DOCUMENTALITY

Common pitfalls & best practices
for technical documentation

$whoami

Hadar Cohen

C++ developer

Product ops @Datree

Developer advocate @Port

Experience documentation from both sides

What's on the agenda?

- Why is documentation more critical than you think?
- Common mistakes
- Best practices
- Tools & integrations

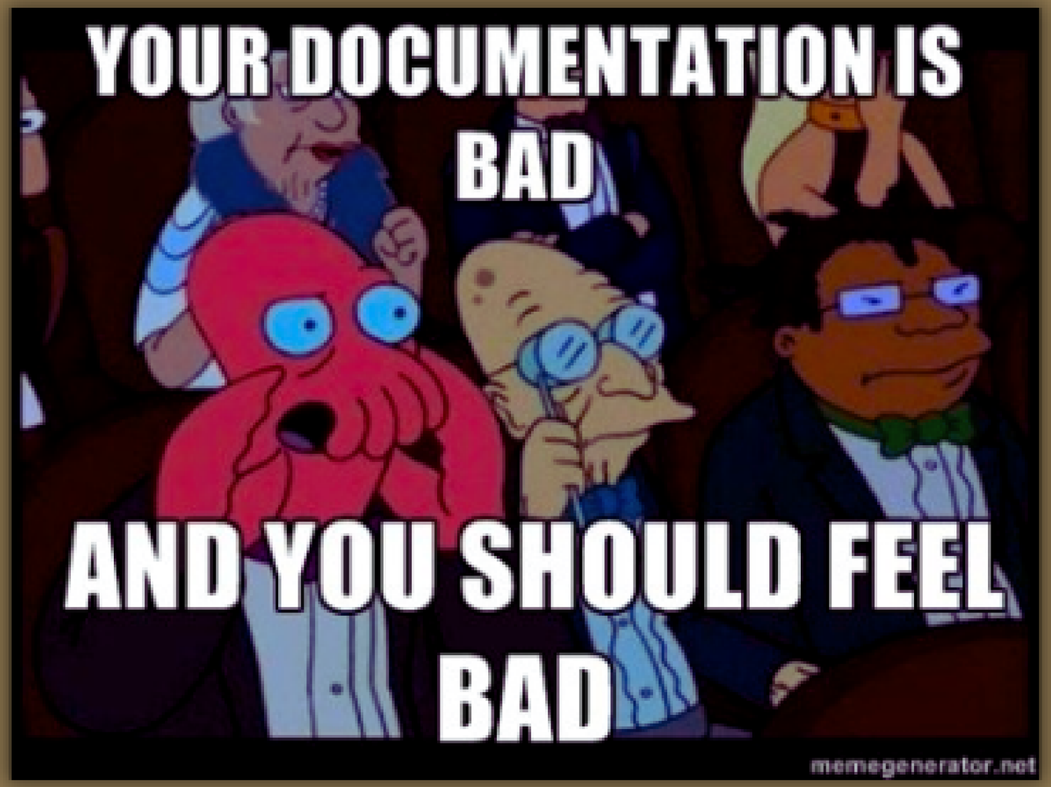
We all write documentation!

We all write documentation!

- Code comments
- PRs
- API documentation
- Github readme
- New feature docs
- Product videos/blogs

We all write documentation!

- Code comments
- PRs
- API documentation
- Github readme
- New feature docs
- Product videos/blogs



We've all experienced bad docs...
Don't be that guy!

Why is documentation so critical?

Why is documentation so critical?

- An early and critical touch point

Why is documentation so critical?

- An early and critical touch point
- Bad documentation can be an immediate turn-off

Why is documentation so critical?

- An early and critical touch point
- Bad documentation can be an immediate turn-off
- Build trust and engagement with your users

Why is documentation so critical?

- An early and critical touch point
- Bad documentation can be an immediate turn-off
- Build trust and engagement with your users
- Prevent "kitbag" questions

Dawid Ostrowski • 2nd
Head of Google Developer Experts Program
6mo • Edited •

+ Follow

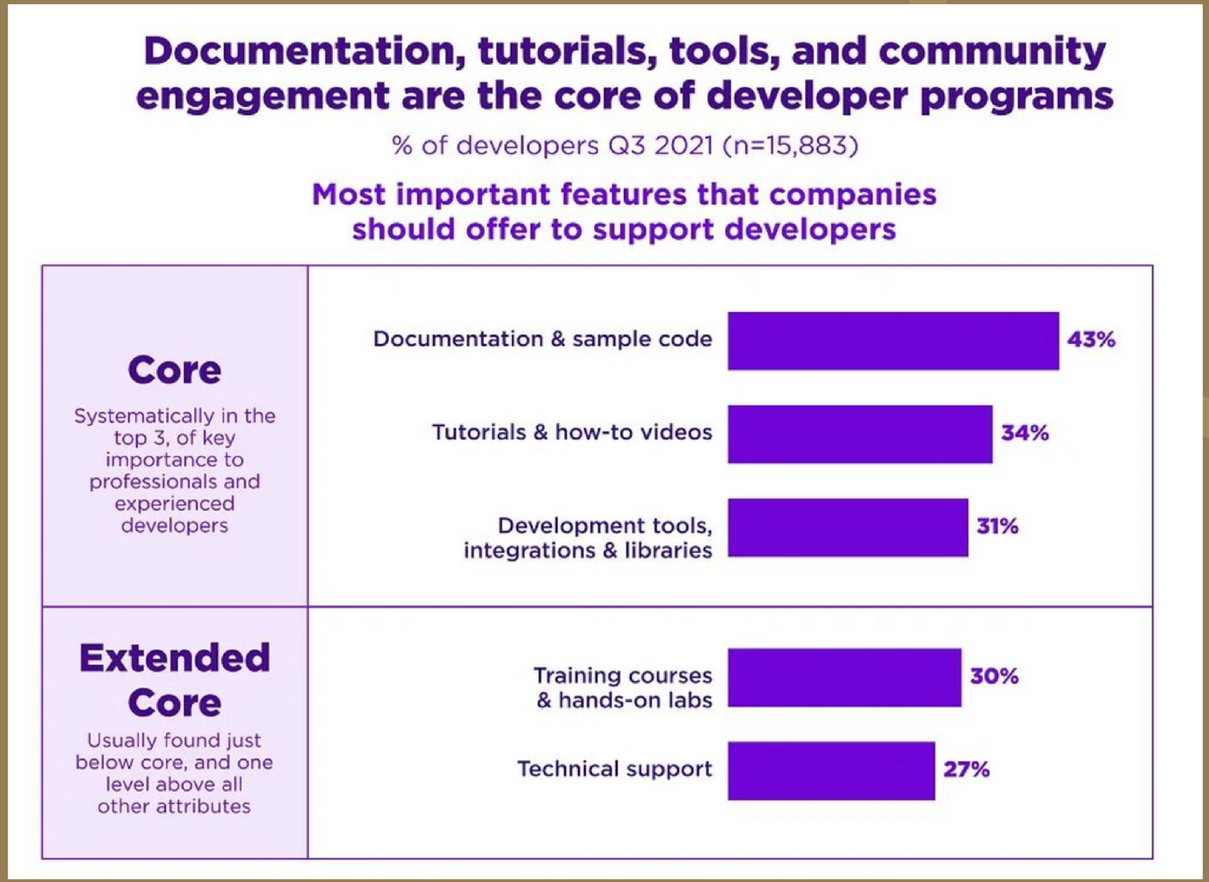
Weekend insights - What developers really need?

