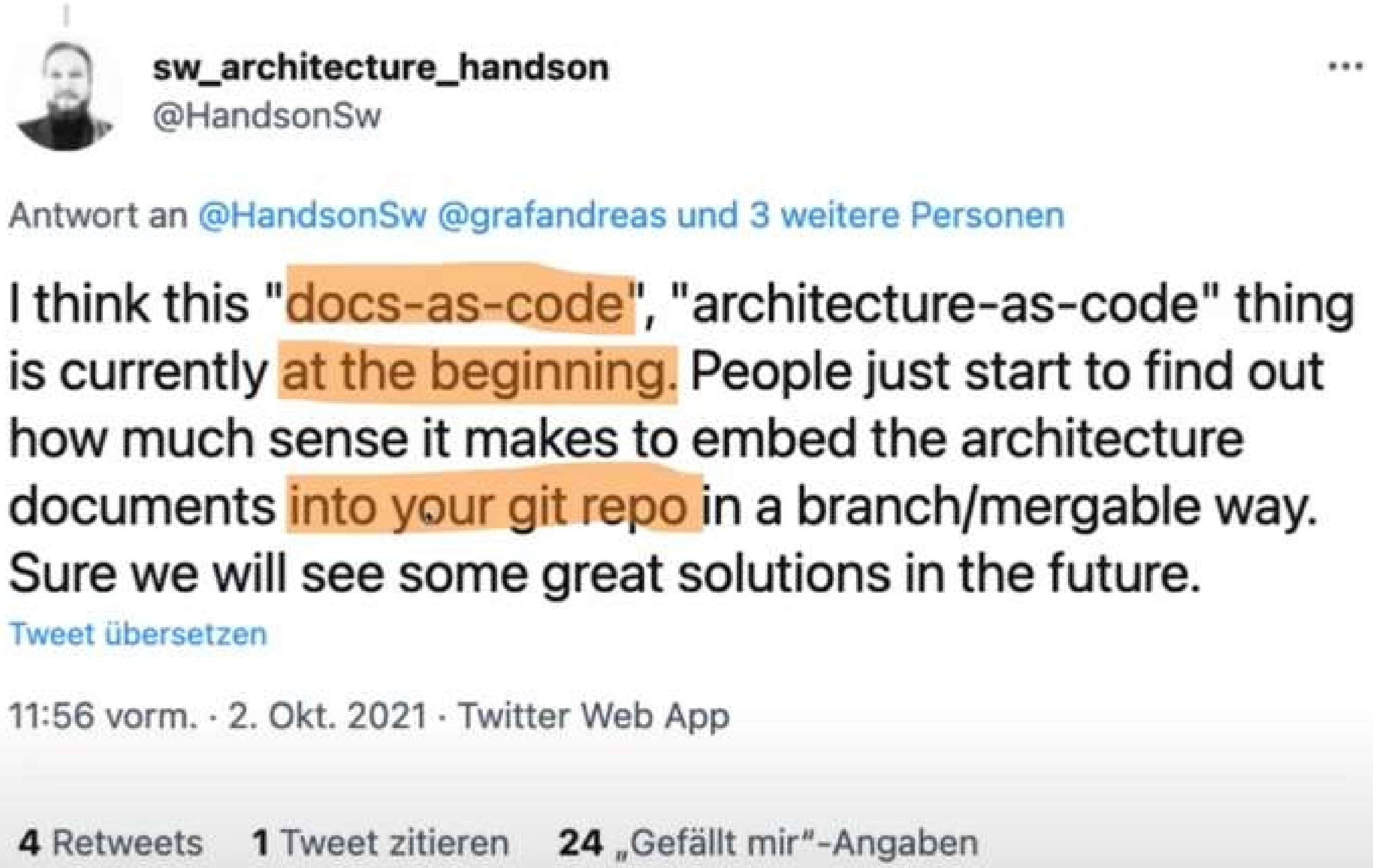
Docs-as-Code with Sphinx

Arnoldas Bagdonas at gmail dot com



sw_architecture_handson
@HandsonSw

Antwort an @HandsonSw @grafandreas und 3 weitere Personen

I think this "docs-as-code", "architecture-as-code" thing is currently at the beginning. People just start to find out how much sense it makes to embed the architecture documents into your git repo in a branch/mergable way. Sure we will see some great solutions in the future.

Tweet übersetzen

11:56 vorm. · 2. Okt. 2021 · Twitter Web App

4 Retweets **1** Tweet zitieren **24** „Gefällt mir"-Angaben

Introduction

- What is Docs-as-Code?

  - A methodology where documentation is created, reviewed, versioned, and published using the same workflows as software source code.
  - Uses Git, CI/CD, code review, linters, automation.

- Why Sphinx?

  - Originally built for Python, but language-agnostic.
  - Powerful ecosystem (themes, extensions).
  - Generates high-quality HTML/PDF/epub.
  - Integrates deeply with codebases.

The Docs-as-Code Philosophy

- Documentation lives in the repo → not a separate tool/wiki.

- Write docs in plain text (reStructuredText or Markdown).

- Automation handles the publishing.

- PRs drive documentation quality:
    - Proposed changes → code reviewers evaluate technical accuracy and clarity.
    - Docs stay versioned alongside code.

The Docs-as-Code Benefits

- Version control: docs tied to code versions.

- Transparency: anyone can diff/read history.

