Technological foundations of software development

Document, license, publish, maintain your software

Objectives of the session

This session aims to familiarize you with the methods and tools for Document, license, publish, maintain your software.
In particular, we will cover tools for Java, Python, JavaScript.

Technological foundations of software development

Document, license, publish, maintain your software

Part 1: Documenting your code

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

Documenting your code: Why ?

- Explain how the program works
- Explain how to use the program

https://en.wikipedia.org/wiki/Software_documentation

https://www.commitstrip.com/en/2015/06/29/as-the-last-resort/?

Slide from: Software Engineering, part 1.

Types of documentation

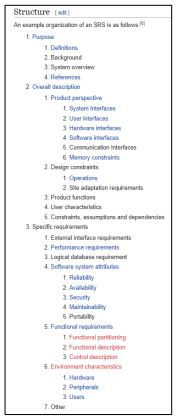
- **Requirements** - Statements that identify the attributes, capabilities, characteristics or qualities of a system. It is the basis for what will be or has been implemented.
- **Architecture/Design** - An overview of the software. Includes the relationships with an environment and the construction principles to be used in the design of software components.
- **Technical** - Documentation of code, algorithms, interfaces and APIs.
- **End User** - Manuals for the end user, system administrators, and support staff.
- **Marketing** - How to market the product and market demand analysis.

Requirements specification document

specification (in software engineering)
"production of a document that can be systematically reviewed, evaluated, and approved"
— ISO/IEC TR 19759:2015 Software Engineering Body of Knowledge (SWEBOK)

software requirements specification (SRS)
"structured collection of the essential requirements [functions, performance, design constraints and attributes] of the software and its external interfaces"
— IEEE 1012-2016 - IEEE Standard for System, Software, and Hardware Verification and Validation

see also https://en.wikipedia.org/wiki/Software_requirements_specification


Structure [edit]
An example organization of an SRS is as follows:[9]
1. Purpose
  1. Definitions
  2. Background
  3. System overview
  4. References
2. Overall description
  1. Product perspective
    1. System interfaces
    2. User interfaces
    3. Hardware interfaces
    4. Software interfaces
    5. Communication interfaces
    6. Memory constraints
  2. Design constraints
    1. Operations
    2. Site adaptation requirements
  3. Product functions
  4. User characteristics
  5. Constraints, assumptions and dependencies
3. Specific requirements
  1. External interface requirements
  2. Performance requirements
  3. Logical database requirement
  4. Software system attributes
    1. Reliability
    2. Availability
    3. Security
    4. Maintainability
    5. Portability
  5. Functional requirements
    1. Functional partitioning
    2. Functional description
    3. Control description
  6. Environment characteristics
    1. Hardware
    2. Peripherals
    3. Users
  7. Other

example organization of a SRS document
source: https://en.wikipedia.org/wiki/Software_requirements_specification

Architecture/design

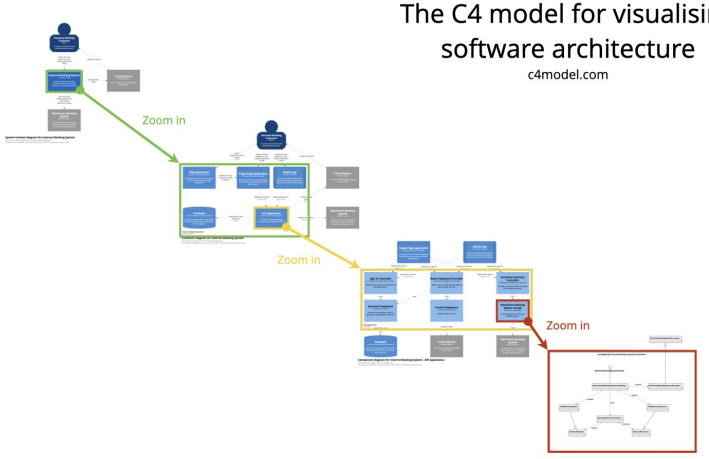
Software architecture description
the set of practices for expressing, communicating and analysing software architectures (also called architectural rendering), and the result of applying such practices through a work product expressing a software architecture

— ISO/IEC/IEEE 42010 Systems and software engineering — Architecture description

existing languages such as the UML can be used as Architecture description languages for analysis, design, and implementation of software-based systems as well as for modeling business and similar processes.

