Request for contributions

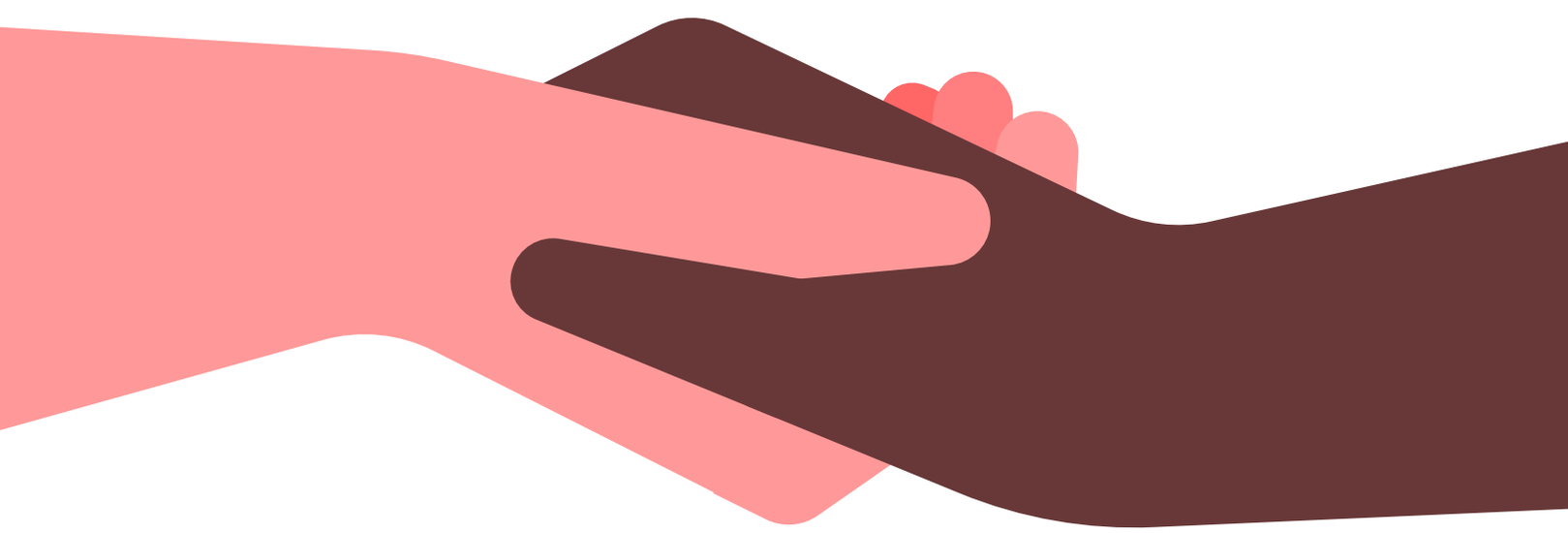
Standard for Public Code

What public code is and how to implement it for:

P Policy makers

M Management

D Designers and developers



Version 0.2.2

github.com/publiccodenet/standard


DPG

Authors

The Foundation for Public Code

https://publiccode.net

- Arnout Schuijff
- Ben Cerveny
- Elena Findley-de Regt
- Claus Mullie
- Boris van Hoytema
- Mirjam van Tiel
- Eric Herman
- Jan Ainali
- Felix Faassen
- Alba Roza

Amsterdam University of Applied Sciences (AUAS), Faculty of Digital Media and Creative Industries, Lectorate of Play & Civic Media

- Martijn de Waal

Gemeente Amsterdam

- Tamas Erkelens
- Mark van der Net
- Maurits van der Schee

Code For NL

- Johan Groenen
- Edo Plantinga

Nordic Institute for Interoperability Solutions (NIIS)

https://niis.org

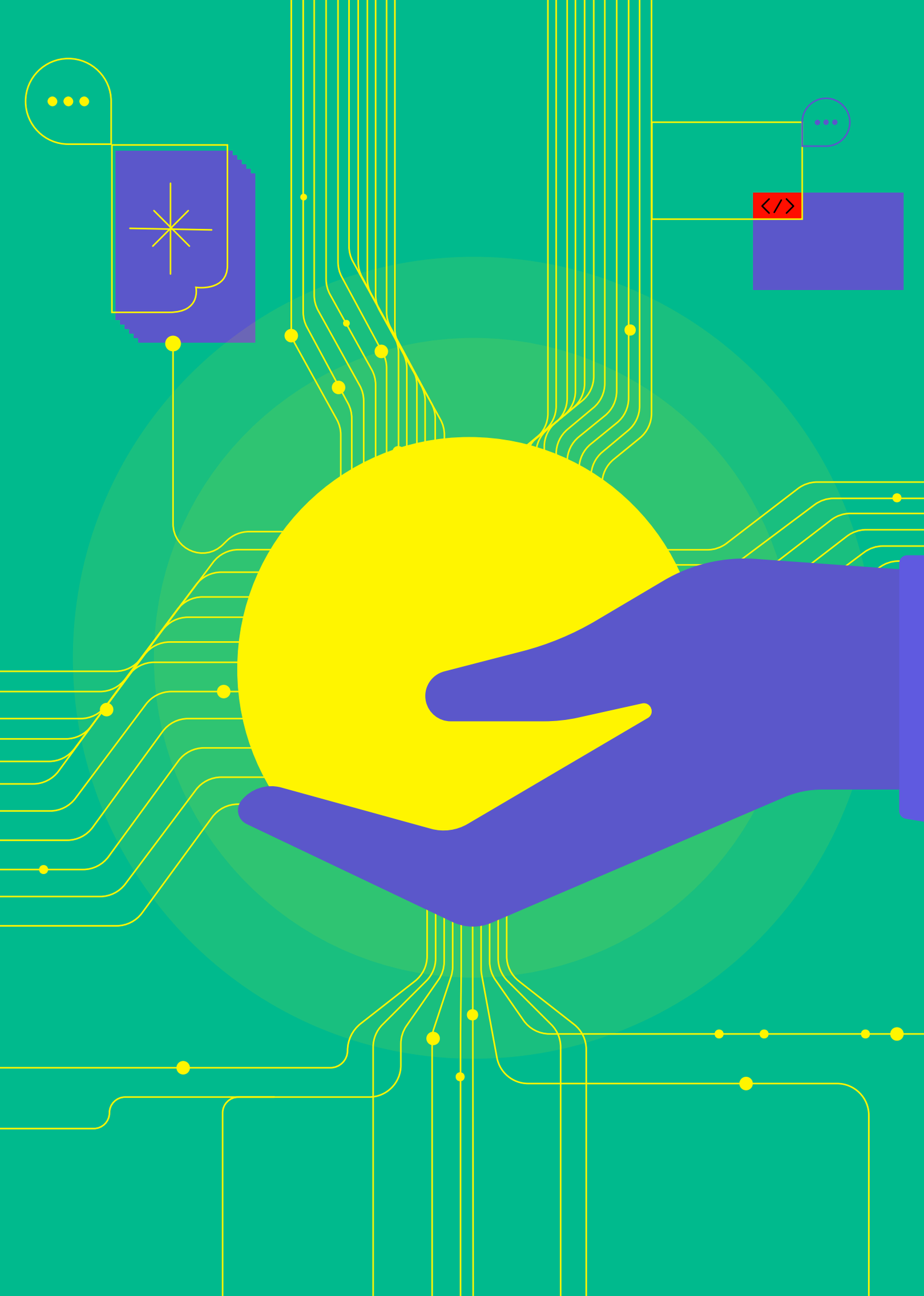
- Petteri Kivimäki

Individual contributors

- Floris Deerenberg
- Timo Slinger
- Bert Spaan
- David Barberi

- Paul Keller
- Sky Bristol
- Marcus Klaas de Vries
- Arnout Engelen
- Ngô Ngọc Đức Huy
- Mauko Quiroga
- Charlotte Heikendorf

Table of Contents



Introduction

The Standard for Public Code is a set of criteria that supports public organizations in developing and maintaining software and policy together.