What are the most important features that developer programs should offer to support developers?

Must haves:
1. Documentation & sample code
2. Tutorials & how-to videos
3. Development tools, integrations & libraries

Also important:
1. Training courses & hands-on labs
2. Technical support

[Developer Nation Q3 2021 - SlashData]



Common Mistakes

Submitting patches: the essential guide to getting your code into the kernel

For a person or company who wishes to submit a change to the Linux kernel, the process can sometimes be daunting if you're not familiar with "the system." This text is a collection of suggestions which can greatly increase the chances of your change being accepted.

This document contains a large number of suggestions in a relatively terse format. For detailed information on how the kernel development process works, see A guide to the Kernel Development Process. Also, read Linux Kernel patch submission checklist for a list of items to check before submitting code. For device tree binding patches, read Submitting Devicetree (DT) binding patches.

This documentation assumes that you're using `git` to prepare your patches. If you're unfamiliar with `git`, you would be well-advised to learn how to use it, it will make your life as a kernel developer and in general much easier.

Some subsystems and maintainer trees have additional information about their workflow and expectations, see Documentation/process/maintainer-handbooks.rst.

Obtain a current source tree

If you do not have a repository with the current kernel source handy, use `git` to obtain one. You'll want to start with the mainline repository, which can be grabbed with:

```
git clone git://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git
```

Note, however, that you may not want to develop against the mainline tree directly. Most subsystem maintainers run their own trees and want to see patches prepared against those trees. See the **T:** entry for the subsystem in the MAINTAINERS file to find that tree, or simply ask the maintainer if the tree is not listed there.

Describe your changes

Describe your problem. Whether your patch is a one-line bug fix or 5000 lines of a new feature, there must be an underlying problem that motivated you to do this work. Convince the reviewer that there is a problem worth fixing and that it makes

Literature & Whitepapers

- ScreenFlow Product Sheet
- Video Tutorial Solution Brief

Customer Case Studies

- Adrian Tucker, Spanish Musician, YouTuber, and Influencer, Builds A Loyal Community with ScreenFlow
- Aaron Nace of Phlearn
- David Besozzi, Educator
- David Chartier, AgileBits
- Jeffrey Bradbury of TeacherCast.net
- Joshua Rosenbaum of MailChimp
- Full Sail University selects ScreenFlow for Media Design curriculum
- Peter Swartz with MacProVideo.com
- Piano Groove uses ScreenFlow to create videos for step-by-step online jazz piano instruction

Documentation

- ScreenFlow Online Help
- ScreenFlow User Guide
- ScreenFlow Tutorial
- ScreenFlow Keyboard Shortcuts
- Telestream Premium Support Guide

Video Tools

- Made with Screenflow Intro/Outro

Literature & Whitepapers

ScreenFlow Produ

Video Tutorial Solu

Custome

Adrian Tucker, Spa
Community with S

Aaron Nace of Phl

David Besozzi, Edu

David Chartier, Ag

Jeffrey Bradbury c

Joshua Rosenbau

Full Sail University

Peter Swartz with

Piano Groove uses
piano instruction

Documentation

Home About Download Documentation Community Developers Support Donate Your account

31st August 2023: PostgreSQL 16 RC1 Released!

Quick Links

- Documentation
- Manuals
  - Archive
- Release Notes
- Books
- Tutorials & Other Resources
- FAQ
- Wiki

Documentation

View the manual

Manuals

You can view the manual for an older version or download a PDF of a manual from the below table.

Online Version	PDF Version
16 rc	A4 PDF (14.2 MB) • US PDF (14.1 MB)
15 / Current	A4 PDF (13.6 MB) • US PDF (13.5 MB)
14	A4 PDF (13.4 MB) • US PDF (13.2 MB)
13	A4 PDF (13.0 MB) • US PDF (12.9 MB)
12	A4 PDF (12.7 MB) • US PDF (12.6 MB)
11	A4 PDF (12.4 MB) • US PDF (12.3 MB)
Development snapshot	PDF version not available

DealerPeak

Product Resources Support About Contact Us

Request Demo

How does our API work?

1. A client application initiates an API call to retrieve information also known as a request. This request is processed from an application to the web server via the API's Uniform Resource Identifier (URI) and includes a request verb, headers, and sometimes, a request body.

2. After receiving a valid request, the API makes a call to the external program or web server.

3. The server sends a response to the API with the requested information.

4. The API transfers the data to the initial requesting application.

The data transfer will vary depending on the web service being used. Regardless, this process of requests and response all happens through an API.

A user interface is designed for use by humans; APIs are designed for use by a computer or application.

