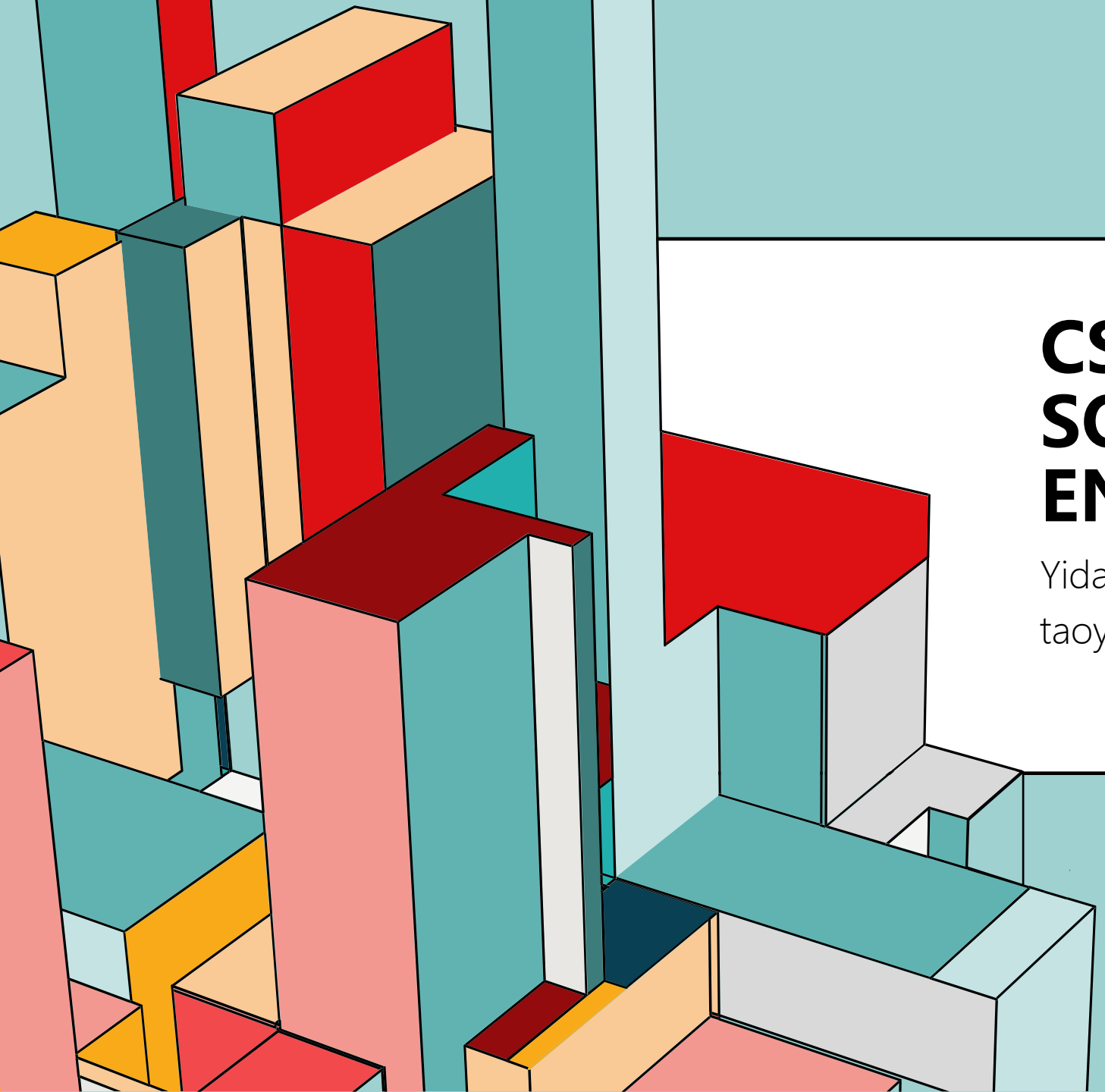


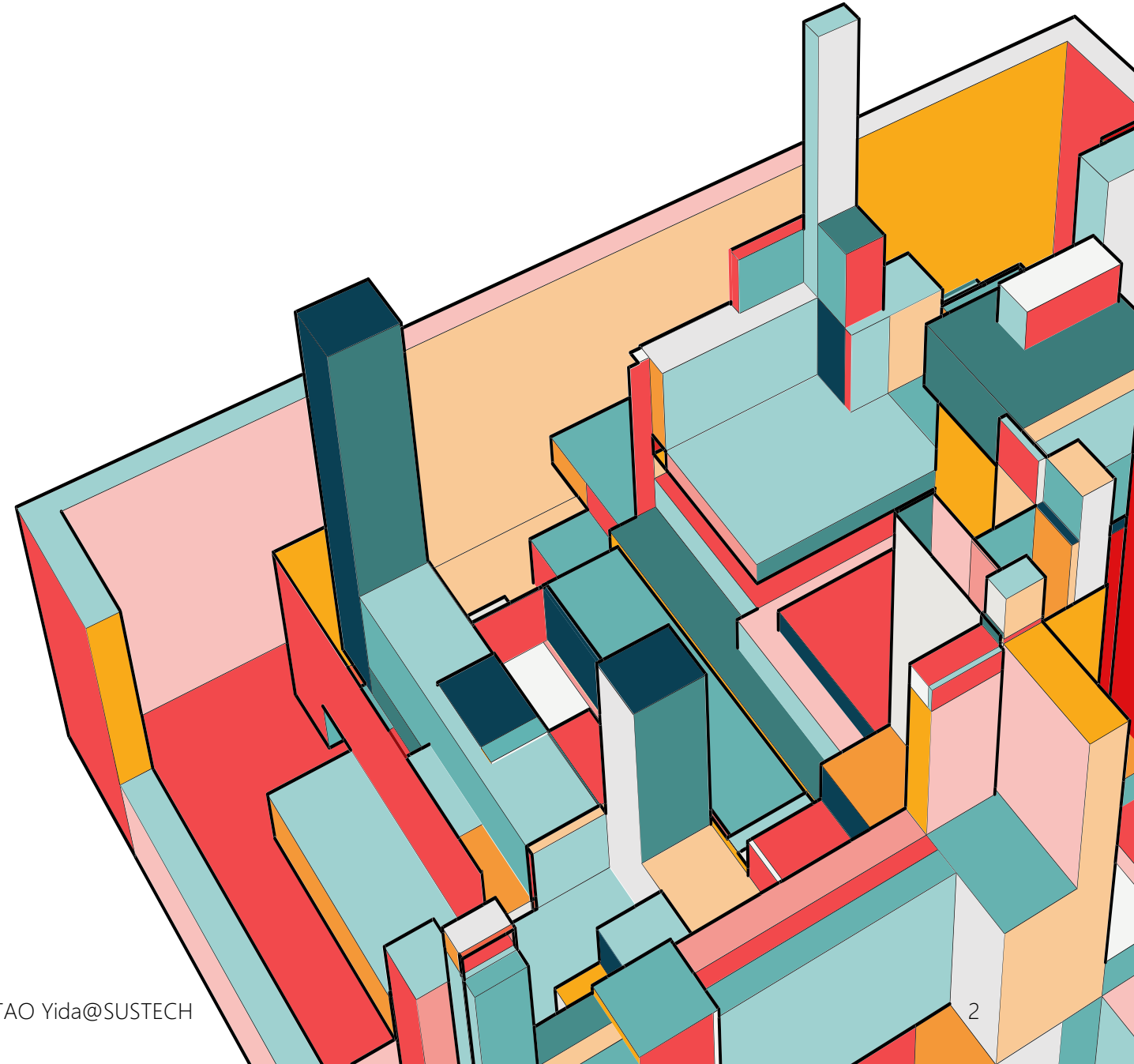
CS304 SOFTWARE ENGINEERING

Yida Tao

taoyd@sustech.edu.cn

LECTURE 9

- Software documentation
- Case study @ Google



SOFTWARE DOCUMENTATION

```java
/**
 * Creates a new String using the character sequence represented by
 * the StringBuffer. Subsequent changes to buf do not affect the String.
 *
 * @param buffer StringBuffer to copy
 * @throws NullPointerException if buffer is null
 */
public String(StringBuffer buffer)
{
  synchronized (buffer)
    {
      offset = 0;
      count = buffer.count;
      // Share unless buffer is 3/4 empty.
      if ((count << 2) < buffer.value.length)
        {
          value = new char[count];
          VMSystem.arraycopy(buffer.value, 0, value, 0, count);
        }
      else
        {
          buffer.shared = true;
          value = buffer.value;
        }
    }
}
```

java.io.String

How to build Teedy from the sources

Prerequisites: JDK 11, Maven 3, NPM, Grunt, Tesseract 4

Teedy is organized in several Maven modules:

- docs-core
- docs-web
- docs-web-common

First off, clone the repository: `git clone https://github.com/sustech-cs304/Teedy`

Launch the build

From the root directory:

```
mvn clean -DskipTests install
```

Run a stand-alone version

From the `docs-web` directory:

```
mvn jetty:run
```

Teedy readme

SOFTWARE DOCUMENTATION

百度paddle paddle文档：https://www.paddlepaddle.org.cn/documentation/docs/zh/api/index_cn.html

ReduceOp

class paddle.distributed.ReduceOp[源代码]

指定规约类操作的逐元素操作类型，需要是下述值之一：

ReduceOp.SUM

ReduceOp.MAX

ReduceOp.MIN

ReduceOp.PROD

代码示例

```
# required: distributed
import paddle
import paddle.distributed as dist

dist.init_parallel_env()
if dist.get_rank() == 0:
    data = paddle.to_tensor([[4, 5, 6], [4, 5, 6]])
else:
    data = paddle.to_tensor([[1, 2, 3], [1, 2, 3]])
dist.all_reduce(data, op=dist.ReduceOp.SUM)
print(data)
# [[5, 7, 9], [5, 7, 9]] (2 GPUs)
```

SOFTWARE DOCUMENTATION

List commits

Works with GitHub Apps

Signature verification object

The response will include a `verification` object that describes the result of verifying the commit's signature. The following fields are included in the `verification` object:

Name	Type	Description
`verified`	`boolean`	Indicates whether GitHub considers the signature in this commit to be verified.
`reason`	`string`	The reason for verified value. Possible values and their meanings are enumerated in table below.

Code samples

GET /repos/{owner}/{repo}/commits

cURL JavaScript GitHub CLI

```
curl \
  -H "Accept: application/vnd.github+json" \
  -H "Authorization: Bearer <YOUR-TOKEN>"\
  -H "X-GitHub-Api-Version: 2022-11-28" \
  https://api.github.com/repos/OWNER/REPO/commits
```

GitHub REST API Documentation:
https://docs.github.com/en/rest/commits/commits?apiVersion=2022-11-28

EXTERNAL SOFTWARE DOCUMENTATION

- **End-user documentation**: gives end users basic instructions on how to use, install and troubleshoot the software (e.g., user guides, tutorials).
- **Enterprise user documentation**: used for IT staff who deploy the software across the enterprise.
- **Just-in-time documentation**: provides end users with support documentation at the exact time they will need it (e.g., FAQ pages, how-to documents)

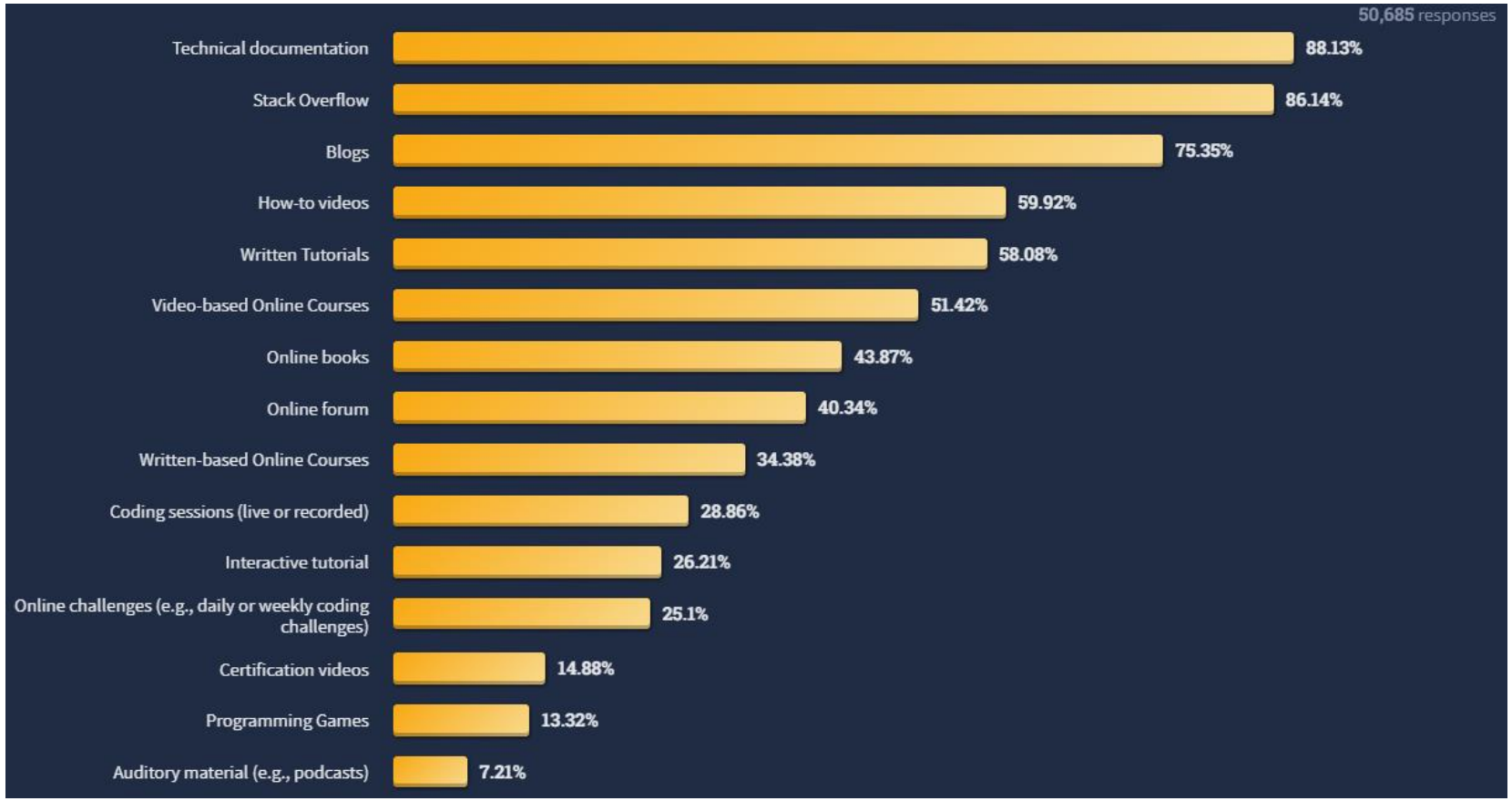
INTERNAL SOFTWARE DOCUMENTATION

- **Administrative documentation**: the high-level administrative guidelines, roadmaps and product requirements for the software development team and project managers working on the software (e.g., status reports, meeting notes).

- **Developer documentation**: provides instructions to developers for building the software and guides them through the development process.
  - Requirements documentation
  - Architecture & design documentation
  - Code Comments
  - API documentation
  - Readme, release notes, etc.