Anyone developing software or policy for a public purpose can use this standard to work towards higher quality public services that are more cost effective, with less risk and more control.

This introduction introduces the term public code, explains why this is important, and introduces the process through which software and policy code can become certified public code.

Definition of public code

Public code is both computer source code (such as software and algorithms) and public policy executed in a public context, by humans or machines. Public code is explicitly distinct from regular software because it operates under fundamentally different circumstances and expectations.

Why public code?

There are many reasons for why public code is relevant now.

Software code == legal code

Software is public infrastructure.

In the 21st century, software can be considered vital public infrastructure. It is increasingly not just the expression of existing policy but the originator of new policy – for example where algorithms decide which districts need extra social services or policing.

Software mechanics, algorithms and data collection have become key elements in the execution of public policies. Computer code now executes policies that have been

codified in legal code through democratic procedures. Both forms of code set conditions for society to function according to democratically set public values, the latter executed by humans, the former by machines. In other words, software code has increasingly started to equal legal code.

Software should therefore be subject to the principles of democratic governance.

Traditional public software procurement

Current public software production methods have not served public service delivery very well.

In the last decade, public organizations that purchased complete software solutions have sometimes been surprised to discover that they:

- can't change their software to reflect changing policy or take advantage of new technology
- don't have access to their data as it's locked into proprietary systems
- are asked to pay ever increasing license fees

Technological sovereignty and democratic accountability

Public institutions, civil servants and residents deserve better.

We believe the software that runs our society can no longer be a black box, controlled by outside companies that keep the underlying logic on which their software operates hidden in proprietary codebases. Instead, governments and the people they serve need technological sovereignty – allowing them to set and control the functioning of public software, just like they are able to set and control policy that is legally formulated in laws. Citizens and civil society actors need this software to be transparent and accountable.

The design of software as essential civic infrastructure should honor digital citizens' rights.

Designing truly public software

Public code is at the core of modern public institutions, shapes the work of civil servants and affects the lives of almost all residents.

Public software must therefore be:

- transparent
- accountable
- understandable for its constituents

It must reflect the values of the society it serves, for example by being inclusive and non-discriminatory.

Most proprietary software systems currently used by public organizations do not meet these requirements. Public code – software built to operate with and as public infrastructure, along with the arrangements for its production – does.

Values of public code

We consider public code to have these core values:

- Inclusive
- Usable
- Open
- Legible
- Accountable
- Accessible
- Sustainable

How public code works

Public code is open source software meant for fulfilling the essential role of public organizations. Through use, other administrations contribute back to the software, so that its development and maintenance become truly collaborative.

Being open unlocks many other things.

Local responsibility and democratic accountability are ensured when a public organization implements and maintains their own public code. By being open and with a broader contributor base, the software is more secure – it

benefits from many eyes spotting potential flaws. Many contributors share the maintenance work to keep it functional and modern, which reduces future technical debt. The shared workload is more sustainable now and in the future. Its openness makes the code and its data more easily adaptable in the future – it will be easier to retool, repurpose or retire. This all results in lower risk public infrastructure.

This pooling of resources lets public administrations give extra attention to how to customize the software so it works best in each local context - creating better user experiences for their end users (residents or citizens).

Economics of public code

Public code offers a better economic model for public organizations as well as for commercial companies. It's an alternative to traditional software procurement which increases local control and economic opportunity.

Designed from the start to be open, adaptable and with data portability, it can be developed by in-house staff or trusted vendors. Because the code is open, the public administration can change vendors if they need to. Open code increases opportunities for public learning and scrutiny, allowing the public administration to procure smaller contracts – thereby making it easier for local small and medium enterprises to bid. Public administrations can use their own software purchasing to stimulate innovation and competition in their local economy.

This can be seen as investment leading to future economic growth – more vendors will be necessary due to growing technology demand.

Procuring public code

Public code can be used and developed by permanent in-house development teams, contractors or outsourced suppliers. Vendors to public organizations can include public code in their bids for contracts.

To use existing public code, you need to specify in your budget and project design that your new solution will use that codebase. To encourage an innovative approach to

adapting the public code to your context, you could describe the service or outcome in your contract.

Standard compliance or certification process

The Foundation for Public Code ensures that codebases under its stewardship (and not in incubation or the attic) are compliant with the Standard for Public Code. This makes clear to potential users and contributors that the codebase is of high quality, and updates will be too.

The audit performed by the Foundation for Public Code is meant to complement machine testing, as machines are great at testing things like syntax and whether outcomes align with expectations. Things meant for humans, such as testing whether documentation is actually understandable and accessible in context, the commit messages make sense, and whether community guidelines are being followed are impossible for machines to test.

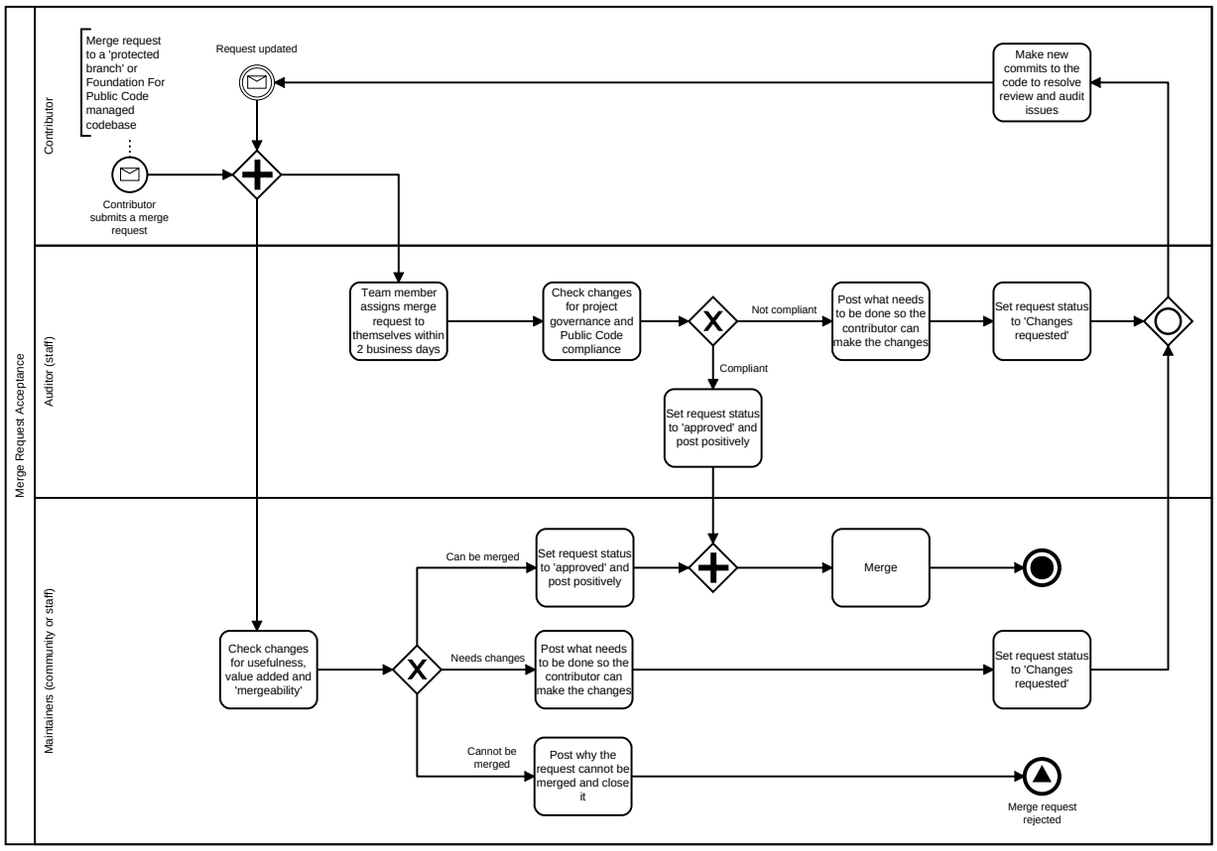
The audit tests the entire codebase, including source code, policy, documentation and conversation for compliance with both the standards set out by the Foundation for Public Code and the standards set out in the codebase itself.

