Documentation for software developers

@PeterHilton

http://hilton.org.uk/

What’s wrong with software system documentation?

README-Driven Development

README Driven Development

'… we have projects with short, badly written, or entirely missing documentation…

There must be some middle ground between reams of technical specifications and no specifications at all.

And in fact there is.

That middle ground is the humble README.'

http://tom.preston-werner.com/2010/08/23/readme-driven-development.html

Why doesn’t software always have a good README?

Group exercise

1. Pick the top ten things to include in a README, one README section heading per sticky note.
2. Number your README sections 1 to 10 in order of importance.
3. Stick your notes in a column on the flip chart.
4. Choose one person per group to present your top 10.

Peter’s (current) top 10 README sections

1. What it is
2. Purpose
3. Usage/examples
4. Installation
5. Asking questions
6. Building from source
7. Contributing
8. Authors/maintainers
9. License
10. Testing

Other suggestions

Code of conduct
Status (does it work?)
Scope
Components
Architecture
Prerequisites
Jargon
Inputs & outputs

Deployment
Target users - who it’s for
How it works
Copyright information
Contributors
To do list
Change log
Features

Three questions to ask about any tool

1. Who invented this tool?
2. What problem was he/she trying to solve?
3. Do I have that problem?

‘Many people […] are prone to believing the people who developed the tools have done their thinking for them’

https://vanguard-method.net/library/systems-principles/three-questions-to-ask-about-any-tool/

Bonus exercise

1. Pick some GitHub repos, e.g. software you use
2. Note the repo URL and number of stars
3. Award 1 point for each section on my list the README has (What it is, Purpose, Usage/examples, Installation, Asking questions, Building from source, Contributing, Authors/maintainers, License, Testing)
4. Award 0.5 points for a link to the info on another page

Name	★★★	Score	What	Purpose	Usage	Install	Build	Ask	Contrib.	Authors	Test	License
httpie	28417	7.5	1	1	1	1		1	0.5	1		1
Homebrew	6431	6.0	0.5	0.5	0.5	0.5		1	1	1		1
validator.js	7223	6.0	1	1	1	1					1	1
fish	7215	5.5	1	0.5	1	0.5	1	1	0.5			
Moment	30208	5.5	1	1	0.5	0.5		0.5	1			1
Jekyll	28909	5.0	1	1	0.5	0.5		1	0.5			0.5
atom	35272	4.5	1	1		1	0.5					1
Lodash	21925	4.5	0.5	1	0.5	1			0.5			1
ImageOptim	3722	4.0	1	0.5	0.5	0.5	1			0.5		
Liquid	5316	3.5	1	1	1				0.5			
NodeCSV	1535	3.5	0.5	1	1	1						
Request	14548	3.5	0.5	1	1	1						

The README is
(the short version of)
the complete system
documentation

Technical writing techniques

Checklist of technical writing techniques

1. Start with the ‘why?’ - explain benefits up-front
2. Examples before definitions
3. Just-in-time, not just-in-case
4. Teach this stuff!
5. Don’t humiliate the reader
6. Handle or declare new concepts
7. ‘So what and why do I care?’

Exercise (in pairs)

1. Review the checklist on the handout.
2. Read the answers to the Stack Overflow question.
3. Use the checklist to find opportunities to improve the text.
4. Highlight the text and mark it with the checklist number.

A monad is a data type that encapsulates a value and to which essentially two operations can be applied:

- `return x` creates a value of the monad type that encapsulates `x`
- `m >>= f` (read it as 'the bind operator') applies the function `f` to the value in the monad `m`

That's what a monad is. There are a few more technicalities, but basically those two operations define a monad. The real question is what a monad does, and that depends on the monad — lists are monads, Maybes are monads, IO operations are monads. All that it means when we say those things are monads is that they have the monad interface of `return` and `>>=`.

In OO terms, a monad is a fluent container.

The minimum requirement is a definition of `class <A> Something` that supports a constructor `Something(A a)` and at least one method `Something<B> flatMap(Function<A, Something<B>>)`.

