Software Documentation

Kim Brugger

March 2019

It doesn’t matter how good your software is, because if the documentation is not good enough, people will not use it.

-Daniele Procida

Documentation: Outline

Introduction

Coding Style

Documentation

Best practices

Reflection

Intro: how many keystrokes do you have left?

Let's assume you'll live to at least **90**. That's **70** years left. We'll assume you spend **half your working day** typing full speed. That's **4** hours a day of nothing but typing. There's an average of **5** letters in a word in English.

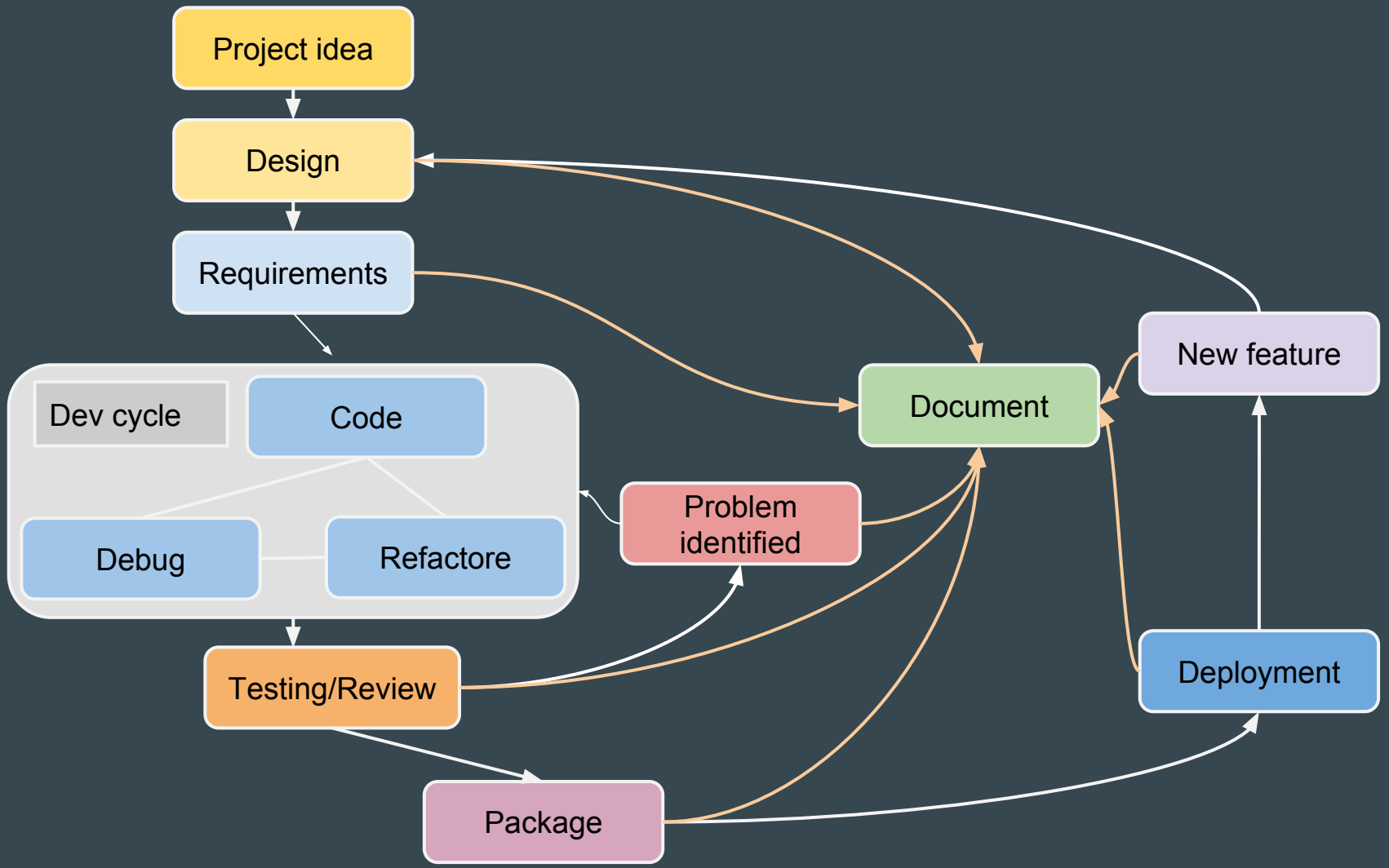
That means, in your remaining lifetime you have these left in your hands:

- **80,639,999** Keystrokes Left
- **575,999** Tweets Left
- **26** Novels Left
- **161** Computer Programs Left
- **8,063** Love Letters Left

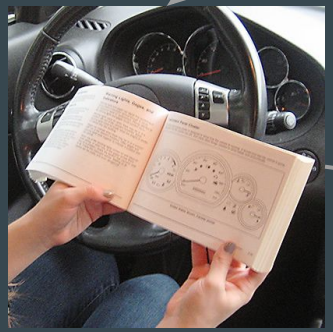
Or

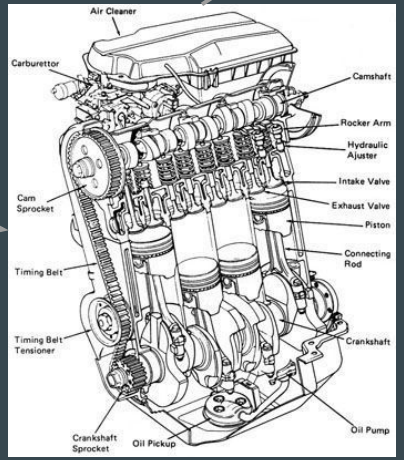
- **403,199** Emails to your boss left

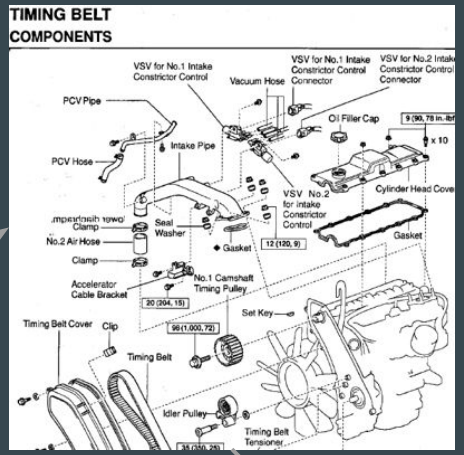
where do you want the gift of your keystrokes to go?

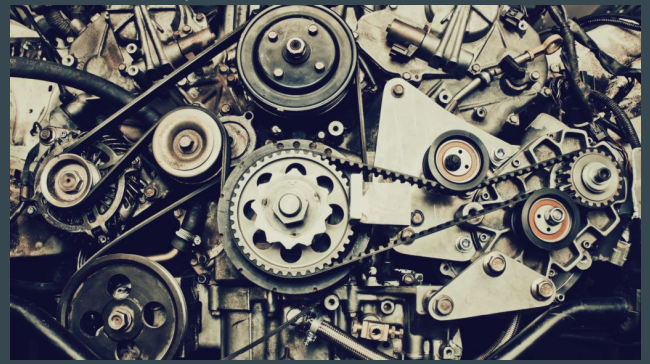

Project idea
Design
Requirements
Dev cycle
Code
Debug
Refactore
Testing/Review
Package
Document
Problem identified
New feature
Deployment