How the process works

Every time a contribution is suggested to a codebase – through for instance a merge request – the codebase stewards of the Foundation for Public Code will audit the contribution for compliance with the Standard for Public Code. New contributions can only be adopted into the codebase after they have been approved as compliant with the Standard for Public Code, and have been reviewed by another contributor.

https://about.publiccode.net/roles/

The audit is presented as a review of the contribution. The codebase steward gives line by line feedback and compliance, helping the contributor to improve their contribution. The merge request cannot be fulfilled until the codebase stewards have approved the contribution.



Certifying an entire codebase versus a contribution

For the codebase to be completely certified every meaningful line of code, and the commits behind the code, need to meet the Standard.

If codebases have been completely audited from the first merge request they can be immediately certified as compliant with the Standard for Public Code.

If the audit process is added to an existing codebase, the new merge requests can be certified, but the existing code cannot be certified. By auditing every new merge request the codebase can move incrementally towards being completely certified.

The goals for the Standard for Public Code

This Standard supports developers, designers, business management and policy makers to:

- develop high quality software and policy for better public service delivery
- develop codebases that can be reused across contexts and collaboratively maintained

- reduce technical debt and project failure rate
- have more granular control over, and ability to make decisions about, their IT systems
- improve vendor relationships with a better economic model

The Foundation for Public Code helps public organizations share and adopt open source software, build sustainable developer communities and create a thriving ecosystem for public code. It does this through codebase stewardship. For this process the codebase stewards use the Standard for Public Code to make sure the code it stewards is high quality as well as collaboratively maintainable.

https://publiccode.net/

Potential users of codebases tested against the Standard for Public Code can expect them to be highly reusable, easily maintainable and of high quality.

The Standard for Public Code does this by:

- setting out a common terminology for public code development
- establishing measures to help develop high quality public code
- providing guidance on how to fulfill its criteria and operationalize compliance

The Standard for Public Code is meant to be time and technology independent.

Who this is for

The Standard for Public Code is for the people who create and reuse public code:

- policy makers
- business and project management
- developers and designers

These people work at:

- public organizations: institutions and administrations
- consultancies and vendors of information technology and policy services to public organizations

It is not aimed at public organizations' end users (residents or citizens), journalists or academics.

Further reading

- "Modernising Public Infrastructure with Free Software" whitepaper by the Free Software Foundation Europe. https://download.fsfe.org/campaigns/pmpc/PMPC-Modernising-with-Free-Software.pdf

Videos on public code

- Collaborative Code is the Future of Cities @ DecidimFest 2019. Talk by Ben Cerveny on the background behind the Foundation for Public Code. https://www.youtube.com/watch?v=cnJtnZ9Cx1o
- Public Money? Public Code! - Panel @ Nextcloud Conference 2019. Touches on topics like procurement, law and more. https://youtube.com/watch?v=QHFkD4xfd6c

Get involved

This standard is a living document. Read our contributor guide to learn how you can make it better.

Readers guide

The Standard describes a number of criteria. All criteria have consistent sections that make it clear how to create great public code. Below is a brief explanation of each of these sections and how they are used within the criteria of the Standard.

Requirements

This section lists what needs to be done in order to comply with the standard.

In order to limit ambiguity, the key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in IETF RFC 2119.

https://tools.ietf.org/html/rfc2119

Why this is important

This section explains why it is important for the users and contributors of this codebase that these requirements are followed.

What this does not do

This section manages expectation by explaining what following the requirements will not save you from.

This helps:

- with applying the Standard correctly
- make sure no unexpected things pop up

How to test

This section offers actions you can take to see if a contribution is compliant with the Standard. This is key if you want to operationalize the Standard.

We've tried to word it so that someone who is not intimately acquainted with the subject matter can still do a basic check for compliance.

P Policy makers: what you need to do

This section tries to specifically speak to policy makers by offering them concrete actions they can perform in their role.

Policy makers set the priorities and goals of projects and may be less technologically experienced.

M Management: what you need to do

This section tries to specifically speak to management by offering concrete actions they can perform in their role.

Management is responsible for on-time project delivery, stakeholder management and continued delivery of the service. For this they are wholly reliant on both the policy makers as well as the developers and designers. They need to create the right culture, line up the right resources and provide the right structures to deliver great services.

D Developers and designers: what you need to do

This section tries to specifically speak to developers and designers by offering them concrete actions they can perform in their role.

Developers are usually more technically aligned and have more impact on the delivery of services than the previous groups.

Glossary

Code

Any explicitly described system of rules. This includes laws, policy and ordinances, as well as source code that is used to build software. Both of these are rules, some executed by humans and others by machines.

Codebase

Any discrete package of code (both source and policy), the tests and the documentation required to implement a piece of policy or software.

This can be – for example – a document or a version-control repository.

Continuous integration

In software engineering, continuous integration (CI) is the practice of merging all developers' working copies to a development branch of a codebase as frequently as reasonable.

Different contexts

Two contexts are different if they are different public organizations or different departments for which there is not one decision maker that could make collaboration happen naturally.

General public

The public at large: end users of the code and the services based upon it.

For example, a city's residents are considered end users of a city's services and of all code that powers these services.

Open source

Open source is defined by the Open Source Initiative in their Open Source Definition.

https://opensource.org/osd-annotated

Open standard

An open standard is any standard that meets the Open Source Initiative's Open Standard Requirements.

https://opensource.org/osr

Policy

A policy is a deliberate system of principles to guide decisions and achieve rational outcomes. A policy is a statement of intent, and is implemented as a procedure or protocol. Policies are generally adopted by a governance body within an organization. Policies can assist in both subjective and objective decision making.

Public policy is the process by which governments translate their political vision into programs and actions to deliver outcomes.

At the national level, policy and legislation (the law) are usually separate. The distinction is often more blurred in local government.

In the Standard the word 'policy' refers to policy created and adopted by public organizations such as governments and municipalities.

Public code

Public code is both computer source code (such as software and algorithms) and public policy executed in a public context, by humans or machines.

Because public code serves the public interest, it should be open, legible, accountable, accessible and sustainable.

By developing public code independently from but still implementable in the local context for which it was developed, as well as documenting the development

process openly, public code can provide a building block for others to:

- re-implement in their local context
- take as a starting point to continue development
- use as a basis for learning

To facilitate re-use, public code should be either released into the public domain or licensed with an open license that permits others to view and reuse the work freely and to produce derivative works.