API documentation available upon request.

oops

NodeGit

Getting Started | Guides | API Documentation | Source

API Docs

Version 0.26.1

AnnotatedCommit

Class Methods

AnnotatedCommit.**fromFetchhead** (repo, branch_name, remote_url, id) ASYNC

AnnotatedCommit.**fromRef** (repo, ref) ASYNC

AnnotatedCommit.**fromRevspec** (repo, revspec) ASYNC

AnnotatedCommit.**lookup** (repo, id) ASYNC

Instance Methods

AnnotatedCommit#**id** () SYNC

AnnotatedCommit#**ref** () SYNC

Apply

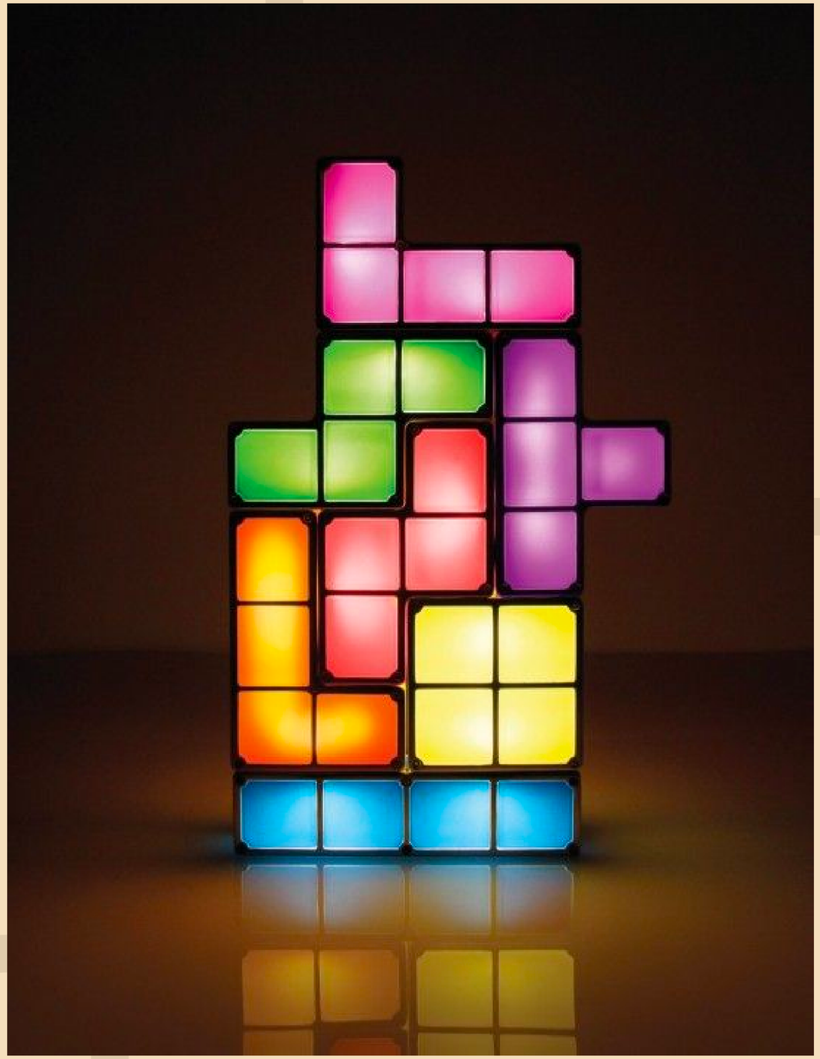