Introduction: levels of detail



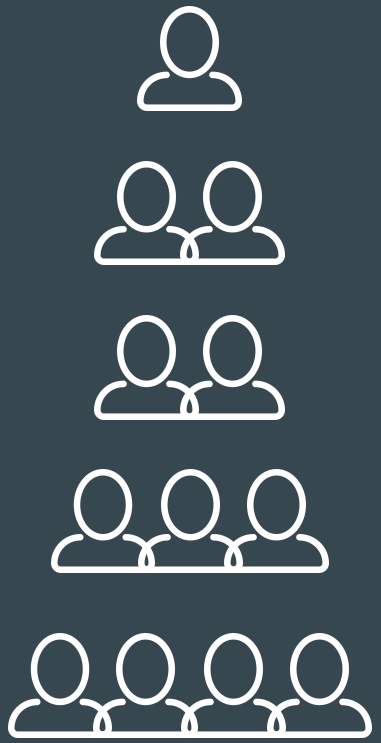


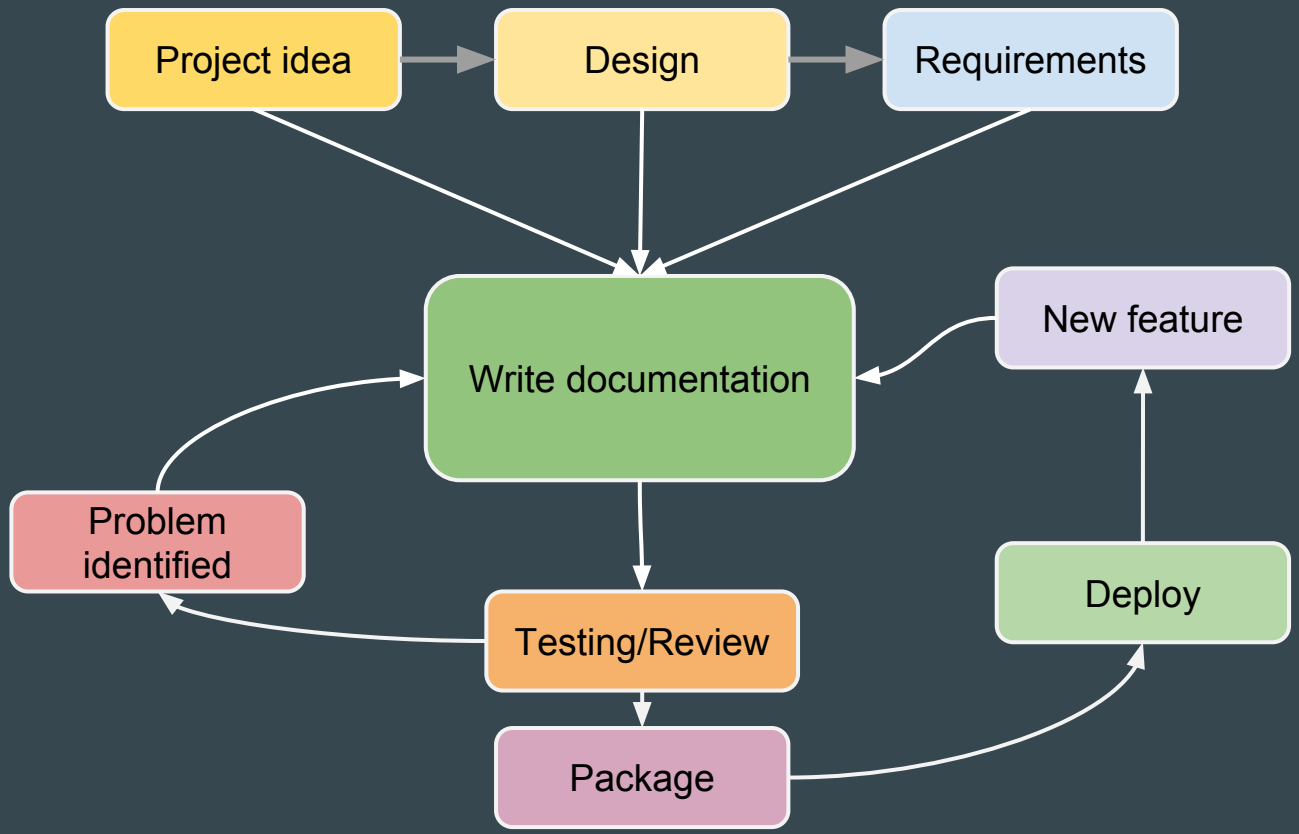




Introduction: Target your Audience

Code documentation	Detailed information on how things work
Technical	API documentation
Architecture	How the software works in greater detail, and how different components interact
Manual	High level usage of software, eg: installation or compiling
Website	To attract uses




