs, so the task gets pushed off to the very end.
- How can we integrate the documentation process into a workflow that developers will enjoy?

What If We Treat the Docs Like Code?

- What if instead of having our docs external to the code, it lived alongside it?
- What if we used Markup languages that developers already know instead of WYSIWYG editor?

What Do We Mean When We Say *"Treat Docs Like Code"?*

- Doc source files are stored in a version control system.
- Build the doc artifacts automatically.
- Ensure a trusted set of reviewers meticulously reviews the docs.
- Docs are tested, both for accuracy and functionality.
- Publish the artifacts without much human intervention.

What Do We Gain From This?

- Promotes collaboration.
- Track documentation mistakes as bugs.
- Include docs in code review.
- Make beautiful, uniform docs.
- Leverage developer tools and workflows.
- Empowers developers to document.

Case Study

- A new team was formed at HomeAway/Vrbo to build a new product.
  - New GitHub org, new team members, everything
- First repository added to org was a documentation repository.
- The docs were the most up to date and well maintained in the department.

Case Study

Jared Wallace 12:48 PM

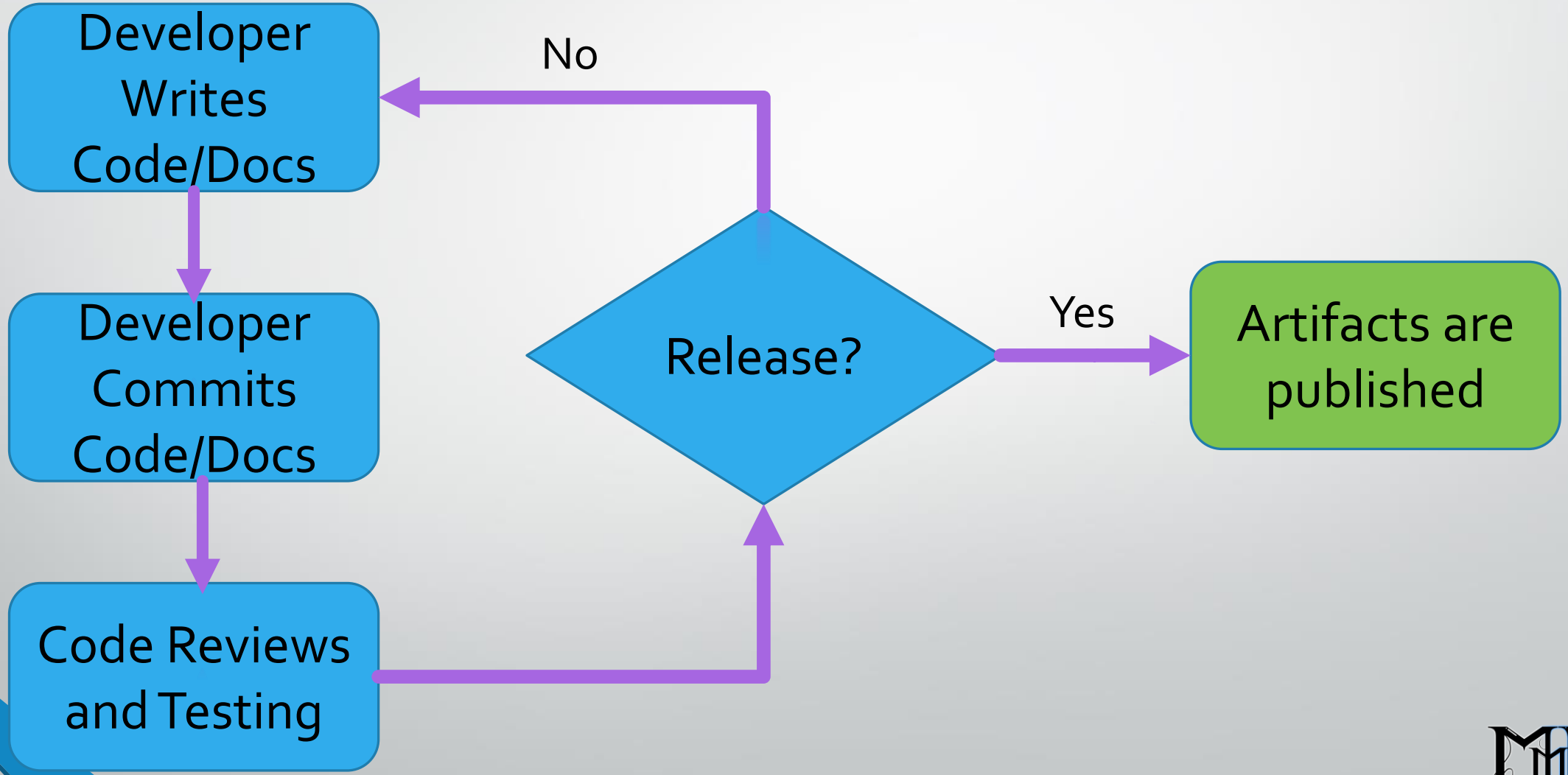
Just wanted to say, the presentation you gave about writing docs as code has led to nothing short of a transformation of how our team functions. We have continued, over the last six months, to improve and expand both our internal team documentation and our “external” facing (company facing) documentation. We have now 29 separate sections in the external docs, and 111(!) in the internal.