Class Methods

Apply.**apply** (repo, diff, location, options) SYNC EXPERIMENTAL

Apply.**toTree** (repo, preimage, diff, options) ASYNC EXPERIMENTAL

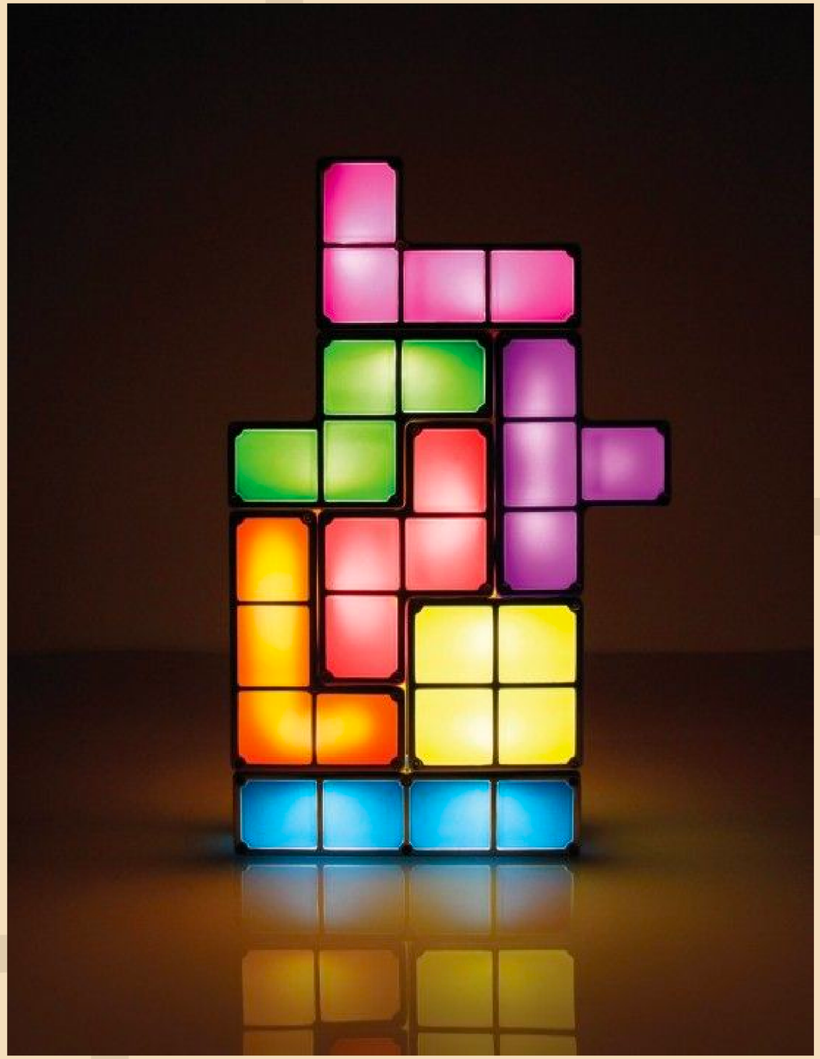
Poor structure

- Messy/long = unreadable



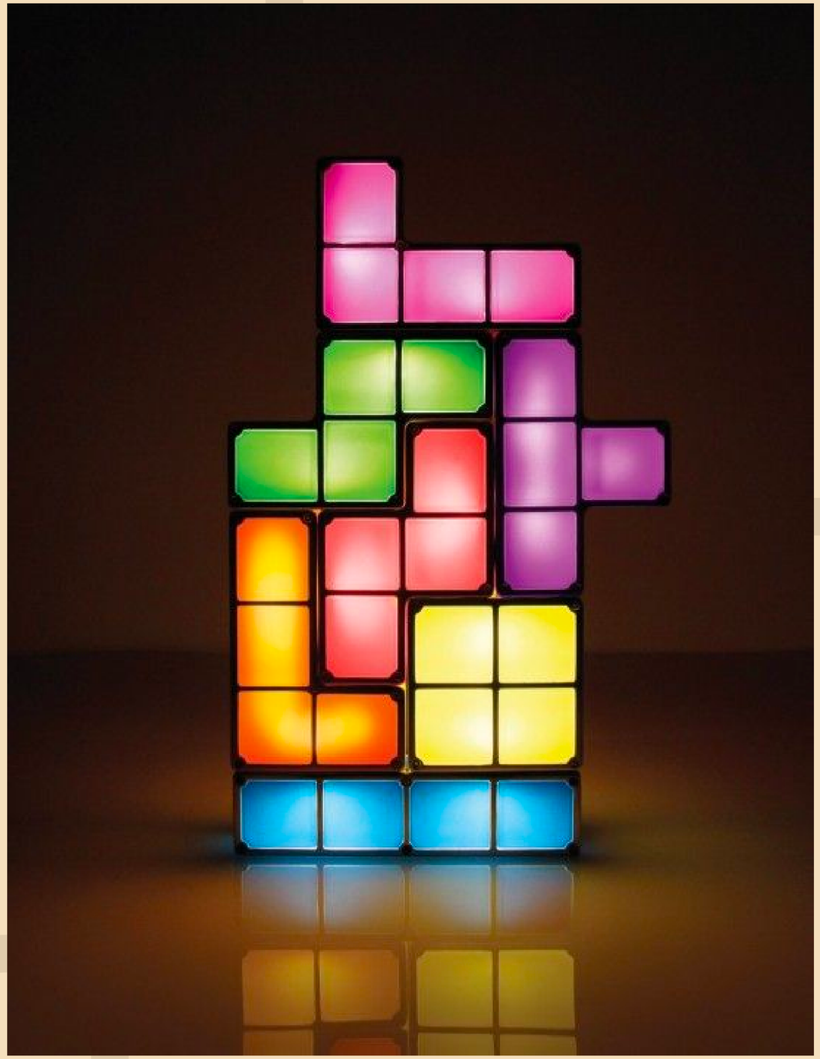
Poor structure

- Messy/long = unreadable
- Hard to find key information

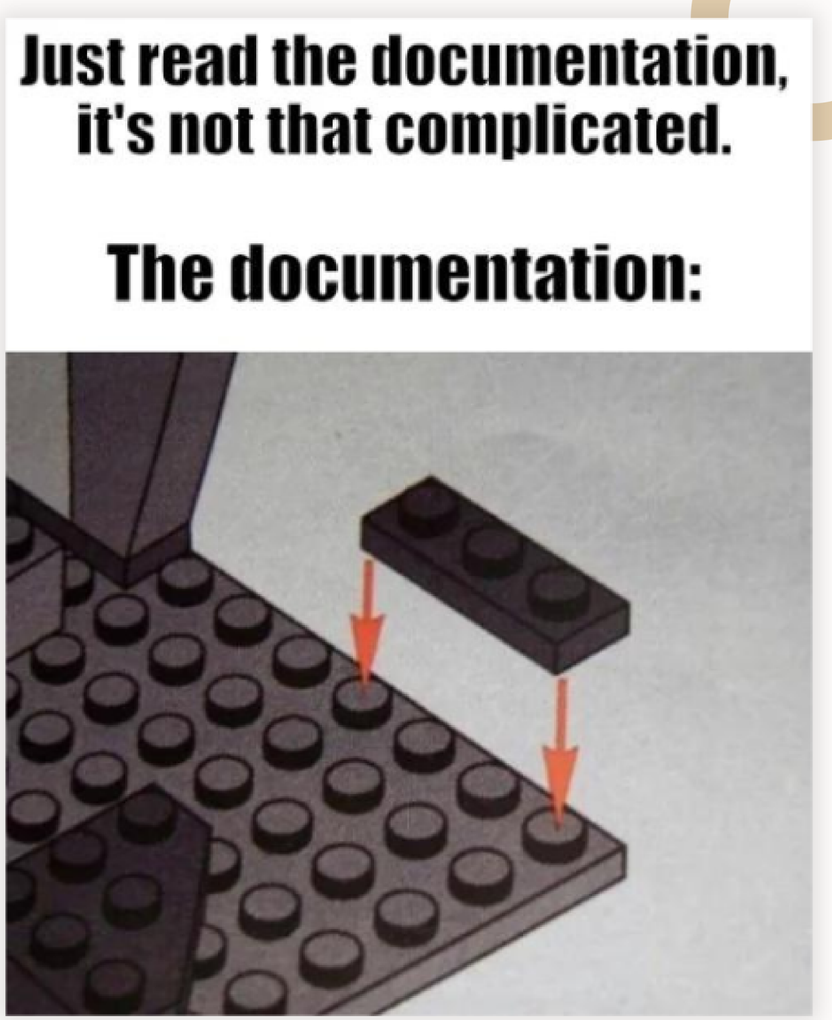


Poor structure

- Messy/long = unreadable
- Hard to find key information
- No basic description/information