Project idea
Design
Requirements
Write documentation
New feature
Problem identified
Testing/Review
Deploy
Package

Coding Style: Standards & guidelines

[Pep8: Style Guide for Python Code]

[Pep20: The Zen of Python]

[Google Python Style Guide]

[python-guide.org style guide]

[Docstring Conventions]

Coding Style: Find your personal style

```
if get_address():
    res = {}
    res['number'] = 52
    res['road'] = 'Kalfarlien'
    res['postcode'] = 5018
    res['city'] = 'Bergen'
    print("{} {}".format(road, nr))
    print("{} {}".format(postcode,
                         city))
```

```
if ( get_address() ):
    res = {}
    res[ 'number'  ] = 52
    res[ 'road'    ] = 'Kalfarlien'
    res[ 'postcode'] = 5018
    res[ 'city'    ] = 'Bergen'

    print( "{} {}".format( road,
                           nr))
    print( "{} {}".format( postcode,
                           city))
```

Coding Style: Valid does not mean good

```
x = {  'a':37,'b':42,

'c':927}

y = 'hello ''world'
z = 'hello '+'world'
a = 'hello {}'.format('world')
class foo  (     object  ):
  def f    (self   ):
    return       37*-+2
  def g(self, x,y=42):
      return y
def f  (   a ) :
  return      37+-+a[42-x :  y**3]
```

```
x = {'a': 37, 'b': 42, 'c': 927}

y = 'hello ' 'world'
z = 'hello ' + 'world'
a = 'hello {}'.format('world')

class foo(object):
    def f(self):
        return 37 * -+2

    def g(self, x, y=42):
        return y

def f(a):
    return 37 + -+a[42 - x:y**3]
```

Coding Style: Hands on pep8 checking

Use the tool on some of your python code:

http://pep8online.com

What errors/warnings do you see?

Do you agree with the errors?

Documentation: What are useful comments

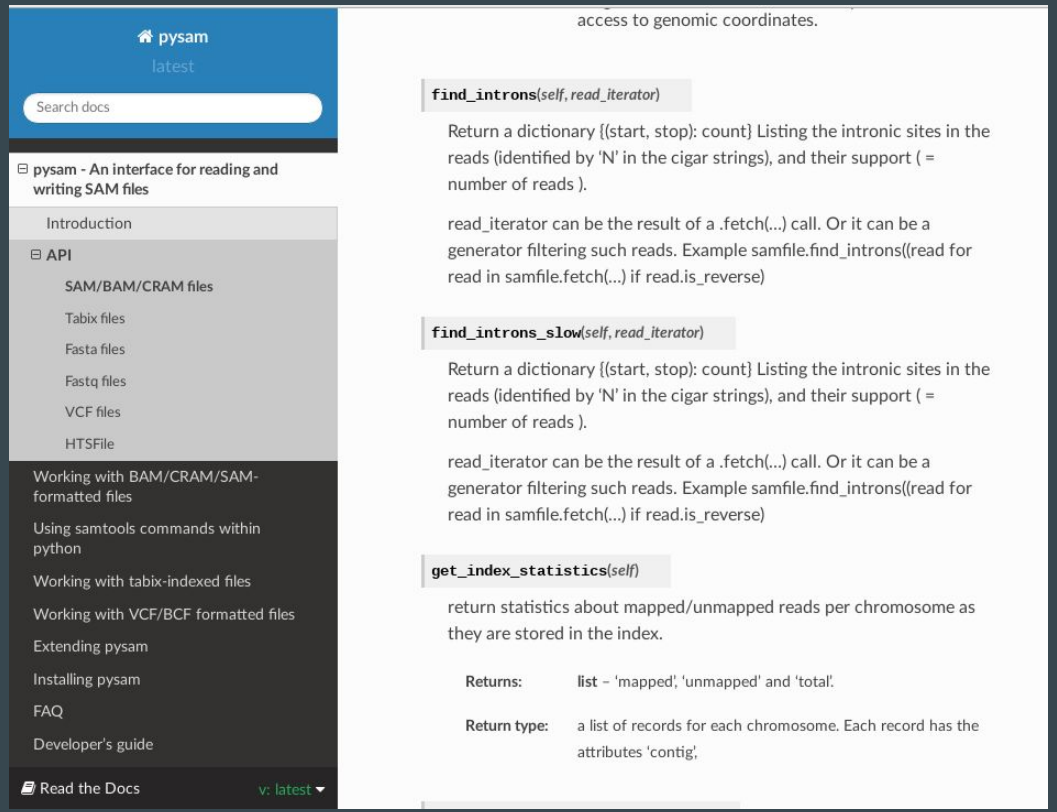
```
# add 1 to counter
i += 1

# Calculate area of circle
area = pi*radius*radius

#cannot be bothered with this right now
# state = set_state( input1, current_state())
state = 1

// Here be dragons. Thou art forewarned
images.sort(ImageSprite::image_path_comparator,5,3);
```