The C4 model for visualising software architecture

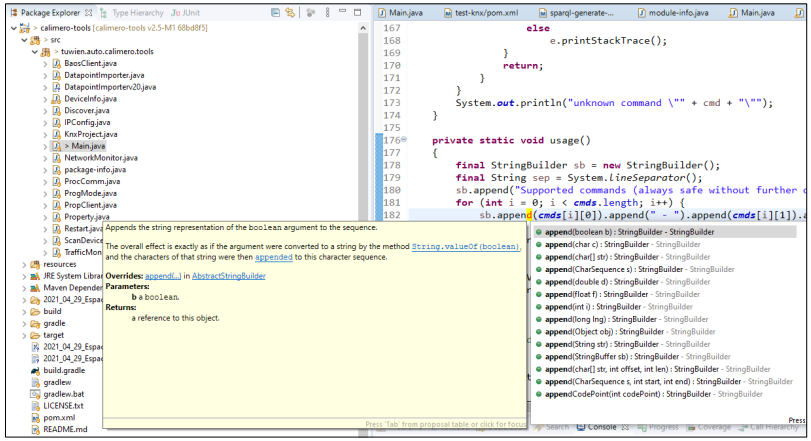
c4model.com



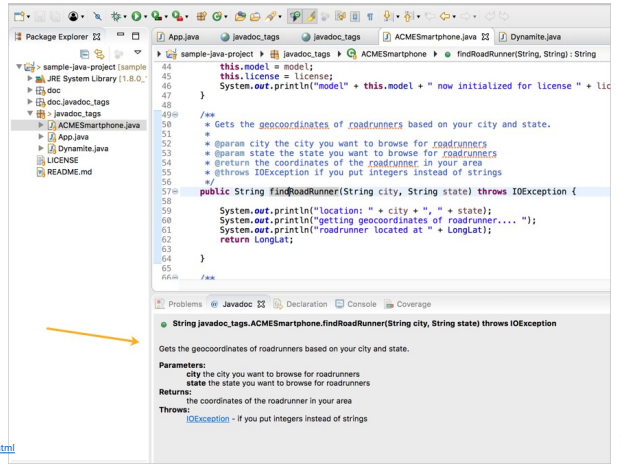
Level 1 Context | Level 2 Containers | Level 3 Components | Level 4 Code

https://c4model.com/

Technical documentation



Technical documentation

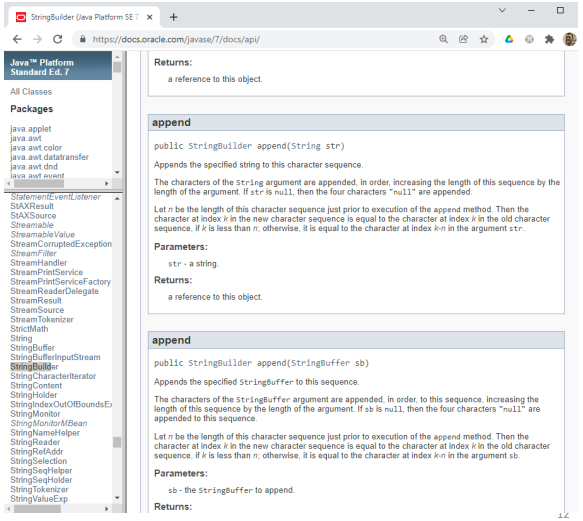


https://www.oracle.com/java/technologies/javase/javadoc-tool.html

Tag	Description	Syntax
@author	Adds the author of a class.	@author name-text
{@code}	Displays text in code font without interpreting the text as HTML markup or nested javadoc tags.	{@code text}
{@docRoot}	Represents the relative path to the generated document's root directory from any generated page.	{@docRoot}
@deprecated	Adds a comment indicating that this API should no longer be used.	@deprecated deprecatedtext
@exception	Adds a **Throws** subheading to the generated documentation, with the classname and description text.	@exception class-name description
{@inheritDoc}	Inherits a comment from the **nearest** inheritable class or implementable interface.	Inherits a comment from the immediate superclass.
{@link}	Inserts an in-line link with the visible text label that points to the documentation for the specified package, class, or member name of a referenced class.	{@link package.class#member label}
{@linkplain}	Identical to {@link}, except the link's label is displayed in plain text than code font.	{@linkplain package.class#member label}
@param	Adds a parameter with the specified parameter-name followed by the specified description to the "Parameters" section.	@param parameter-name description
@return	Adds a "Returns" section with the description text.	@return description
@see	Adds a "See Also" heading with a link or text entry that points to reference.	@see reference
@serial	Used in the doc comment for a default serializable field.	@serial field-description | include | exclude
@serialData	Documents the data written by the writeObject( ) or writeExternal( ) methods.	@serialData data-description
@serialField	Documents an ObjectStreamField component.	@serialField field-name field-type field-description
@since	Adds a "Since" heading with the specified since-text to the generated documentation.	@since release
@throws	The @throws and @exception tags are synonyms.	@throws class-name description
{@value}	When {@value} is used in the doc comment of a static field, it displays the value of that constant.	{@value package.class#field}
@version	Adds a "Version" subheading with the specified version-text to the generated docs when the -version option is used.	@version version-text

https://www.javaguides.net/2018/12/the-javadoc-tags-explained.html
https://www.oracle.com/java/technologies/javase/javadoc-tool.html

- generate HTML pages
- generate javadoc jars



https://www.javaguides.net/2018/12/the-javadoc-tags-explained.html
https://www.oracle.com/java/technologies/javase/javadoc-tool.html

Python docstrings

```python
class DocsBuilder:
    """
    Class used for automatically generating HTML-based documentation from
    Python code. Note that function docstrings must also follow NumPy/SciPy
    docstring formatting conventions.
    """
    def __init__(self, docs_dir='docs', offline=False):
        """
        ... docstring ...
        """
        ... code ...

    def extract(self, code):
        """
        ... docstring ...
        """
        ... code ...


def output(code, filename, path="docs"):
```

https://devguide.python.org/documenting/

Python doc conventions

```
"""
This is a javadoc style.

@param param1: this is a first param
@param param2: this is a second param
@return: this is a description of what is returned
@raise keyError: raises an exception
"""
```