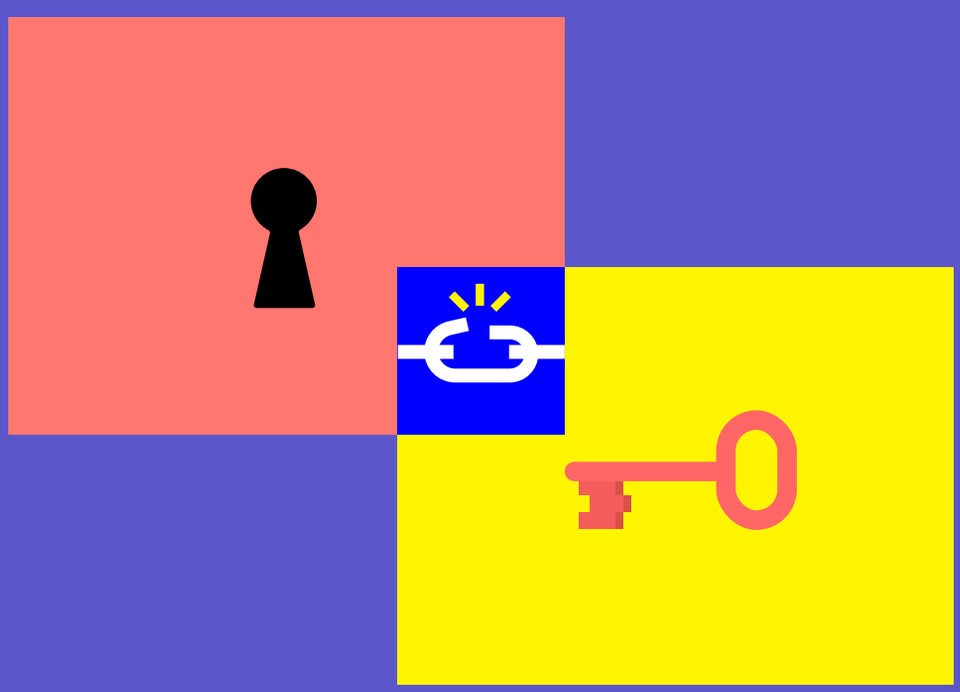
Repository

In revision (or version) control systems, a repository is a data structure which stores metadata for a set of files or directory structure. (source: SVNBook)

Version control

Version control is the management of changes to source code and the files associated with it. Changes are usually identified by a code, termed the *revision number* (or similar). Each revision is associated with the time it was made and the person making the change thus making it easier to retrace the evolution of the code. Revision control systems can be used to compare different versions with each other and to see how content has changed over time.

Criteria



Code in the open

Requirements

- All source code for any policy and software in use (unless used for fraud detection) MUST be published and publicly accessible.
- Contributors MUST NOT upload sensitive information regarding users, their organization or third parties to the repository.
- Any source code not currently in use (such as new versions, proposals or older versions) SHOULD be published.
- The source code MAY provide the general public with insight into which source code or policy underpins any specific interaction they have with an organization.

Why this is important

Coding in the open:

- improves transparency
- increases code quality
- facilitates the auditing processes

What this does not do

- Make source code or policy reusable.
- Make the codebase and the code within it understandable to as many people as possible.

How to test

The source for any version currently in use is published on the internet where it can be seen:

- from outside the original contributing organization
- without the need for any form of authentication or authorization

For each commit, reviewers verify that content does not include sensitive information such as configurations, usernames or passwords, public keys or other real credentials used in production systems.

P Policy makers: what you need to do

- Develop policies in the open.
- Prioritize open and transparent policies.

M Management: what you need to do

- Develop a culture that embraces openness, learning and feedback.
- Collaborate with external vendors and freelancers by working in the open.

D Developers and designers: what you need to do

- Clearly split data and code, in order to meet the requirement about sensitive information above.

Further reading

- Coding in the open by the UK Government Digital Service. https://gds.blog.gov.uk/2012/10/12/coding-in-the-open/
- When code should be open or closed by the UK Government Digital Service. https://www.gov.uk/government/publications/open-source-guidance/when-code-should-be-open-or-closed
- Security considerations when coding in the open by the UK Government Digital Service. https://www.gov.uk/government/publications/open-source-guidance/security-considerations-when-coding-in-the-open
- Deploying software regularly by the UK Government Digital Service. https://www.gov.uk/service-manual/technology/deploying-software-regularly
- How GDS uses GitHub by the UK Government Digital Service. https://gdstechnology.blog.gov.uk/2014/01/27/how-we-use-github/

Bundle policy and source code

Requirements

- A codebase MUST include the policy that the source code is based on.
- A codebase MUST include all source code that the policy is based on.
- All policy and source code that the codebase is based on MUST be documented, reusable and portable.
- Policy SHOULD be provided in machine readable and unambiguous formats.
- Continuous integration tests SHOULD validate that the source code and the policy are executed coherently.

Why this is important

This makes sure access is guaranteed to both the source code and the policy documents to facilitate effective reuse of a codebase.

What this does not do

- Guarantee that a codebase will reflect the bundled policy.
- Make sure packages comply with the local technical infrastructure or legal framework of a given public organization.

How to test

- Policy is provided in machine readable and unambiguous formats.
- Continuous integration tests validate that the source code and policy are executed coherently.

P Policy makers: what you need to do

- Collaborate with developers and designers to ensure there is no mismatch between policy code and source code.
- Provide the relevant policy texts for inclusion in the repository.
- Provide references and links to texts which support the policies.
- Document policy in formats that are unambiguous and machine readable such as Business Process Model and Notation, Decision Model and Notation and Case Management Model Notation.

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

https://www.omg.org/dmn/

https://www.omg.org/cmmn/

- Track policy with the same version control and documentation used to track source code.
- Check in regularly to understand how the non-policy code in the codebase has changed and whether it still matches the intentions of the policy.

M Management: what you need to do

- Keep policy makers, developers and designers involved and connected throughout the whole development process.
- Make sure policy makers, developers and designers are working to the same objectives.

D Developers and designers: what you need to do

- Become familiar with and be able to use the process modelling notation that the policy makers in your organization use.
- Work together with policy makers to ensure there is no mismatch between policy code and source code.
- Give feedback on how to make policy documentation more clear.

Further reading

- Free online tools for building BPMN, CMMN and DMN diagrams at bmpn.io by Camunda.

  https://bpmn.io/

- BPMN Quick Guide by Trisotech.

  https://www.bpmnquickguide.com/view-bpmn-quick-guide/

Create reusable and portable code

Requirements

- The codebase MUST be developed to be reusable in different contexts.
- The codebase MUST be independent from any secret, undisclosed, proprietary or non-open licensed code or services for execution and understanding.
- The codebase SHOULD be in use by multiple parties.
- The roadmap SHOULD be influenced by the needs of multiple parties.
- Configuration SHOULD be used to make code adapt to context specific needs.
- The codebase SHOULD include a machine readable metadata description, for example in a publiccode.yml file.

https://github.com/publiccodeyml/publiccode.yml

- Code and its documentation SHOULD NOT contain situation-specific information.

Why this is important

- Enables other policy makers, developers and designers to reuse what you've developed, to test it, to improve it and contribute those improvements back leading to better quality, cheaper maintainability and higher reliability.
- Makes the code easier for new people to understand (as it's more general).
- Makes it easier to control the mission, vision and scope of the codebase because the codebase is thoughtfully and purposefully designed for reusability.
- Codebases used by multiple parties are more likely to benefit from a self-sustaining community.
- A metadata description file increases discoverability.

- Any contributor is able to test and contribute without relying on the situation-specific infrastructure of any other contributor or deployment.