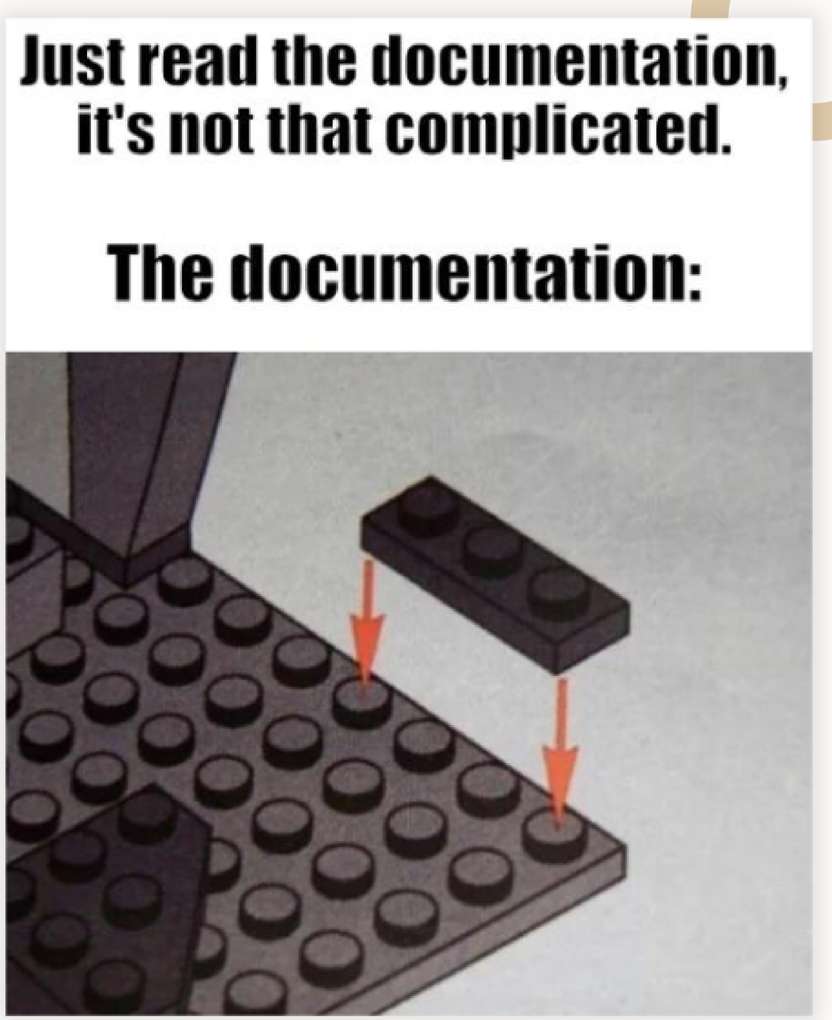


Inconsistent content



Inconsistent content

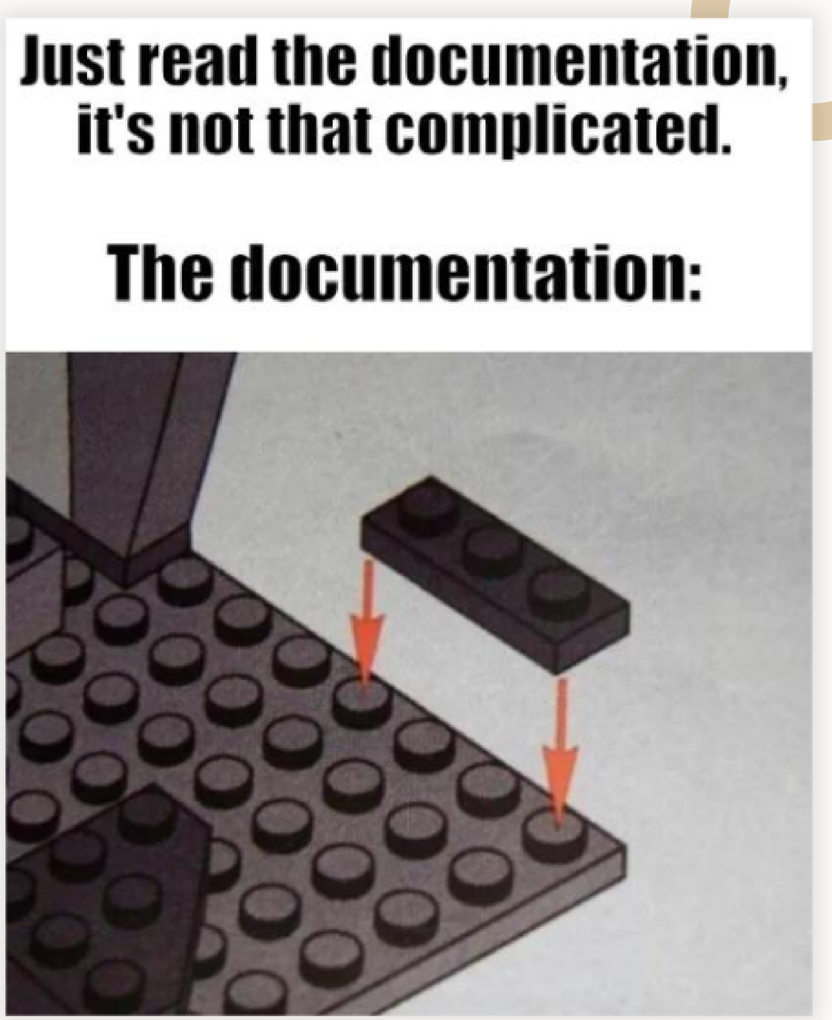
- Same information in multiple places
  Example: token



Inconsistent content

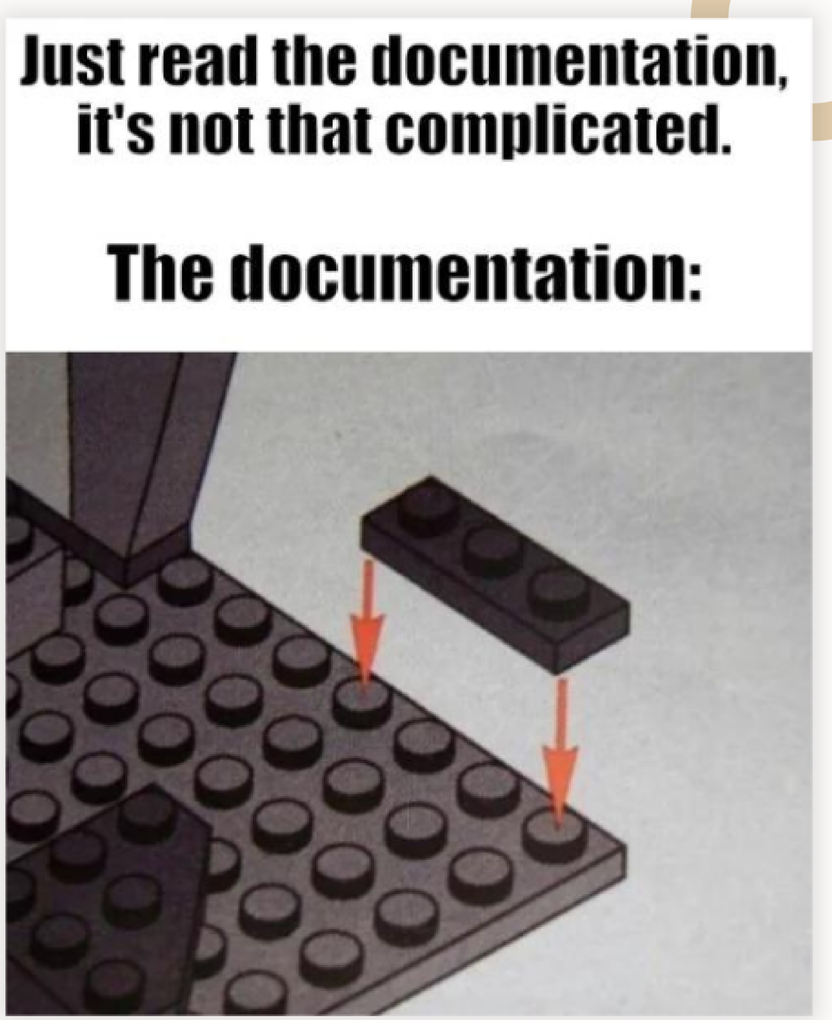
- Same information in multiple places
  Example: token