Epytext

```
"""
This is an example of Google style.

Args:
    param1: This is the first param.
    param2: This is a second param.

Returns:
    This is a description of what is returned.

Raises:
    KeyError: Raises an exception.
"""
```

Google

recommended

```
"""
This is a reST style.

:param param1: this is a first param
:param param2: this is a second param
:returns: this is a description of what is returned
:raises keyError: raises an exception
"""
```

reStructuredText -> used by Sphinx

https://devguide.python.org/documenting/

JavaScript documentation: JSDoc

Initial release	1999; 22 years ago
Latest release	3.6.3 (15 July 2019; 2 years ago)
Type of format	Programming documentation Format
Contained by	JavaScript source files
Extended from	JavaDoc
Open format?	Yes
Website	jsdoc.app

```javascript
/** @class Circle representing a circle. */
class Circle {
/**
 * Creates an instance of Circle.
 *
 * @author: moi
 * @param {number} r The desired radius of the circle.
 */
 constructor(r) {
   /** @private */ this.radius = r
   /** @private */ this.circumference = 2 * Math.PI * r
 }

 /**
  * Creates a new Circle from a diameter.
  *
  * @param {number} d The desired diameter of the circle.
  * @return {Circle} The new Circle object.
  */
 static fromDiameter(d) {
   return new Circle(d / 2)
 }

 /**
  * Calculates the circumference of the Circle.
  *
  * @deprecated since 1.1.0; use getCircumference instead
  * @return {number} The circumference of the circle.
  */
 calculateCircumference() {
   return 2 * Math.PI * this.radius
 }

 /**
  * Returns the pre-computed circumference of the Circle.
  *
  * @return {number} The circumference of the circle.
  * @since 1.1.0
  */
 getCircumference() {
   return this.circumference
 }

 /**
  * Find a String representation of the Circle.
  *
  * @override
  * @return {string} Human-readable representation of this Circle.
  */
 toString() {
   return `[A Circle object with radius of ${this.radius}.]`
 }
}

/**
 * Prints a circle.
 *
 * @param {Circle} circle
 */
function printCircle(circle) {
    /** @this {Circle} */
    function bound() { console.log(this) }
    bound.apply(circle)
}
```

https://en.wikipedia.org/wiki/JSDoc
https://jsdoc.app/

JavaScript documentation: JSDoc

Initial release	1999; 22 years ago
Latest release	3.6.3 (15 July 2019; 2 years ago)
Type of format	Programming documentation Format
Contained by	JavaScript source files
Extended from	JavaDoc
Open format?	Yes
Website	jsdoc.app

Tag	Description
`@author`	Developer's name
`@constructor`	Marks a function as a constructor
`@deprecated`	Marks a method as deprecated
`@exception`	Synonym for `@throws`
`@exports`	Identifies a member that is exported by the module
`@param`	Documents a method parameter; a datatype indicator can be added between curly braces
`@private`	Signifies that a member is private
`@returns`	Documents a return value
`@return`	Synonym for `@returns`
`@see`	Documents an association to another object
`@todo`	Documents something that is missing/open
`@this`	Specifies the type of the object to which the keyword *`this`* refers within a function.
`@throws`	Documents an exception thrown by a method
`@version`	Provides the version number of a library

https://en.wikipedia.org/wiki/JSDoc
https://jsdoc.app/

Self-documenting code

- Make source code easier to read and understand
- Minimize the effort required to maintain or extend legacy systems
- Reduce the need for users and developers of a system to consult secondary documentation sources such as code comments or software manuals
- Facilitate automation through self-contained knowledge representation

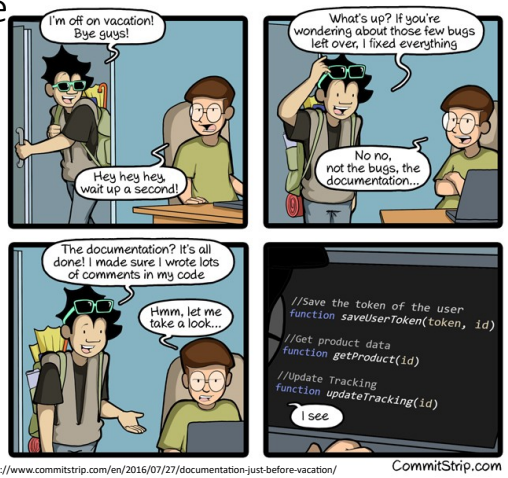
Self-documenting code

- Make source code easier to read and understand
- Minimize the effort required to maintain or extend legacy systems
- Reduce the need for users and developers of a system to consult secondary documentation sources such as code comments or software manuals
- Facilitate automation through self-contained knowledge representation

```c
size_t count_alphabetic_chars(const char *text)
{
    if (text == NULL)
        return 0;

    size_t  count = 0;

    while (*text != '\0')
    {
        if (is_alphabetic(*text))
            count++;
        text++;
    }

    return count;
}
```