Documentation: Python docstring

```
def function_with_pep484_type_annotations(param1: int, param2: str) -> bool:
    """Example function with PEP 484 type annotations.

    Args:
        param1 (int): The first parameter.
        param2 (str): The second parameter.

    Returns:
        The return value. True for success, False otherwise.

    Raises:
        ValueError: If `param2` is equal to `param1`.
   """

# Class help:
import string
help( string )

#function help:
help( max )
```

Documentation: Integrated documentation (sphinx)

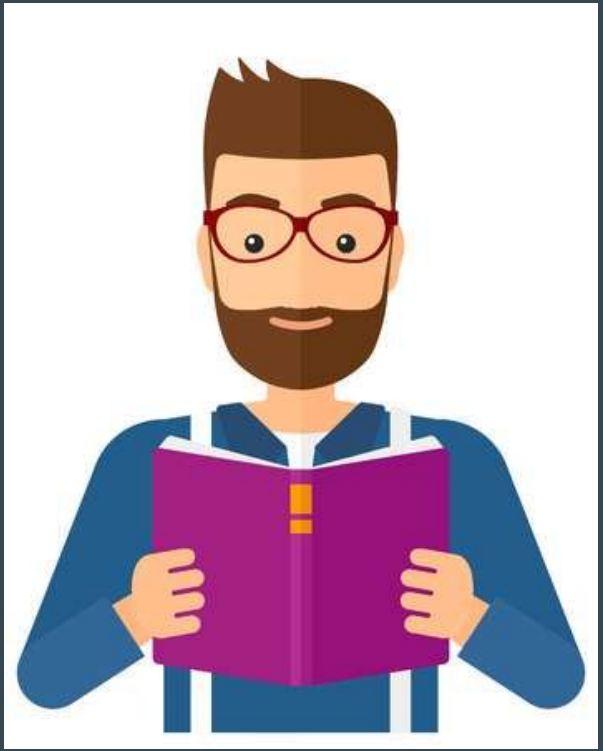
```
.. test documentation master file
word_counter: count words in files
==========================================

:Author: Kim Brugger
:Date: |today|
:Version: |version|

Word_counter is a python package/module …

.. toctree::
   :maxdepth: 1
   :caption: Contents:
   tools
```



Best practices: Repository readme file

Project Title

- Overview
- Getting Started
  - Prerequisites
  - Installing
  - Testing the installation
- Deployment
  - Configuration
  - Security
- Contributing
- Versioning
- Authors
  - Contributors
  - Acknowledgments
- Licence



Best practices: Distribution of documentation

readthedocs.com

Github.io

https://sourceforge.net/

Dedicated project webpage

Best practices: Look at Examples (20 min):

https://github.com/pysam-developers/pysam

https://github.com/pallets/flask

https://github.com/django/django

https://github.com/pandas-dev/pandas

https://github.com/google/yapf

Documentation: reflection & thoughts

What is your current practice?

How could you improve it?

How do you find the documentation of other projects?

Any good practices?

Various resources

https://softdev4research.github.io/4OSS-lesson/

https://semver.org/

https://choosealicense.com/licenses/

https://sphinxcontrib-napoleon.readthedocs.io/en/latest/example_google.html

http://www.sphinx-doc.org/en/master/