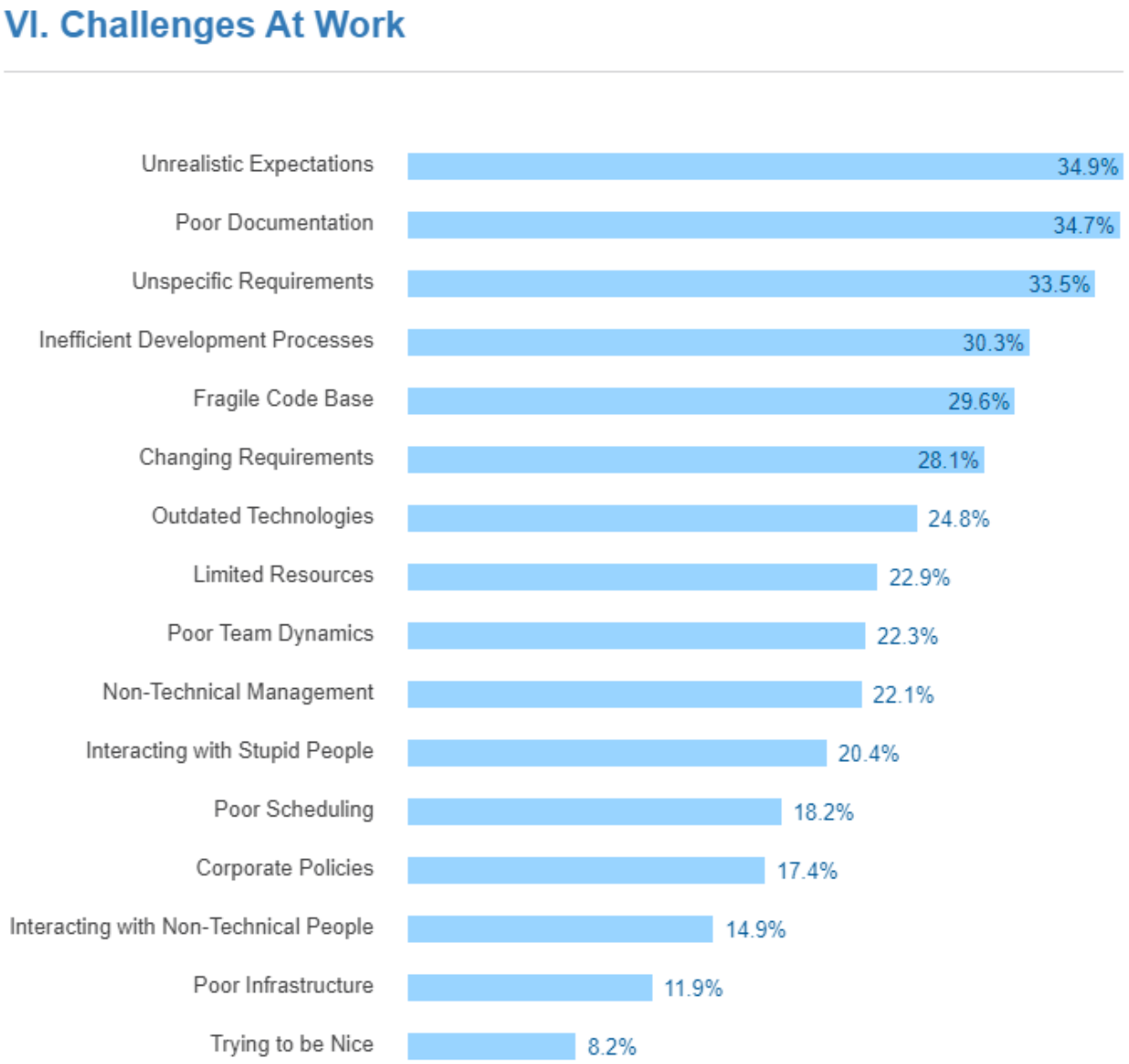
IMPORTANCE OF DOCUMENTATION

What online resources do you use to learn to code?

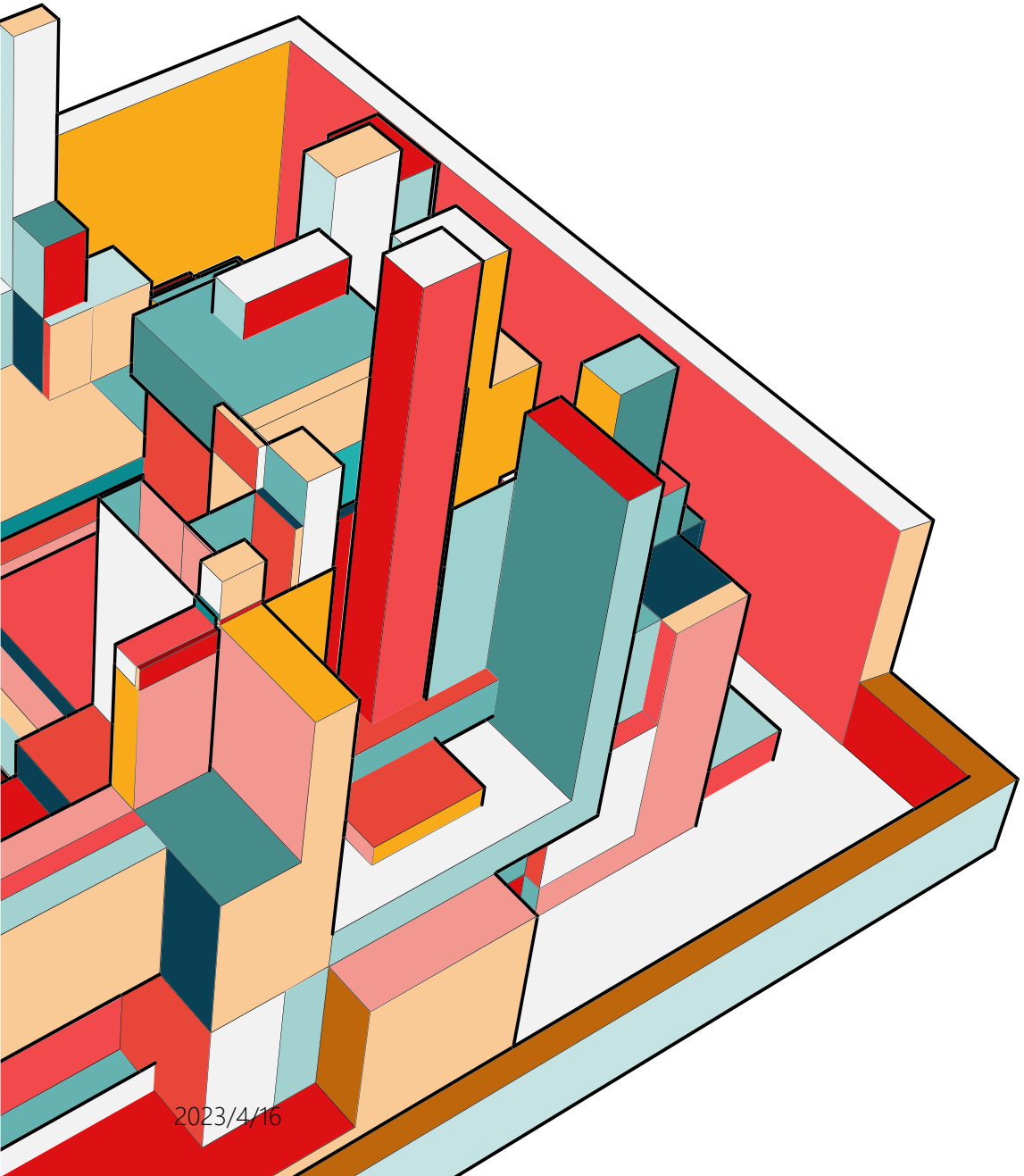


Stack Overflow Developer Survey 2022

POOR DOCUMENTATION IS EVERYWHERE



Source: Stack Overflow Developer Survey 2016



WRITING GOOD DOCUMENTATION

GOOD PRACTICES

SELF-DOCUMENTING CODE

- Self-documenting (or self-describing) code follow naming conventions and structured programming conventions that enable use and understanding of the system without prior specific knowledge
  - Meaningful directory structure
  - Meaningful file/class/method/variable names

- Main goal: allowing developers to understand code at a glance

SELF-DOCUMENTING CODE

```
float a, b, c; a=9.81; b=5; c= .5*a*(b^2);
```

VS

```
const float gravitationalForce = 9.81;
float timeInSeconds = 5;
float displacement = (1 / 2) * gravitationalForce * (timeInSeconds ^ 2);
```

Good code doesn’t need documentation. It’s self-explaining

https://stackoverflow.com/questions/209015/what-is-self-documenting-code-and-can-it-replace-well-documented-code

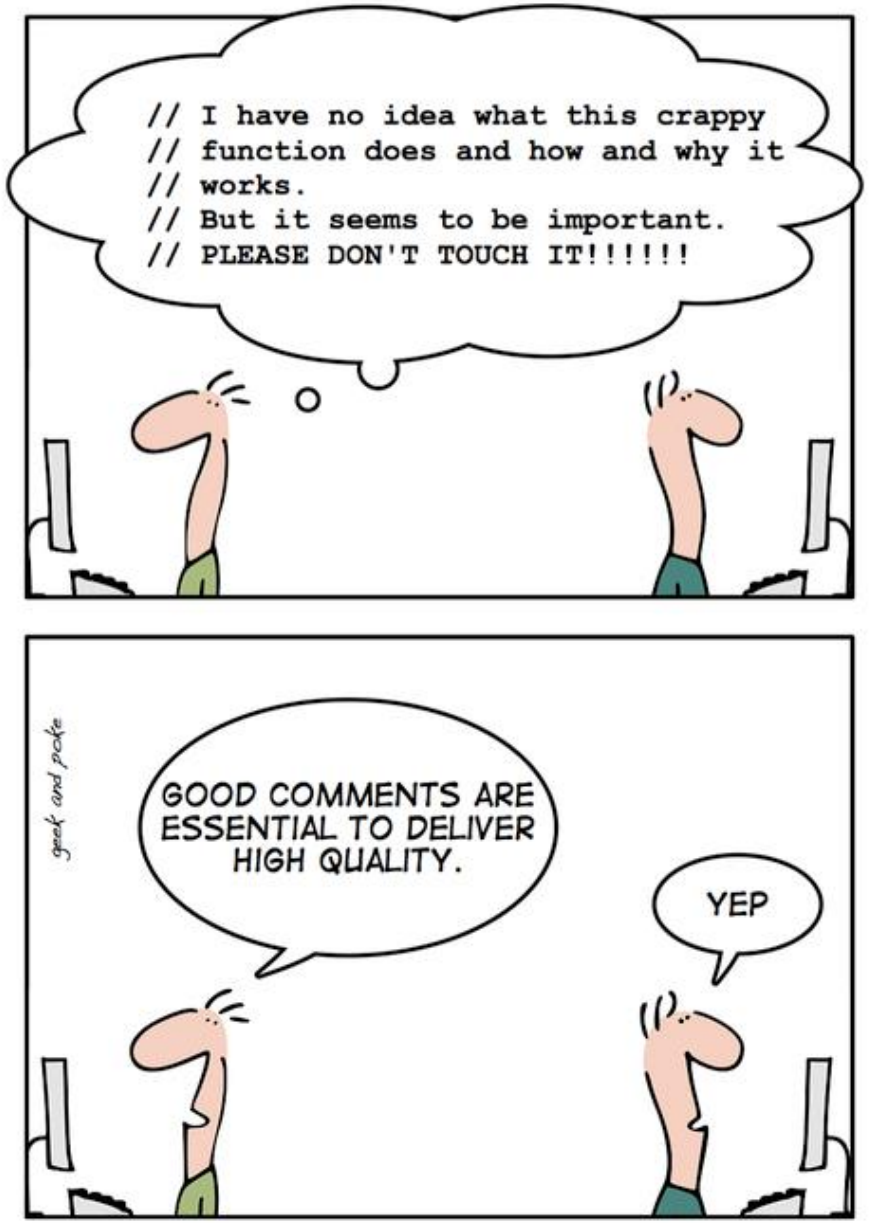
CODE COMMENTS （注释）

- Sometimes when the code alone can’t provide context or clarify intent, the developer may write extra descriptions, i.e., code comments.
- Code comments enhance readability. They facilitate code reviews, refactoring, and maintenance.
- Code comments are ignored by compilers and interpreters when producing the final executable. Thus, they incur no runtime performance overhead. However, too many or unnecessary comments reduce readability.

CODE COMMENTS

"Nothing can be quite so helpful as a well-placed comment.
Nothing can clutter up a module more than frivolous dogmatic comments.
Nothing can be quite so damaging as an old crufty comment that propagates lies and misinformation“

- Clean code by Robert C. Martin.



WRITING GOOD COMMENTS

- Comments should be used only to explain the **intent** behind code that cannot be refactored to explain itself
- Mostly used for providing **additional context**, instead of simply repeating the code
- Should answer **WHY**, instead of WHAT

```
const float a = 9.81; //gravitational force
float b = 5; //time in seconds
float c = (1/2)*a*(b^2) //multiply the time and gravity together to get displacement.
```

VS

```
/* compute displacement with Newton's equation x = v₀t + ½at² */
const float gravitationalForce = 9.81;
float timeInSeconds = 5;
float displacement = (1 / 2) * gravitationalForce * (timeInSeconds ^ 2);
```

https://stackoverflow.com/questions/209015/what-is-self-documenting-code-and-can-it-replace-well-documented-code

WRITING GOOD COMMENTS

```
1           while (! finished) {
2               try {
3                   Thread.sleep(10); // so that program gets time to handle xxx logics
4               }
5               catch (InterruptedException e) {
…                   ……
…               }
…           }
```

Meaningful code comment complements / explains the intention of the code

WRITING GOOD COMMENTS

```
1       //WARNING: 本测试用例需要耗费大约15分钟时间，因此日常集成时不运行
2       @Ignore
3       @Test
4       public void testHighThroughput() {
5       //模拟十万个线程同时进行访问
6       }
```

Meaningful code comment gives necessary warnings

WRITING GOOD COMMENTS

```
1    //Notice：下列数据库连接参数需要改为从外部配置文件中读取
2    String driver = "com.mysql.jdbc.Driver";
3    String url = "jdbc:mysql://localhost:3306/sqltestdb";
4    String user = "root";
5    String password = "123456";
6    Class.forName(driver);
7    con = DriverManager.getConnection(url,user,password);
```

Meaningful code comment reminds about self-admitted technical debt (自承认技术债): developers consciously perform the hack (suboptimal solution) due to time constraints or technical limitations

WRITING GOOD COMMENTS

```
1  public int[] sortNumbers() {
2      //TODO: 需在这里补充真实的排序代码
3      return new int[]{1,2,3,4,5};
4  }
```

Meaningful code comment helps developers locate incomplete/unfinished implementations (//TODO cannot exist in delivered code)

COMMENTS - EXAMPLES

```
// make sure the port is greater or equal to 1024
if (port < 1024) {
  throw new InvalidPortError(port);
}
```

Bad
- No additional information
- WHAT

```
// port numbers below 1024 (the privileged or "well-known ports")
// require root access, which we don't have
if (port < 1024) {
  throw new InvalidPortError(port);
}
```

Okay
- Additional information
- WHY

Examples from https://www.youtube.com/watch?v=uPMxUnBjGG8

COMMENTS - EXAMPLES

```
if (!hasRootPrivileges(port)) {
  throw new InvalidPortError(port);
}

private boolean hasRootPrivileges(int port) {
  // port numbers below 1024 (the privileged or "well-known ports")
  // require root access on unix systems
  return port < 1024;
}
```

Better

- Refactored with meaningful name (`hasRootPrivileges`)

```
final static const HIGHEST_PRIVILEDGED_PORT = 1023;

private boolean hasRootPrivileges(int port) {
  // The privileged or "well-known ports" require root access on unix systems
  return port <= HIGHEST_PRIVILEDGED_PORT;
}
```

Even better

- Turn magic number into a constant with meaningful name
- Comment might no longer be needed