Self-documenting code

- Make source code easier to read and understand
- Minimize the effort required to maintain or extend legacy systems
- Reduce the need for users and developers of a system to consult secondary documentation sources such as code comments or software manuals
- Facilitate automation through self-contained knowledge representation



https://www.commitstrip.com/en/2016/07/27/documentation-just-before-vacation/

Self-documenting code

1. Move code to function

```c
var width = (value - 0.5) * 16;
```

```c
var width = emToPixels(value);

function emToPixels(ems) {
    return (ems - 0.5) * 16;
}
```

https://multi-programming.com/blog/self-documenting-code

Self-documenting code

2. Expression is replaced with a variable

```
var isVisible = el.offsetWidth && el.offsetHeight;
if(!isVisible) {
}
```

```
return a * b + (c / d);
```

```
var multiplier = a * b;
var divisor = c / d;
return multiplier + divisor;
```


Self-documenting code

3. Class and module interfaces

```
class Box {
    setState(state) {
        this.state = state;
    }

    getState() {
        return this.state;
    }
}
```

```
class Box {
    open() {
        this.state = 'open';
    }

    close() {
        this.state = 'closed';
    }

    isOpen() {
        return this.state === 'open';
    }
}
```


Self-documenting code

4. Group your code

```
var foo = 1;

blah()
xyz();

bar(foo);
baz(1337);
quux(foo);
```


Self-documenting code

5. Give another name to the function

Omit to use obscure words:

```
handleButtons(), manageLinks()
```

Use active verbs like ”send”:

```
cutLawn(), sendFolder()
```

Self-documenting code

6. Give other names to variables

Do not use silly variable names

```
a, b, c, variable1, other, …
```

7. Follow the same naming conventions

```
var element = getElement();
```

```
var node = getElement();
```


Self-documenting code

8. Avoid utilizing syntax tricks

```
imStrange && doMagic();
```

```
if(imTricky) {
    doMagic();
}
```


Self-documenting code

9. Use named constants

```
const BUY_HAPPINESS = 42;
```

10. Omit boolean flags

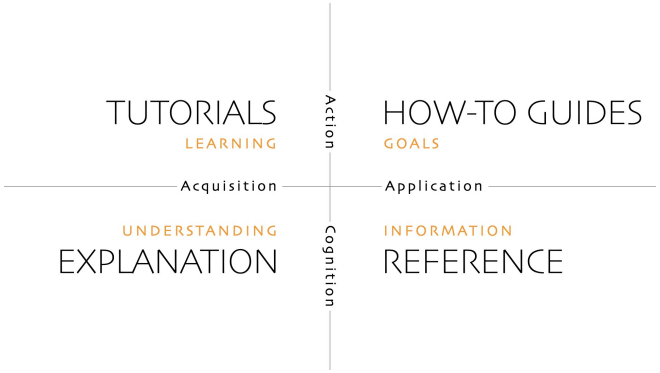
```
myStuff.setData({ x: 1 }, true);
```

```
myStuff.mergeData({ x: 1 });
```


Documentation for the end user

Diátaxis

A systematic approach to technical documentation authoring.



Documentation for the end user

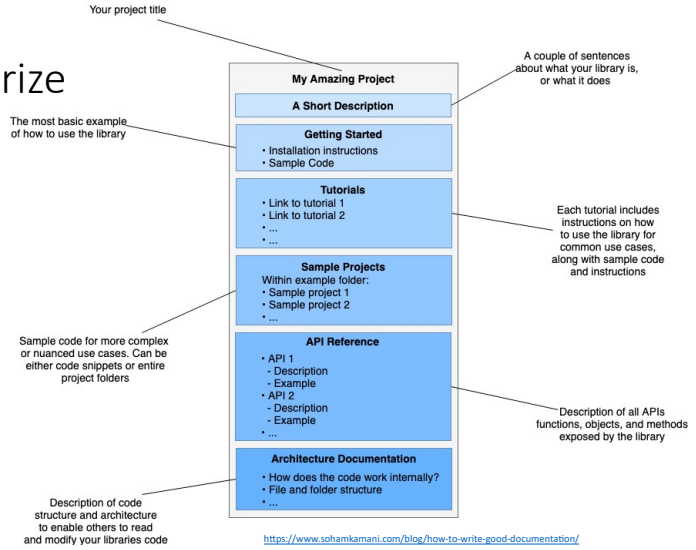
Diátaxis

A systematic approach to technical documentation authoring.