Arguably, it also counts if your monad class has any methods with signature `Something<B> work()` which preserves the class's rules -- the compiler bakes in `flatMap` at compile time.

Why is a monad useful? Because it is a container that allows chain-able operations that preserve semantics. For example, `Optional<?>` preserves the semantics of `isPresent` for `Optional<String>`, `Optional<Integer>`, `Optional<MyClass>`, etc. […]

You should first understand what a functor is. Before that, understand higher-order functions.

A *higher-order function* is simply a function that takes a function as an argument.

A *functor* is any type construction `T` for which there exists a higher-order function, call it `map`, that transforms a function of type `a -> b` (given any two types `a` and `b`) into a function `T a -> T b`. This `map` function must also obey the laws of identity and composition such that the following expressions return true for all `x`, `p`, and `q` (Haskell notation):

```
map id = id
map (p . q) = map p . map q
```

For example, a type constructor called `List` is a functor if it comes equipped with a function of type `(a -> b) -> List a -> List b`

More techniques (for longer text)

1. ‘What are you trying to say here?’
2. ‘Give three big take-aways!’
3. Get to the ‘how’
4. Don’t ask the reader to rewind
5. ‘What are you trying to say? Write that down!’
6. Don’t count words
7. Avoid cheerleading - say why it’s better

Documentation process design

Software documentation process

How does software documentation fit into your project?

Who writes the documentation?

When do they write or update it?

Group exercise 1. Documentation tasks (5 min)

1. Identify and name documentation tasks
2. Make a timeline to indicate flow:
   arrange simultaneous tasks vertically
3. Use your phone for photo back-ups!

Group exercise 2. Process events (5 min)

What happens during the software development process in between the documentation tasks?

How do you know a task should start, or that it is complete?

1. Add interesting events in between your tasks.
2. Use different colour stickies.

Group exercise 3. Documentation roles (5 min)

1. List roles involved in your process, on a separate piece of paper.
2. Choose a good name for each role.
3. Annotate the sticky notes for the tasks with the role name or initials.

Documentation process wrap-up

Which documentation tasks do you include in your process?

How much time should your team spend on documentation each week?

How do you use documentation to reduce project risk?

What tools and production pipelines do you use?

Workshop wrap-up (retrospective)

What went well?

Would could have gone better?

What would you change?

Other documentation topics

Bonus topics

1. Reducing the need for documentation
2. Documentation types
3. Production pipelines
4. Agile software development documentation
5. Code comments 😬

Reducing the need for documentation

Simplified/standard architecture (fewer diagrams)

Better naming and cleaner code (*fewer* code comments)

Automated acceptance tests (replacing functional specs)

Automated builds (*shorter* installation instructions)

However, we cannot replace many kinds of docs with code.

Documentation types

README

Installation instructions

Tutorial

API reference

Contributor’s guide

Architecture diagram

Component inventory

UML diagrams (various)

Data dictionary

Business process model

Business rules

Architecture Decision Record

Production pipelines

Word processor docs 😭

Wiki, e.g. Confluence

Google Docs

Markdown + GitHub repo

Markdown + Jekyll

(GitHub Pages)

AsciiDoc → HTML + PDF

reStructuredText + Sphinx

(readthedocs.org)

Agile documentation

The purpose of documentation is risk reduction.

Agile software development manages risks differently.

From a lean software development perspective, documentation is waste. Minimum Viable Docs FTW!

How do you include documentation in agile development?

Code comments 😬

Comments are a feature of almost all languages,

but remain an almost taboo topic.

Developers will go to bizarre lengths to avoid comments

but usually fail to write code so good they don’t need any.

Comments are not the enemy:

meetings are the enemy!

Wrap-up

Wrapping up

We need system documentation,

but we don’t usually need very much.

Technical writing is not a mysterious black art:

as with coding, you can learn techniques and improve.

Session feedback welcome in person or via Twitter

I’m here all week!

Ask for details about this workshop at your company

@PeterHilton

http://hilton.org.uk/presentations/documentation-workshop