- Everybody's got a different flavor
  Example: CLI/terminal



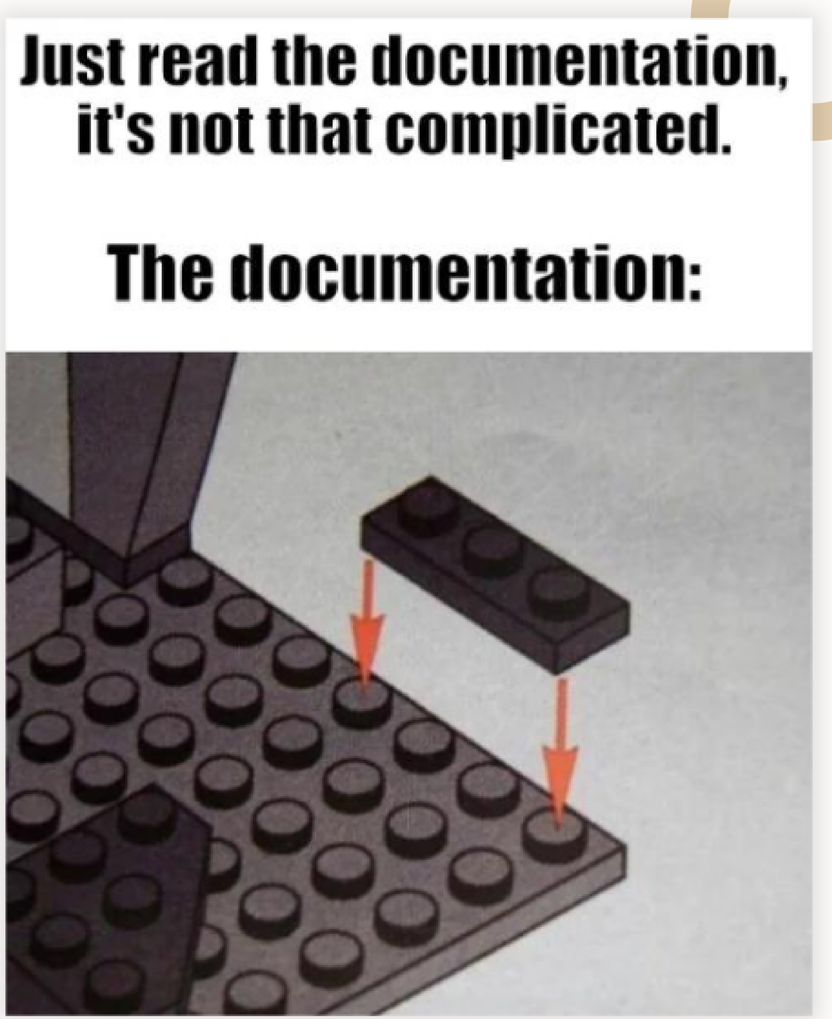
Inconsistent content

- Same information in multiple places
  Example: token
- Everybody's got a different flavor
  Example: CLI/terminal
- Information is not up-to-date
  Examples: broken links, outdated information



Inconsistent content

- Same information in multiple places
  Example: token
- Everybody's got a different flavor
  Example: CLI/terminal
- Information is not up-to-date
  Examples: broken links, outdated information
- Formal vs. informal language
  Examples: emojis, direct speak

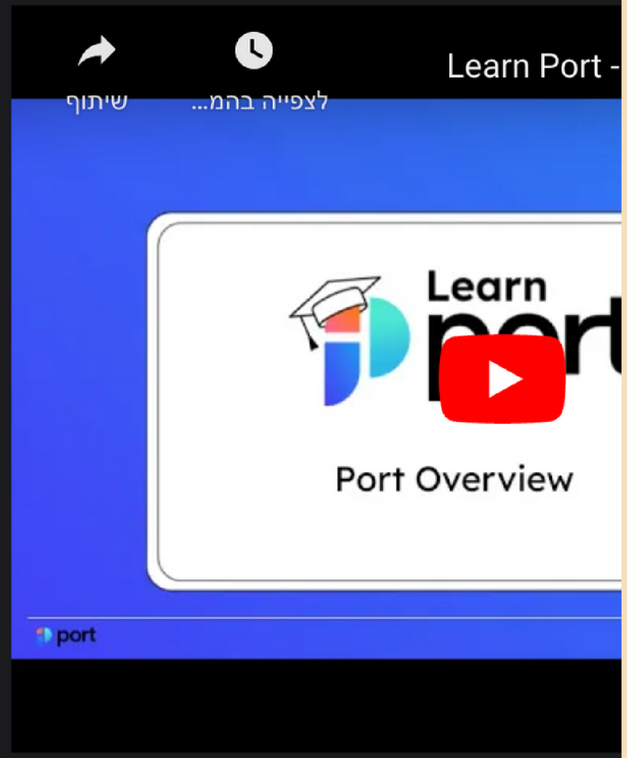


Port Documentation

- 👋 Port Overview
- ⏱️ Quickstart
- 🔧 Setup Methods
- 🏗️ Build Software Catalog
- ⚡ Create Self-Service Experiences
- 📊 Promote Scorecards
- ✏️ Customize Pages, Dashboards & Plugins
- 🔍 Search & Query
- 🔒 SSO & RBAC
- 🧪 API Reference
- ❓ FAQ
- 📚 Resources