Describes how the program is installed and used

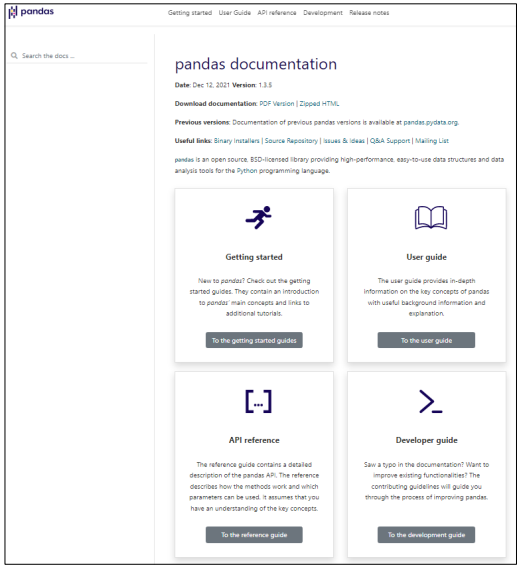
- **Tutorials:** *A tutorial is an **experience** that takes place under the guidance of a tutor. A tutorial is always **learning-oriented**.*
- **How-to guides:** *How-to guides are **directions** that guide the reader through a problem or towards a result. How-to guides are **goal-oriented**.*
- **References:** *Reference guides are **technical descriptions** of the machinery and how to operate it. Reference material is **information-oriented**.*
- **Explanations:** *Explanation is a discursive treatment of a subject, that permits reflection. Explanation is **understanding-oriented**.*

To summarize



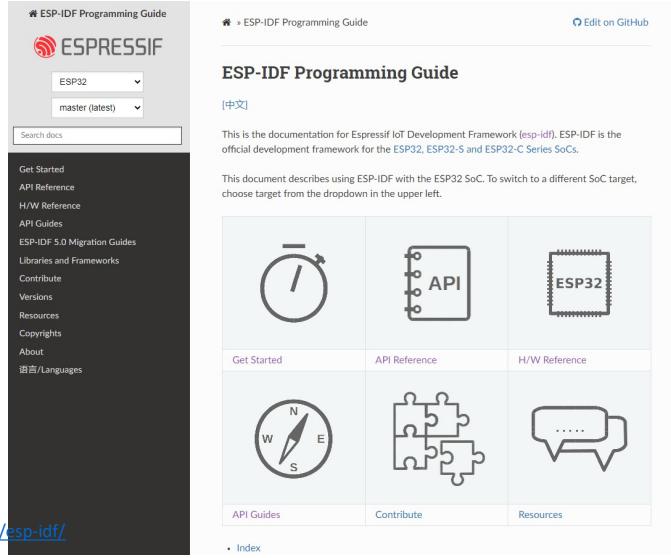
https://www.sohamkamani.com/blog/how-to-write-good-documentation/

Case study 1



https://pandas.pydata.org/docs/

Case study 2



https://docs.espressif.com/projects/esp-idf/

The README file

- A description of what the project is for.
  - What is this repo or project? (You can reuse the repo description you used earlier because this section doesn't have to be long.)
  - How does it work?
  - Who will use this repo or project?
  - What is the goal of this project?
- Instructions for how to develop, use, and test the code.
- Instructions for how people can help.
- List the licensing information for your project.
- List the contact information for your team as well as where to ask questions.

see https://github.com/18F/open-source-guide/blob/18f-pages/pages/making-readmes-readable.md

To read for MCQ test

- The Diátaxis systematic approach to creating better documentation:
  - Tutorials
  - How-to guides
  - Reference
  - Explanation

Technological foundations of software development

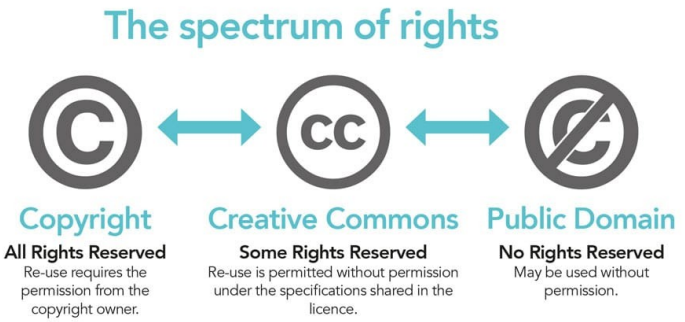
Document, license, publish, maintain your software

Part 2: Software licenses

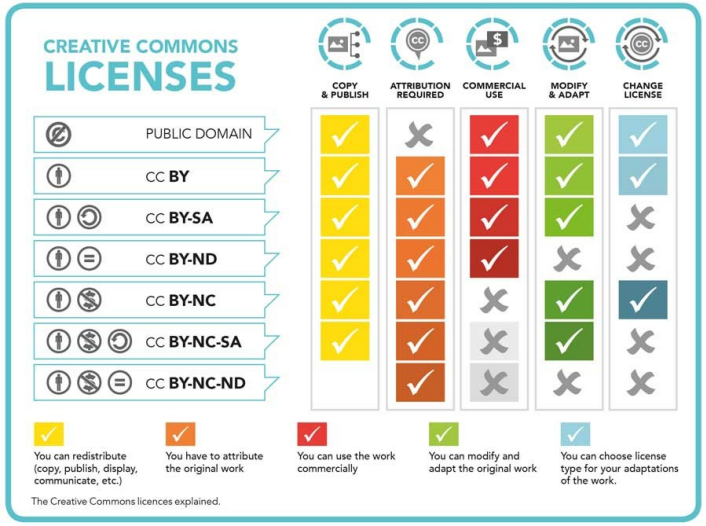
ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

Creative Commons

- Set of licenses for published works



https://researchoutreach.org/articles/thought-leaders/license-to-share-how-the-creative-commons-licensing-system-encourages-the-remixing-and-reuse-of-published-materials/



https://researchoutreach.org/articles/thought-leaders/license-to-share-how-the-creative-commons-licensing-system-encourages-the-remixing-and-reuse-of-published-materials/

Software license

Contract by which the owner of the copyright on a computer program defines with his co-contractor (operator or user) the conditions under which this program can be used, distributed or modified.