From high level overviews, to architecture diagrams, to detailed guides and requirements, our support burden has been greatly reduced, as we can now quickly point to consistently correct and clear documentation for most common questions.

Finally, from our teams perspective, we have reduced time to onboard a new member down to just a few days - thanks to comprehensive docs on gaining access to critical systems, to detailed runbooks on setting up the developer environment, to exact steps to resolve common issues, to detailed explanations of alerts and metrics and what they measure - I could go on and on.

I have had several comments from other teams who admire our docs, and who, after seeing ours, began their own effort to do the same. I think that’s perhaps the most powerful testament, that folks who did not hear the original presentation, were so impressed by the results that they reached out for information on how to bring that effort back to their teams.

How Does This Change the Workflow?



Tooling and CI/CD for Documentation

What is Continuous Integration and Continuous Deployment (CI/CD)?

- Continuous Integration (CI) means that code is continuously tested, integrated with other code changes, and merged. [1]
- Continuous deployment (CD) means that code is continuously deployed with each patch to the entire code base. [1]

1 Gentle, Docs like Code

What Does This Mean for Docs?

For docs, CI/CD means:

- Building a full artifact of the docs with each patch.
- Continuously testing content with each patch.
- Publishing automatically with every release.
- Versioning of the docs.

Types of Documentation

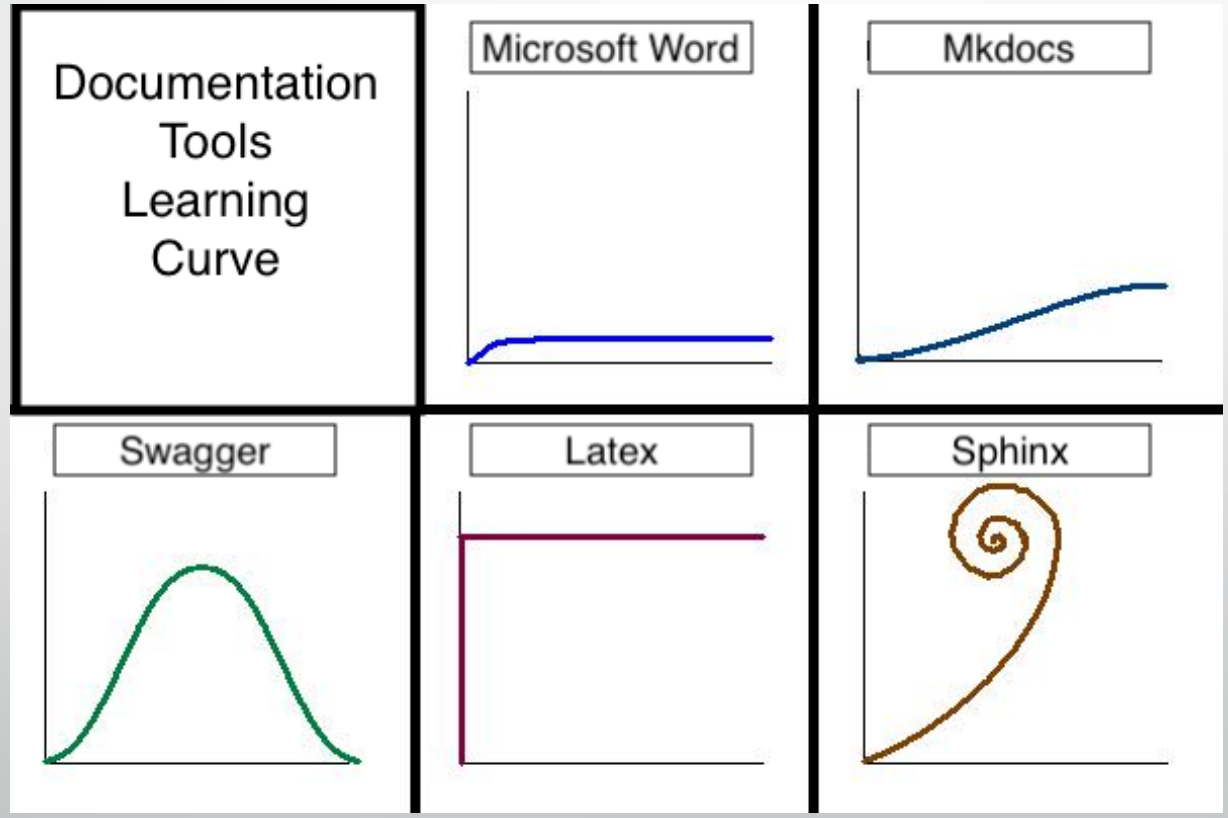
- Long form documentation – User Guides, Getting Started, FAQs, etc.
- Functional documentation – RESTful APIs, SDKs, man pages.

Documentation Tools

- Static site generators
  - Good for long form documentation, FAQs, runbooks.
- Source code based documentation generators
  - Documentation lives inside the code, pydoc, Javadoc.
  - Some even generate clients for testing like Swagger.
- System documentation generators
  - ronn – markdown based man page generator.

Documentation Tools

- The more powerful your documentation tool is, the more complex it is to use.



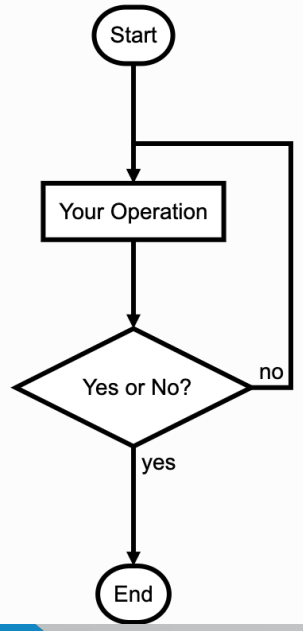
Documentation Tools - Mkdocs

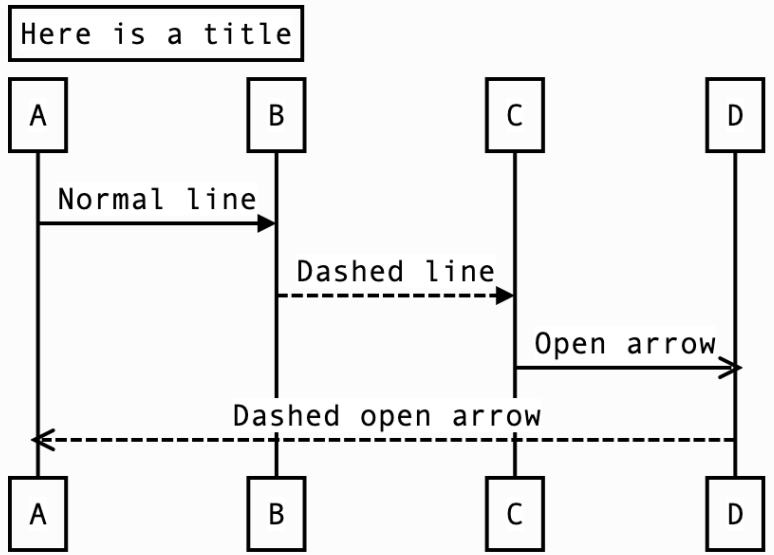
- Markdown based documentation, yaml based config file.
- Time to Hello World is ~30 seconds.
- Easy to configure, many extensions and themes supported.
- Python based. Easy to extend and modify.