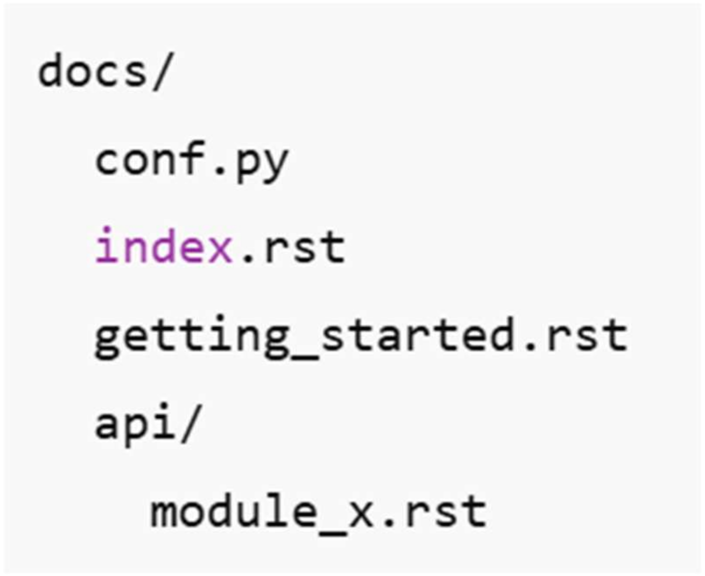
- Collaboration: engineers treat doc changes like feature changes.

- No vendor lock-in.

Why Sphinx Works Well

- Autodoc: generates API docs automatically from docstrings.
- Doctest: executes example code snippets in the documentation to ensure they run exactly as written.
- Cross-referencing between APIs, modules, or external packages.
- Customizable themes.
- Plugin ecosystem.
- Output Formats.
- CI/CD Integration.

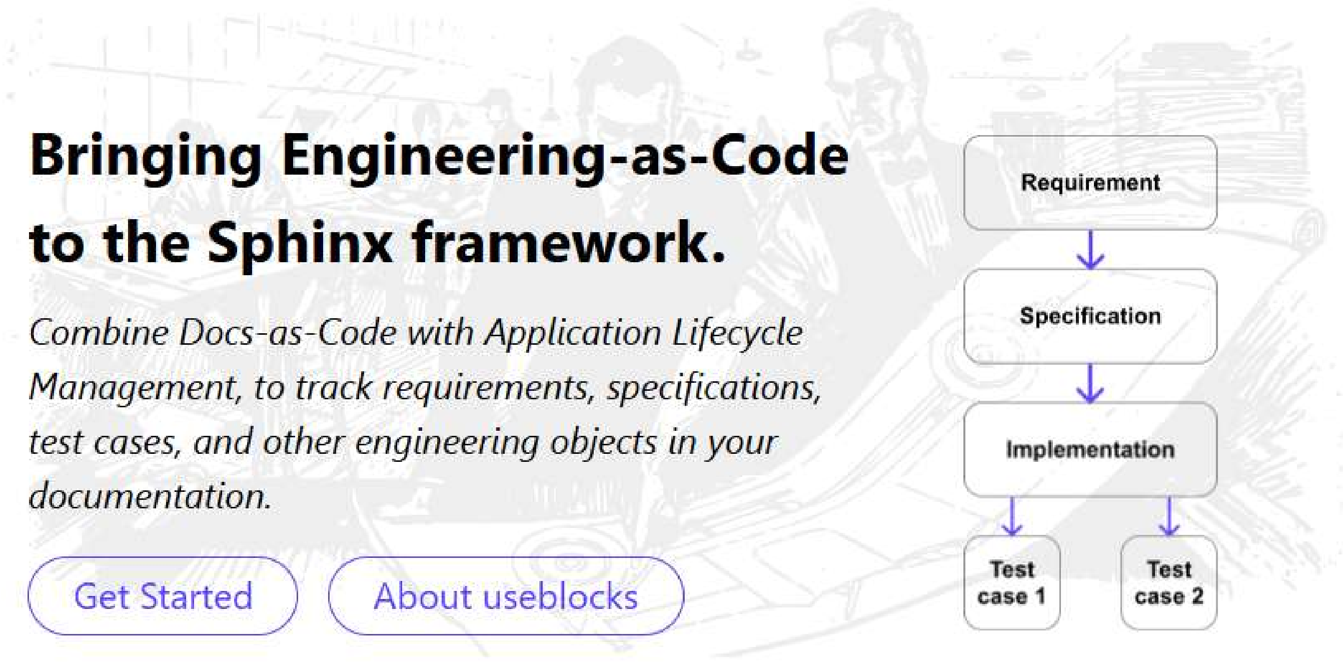
Basic Workflow Example



```
docs/
  conf.py
  index.rst
  getting_started.rst
  api/
    module_x.rst
```

- Developer workflow:
  1. Create or modify .rst/.md files.
  2. Run make all.
  3. Preview locally.
  4. Commit and open a PR.
  5. CI builds docs and reports issues.
  6. Merge → docs auto-deploy.

Example Use Case


Bringing Engineering-as-Code to the Sphinx framework.

Combine Docs-as-Code with Application Lifecycle Management, to track requirements, specifications, test cases, and other engineering objects in your documentation.

https://sphinx-needs.readthedocs.io/en/latest/

Limitations to Acknowledge

- rST’s learning curve.

- Theme customization can be tricky.

- Autodoc works best with Python.

- Requires developer buy-in (but Docs-as-Code helps with that).

Best Practices

- Keep docs close to code — same PR if possible.

- Use clear structure + consistent style guide.

- Write examples and quickstarts early.

- Automate everything (formatting, broken link checks).

- Review docs as part of definition of done.

Conclusion

- Docs-as-Code + Sphinx enables:

  - Continuous, versioned, trustworthy documentation.
  - Reduced drift between code and docs.
  - Developer-friendly workflows.
  - Automated, repeatable publishing.