Examples from https://www.youtube.com/watch?v=uPMxUnBjGG8

COMMENTS - DESCRIBING WHY

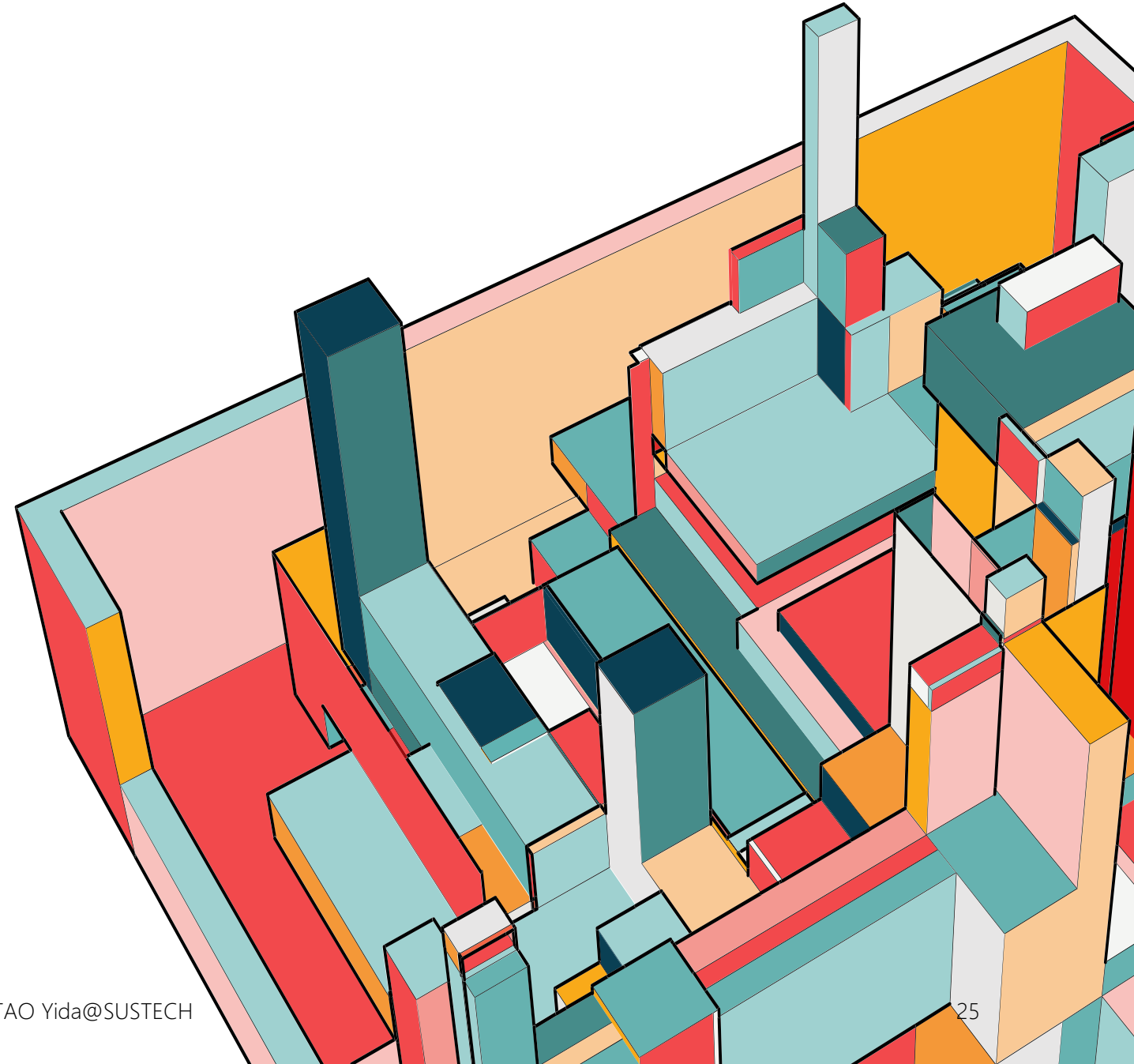
- Describe the design decisions to a class
- Describe the limitations of an implementation
- Describe usage assumptions of APIs
- Describe the purpose and intents of each file

COMMENTING PRINCIPLES

- The best documentation is the code itself
- Make the code self-explainable and self-documenting
- Do not document bad code, refactor or rewrite it!
- WHY (rationales), not WHAT (implementations)

JAVADOC

Javadoc (included in JDK) automatically generates API documentation from comments present in the Java source code.



JAVADOC COMMENT SYNTAX

- Javadoc comments structure look very similar to a regular multi-line comment, but the key difference is the extra asterisk at the beginning
- Javadoc style comments may contain HTML tags as well.

```
// This is a single line comment

/*
 * This is a regular multi-line comment
 */

/**
 * This is a Javadoc
 */
```

https://www.baeldung.com/javadoc

JAVADOC COMMENT SYNTAX

- Javadoc comments may be placed above any class, method, or field which we want to document.

- These comments are commonly made up of two sections:
  - The description of what we're commenting on
  - The standalone block tags (marked with the “@” symbol) which describe specific meta-data

https://www.baeldung.com/javadoc

JAVADOC TAGS

@author	A person who made significant contribution to the code. Applied only at the class, package, or overview level. Not included in Javadoc output. It's not recommended to include this tag since authorship changes often.
@param	A parameter that the method or constructor accepts. Write the description like this: @param count Sets the number of widgets you want included.
@deprecated	Lets users know the class or method is no longer used. This tag will be positioned in a prominent way in the Javadoc. Accompany it with a @see or {@link} tag as well.
@return	What the method returns.
@see	Creates a see also list. Use {@link} for the content to be linked.
{@link}	Used to create links to other classes or methods. Example: {@link Foo#bar} links to the method `bar` that belongs to the class `Foo`. To link to the method in the same class, just include #bar.
@since 2.0	The version since the feature was added.
@throws	The kind of exception the method throws. Note that your code must indicate an exception thrown in order for this tag to validate. Otherwise Javadoc will produce an error. @exception is an alternative tag.
@Override	performs a check to see if the method is an override. used with interfaces and abstract classes.

https://idratherbewriting.com/java-javadoc-tags/

JAVADOC AT CLASS LEVEL

```java
/**
* Hero is the main entity we'll be using to . . .
*
* Please see the {@link com.baeldung.javadoc.Person} class for true identity
* @author Captain America
*
*/
public class SuperHero extends Person {
    // fields and methods
}
```

https://www.baeldung.com/javadoc

JAVADOC AT METHOD LEVEL

```
/**
 * Returns the element at the specified position in this list.
 *
 * @param  index index of element to return.
 * @return the element at the specified position in this list.
 * @throws    IndexOutOfBoundsException if index is out of range <tt>(index
 *            &lt; 0 || index &gt;= size())</tt>.
 */
public E get(int index) {
    RangeCheck(index);

    return elementData[index];
}
```

java.util.ArrayList

JAVADOC GENERATION

```
C:> javadoc -d C:\home\html -sourcepath C:\home\src java.my.package
```

```
/**
 * Returns the element at the specified position in this list.
 *
 * @param  index index of element to return.
 * @return the element at the specified position in this list.
 * @throws    IndexOutOfBoundsException if index is out of range <tt>(index
 *            &lt; 0 || index &gt;= size())</tt>.
 */
public E get(int index) {
    RangeCheck(index);

    return elementData[index];
}
```

get

```
public abstract E get(int index)
```

Returns the element at the specified position in this list.

Specified by:

`get` in interface `List<E>`

Parameters:

`index` - index of the element to return

Returns:

the element at the specified position in this list

Throws:

`IndexOutOfBoundsException` - if the index is out of range (index < 0 || index >= size())

DOCUMENTATION TOOLS & FRAMEWORKS

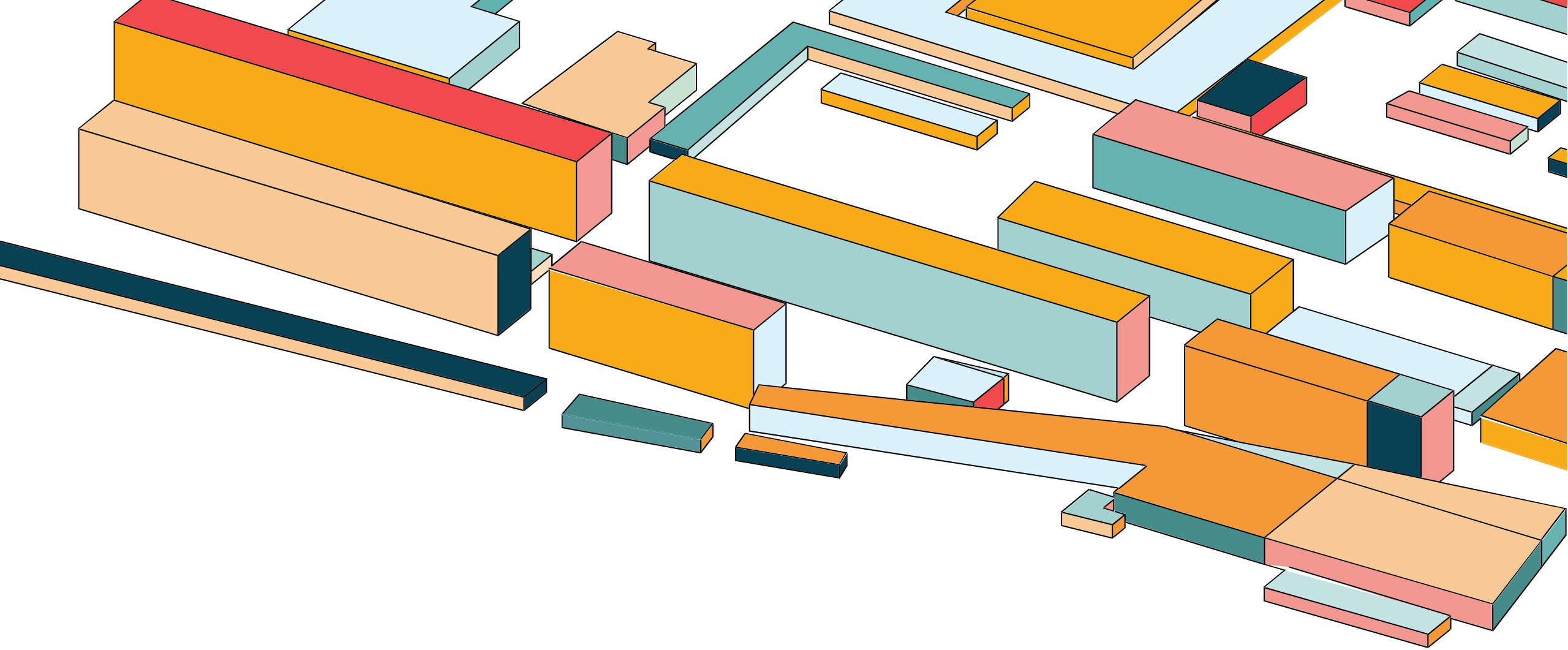
- Python: Python has its built-in documentation tool called PyDoc, which generates HTML documentation from Python code.
- JavaScript: JSDoc is an open source API documentation generator for Javascript. It allows developers to document their code through comments.

https://www.baeldung.com/javadoc

DOCUMENTATION TOOLS & FRAMEWORKS

- Sphinx: a documentation generator written and used by the Python community. https://www.sphinx-doc.org/en/master/
- Swagger: a popular framework for documenting REST APIs. https://swagger.io/docs/
- GitBook: a modern documentation platform where teams can document everything from products to internal knowledge bases and APIs. https://www.gitbook.com/

https://www.baeldung.com/javadoc



LESS RECOGNIZED DOCUMENTATION

TESTS

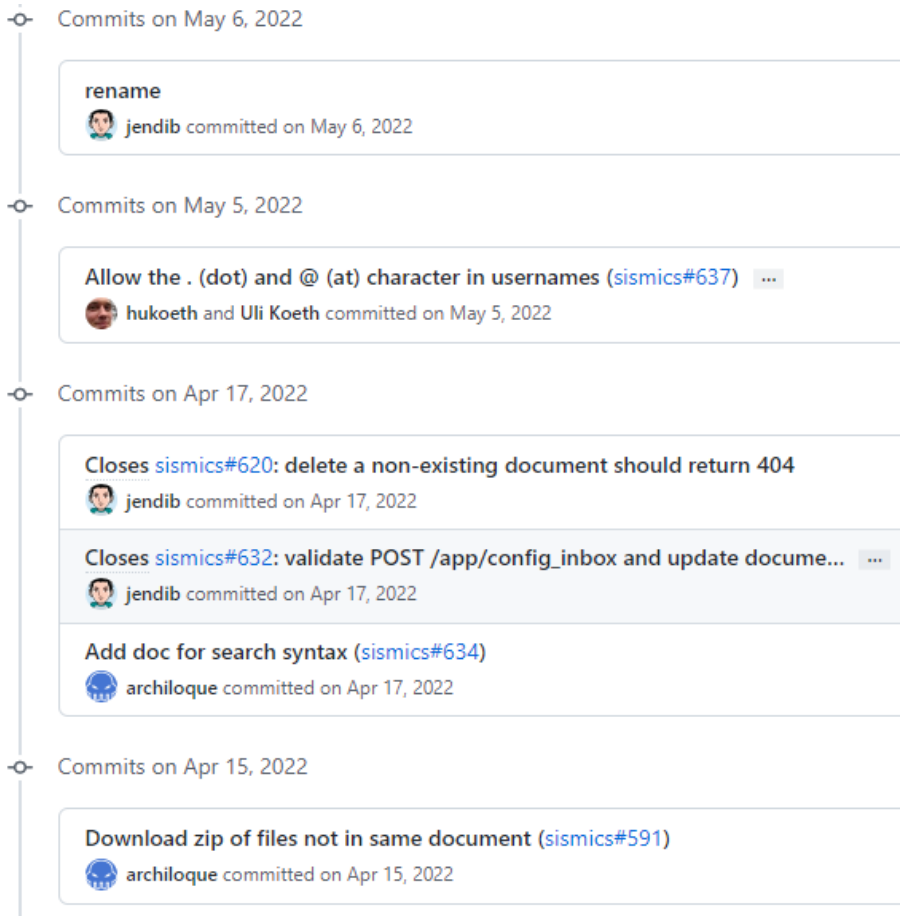
- Tests could be used to “document” how the code should behave
- Tests provide examples of how to use an API, with assertions
- Tests also give examples of edge case scenarios

```
void shouldRetrieveValuesInOrderTheyAreAdded()
void shouldThrowExceptionIfStackIsEmpty()
void shouldThrowExceptionIfMaxThresholdIsReached()
```

Sample tests for Stack

GIT COMMIT MESSAGES

- Gives context of the change
- Explain “why this change is needed?” kinds of questions
- Commit messages live with the code with no clutter
- Always up-to-date w.r.t. the code changes



README

- Provide high-level context such as
  - What's this project is for?
  - Design choices
  - How to install
  - How to use
  - How to contribute
  - License
  - ……

README.md

Product Name

> Short blurb about what your product does.

npm v0.3.0 | build passing | downloads 1k/month

One to two paragraph statement about your product and what it does.