Software license

Software licenses and rights granted in context of the copyright according to Mark Webbink.[2] Expanded by freeware and sublicensing.

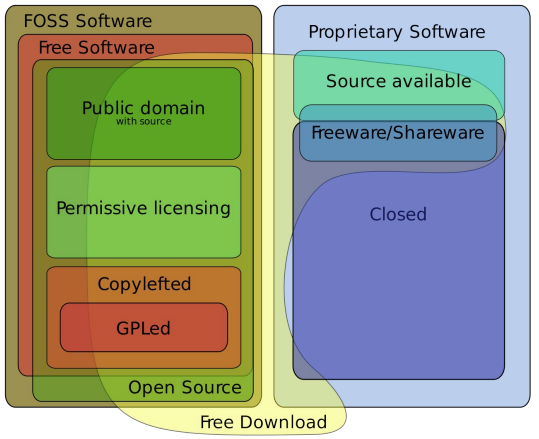
Rights granted	Public domain	Permissive FOSS license (e.g. BSD license)	Copyleft FOSS license (e.g. GPL)	Freeware/Shareware/ Freemium	Proprietary license	Trade secret
Copyright retained	No	Yes	Yes	Yes	Yes	Very strict
Right to perform	Yes	Yes	Yes	Yes	Yes	No
Right to display	Yes	Yes	Yes	Yes	Yes	No
Right to copy	Yes	Yes	Yes	Often	No	Lawsuits are filed by the owner against copyright infringement the most
Right to modify	Yes	Yes	Yes	No	No	No
Right to distribute	Yes	Yes, under same license	Yes, under same license	Often	No	No
Right to sublicense	Yes	Yes	No	No	No	No
Example software	SQLite, ImageJ	Apache web server, ToyBox	Linux kernel, GIMP, OBS	Irfanview, Winamp, *League of Legends*	Windows, the majority of commercial video games and their DRMs, Spotify, xSplit, TIDAL	Server-side Cloud computing programs and services, forensic applications, and other line-of-business work.

Free software foundation

Free Software Foundation

FREE SOFTWARE FOUNDATION

Abbreviation	FSF
Formation	October 4, 1985; 36 years ago[1]
Founder	Richard Stallman (GNU project)
Type	501(c)(3) non-profit organization
Legal status	501(c)(3)
Purpose	Educational
Headquarters	Boston, Massachusetts, US
Region served	Worldwide
Membership	Individuals
President	Geoffrey Knauth
Budget	$1,373,645 in FY 2017[2]
Staff	14[3]
Website	www.fsf.org



Open Source licenses

Open source licenses are licenses that comply with the Open Source Definition https://opensource.org/osd

— in brief, they allow software to be freely used, modified, and shared.

Apache License 2.0
BSD 3-Clause "New" or "Revised" license
BSD 2-Clause "Simplified" or "FreeBSD" license
GNU General Public License (GPL)
GNU Library or "Lesser" General Public License (LGPL)
MIT license

Mozilla Public License 2.0
Common Development and Distribution License
Eclipse Public License version 2.0
...

FOSS Licenses (free and open-source software)

	Apache	BSD	MIT	GPL	LGPL	AGPL
Type	Permissive	Permissive	Permissive	Copyleft	Copyleft	Copyleft
Provides copyright protection	✓TRUE	✓TRUE	✓TRUE	✓TRUE	✓TRUE	✓TRUE
Can be used in commercial applications	✓TRUE	✓TRUE	✓TRUE	✓TRUE	✓TRUE	✓TRUE
Provides an explicit patent license	✓TRUE	✗FALSE	✗FALSE	✗FALSE	✗FALSE	✗FALSE
Can be used in proprietary (closed source) projects	✓TRUE	✓TRUE	✓TRUE	✗FALSE	✗FALSE partially	✗FALSE for web
Popular open-source and free projects	Kubernetes Swift Firebase	Django React Flutter	Angular.js JQuery, .NET Core Laravel	Joomla Notepad++ MySQL	Qt SharpDevelop	SugarCRM Launchpad

https://mogod-software.medium.com/understanding-open-source-and-free-software-licensing-c0fa600106c9

Choose an open source license

An open source license protects contributors and users. Businesses and savvy developers won't touch a project without this protection.

Which of the following best describes your situation?

I need to work in a community.

Use the **license preferred by the community** you're contributing to or depending on. Your project will fit right in.

If you have a dependency that doesn't have a license, ask its maintainers to **add a license**.

I want it simple and permissive.

The **MIT License** is short and to the point. It lets people do almost anything they want with your project, like making and distributing closed source versions.

Babel, **.NET Core**, and **Rails** use the MIT License.

I care about sharing improvements.

The **GNU GPLv3** also lets people do almost anything they want with your project, *except* distributing closed source versions.