Mkdocs – Extensions - UML

Flow Chart

flowchart.js documentation and live editor

```
```flow
st=>start: Start
op=>operation: Your Operation
cond=>condition: Yes or No?
e=>end

st->op->cond
cond(yes)->e
cond(no)->op
```
```



Sequence Diagram

js-sequence-diagrams documentation and live editor

```
```sequence
Title: Here is a title
A->B: Normal line
B-->C: Dashed line
C->>D: Open arrow
D-->>A: Dashed open arrow
```
```



Documentation Tools - Sphinx

- reStructured Text based documentation tool with support for Markdown.
- Most common tool for creating SDK documentation from in-code documentation.
- Can output to literally *any* media format you can imagine, including Latex.
- Might be sentient, currently uncertain.

Sphinx – Documentation Testing

```python
def hello_python():
    """A Hello World function.

    .. note:: These docs are generated with Sphinx.

    :Example:

    >>> from megger_test import megger_test
    >>> megger_test.hello_python()
    Hello Python Library
    """
    print("Hello Python Library")
```

```
Document: megger_test.megger_test
---------------------------------
1 items passed all tests:
   7 tests in default
7 tests in 1 items.
7 passed and 0 failed.
Test passed.

Doctest summary
===============
    7 tests
    0 failures in tests
    0 failures in setup code
    0 failures in cleanup code
build succeeded.
```

Documentation Tools - ronn

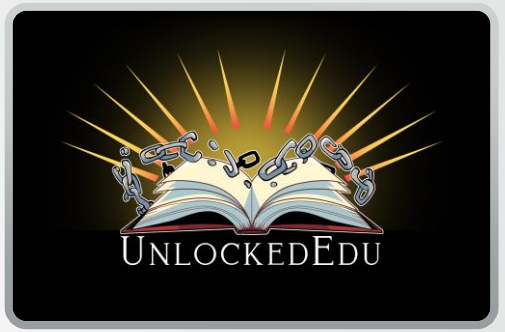
```
ronn(1) -- convert markdown files to manpages
==========================================

## SYNOPSIS

`ronn` [<format>...] <file>...<br>
`ronn` `-m`|`--man` <file>...<br>
`ronn` `-S`|`--server` <file>...<br>
`ronn` `--pipe` <file><br>
`ronn` &lt; <file>

## DESCRIPTION

**Ronn** converts textfiles to standard roff-formatted UNIX manpages or HTML.
ronn-format(7) is based on markdown(7) but includes additional rules and syntax
geared toward authoring manuals.

In its default mode, `ronn` converts one or more input <file>s to HTML or roff
output files. The `--roff`, `--html`, and `--fragment` options dictate which
output files are generated. Multiple format arguments may be specified to
generate multiple output files. Output files are named after and written to the
same directory as input <file>s.
```

Documentation Tools - ronn

```
RONN(1)                                                          RONN(1)



NAME
       ronn - convert markdown files to manpages

SYNOPSIS
       ronn [format...] file...
       ronn -m|--man file...
       ronn -S|--server file...
       ronn --pipe file
       ronn < file

DESCRIPTION
       Ronn  converts  textfiles  to  standard roff-formatted UNIX manpages or
       HTML. ronn-format(7) is based on markdown(7)  but  includes  additional
       rules and syntax geared toward authoring manuals.
```