Installation

Information on how to install your product

Usage

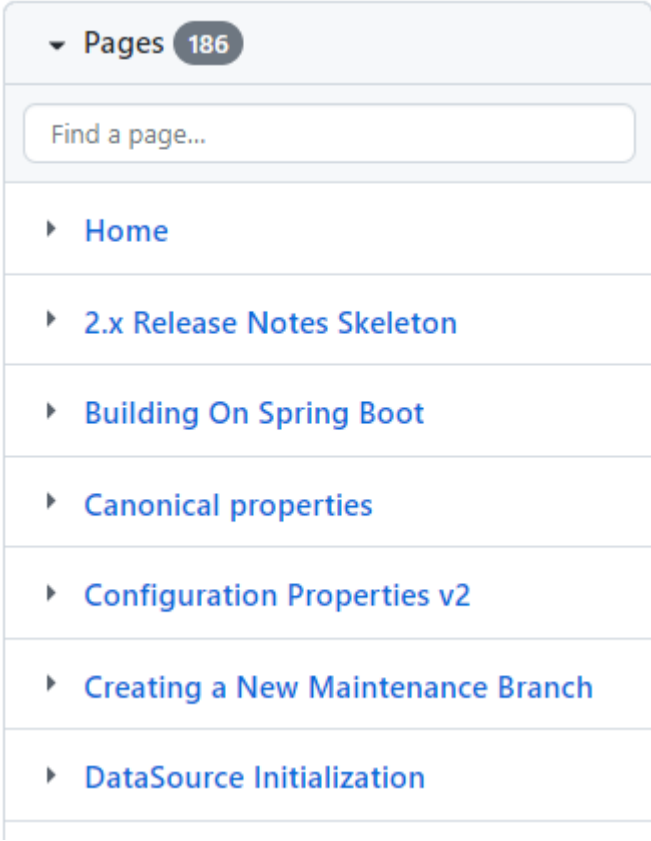
Information on how to use your product

Contributing

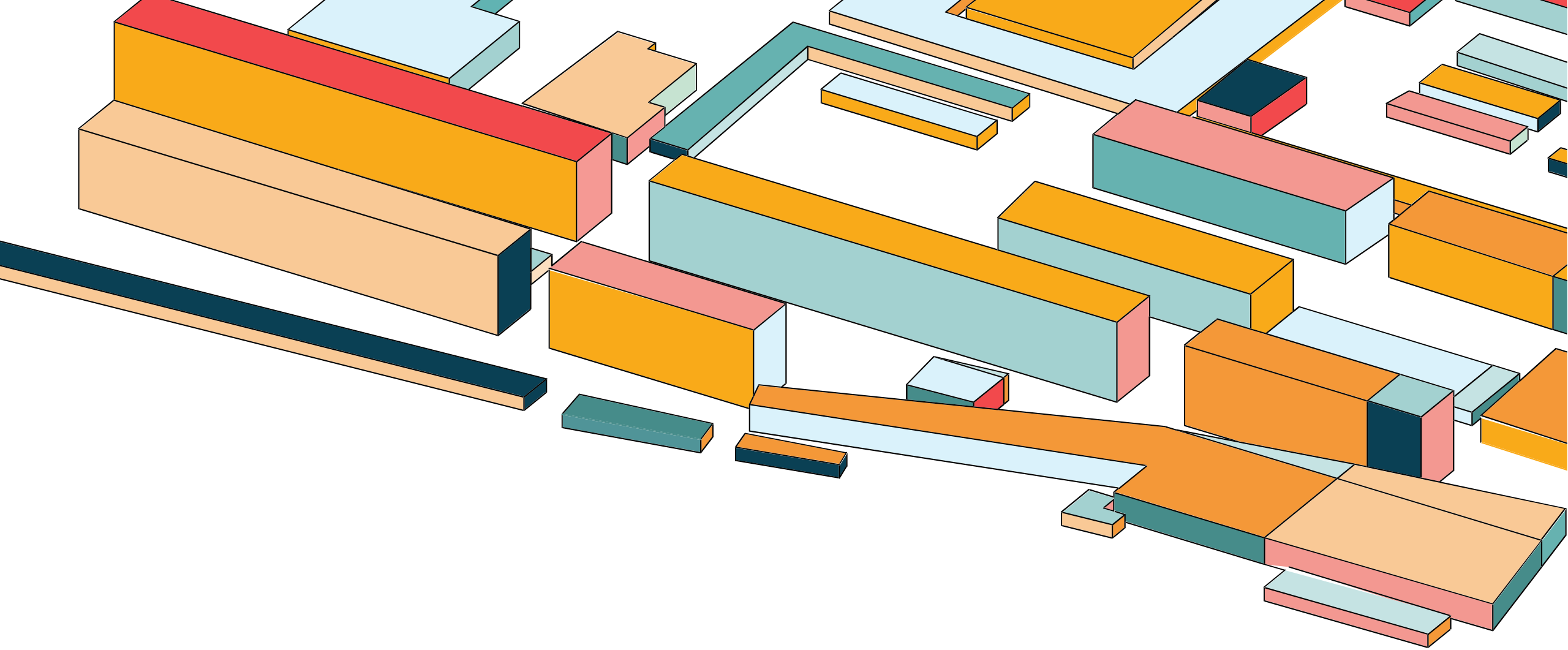
Add the hall-of-fame widget to highlight your contributors

WIKIS

- Every repo on GitHub comes equipped with a section for hosting documentation, called a wiki.
- You can use wiki to share long-form content about your project, such as how to use it, how you designed it, or its core principles.
- A README file quickly tells what your project can do, while you can use a wiki to provide additional documentation.



https://github.com/spring-projects/spring-boot/wiki



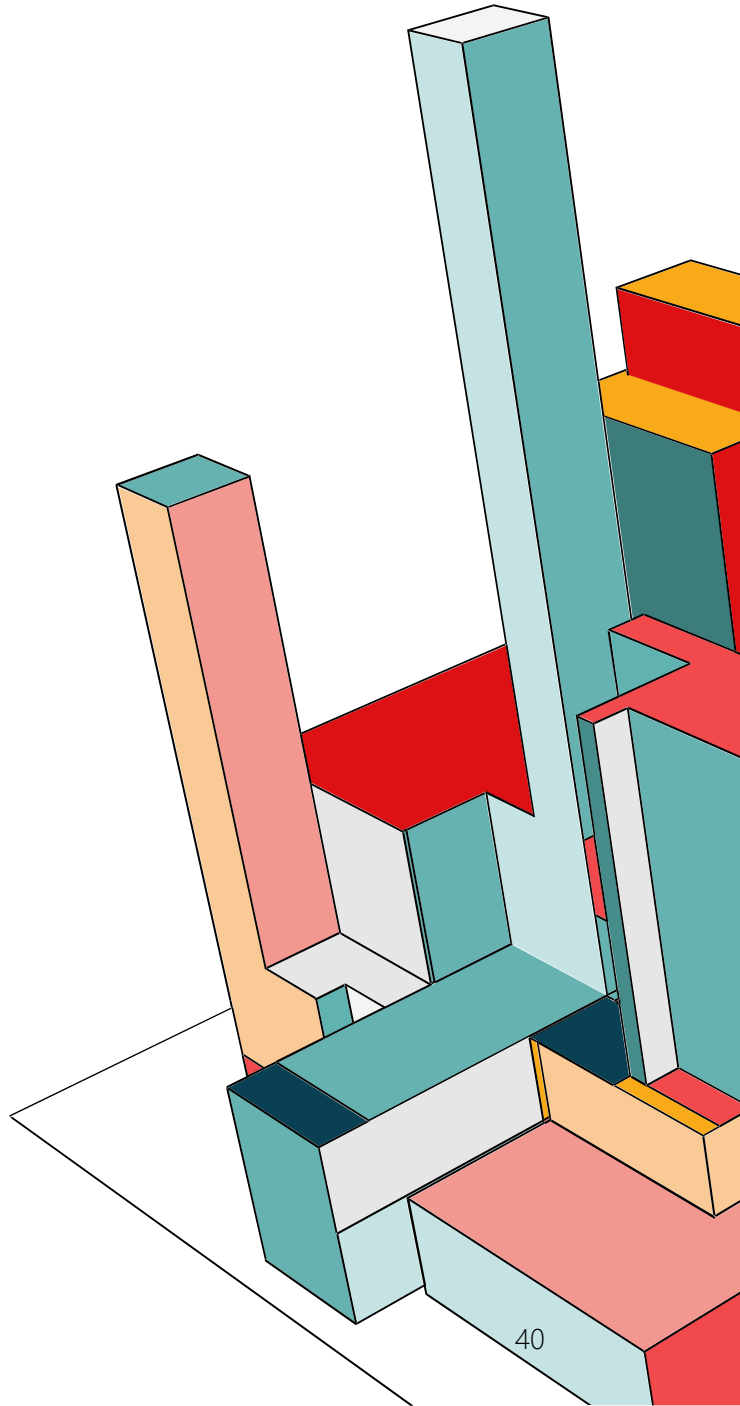
CASE STUDY: G3DOC @ GOOGLE

https://www.usenix.org/conference/srecon16europe/program/presentation/macnamara

ONCE UPON A TIME ...

48%

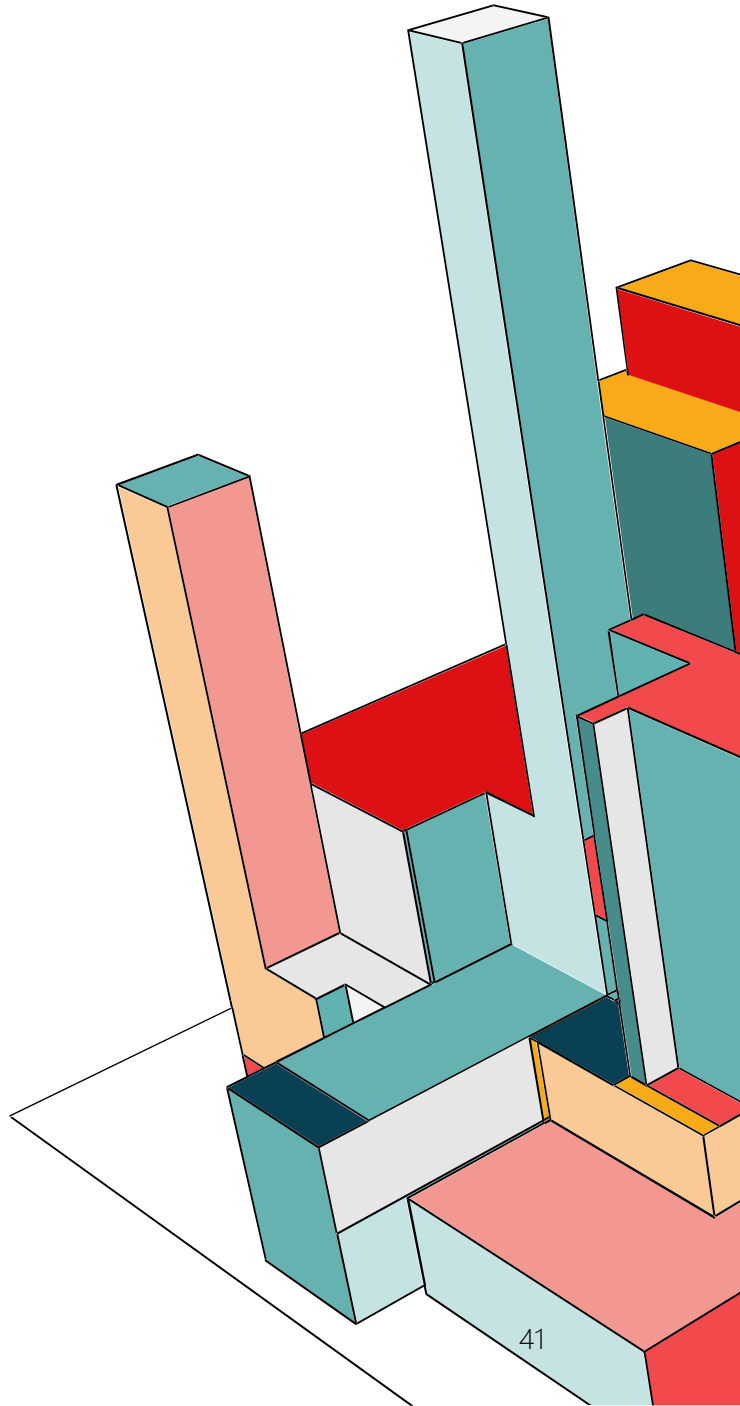
Google engineers citing documentations as productivity issue (Googlegeist 2014)



ONCE UPON A TIME ...

50%+

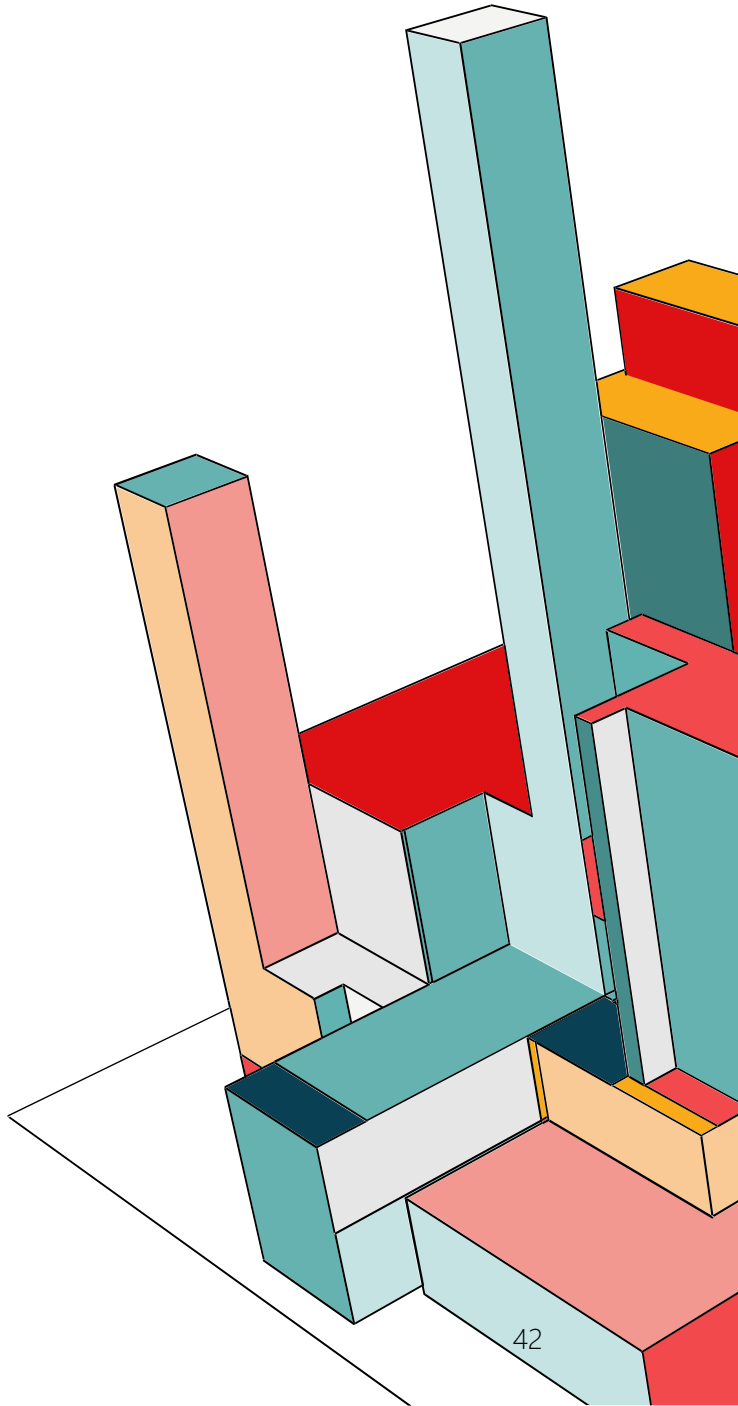
Google SRE issues cited problems with documentation (Googlegeist 2014)



ONCE UPON A TIME . . .