Ansible, **Bash**, and **GIMP** use the GNU GPLv3.

https://choosealicense.com/

Display the license in your Git repository

Determining the location of your license

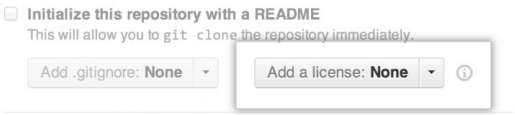
Most people place their license text in a file named `LICENSE.txt` (or `LICENSE.md` or `LICENSE.rst`) in the root of the repository; here's an example from Hubot.

Some projects include information about their license in their README. For example, a project's README may include a note saying "This project is licensed under the terms of the MIT license."

As a best practice, we encourage you to include the license file with your project.

Applying a license to a repository with an existing license

The license picker is only available when you create a new project on GitHub. You can manually add a license using the browser. For more information on adding a license to a repository, see "Adding a license to a repository."



https://docs.github.com/en/repositories/managing-your-repositorys-settings-and-features/customizing-your-repository/licensing-a-repository

Technological foundations of software development

Document, license, publish, maintain your software

Part 3: Publish your software

ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/

Publish your software

Software publishing is a term that describes the overall process of designing and distributing a software package or collection of software packages.

https://www.computerhope.com/jargon/s/softpubl.htm

Package repositories

The Central Repository

Java

- Guide to start publishing your code: https://central.sonatype.org/publish/publish-guide/
  - sign up
  - generate artifacts: binaries, source, javadoc
  - sign your artifacts (GPG)
  - publish : `> mvn clean deploy nexus-staging:release`

```
<distributionManagement>
  <snapshotRepository>
    <id>ossrh</id>
    <url>https://s01.oss.sonatype.org/content/repositories/snapshots</url>
  </snapshotRepository>
</distributionManagement>
<build>
  <plugins>
    <plugin>
      <groupId>org.sonatype.plugins</groupId>
      <artifactId>nexus-staging-maven-plugin</artifactId>
      <version>1.6.7</version>
      <extensions>true</extensions>
      <configuration>
        <serverId>ossrh</serverId>
        <nexusUrl>https://s01.oss.sonatype.org/</nexusUrl>
        <autoReleaseAfterClose>true</autoReleaseAfterClose>
      </configuration>
    </plugin>
    ...
  </plugins>
</build>
```

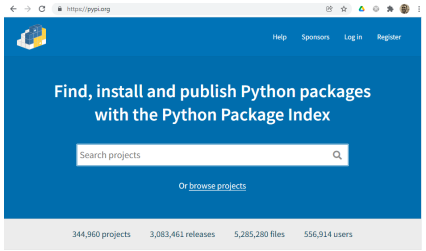
example with Maven: in the `pom.xml`

```
<settings>
  <servers>
    <server>
      <id>ossrh</id>
      <username>your-jira-id</username>
      <password>your-jira-pwd</password>
    </server>
  </servers>
</settings>
```

example with Maven: in the `~/.m2/settings.xml`

Package repositories

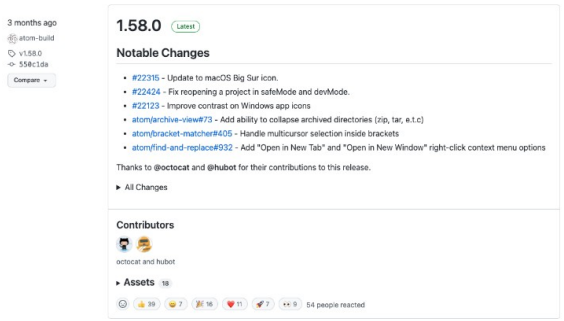
python Package Index



- Guide to start publishing your code: https://packaging.python.org/en/latest/tutorials/packaging-projects/

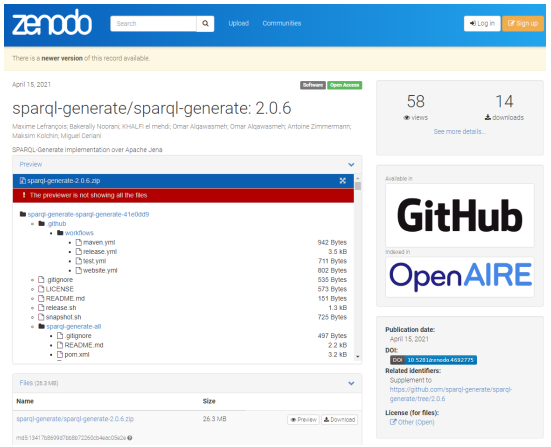
Publish on github/gitlab

- https://docs.github.com/en/repositories/releasing-projects-on-github/managing-releases-in-a-repository