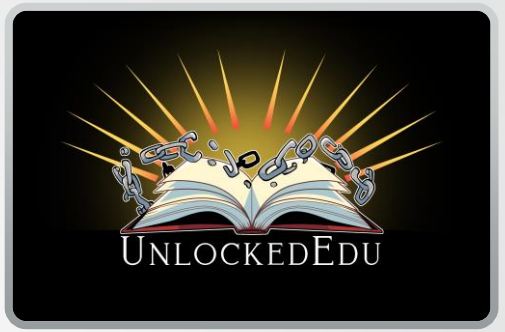
DEMO

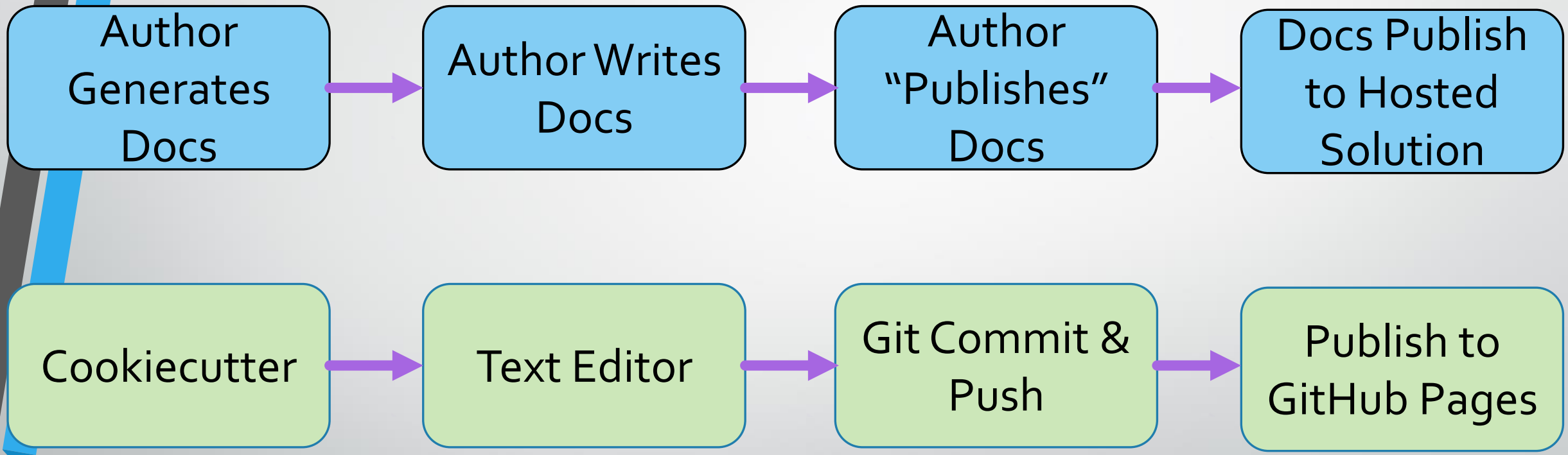
My Issue



- Need to create many open source texts, all with a similar format, that is production ready out of box.
  - I don't want anyone to worry about building the texts. They should just appear.
- Want a workflow that jump-starts docs writers.

My Solution

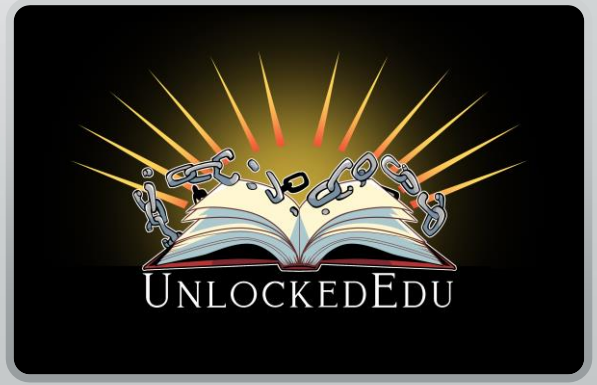

UnlockedEdu


Author Generates Docs
Author Writes Docs
Author "Publishes" Docs
Docs Publish to Hosted Solution
Cookiecutter
Text Editor
Git Commit & Push
Publish to GitHub Pages

Can I Try This Myself?