What this does not do

- Get others to reuse the codebase.
- Build a community.
- Shift responsibility for documentation, support, bug-fixing, etc. to another party.

How to test

- Ask someone in a similar role at another organization if they could reuse the codebase and what that would entail.
- Codebase is in use by multiple parties or in multiple contexts.
- For each commit, reviewers verify that content does not include situation-specific data such as hostnames, personal and organizational data, or tokens and passwords.

P Policy makers: what you need to do

- Document your policy with enough clarity and detail that it can be understood outside of its original context.
- Make sure your organization is listed as a known user by the codebase.

M Management: what you need to do

- Make sure that stakeholders and business owners understand that reusability is an explicit goal of the codebase as it reduces technical debt and provides sustainability for it.

D Developers and designers: what you need to do

- Source should be designed for reuse by other users and organizations.

- Source should be designed to solve a general problem instead of a specific one.
- Someone in a similar organization facing a similar problem should be able to use your solution.

Further reading

- Making source code open and reusable by the UK Government Digital Service.

https://www.gov.uk/service-manual/technology/making-source-code-open-and-reusable

Welcome contributors

Requirements

- The codebase MUST allow anyone to submit suggestions for changes to the codebase.
- The codebase MUST include contribution guidelines explaining what kinds of contributions are welcome and how contributors can get involved, for example in a `CONTRIBUTING` file.
- The codebase SHOULD advertise the committed engagement of involved organizations in the development and maintenance.
- The codebase SHOULD document the governance of the codebase, contributions and its community, for example in a `GOVERNANCE` file.
- The codebase SHOULD have a publicly available roadmap.
- The codebase MAY include a code of conduct for contributors.

Why this is important

- Helps newcomers understand and trust your codebase community's leadership.
- Prevents the community that works on a codebase splitting because there is no way to influence its goals and progress – resulting in diverging communities.
- Helps users decide to use one codebase over another.

What this does not do

- Guarantee others will join the community.
- Guarantee others will reuse the codebase.

How to test

- It's possible to submit suggestions for changes to the codebase.

- There are contribution guidelines.
- Codebase governance is clearly explained, including how to influence codebase governance.

P Policy makers: what you need to do

- Add a list to the codebase of any other resources that policy experts, non-governmental organizations and academics would find useful for understanding or reusing your policy.
- Consider adding contact details so that other policy makers considering reuse can ask you for advice.

M Management: what you need to do

- Make sure the documentation explains how your organization is involved in the codebase, what resources it has available for it and for how long.
- Support your experienced policy makers, developers and designers to stay part of the community for as long as possible.

D Developers and designers: what you need to do

- Respond promptly to requests.
- Keep your management informed of the time and resources you require to support other contributors.

Further reading

- Building welcoming communities by Open Source Guides. https://opensource.guide/building-community/
- The Contributor Covenant is a model code of conduct. https://www.contributor-covenant.org/version/1/4/code-of-conduct
- Leadership and governance for growing open source community projects, by Open Source Guides. https://opensource.guide/leadership-and-governance/
- Building online communities by Pieter Hintjens (long read!). http://hintjens.com/blog:117

Make contributing easy

Requirements

- The codebase MUST have a public issue tracker that accepts suggestions from anyone.
- The codebase MUST include instructions for how to privately report security issues for responsible disclosure.
- The documentation MUST link to both the public issue tracker and submitted codebase changes, for example in a `README` file.
- The codebase MUST have communication channels for users and developers, for example email lists.
- The documentation SHOULD include instructions for how to report potentially security sensitive issues on a closed channel.

Why this is important

- Enables users to fix problems and add features to the shared codebase leading to better, more reliable and feature rich software.
- Allows collaborative uptake of shared digital infrastructure.
- Helps users decide to use one codebase over another.

What this does not do

- Guarantee others will reuse the codebase.

How to test

- There’s a public issue tracker.
- It’s possible to participate in a discussion with other users about the software.

P Policy makers: what you need to do

- Track policy issues in the codebase, so that a relevant external policy expert can volunteer help.

M Management: what you need to do

- Track management issues in the codebase, so that external managers with relevant experience can volunteer help.
- Support your experienced policy makers, developers and designers to keep contributing to the codebase for as long as possible.

D Developers and designers: what you need to do

- Respond promptly to requests.
- Keep your management informed of the time and resources you require to support other contributors.

Further reading

- How to inspire exceptional contributions to your open-source project
  https://www.netdata.cloud/blog/open-source-contributions/
- The benefits of coding in the open by the UK Government Digital Service.
  https://gds.blog.gov.uk/2017/09/04/the-benefits-of-coding-in-the-open/
- Verdaccio's security policy is a really nice example.
  https://github.com/verdaccio/verdaccio/blob/master/SECURITY.md

Maintain version control

Requirements

- The community MUST have a way to maintain version control for the code.
- All files in a codebase MUST be version controlled.
- All decisions MUST be documented in commit messages.
- Every commit message MUST link to discussions and issues wherever possible.
- The codebase SHOULD be maintained in a distributed version control system.
- Contributors SHOULD group relevant changes in commits.
- Maintainers SHOULD mark released versions of the codebase, for example using revision tags or textual labels.
- Contributors SHOULD prefer file formats where the changes within the files can be easily viewed and understood in the version control system.
- Contributors MAY sign their commits and provide an email address, so that future contributors are able to contact past contributors with questions about their work.

Why this is important

Version control means keeping track of changes to the code over time. This allows you to create structured documentation of the history of the codebase. This is essential for collaboration at scale.

Distributed version control enables you to:

- have a full copy of the code and its history

- revert to an earlier version of the codebase whenever you want to
- record your changes and the reasons why you made them, to help future developers understand the process
- compare two different versions
- work on changes in parallel as a team before merging them together
- continue to work when the network is unavailable, merging changes back with everyone else's at a later date

What this does not do

- Substitute for advertising maturity.
- Guarantee the code executes correctly.
- Guarantee collaborators.

How to test

- The codebase is kept in version control using software such as Git.
- All commit messages explain:
  - why the change was made,
  - what the discussion about the change was or where to find it (with a URL).
- It is possible to access a specific version of the codebase, for example through a revision tag or a textual label.

P Policy makers: what you need to do

- If a new version of the codebase is created because of a policy change, make sure it's clear in the documentation:
  - what the policy change is,
  - how it's changed the codebase.

For example, adding a new category of applicant to a codebase that manages granting permits would be considered a policy change.

M Management: what you need to do

- Support policy makers, developers and designers to be clear about what improvements they're making to the codebase - making improvements isn't a public relations risk.

D Developers and designers: what you need to do

- Write clear commit messages so that it is easy to understand why the commit was made.
- Mark different versions so that it is easy to access a specific version, for example using revision tags or textual labels.
- Write clear commit messages so that versions can be usefully compared.
- Work with policy makers to describe how the source code was updated after a policy change.

Further reading

- Producing OSS: Version Control Vocabulary by Karl Fogel. https://producingoss.com/en/vc.html#vc-vocabulary
- Maintaining version control in coding by the UK Government Digital Service. https://www.gov.uk/service-manual/technology/maintaining-version-control-in-coding
- GitHub Learning Lab for learning how to use GitHub or refresh your skills. https://lab.github.com/
- Git Cheat Sheet a list with the most common used git commands. https://education.github.com/git-cheat-sheet-education.pdf

Require review of contributions

Requirements

- All contributions that are accepted or committed to release versions of the codebase MUST be reviewed by another contributor.
- Reviews MUST include source, policy, tests and documentation.
- Reviewers MUST provide feedback on all decisions to not accept a contribution.
- Contributions SHOULD conform to the standards, architecture and decisions set out in the codebase in order to pass review.
- Reviews SHOULD include running both the code and the tests of the codebase.
- Contributions SHOULD be reviewed by someone in a different context than the contributor.
- Version control systems SHOULD not accept non-reviewed contributions in release versions.
- Reviews SHOULD happen within two business days.
- Reviews MAY be performed by multiple reviewers.