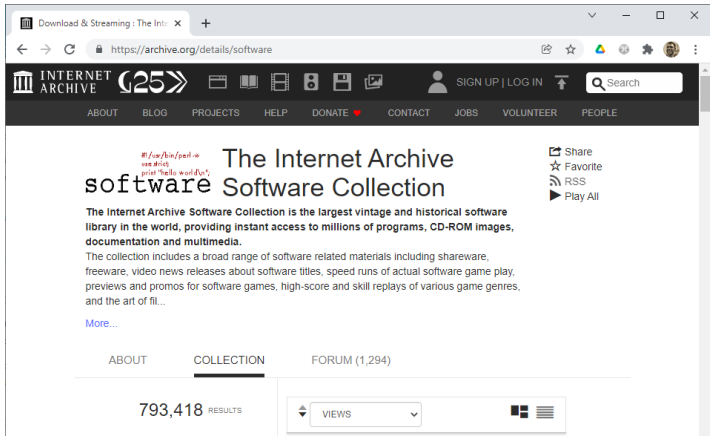
Researchers: get a DOI with Zenodo

- Digital Object Identifiers (DOIs)
- For academics, you can now publish your software (or data) and obtain a permanent identifier so that it can be cited in scientific publications



https://docs.github.com/en/repositories/archiving-a-github-repository/referencing-and-citing-content

Software archives

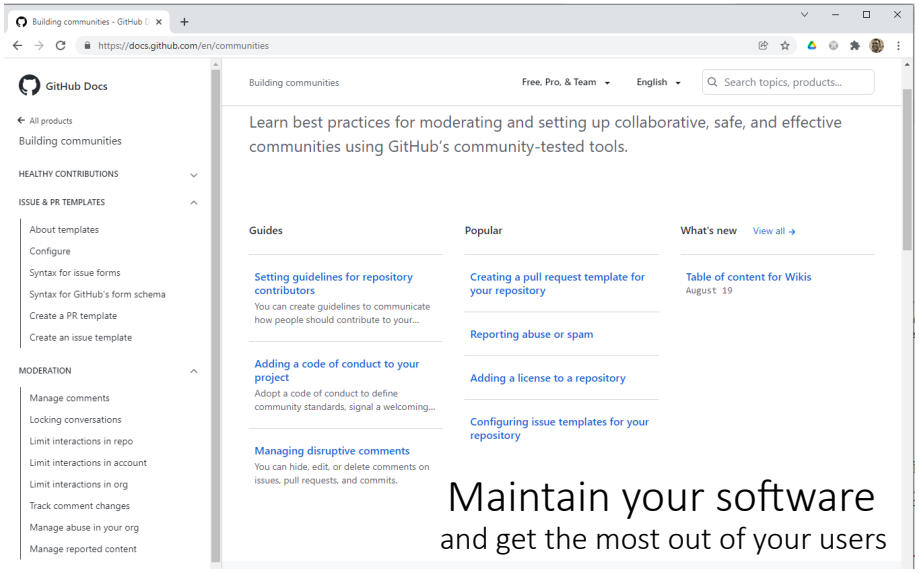


Technological foundations of software development

Document, license, publish, maintain your software

Part 4: Maintaining your software

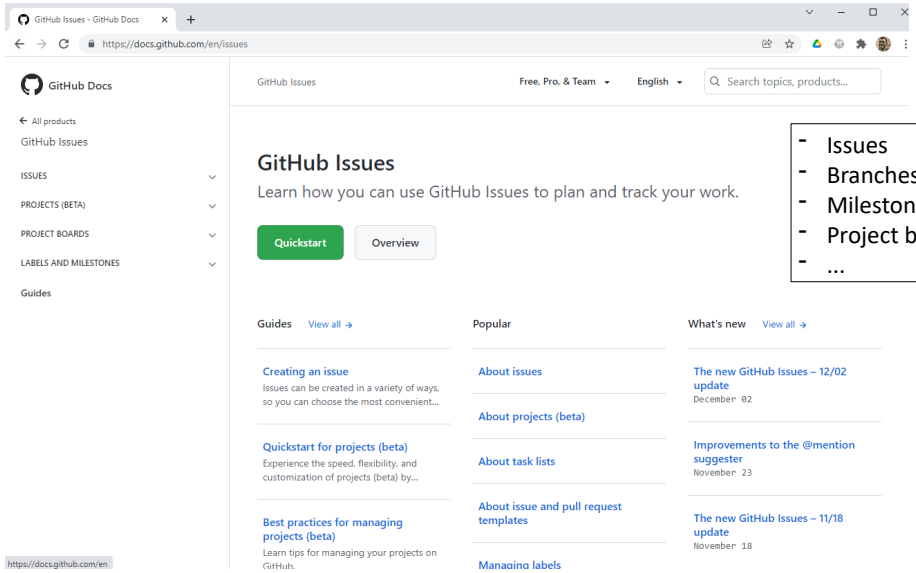
ICM – Computer Science Major – Course unit on Technological foundations of computer science
M1 Cyber Physical and Social Systems – Course unit on CPS2 engineering and development, Part 2: Technological foundations of software development
Maxime Lefrançois https://maxime-lefrancois.info
online: https://ci.mines-stetienne.fr/cps2/course/tfsd/



Maintain your software

and get the most out of your users

53



- Issues
- Branches
- Milestones
- Project boards
- ...

54

... Your turn

Technological foundations of software development

Document, license, publish, maintain your software

Complete the TODO section:

https://ci.mines-stetienne.fr/cps2/course/tfsd/course-6.html#_todos