Troubles with docs

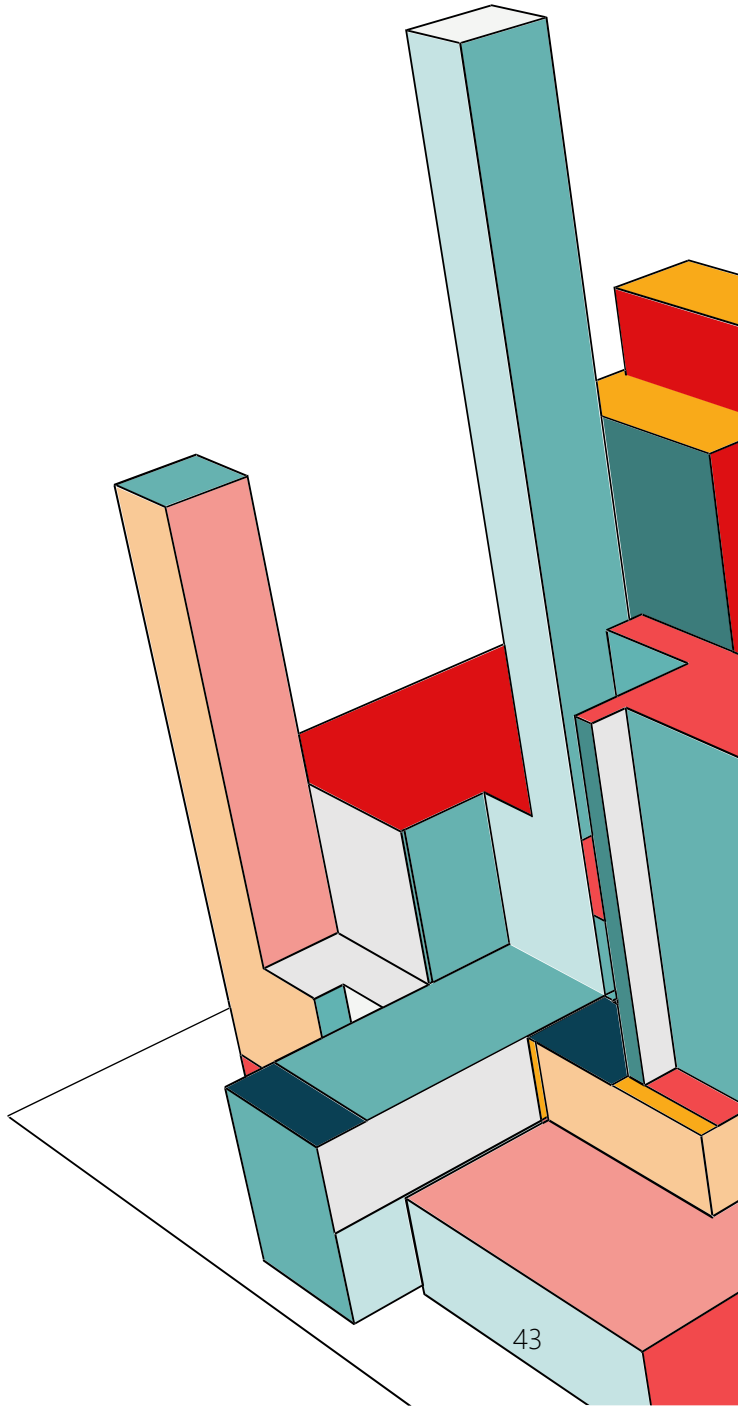
- They “don’t exist”
- They are “impossible to find”
- They are “wrong”



ONCE UPON A TIME . . .

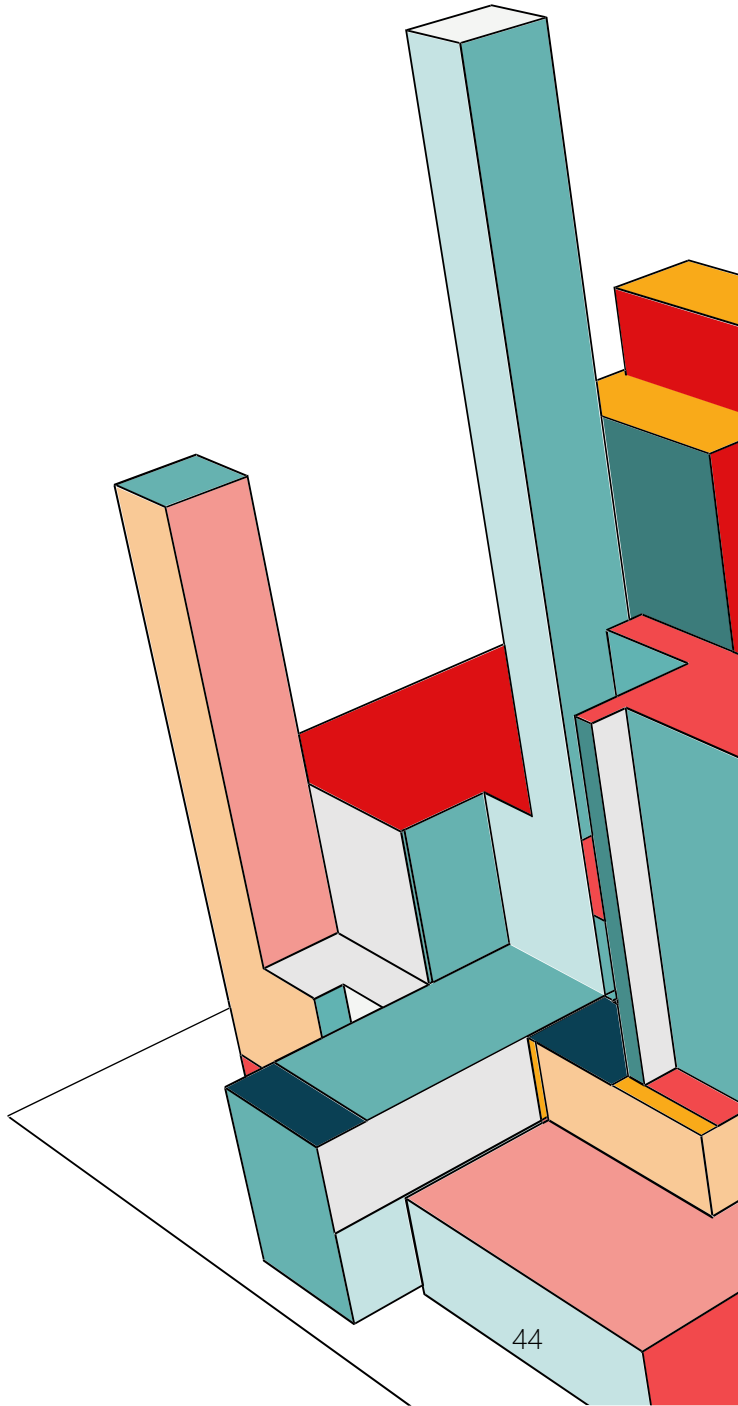
Troubles with writing docs

- “No time”
- “No incentive”



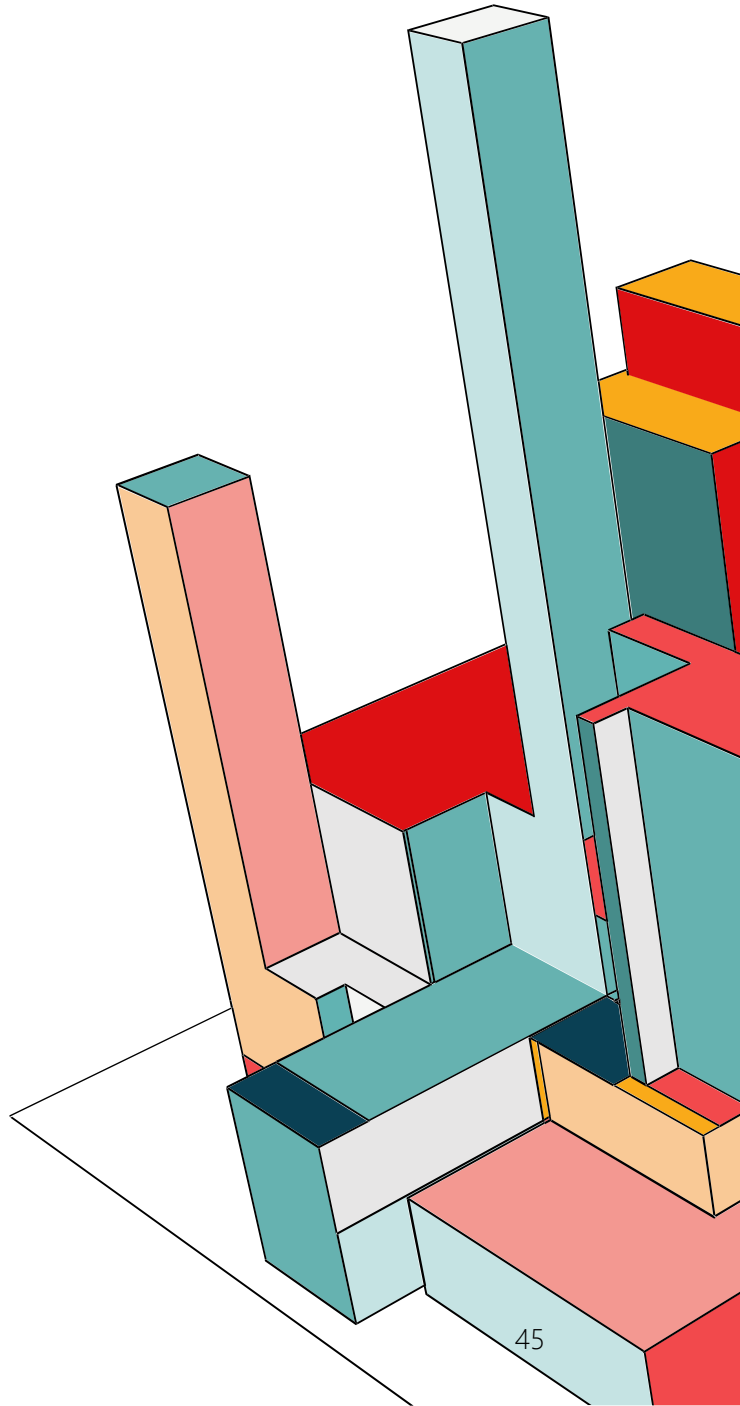
NO DOC CULTURE

Docs: Everybody’s problem, nobody’s jobs



PREVIOUS EFFORTS ... FAILED

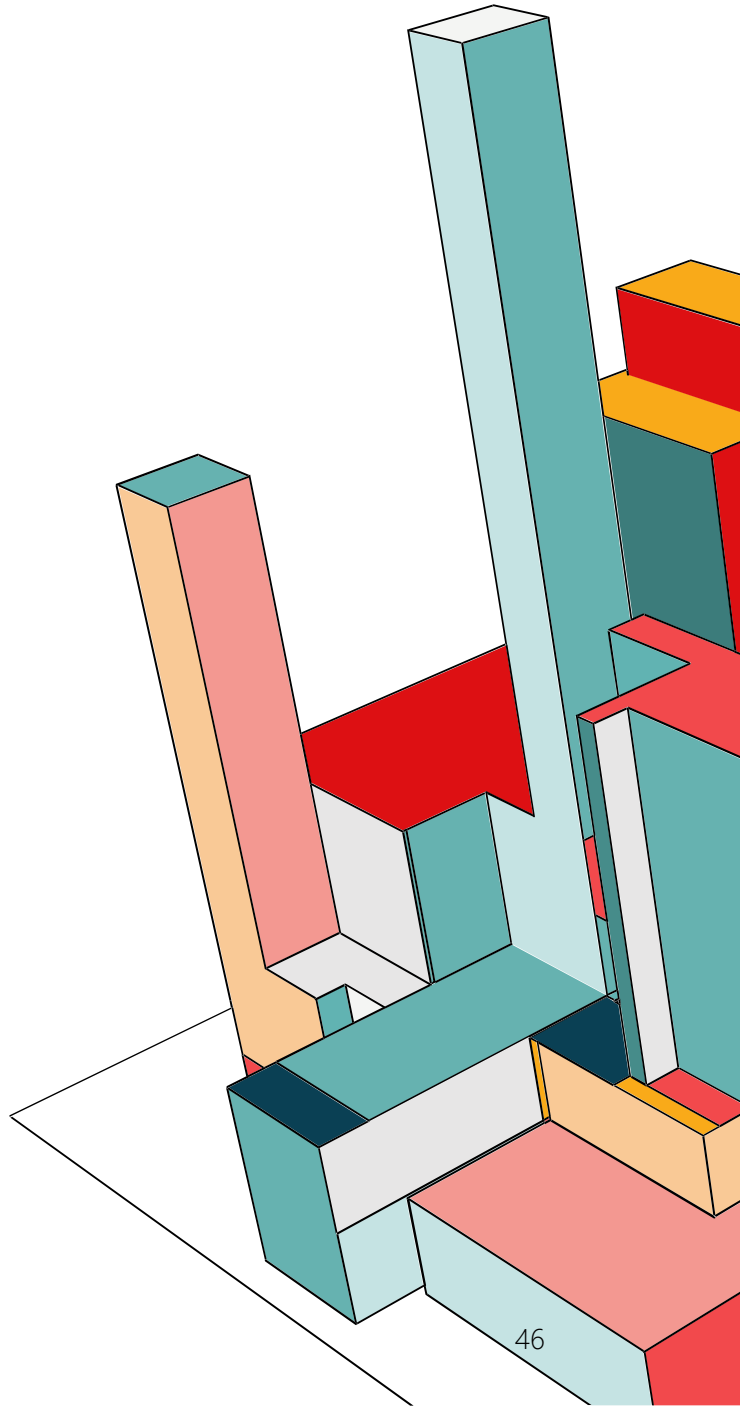
- Fix the culture (bottom up)
- Fix the tooling (top down)



LESSONS LEARNED

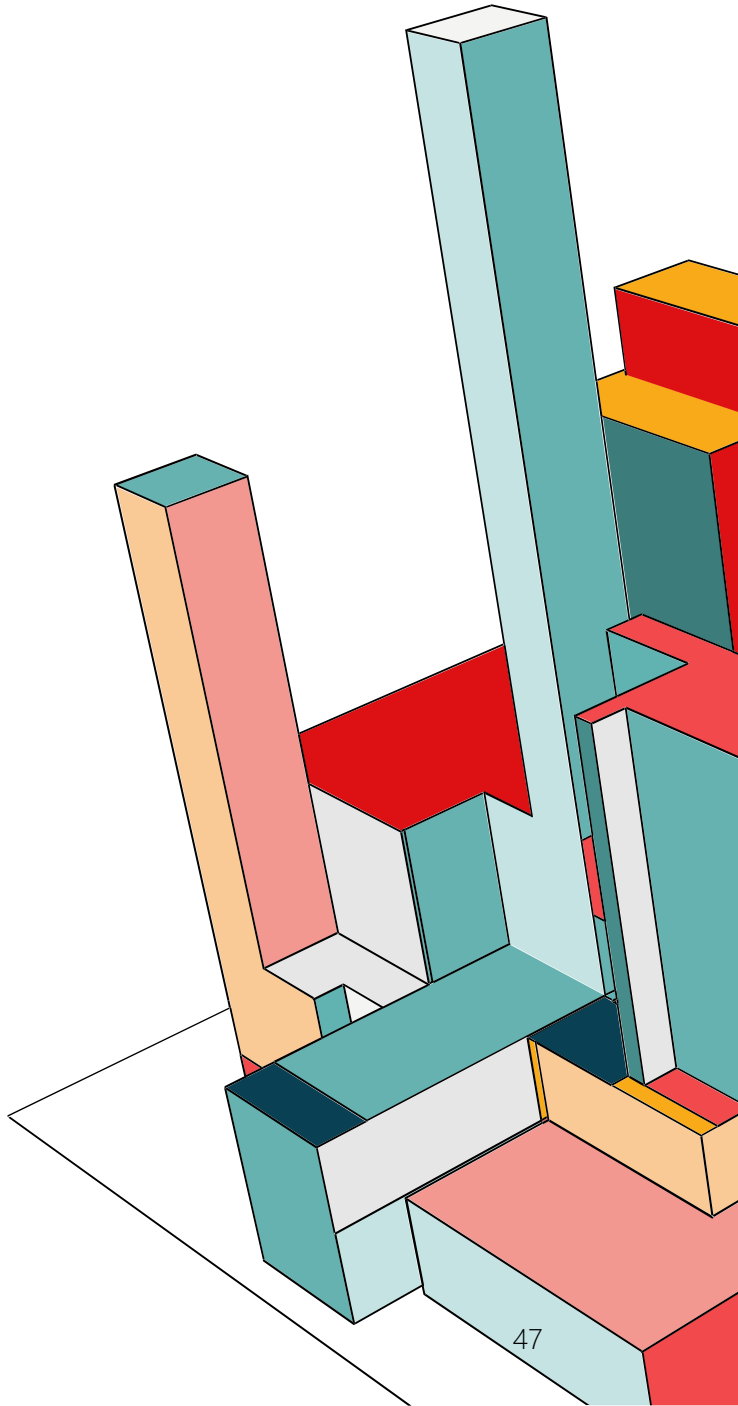
Fragmentation is bad

- Hard to create docs
  - Context switching
  - Horrible tooling
- Hard to maintain docs
  - No explicit connection to code
  - No clear ownership
- Hard to find docs