🏠 › 👋 Port Overview

👋 Port Overview

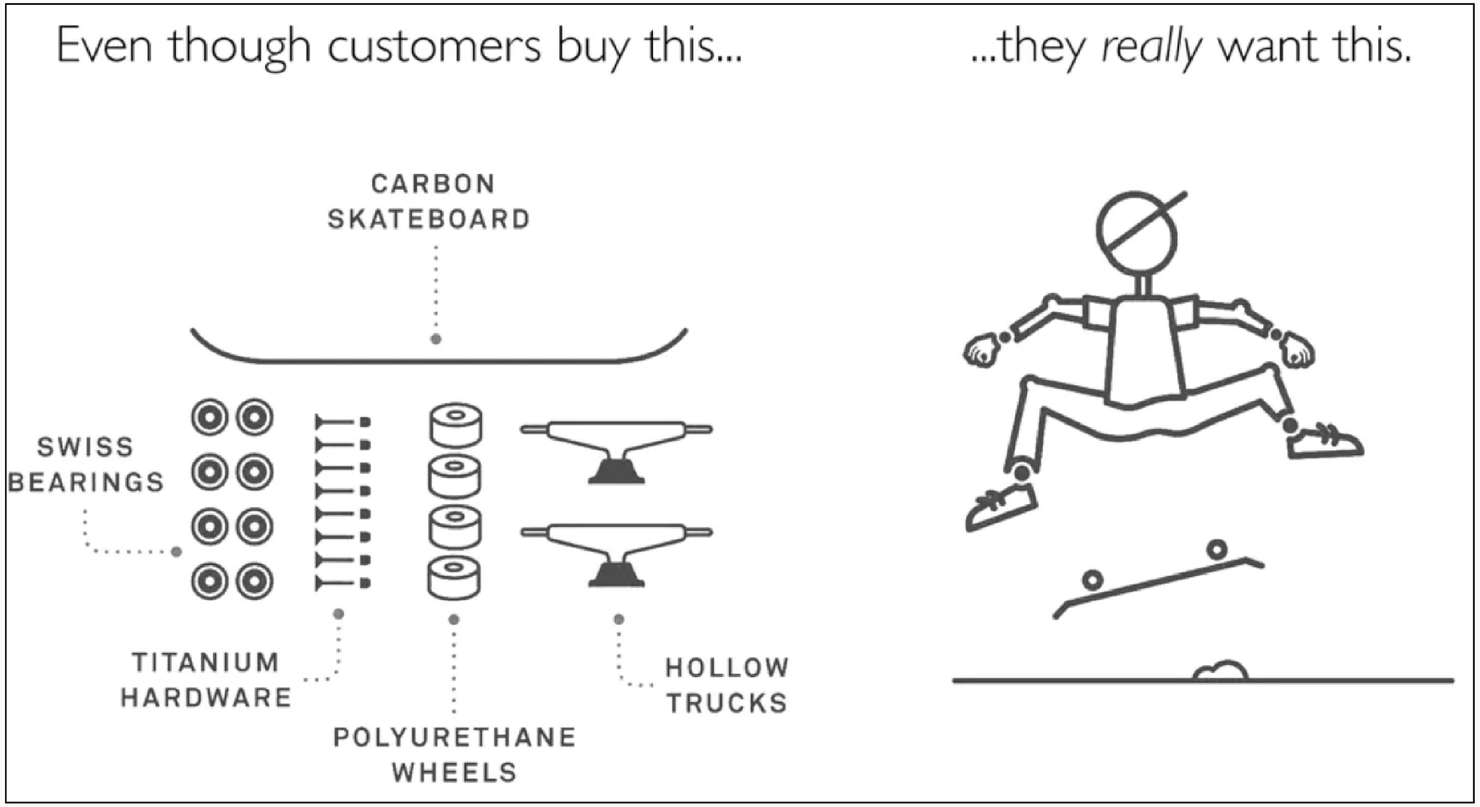


Best practices

JTBD

(job-to-be-done)

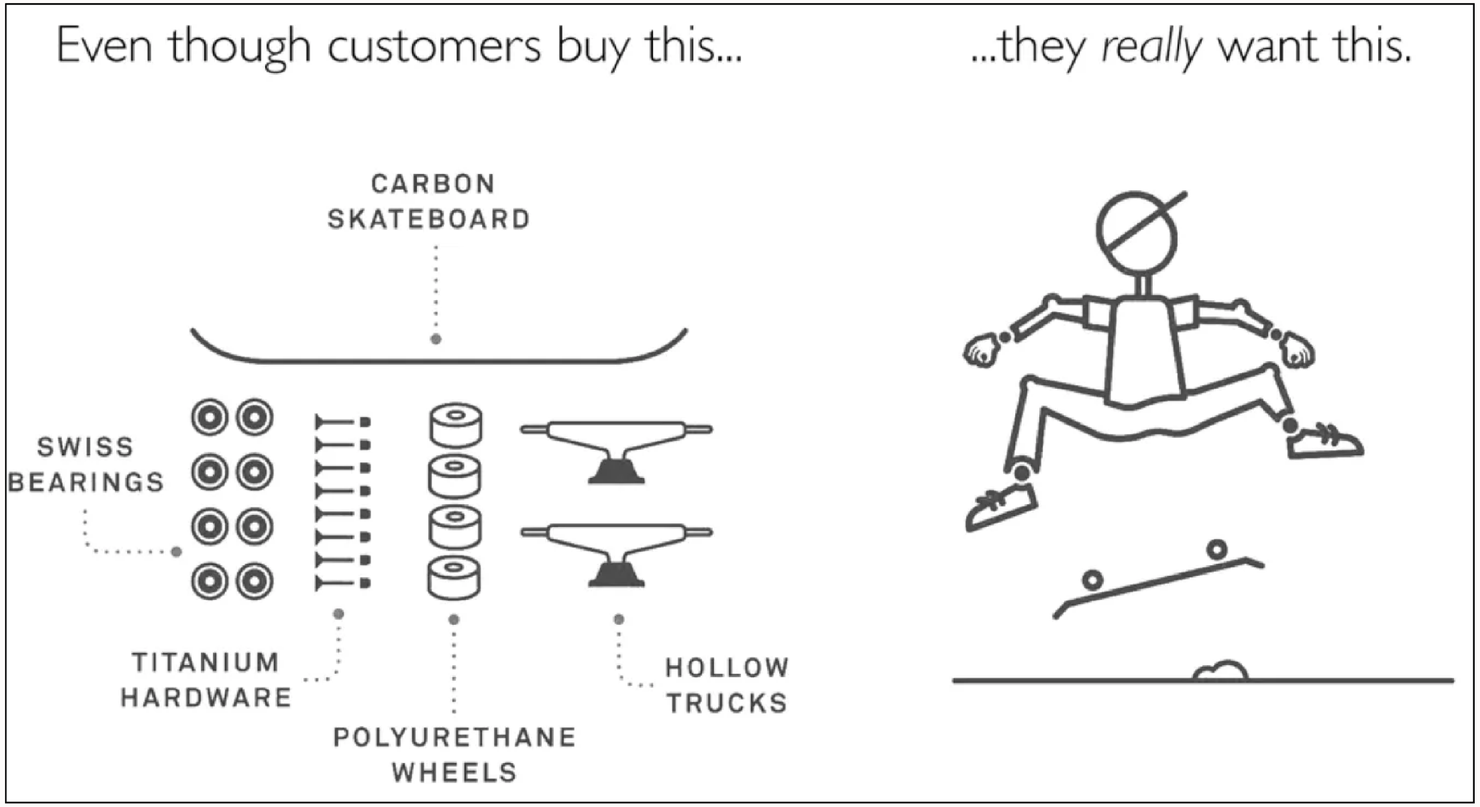
JTBD
(job-to-be-done)