Why this is important

- Increases codebase quality.
- Reduces security risks as well as operational risks.
- Creates a culture of making every contribution great.
- Catches the most obvious mistakes that could happen.
- Gives contributors the security that their contributions are only accepted if they really add value.
- Assures contributors of a guaranteed time for feedback or collaborative improvement.

What this does not do

- Guarantee the right solution to a problem.
- Mean that reviewers are liable.
- Absolve a contributor from writing documentation and tests.
- Provide you with the right reviewers.

How to test

- Every commit in the history has been reviewed by a different contributor in a different context.

P Policy makers: what you need to do

- Institute a ‘four eyes’ policy where everything, not just code, is reviewed.
- Use a version control system or methodology that enables review and feedback.

M Management: what you need to do

- Make delivering great code a shared objective.
- Make sure writing and reviewing contributions to source, policy, documentation and tests are considered equally valuable.
- Create a culture where all contributions are welcome and everyone is empowered to review them.
- Make sure no contributor is ever alone in contributing to a codebase.

D Developers and designers: what you need to do

- Ask other contributors on the codebase to review your work, in your organization or outside of it.
- Try to respond to others’ requests for code review promptly, initially providing feedback about the concept of the change.

Further reading

- How to review code the GDS way by the UK Government Digital Service.

  https://gds-way.cloudapps.digital/manuals/code-review-guidelines.html#content

- Branch protection on GitHub and GitLab.

  https://help.github.com/en/articles/about-protected-branches

  https://about.gitlab.com/2014/11/26/keeping-your-code-protected/

- The Gentle Art of Patch Review

  https://sage.thesharps.us/2014/09/01/the-gentle-art-of-patch-review/

Document codebase objectives

Requirements

- The codebase MUST contain documentation of its objectives – like a mission and goal statement – that is understandable by designers and developers so that they can use or contribute to the codebase.
- Codebase documentation SHOULD clearly describe the connections between policy objectives and codebase objectives.
- The codebase MAY contain documentation of its objectives for the general public.

Why this is important

Documenting codebase objectives:

- provides an easy way for people to decide whether this codebase is interesting for them now or in the future.
- helps scope your own development.
- clearly communicates to other stakeholders and contributors what the codebase is for.

What this does not do

- Guarantee that the codebase achieves the stated objective(s).
- Guarantee contributions to the codebase.
- Prevent other codebases from attempting to achieve the same objectives.

How to test

There is an entry for the codebase objectives, mission or goal in the codebase documentation.

P Policy makers: what you need to do

- Add the policy objectives to the codebase documentation, for example in the `README`.
- Include relevant policies which impact the community, codebase, and development like value and ethics based policies, for example accessibility or equal opportunity.

M Management: what you need to do

- Add the organizational and business objectives to the codebase documentation, for example in the `README`.

D Developers and designers: what you need to do

- Add the technology and design objectives to the codebase documentation, for example in the `README`.

Further reading

- How to write project objectives by Marek Hajduczenia.

http://grouper.ieee.org/groups/802/3/RTPGE/public/may12/hajduczenia_01_0512.pdf

Document the code

Requirements

- All of the functionality of the codebase – policy as well as source – MUST be described in language clearly understandable for those that understand the purpose of the code.
- The documentation of the codebase MUST contain:
  - a description of how to install and run the source code,
  - examples demonstrating the key functionality.
- The documentation of the codebase SHOULD contain:
  - a high level description that is clearly understandable for a wide audience of stakeholders, like the general public and journalists,
  - a section describing how to install and run a standalone version of the source code, including, if necessary, a test dataset,
  - examples for all functionality.
- There SHOULD be continuous integration tests for the quality of the documentation.
- The documentation of the codebase MAY contain examples that make users want to immediately start using the codebase.
- The code MAY be tested by using examples in the documentation.

Why this is important

- Users can start using and contributing more quickly.
- You help people discover the codebase, especially people asking ‘is there already code that does something like this’.
- This provides transparency into your organization and processes.

What this does not do

- Contribute directly to more reusable, portable code (see Create reusable and portable code).

How to test

- Other stakeholders, professionals from other public organizations and the general public find the documentation clear and understandable.
- Documentation is generated from code.
- Links and images are automatically tested.

P Policy makers: what you need to do

- Check in regularly to understand how the non-policy code in the codebase has changed.
- Give feedback on how to make non-policy documentation more clear.

M Management: what you need to do

- Try to use the codebase.
- Make sure you understand both the policy and source code as well as the documentation.

D Developers and designers: what you need to do

- Check in regularly to understand how the non-source code in the codebase has changed.
- Give feedback on how to make non-source documentation more clear.

Further reading

- Documentation guide by Write the Docs.

https://www.writethedocs.org/guide/

Use plain English

Requirements

- All codebase documentation MUST be in English.
- All code MUST be in English, except where policy is machine interpreted as code.
- Any translation MUST be up to date with the English version and vice versa.
- There SHOULD be no acronyms, abbreviations, puns or legal/non-English/domain specific terms in the codebase without an explanation preceding it or a link to an explanation.
- The name of the codebase SHOULD be descriptive and free from acronyms, abbreviations, puns or organizational branding.
- Documentation SHOULD aim for a lower secondary education reading level, as recommended by the Web Content Accessibility Guidelines 2.

https://www.w3.org/WAI/WCAG21/quickref/?showtechniques=315#readable

- Any code, documentation and tests MAY have a translation.

Why this is important

- Makes the codebase and what it does understandable for a wider variety of stakeholders in multiple contexts.
- Helps with the discoverability of the codebase.
- Can help you meet the European Union accessibility directive, which requires most public sector information to be accessible.

https://ec.europa.eu/digital-single-market/en/web-accessibility

What this does not do

- Make explanations of the codebase's functionality understandable.
- Make your organization's jargon understandable without an explanation.

How to test

- Check that translations and the English version have the same content.
- Validate that no unexplained acronyms, abbreviations, puns or legal/domain specific terms are in the documentation.
- Test the documentation for grammar using Grammarly.

https://www.grammarly.com/

- Test the documentation for readability using Hemingway text editor.

https://hemingwayapp.com/

- Ask someone outside of your context if they understand your content (for example, a developer working on a different codebase).

P Policy makers: what you need to do

- Frequently test with other management, designers and developers in the process if they understand what you are delivering and how you document it.

M Management: what you need to do

- Try to limit the use of acronyms, abbreviations, puns or legal/domain specific terms in internal communications in and between teams and stakeholders.
- Be critical of documentation and descriptions in proposals and changes - if you don't understand something, others will probably also struggle with it.

D Developers and designers: what you need to do

- Frequently test with policy makers and management if they understand what you are delivering and how you document it.

Further reading

- Text of the Web Content Accessibilty Guidelines 2.1, Guideline 3.1 Readable - make text content readable and understandable.

https://www.w3.org/TR/WCAG21/#readable

- Upgoer 5 text editor - only allows 1000 most common words. https://splasho.com/upgoer5/
- Definition of plain language by United States General Services Administration. https://www.plainlanguage.gov/about/definitions/

Use open standards

Requirements

- For features of a codebase that facilitate the exchange of data the codebase MUST use an open standard that meets the Open Source Initiative Open Standard Requirements.

https://opensource.org/osr

- If no existing open standard is available, effort SHOULD be put into developing one.
- Standards that are machine testable SHOULD be preferred over those that are not.
- Functionality using features from a standard that is not an open standard MAY be provided if necessary, but only in addition to compliant features.
- All non-compliant standards used MUST be recorded clearly in the documentation.
- The codebase SHOULD contain a list of all the standards used with links to where they are available.