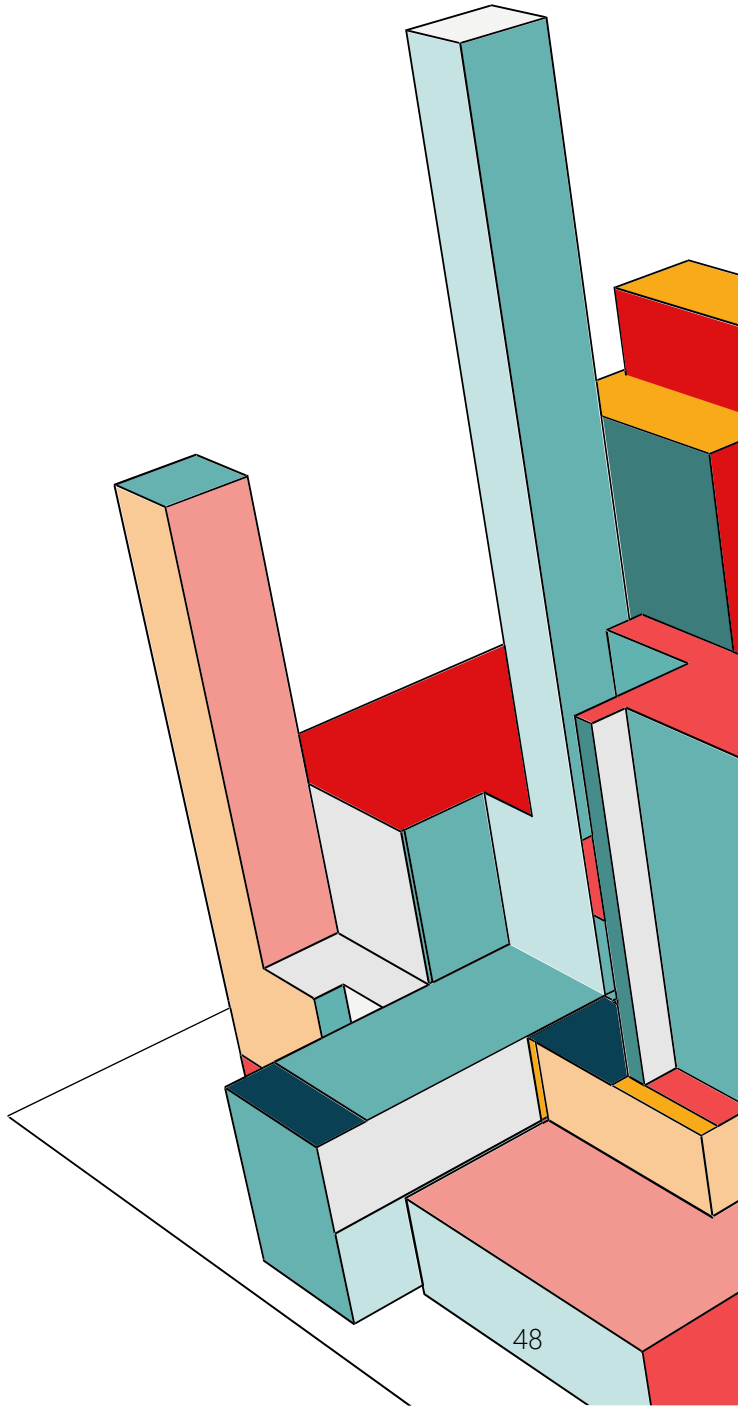
LESSONS LEARNED

Documentation will never be part of the engineering culture until it is integrated into the codebase and workflow



g3doc

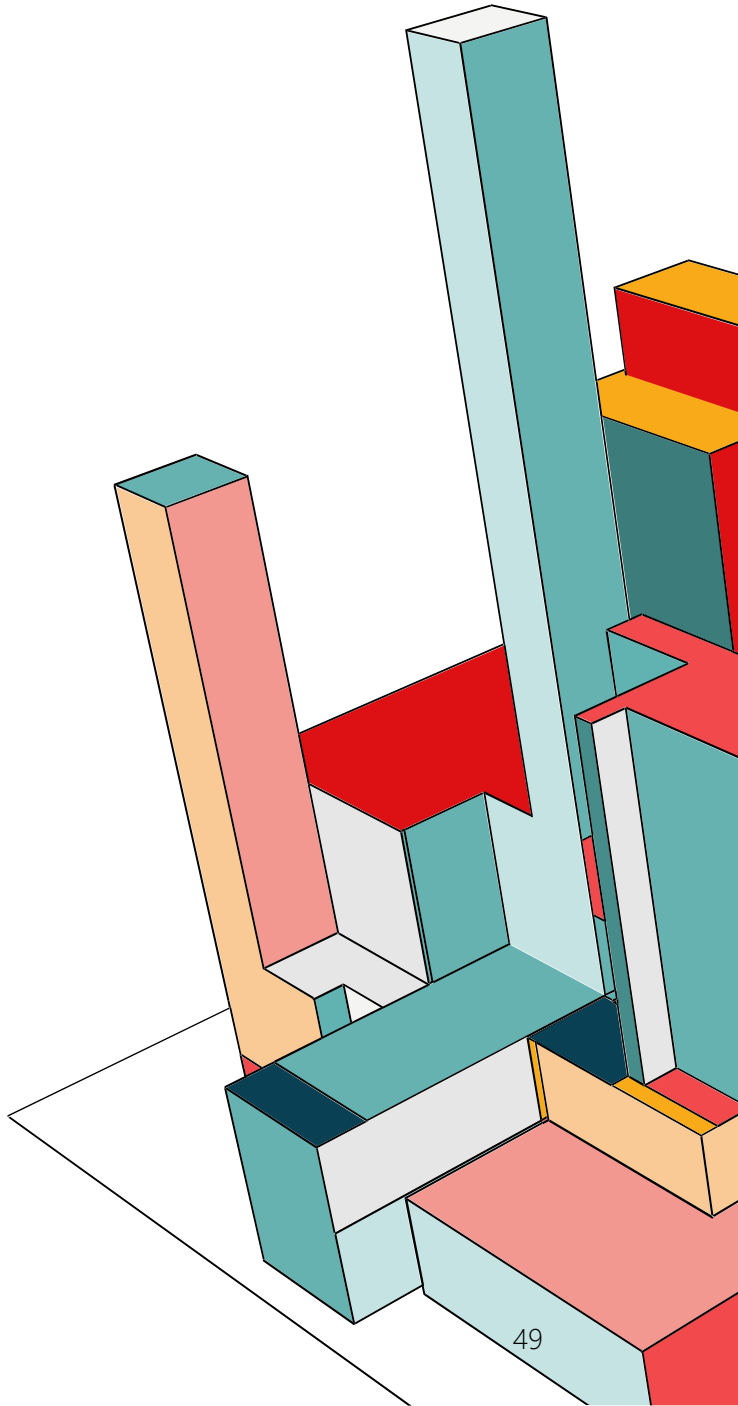
Radically simple documentation



g3doc vision

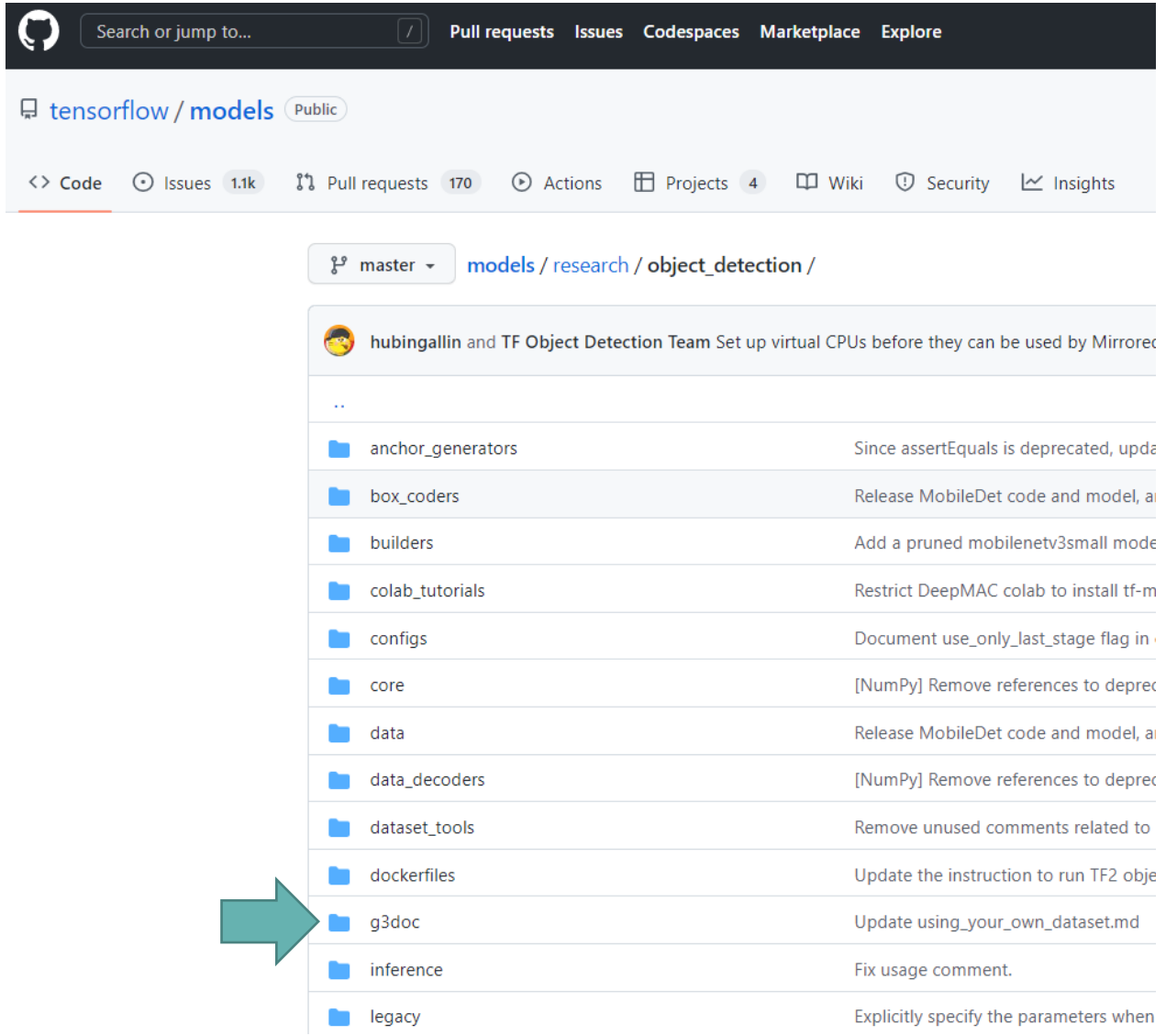
Engineers find, create, and maintain docs using their regular workflow and developers tools