JTBD
(job-to-be-done)

Who is my reader?

What does he/she need?



JTBD
(job-to-be-done)

HubSpot	❌ Users don't hire HubSpot to provide CRM software and marketing automation. ✅ They hire HubSpot to grow their business.
Spotify	❌ Users don't hire Spotify to browse, download, and stream media. ✅ They hire Spotify to easily and affordably listen to the music they love during the most mundane—and the most important—moments of their lives.
Verizon	❌ Users don't hire Verizon to make phone calls. ✅ They hire Verizon to reliably connect with their loved ones, no matter how far away they might be.
Research Hub	❌ Users don't hire Research Hub to automate screening, scheduling, and reminder messages. ✅ They hire Research Hub for the peace-of-mind that all the logistical aspects of a study are handled—without the time-suck of doing it manually.

Useful tips

Useful tips

- Examples

Useful tips

- Examples
- Search-oriented

Useful tips

- Examples
- Search-oriented
- Distinguish between concepts/information and tasks/CTA

Useful tips

- Examples
- Search-oriented
- Distinguish between concepts/information and tasks/CTA
- Rubber-ducking & peer review

Useful tips

- Examples
- Search-oriented
- Distinguish between concepts/information and tasks/CTA
- Rubber-ducking & peer review
- Visualize information correctly

Tools & Integrations

Tools & integrations

User engagement

giscus

INTERCOM

Platforms

Docusaurus

readme

AI tools

aicommits

Scribe

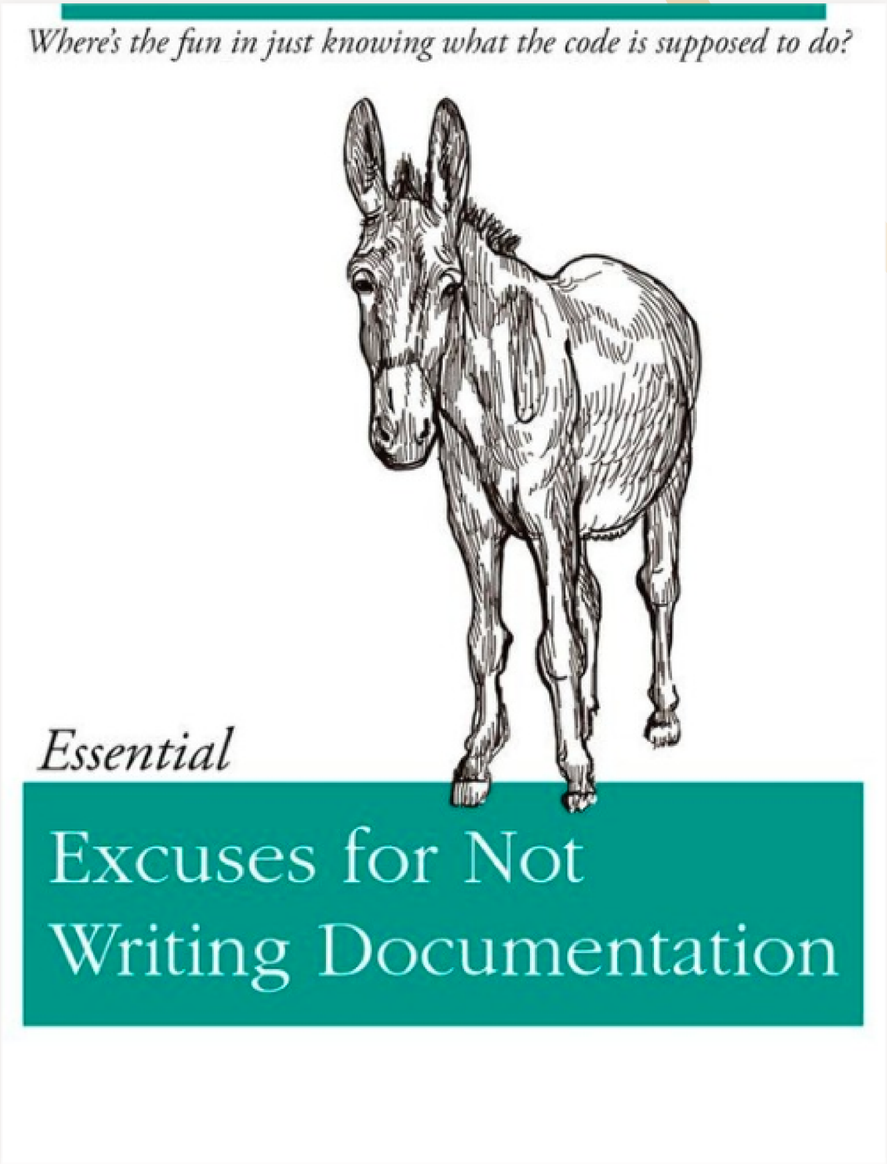
Visualization

snappify.io

Search

algolia

Thanks!



hadar@getport.io