Why this is important

- Creates interoperability between systems.
- Reduces possible vendor lock-in.
- Guarantees access to the knowledge required to reuse and contribute to the codebase.

What this does not do

- Make it understandable how to use the software.

How to test

- The documentation includes a list of the standards.
- The standards used for all features that offer interoperability with other components and systems are freely and publicly available on the internet.

P Policy makers: what you need to do

- Mandate use of open standards everywhere possible.
- Prohibit procurement of technology that does not use open standards.

M Management: what you need to do

- Consider including open standard compliance assessment in code reviews.

D Developers and designers: what you need to do

- Add continuous integration tests for compliance with the standards.

Further reading

- Open Standards principles, policy paper of the UK Cabinet Office.
  https://www.gov.uk/government/publications/open-standards-principles/open-standards-principles

Use continuous integration

Requirements

- All functionality in the source code MUST have automated tests.
- Contributions MUST pass all automated tests before they are admitted into the codebase.
- Contributions MUST be small.
- The codebase MUST have active contributors.
- Source code test and documentation coverage SHOULD be monitored.
- Policy and documentation MAY have testing for consistency with the source and vice versa.
- Policy and documentation MAY have testing for style and broken links.

Why this is important

- Using continuous integration:
  - allows you to quickly identify problems with the codebase,
  - enables risk taking and focusing on problem solving while minimizing stress for the contributors,
  - lowers barriers for new contributors by reducing the amount of understanding necessary to suggest changes,
  - leads to more maintainable code,
  - speeds up the development cycle.
- Smaller, more regular contributions are typically easier to evaluate and lower risk compared to large infrequent changes.
- Codebases in active development more reliably provide opportunities for collaboration and feedback.

What this does not do

- Create a fault tolerant infrastructure that will work and scale perfectly.
- Create meaningful tests.
- Test for real life situations.
- Guarantee the code will deliver exactly the same policy result.

How to test

- There are tests present.
- Code coverage tools check whether coverage is at 100% of the code.
- Contributions are only admitted into the codebase after all of the tests are passed.
- Everyone working on the codebase integrates their work at least once a day.
- There are contributions from within the last three months.

P Policy makers: what you need to do

- Involve management as well as developers and designers as early in the process as possible and keep them engaged throughout development of your policy.
- Make sure there are also automated tests set up for policy documentation.
- Fix policy documentation promptly if it fails a test.
- Make sure the code reflects any changes to the policy (see Maintain version control).

M Management: what you need to do

- Make sure to test with real end users as quickly and often as possible.
- Procure consultancy services that deliver small parts very often instead of large parts less frequently.
- After a large failure, encourage publication of incident reports and public discussion of what was learned.

D Developers and designers: what you need to do

- Help management and policy makers test their contributions, by for example testing their contributions for broken links or style.
- Structure code written to handle conditions which are difficult to create in a test environment in such a way that the conditions can be simulated during testing. Forms of resource exhaustion such as running out of storage space and memory allocation failure are typical examples of difficult to create conditions.
- Tune the test code coverage tools to avoid false alarms resulting from inlining or other optimizations.
- Deploy often.

Further reading

- What is continuous integration by Martin Fowler.

  https://www.martinfowler.com/articles/continuousIntegration.html

- What is continuous delivery by Jez Humble.

  https://www.continuousdelivery.com/

- Use continuous delivery by the UK Government Digital Service.

  https://gds-way.cloudapps.digital/standards/continuous-delivery.html

- Quality assurance: testing your service regularly by the Government Digital Service (United Kingdom).

  https://www.gov.uk/service-manual/technology/quality-assurance-testing-your-service-regularly

Publish with an open license

Requirements

- All code and documentation MUST be licensed such that it may be freely reusable, changeable and redistributable.
- Software source code MUST be licensed under an OSI-approved open source license.

https://opensource.org/licenses/category

- All code MUST be published with a license file.
- Contributors MUST NOT be required to transfer copyright of their contributions to the codebase.
- All source code files in the codebase SHOULD include a copyright notice and a license header that are machine readable.
- Codebases MAY have multiple licenses for different types of code and documentation.

Why this is important

- Makes it possible for anyone to see the code and know that they can and how they can reuse it.

What this does not do

- Prevent use of the code by any specific actors.

How to test

- There is at least 1 license file present in the codebase, usually called `license`.
- The license for the source code is on the OSI-approved Open Source license list and the license for documentation is conformant to the Open Definition.

https://opensource.org/licenses/category

https://opendefinition.org/licenses/

- Check for machine-readable licenses with tools like Licensee or REUSE. https://github.com/licensee/licensee

P Policy makers: what you need to do

- Develop policy that requires code to be open source.
- Develop policy that disincentivizes non-open source code and technology in procurement.

M Management: what you need to do

- Only work with open source vendors that deliver their code by publishing it under an open source license.
- Beware that even though Creative Commons licenses are great for documentation, licenses that stipulate Non-Commercial or No Derivatives are NOT freely reusable, changeable and redistributable and don't fulfill these requirements. https://creativecommons.org/licenses/

D Developers and designers: what you need to do

- Add a new `license` file to every new codebase created.
- Add a copyright notice and a license header to every new source code file created.

Further reading

- Open source definition by the Open Source Initiative - all open source licenses meet this definition. https://opensource.org/osd
- Animated video introduction to Creative Commons by Creative Commons Aotearoa New Zealand. https://creativecommons.org/about/videos/creative-commons-kiwi
- REUSE Initiative specification for unambiguous, human-readable and machine-readable copyright and licensing information. https://reuse.software/spec/
- SPDX License List with standardized, machine-readable abbreviations for most licenses. https://spdx.org/licenses/

Use a coherent style

Requirements

- Contributions MUST adhere to either a coding or writing style guide, either the codebase community's own or an existing one that is advertised in or part of the codebase.
- Contributions SHOULD pass automated tests on style.
- The codebase SHOULD include inline comments and documentation for non-trivial sections.
- The style guide MAY include sections on understanda ble English.

Why this is important

- Enables contributors in different environments to work together on a unified product.
- Unifying vocabularies reduces friction in communication between contributors.

What this does not do

- Help contributors write well or effectively explain what they do.

How to test

- Inspect whether contributions are in line with the style guides specified in the documentation.

P Policy makers: what you need to do

- Create, follow and continually improve on a style guide for policy and documentation as well as document this in the codebase, for example in the `CONTRIBUTING` or `README`.

M Management: what you need to do

- Include written language, source, test and policy standards in your organizational definition of quality.

D Developers and designers: what you need to do

If the codebase does not already have engineering guidelines or other contributor guidance, start by adding documentation to the repository describing whatever is being done now, for example in the `CONTRIBUTING` or `README`. An important purpose of the file is to communicate design preferences, naming conventions, and other aspects machines can't easily check. Guidance should include what would be expected from code contributions in order for them to be merged by the maintainers, including source, tests and documentation. Continually improve upon and expand this documentation as you go with the aim of evolving this documentation into engineering guidelines.

Additionally:

- Use a linter.
- Add linter configurations to the codebase.