Documentation is treated as code



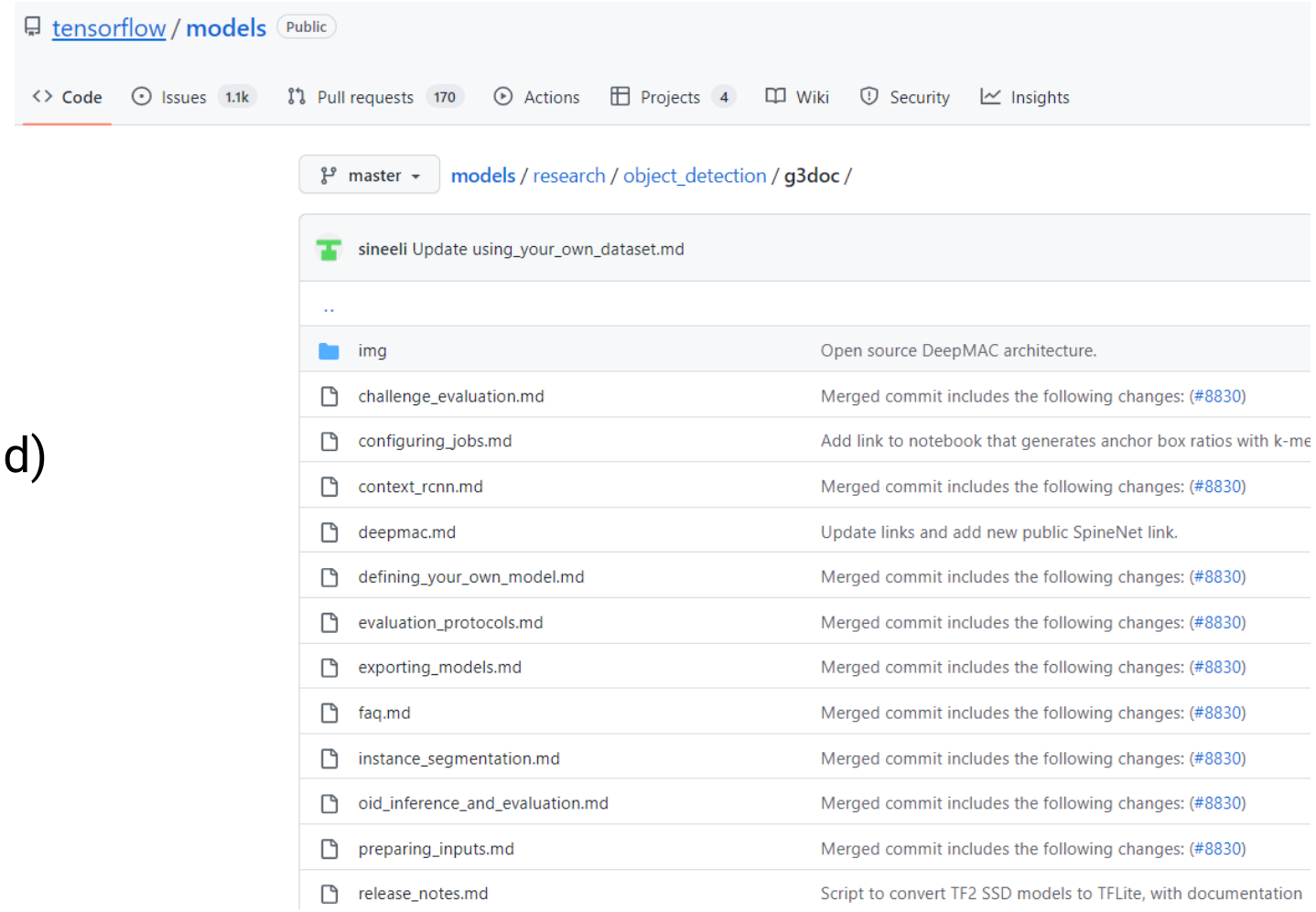
DESIGN 1: DOCS LIVES WITH CODE

Docs are stored side-by-side with the source code in the codebase



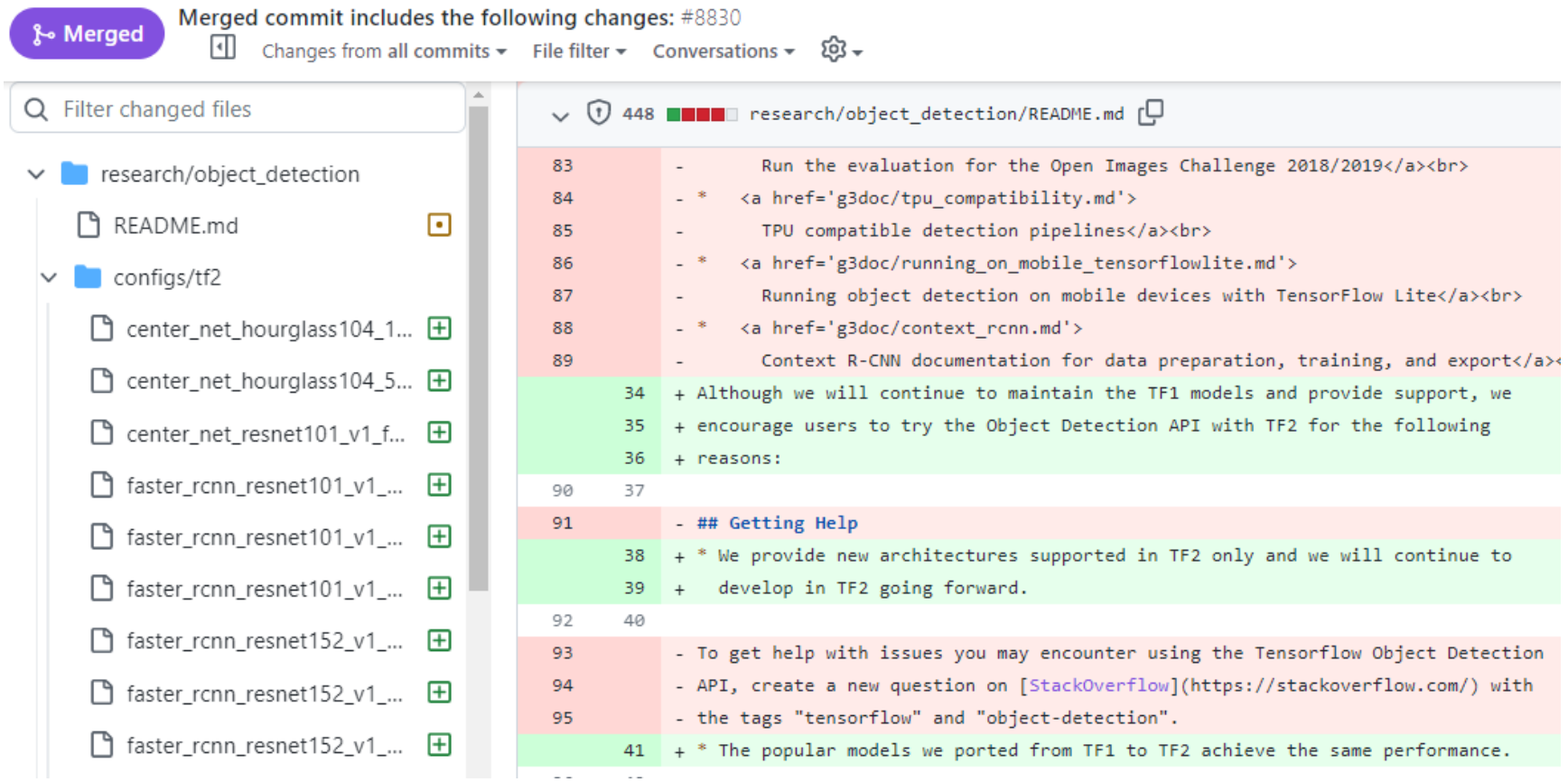
DESIGN 2: USE MARKDOWN

Docs are written in markdown format (.md)



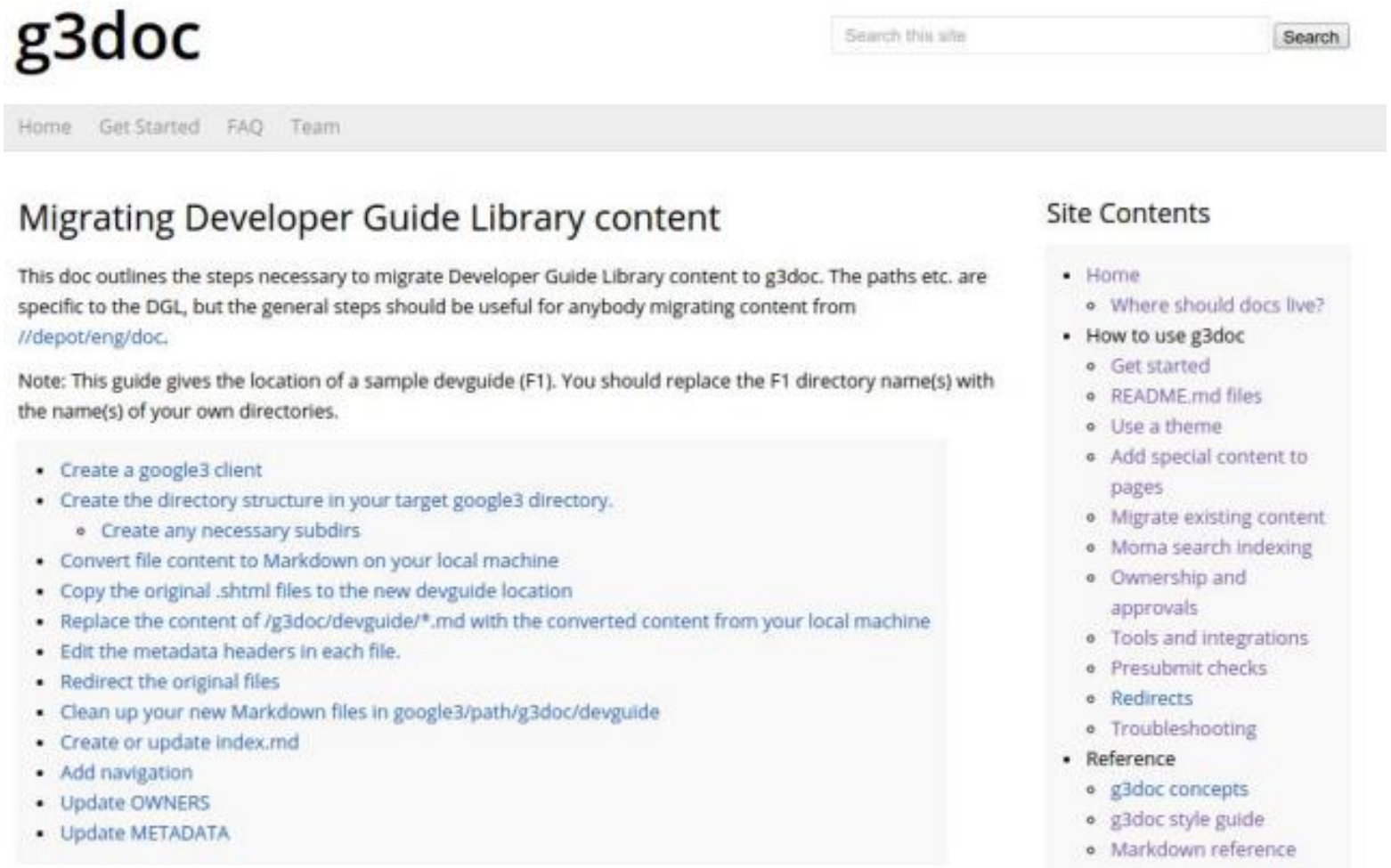
BENEFITS

- Docs are under version control like code
- Allow code review, diff, blame, fixes, and issue tracking



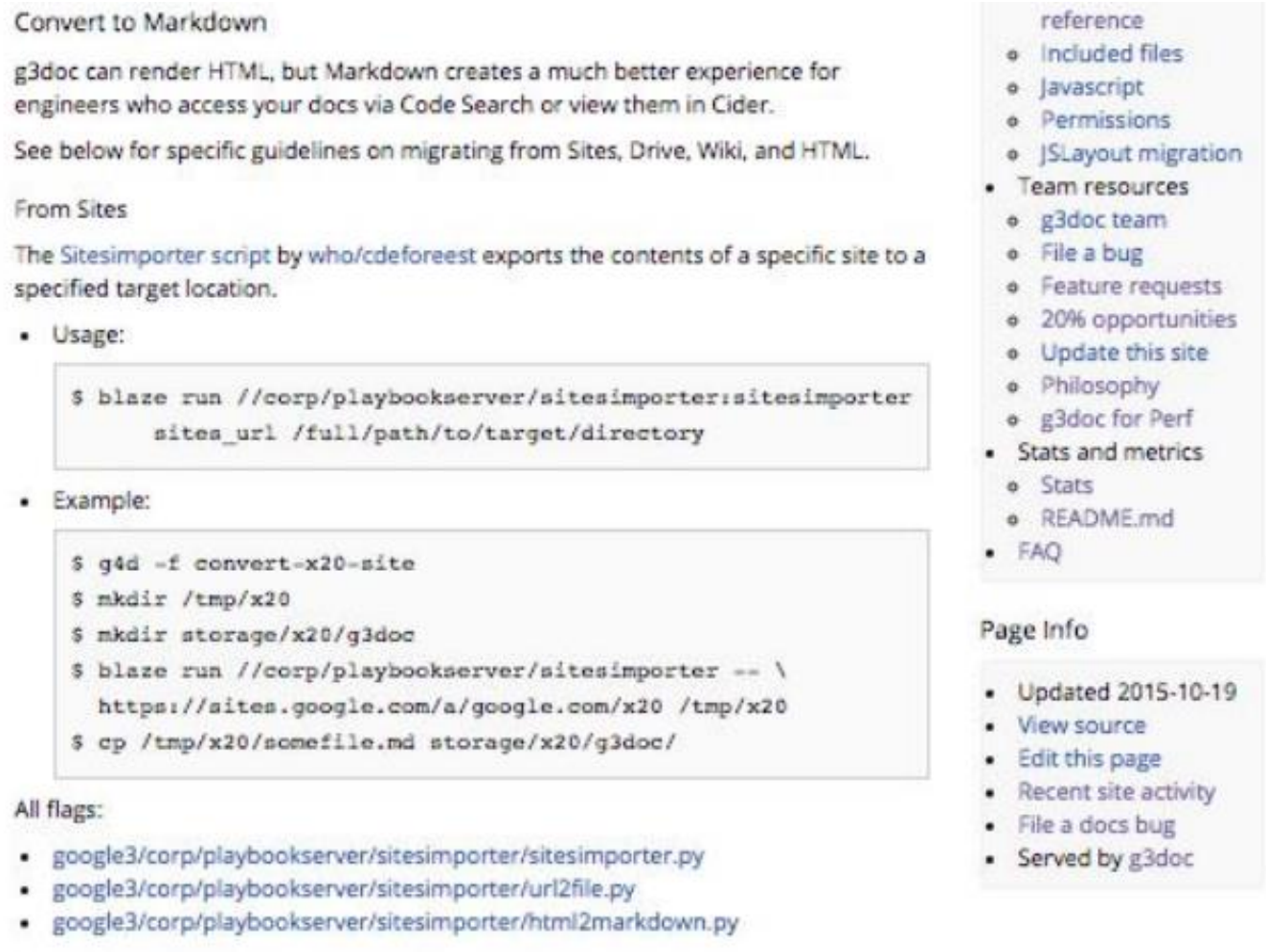
BENEFITS

- g3doc renders beautiful html based on .md files
- Developers focus on the contents, instead of the presentation



BENEFITS

- Docs have rich links to source code
- Easy for code search



Convert to Markdown

g3doc can render HTML, but Markdown creates a much better experience for engineers who access your docs via Code Search or view them in Cider.

See below for specific guidelines on migrating from Sites, Drive, Wiki, and HTML.

From Sites

The Sitesimporter script by who/cdeforeest exports the contents of a specific site to a specified target location.

- Usage:

```
$ blaze run //corp/playbookserver/sitesimporter:sitesimporter
      sites_url /full/path/to/target/directory
```

- Example:

```
$ g4d -f convert-x20-site
$ mkdir /tmp/x20
$ mkdir storage/x20/g3doc
$ blaze run //corp/playbookserver/sitesimporter -- \
  https://sites.google.com/a/google.com/x20 /tmp/x20
$ cp /tmp/x20/somefile.md storage/x20/g3doc/
```

All flags:

- google3/corp/playbookserver/sitesimporter/sitesimporter.py
- google3/corp/playbookserver/sitesimporter/url2file.py
- google3/corp/playbookserver/sitesimporter/html2markdown.py

reference
- Included files
- Javascript
- Permissions
- JSLayout migration

- Team resources
  - g3doc team
  - File a bug
  - Feature requests
  - 20% opportunities
  - Update this site
  - Philosophy
  - g3doc for Perf
- Stats and metrics
  - Stats
  - README.md
- FAQ

Page Info

- Updated 2015-10-19
- View source
- Edit this page
- Recent site activity
- File a docs bug
- Served by g3doc