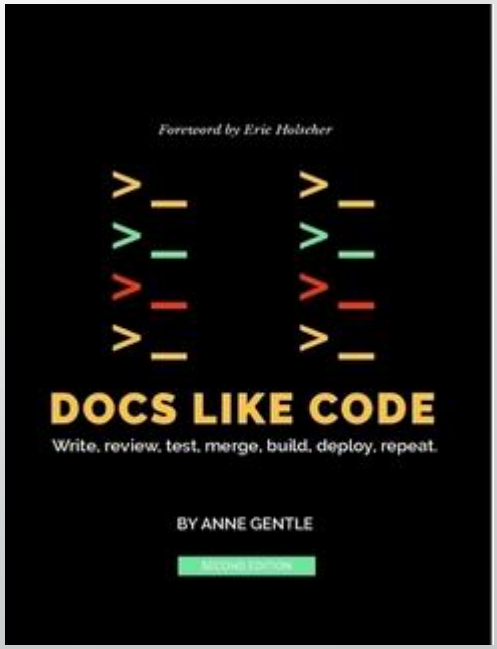
- UnlockedEdu is an Open Source project dedicated to create free and open source educational resources (such as textbooks, curriculums, worksheets, etc.) for use in schools.
- All books are written in Markdown with Mkdocs
- A cookiecutter (project generator written in python) has been created to setup an entire Documentation Pipeline.

https://github.com/UnlockedEdu/documentation-pipeline-generator



Sources

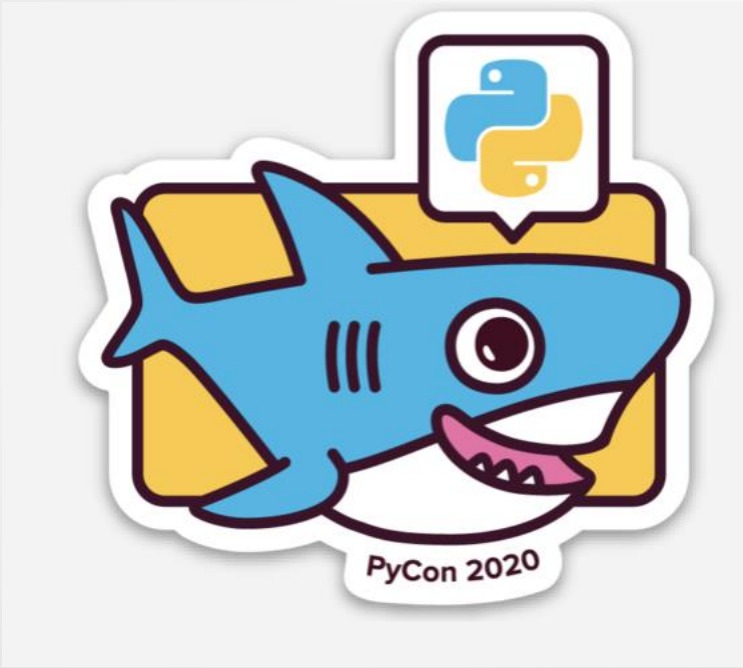
- Gentle, Anne, et al. *Docs like Code: Write, Review, Test, Merge, Build, Deploy, Repeat*. Just Write Click, 2017.



Final Thoughts

- Every job that I've implemented this workflow at, both developer experience and user docs have vastly improved.
- Stop making docs a punishment.
- If your docs suck, people will abandon your project.
- Versioning docs is great. We should do that a lot more.
- Slides will be available at https://www.masonegger.com/talks/ci-docs/

But Wait! There's More



Get your PyCon 2020 Sammy Sticker!

https://do.co/pycon2020