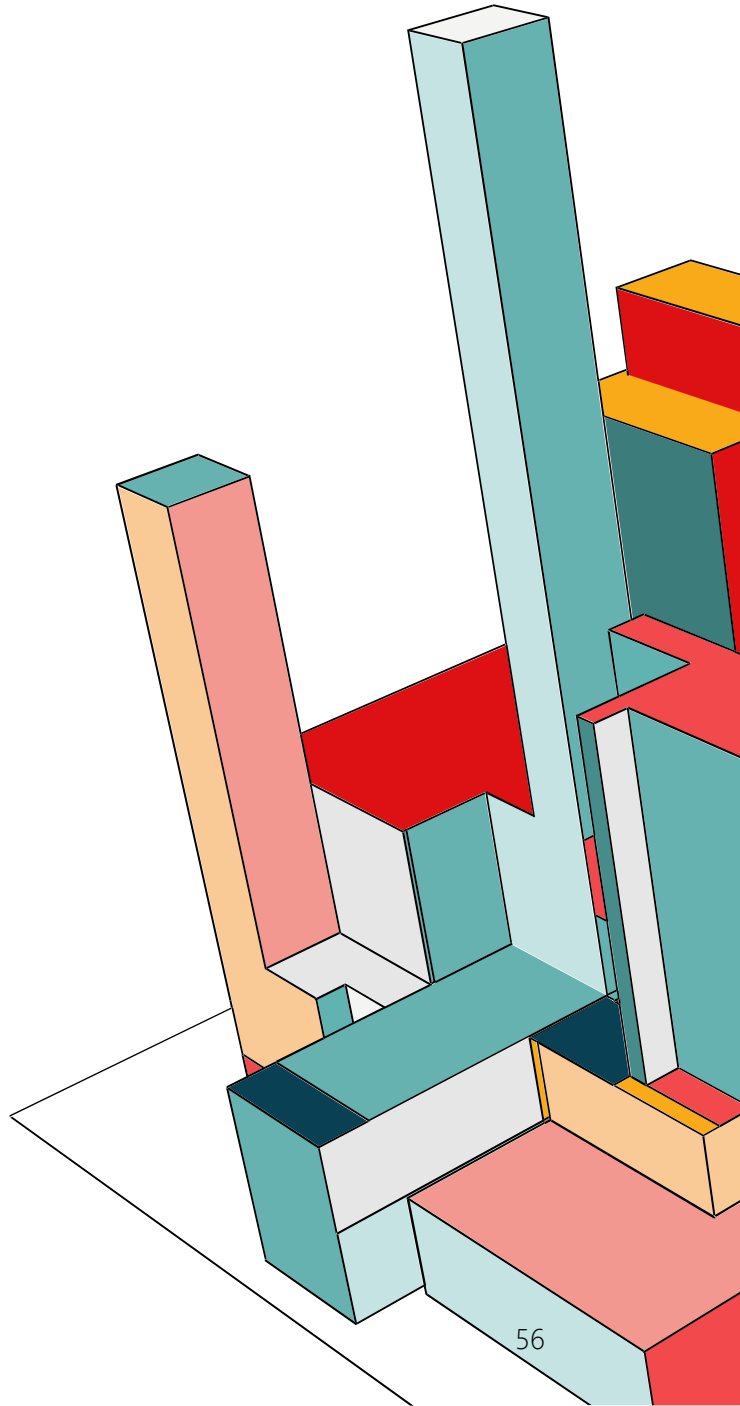
Case study @ Google

GROWTH OF G3DOC

ACTIVELY MAINTAINED

30%

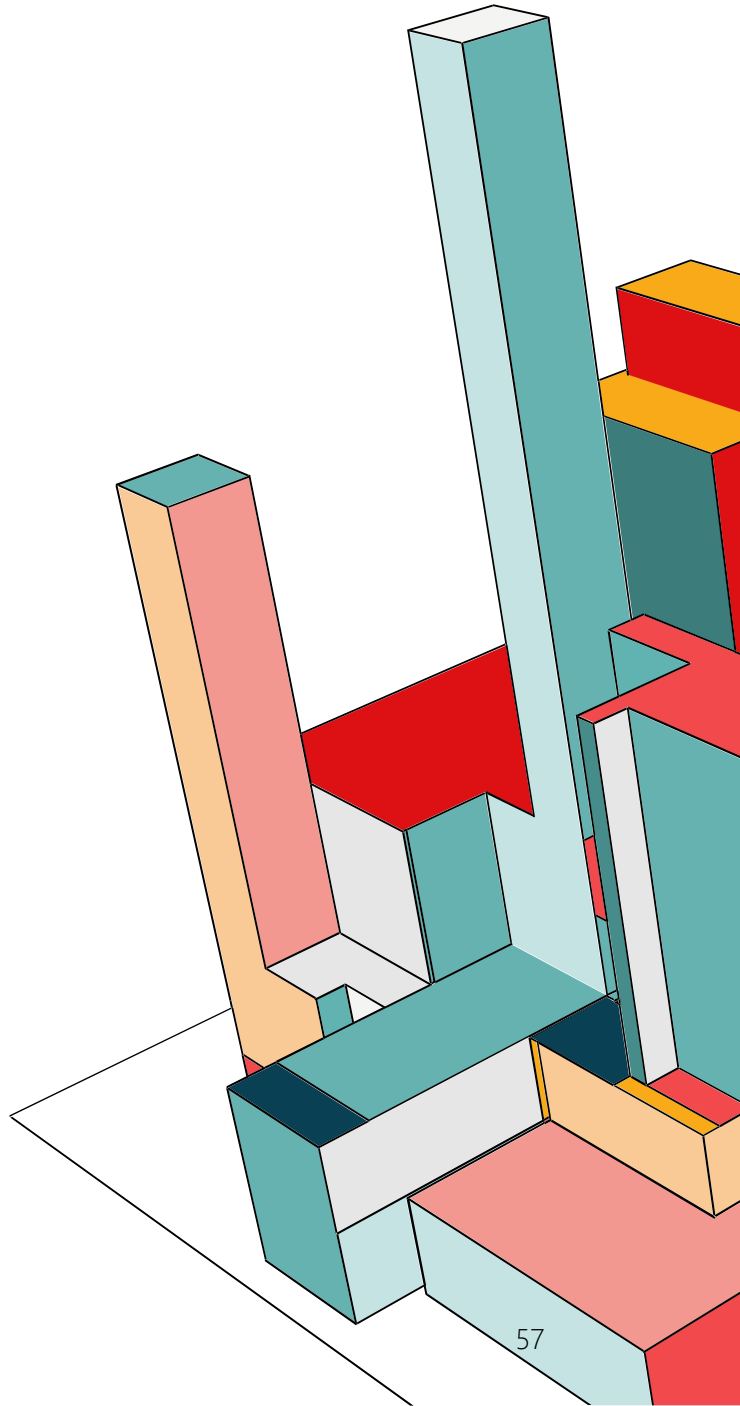
Docs changelists (~commits) contain code



ACTIVELY MAINTAINED

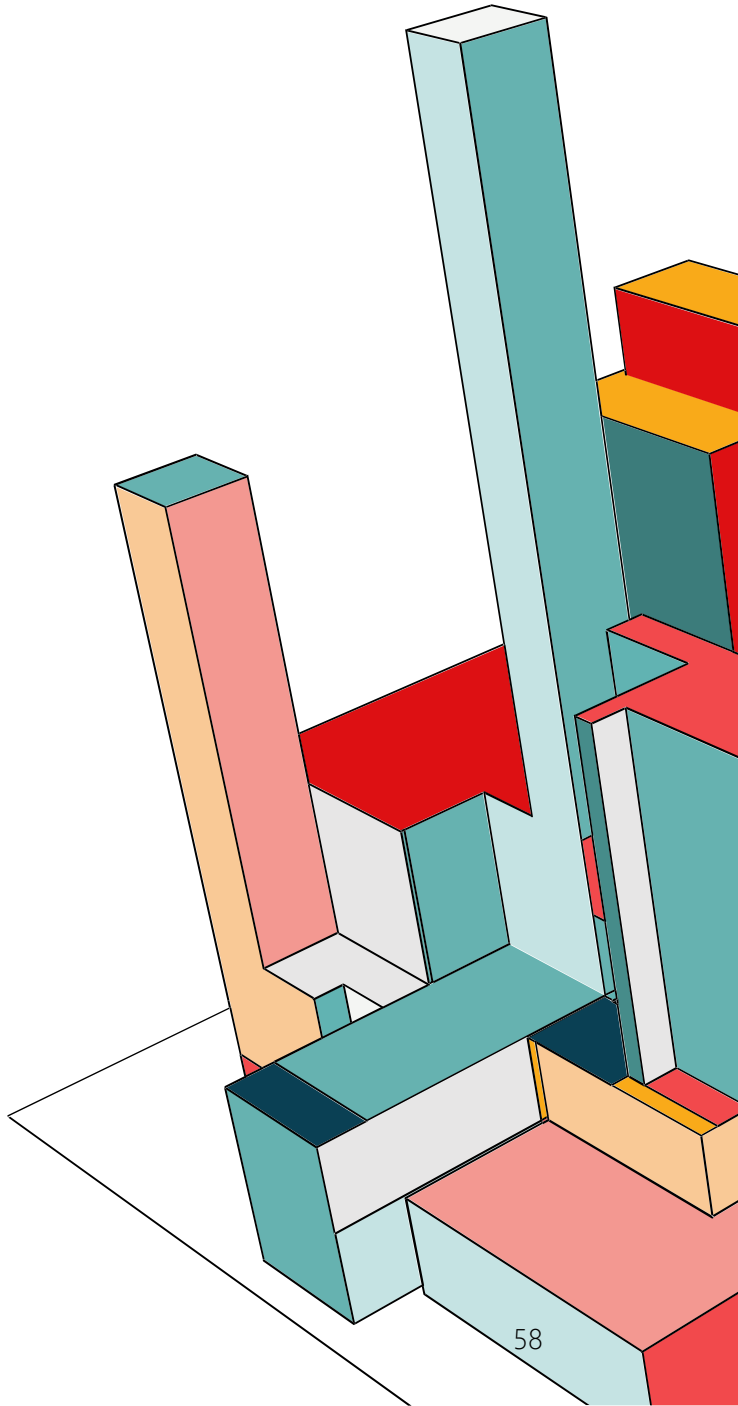
200K files
200K+ doc changelists

3 changelists per file / month



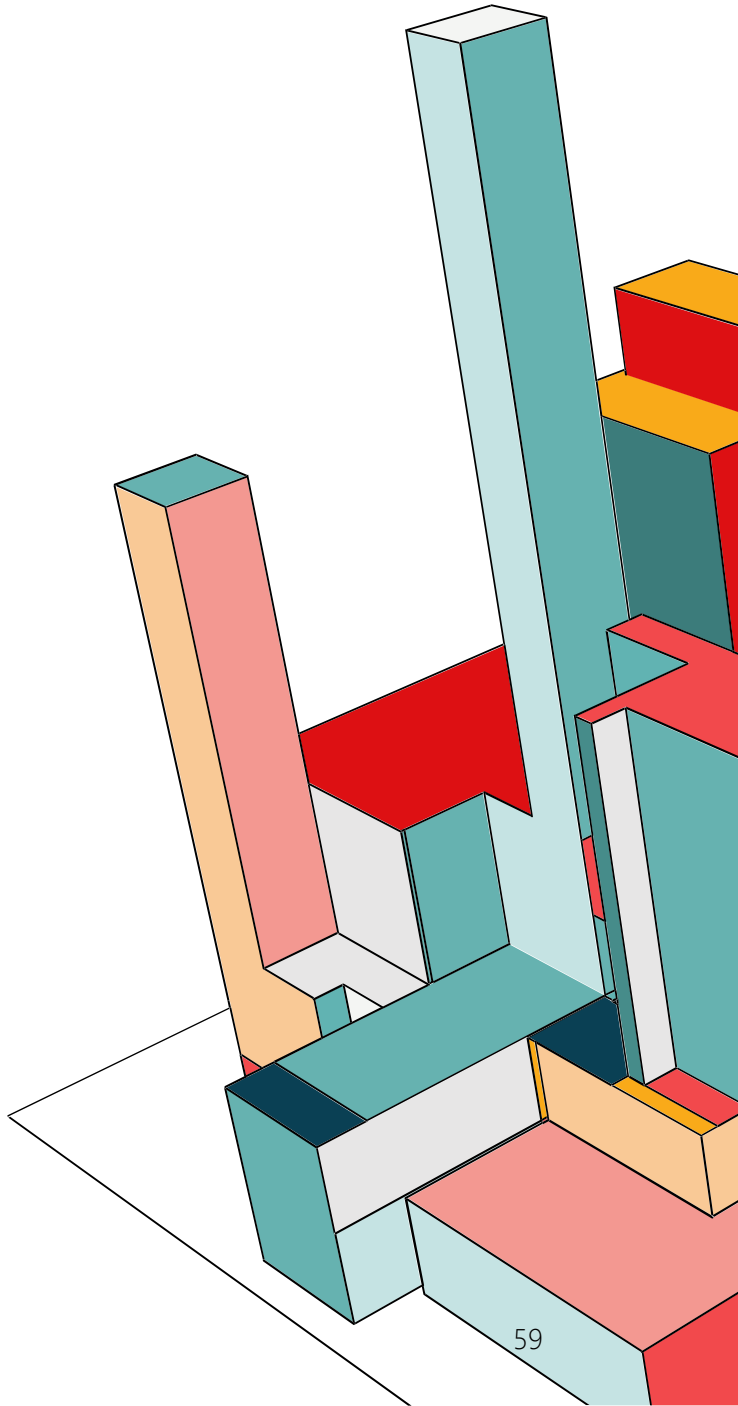
ACTIVELY USED

17K authors



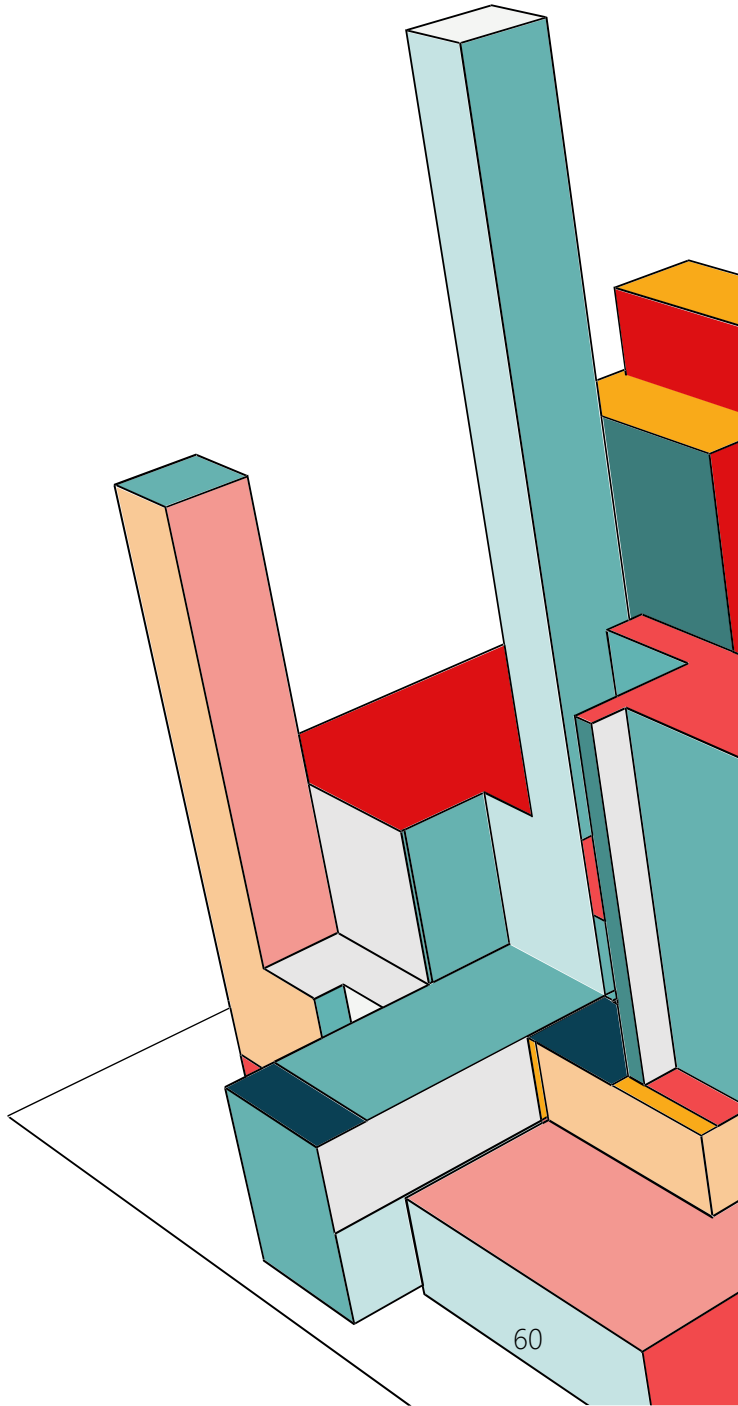
ACTIVELY USED

3.9M 30-day page views



READINGS

- Chapter 10. Documentation. Software Engineering at Google by Titus Winters, et al.
- 第4.2章 代码风格. 现代软件工程基础 by 彭鑫 et al.



NEXT

- Software Testing