Further reading

- List of linters by Hugo Martins. https://github.com/caramelomartins/awesome-linters
- Programming style on Wikipedia. https://en.wikipedia.org/wiki/Programming_style

Document codebase maturity

Requirements

- A codebase MUST be versioned.
- A codebase that is ready to use MUST only depend on other codebases that are also ready to use.
- A codebase that is not yet ready to use MUST have one of these labels:
  - prototype - to test the look and feel, and to internally prove the concept of the technical possibilities,
  - alpha - to do guided tests with a limited set of users,
  - beta - to open up testing to a larger section of the general public, for example to test if the codebase works at scale,
  - pre-release version - code that is ready to be released but hasn't received formal approval yet.
- A codebase SHOULD contain a log of changes from version to version, for example in the `CHANGELOG`.

Why this is important

Clearly signalling a codebase's maturity helps others decide whether to reuse, invest in or contribute to it.

What this does not do

- Guarantee that others will use the code.

How to test

- The codebase has a strategy for versioning which is documented.
- It is clear where to get the newest version.

- The codebase doesn't depend on any codebases marked with a less mature status.

P Policy makers: what you need to do

- When developing policy, understand that any code developed needs to be tested and improved before it can be put into service.
- Consider versioning policy changes, especially when they trigger new versions of the source code.

M Management: what you need to do

- Make sure that services only rely on codebases of equal or greater maturity than the service. For example, don't use a beta codebase in a production service or a prototype codebase in a beta service.

D Developers and designers: what you need to do

- Add a prominent header to every interface that indicates the maturity level of the code.
- Version all releases.
- Especially in 'rolling release' scenarios, the version may be automatically derived from the version control system metadata (for example by using git describe).

https://git-scm.com/docs/git-describe

Further reading

- Service Design and Delivery Process by the Australian Digital Transformation Agency.

https://guides.service.gov.au/topics/service-design-delivery-process/

- Service Manual on Agile Delivery by the UK Government Digital Service.

https://www.gov.uk/service-manual/agile-delivery

- Semantic Versioning Specification used by many codebases to label versions.

https://semver.org/

- What are the Discovery, Alpha, Beta and Live stages in developing a service? [Video 0'0"59] by the UK Government Digital Service.

https://www.youtube.com/watch?v=_cyl7DMhgYc

Contributing to this standard

🙇‍♀️ Thank you for contributing!

We understand that a standard like this can only be set in collaboration with as many public technologists, policy makers and interested folk as possible. Thus we appreciate your input, enjoy feedback and welcome improvements to this project and are very open to collaboration.

We love issues and pull requests from everyone. If you're not comfortable with GitHub, you can email use your feedback at info@publiccode.net.

Problems, suggestions and questions in issues

Please help development by reporting problems, suggesting changes and asking questions. To do this, you can create a GitHub issue for this project in the GitHub Issues for the Standard for Public Code. Or, sign up to the mailing list and send an email to standard@lists.publiccode.net.

https://help.github.com/articles/creating-an-issue/
https://lists.publiccode.net/mailman/listinfo/lists/standard.lists.publiccode.net/
https://github.com/publiccodenet/standard/issues

You don't need to change any of our code or documentation to be a contributor!

Documentation and code in pull requests

If you want to add to the documentation or code of one of our projects you should make a pull request.

If you never used GitHub, get up to speed with Understanding the GitHub flow or follow one of the great free interactive courses in the GitHub learning lab on working with GitHub and working with MarkDown, the syntax this project's documentation is in.

https://guides.github.com/introduction/flow/
https://lab.github.com/

This project is licensed CC-0, which essentially means that the project, along with your contributions is in the public domain in whatever jurisdiction possible, and everyone can do whatever they want with it.

1. Make your changes

This project uses the GitFlow branching model and workflow. When you've forked this repository, please make sure to create a feature branch following the GitFlow model.

https://nvie.com/posts/a-successful-git-branching-model/

Add your changes in commits with a message that explains them. Document choices or decisions you make in the commit message, this will enable everyone to be informed of your choices in the future.

https://robots.thoughtbot.com/5-useful-tips-for-a-better-commit-message

If you are adding code, make sure you've added and updated the relevant documentation and tests before you submit your pull request. Make sure to write tests that show the behavior of the newly added or changed code.

2. Pull request

When submitting the pull request, please accompany it with a description of the problem you are trying to solve and the issue numbers that this pull request fixes.

3. Improve

All contributions have to be reviewed by someone.

It could be that your contribution can be merged immediately by a maintainer. However, usually, a new pull request needs some improvements before it can be merged. Other contributors (or helper robots) might have feedback. If this is the case the reviewing maintainer will help you improve your documentation and code.

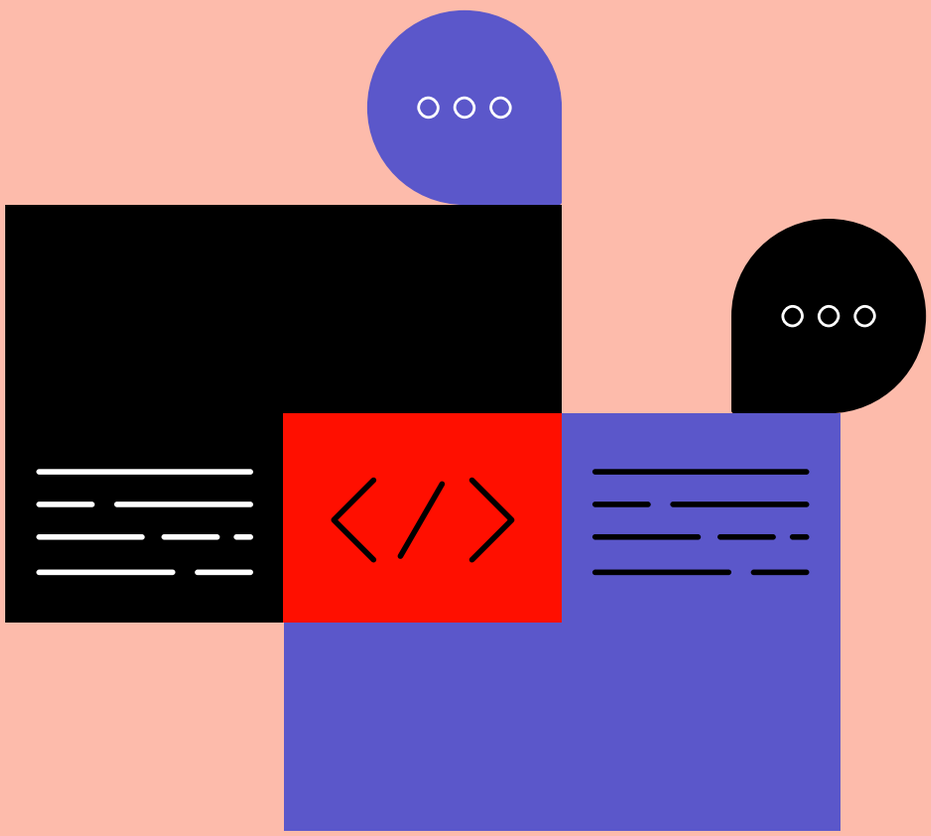
If your documentation and code have passed human review, it is merged.

4. Celebrate

Your ideas, documentation and code have become an integral part of this project. You are the open source hero we need!

In fact, feel free to open a pull request to add your name to the `AUTHORS` file and get eternal attribution.

For more information on how to use and contribute to this project, please read the `README`.



Contributor Covenant Code of Conduct

Our pledge

In the interest of fostering an open and welcoming environment, we as contributors and maintainers pledge to making participation in our project and our community a harassment-free experience for everyone, regardless of age, body size, disability, ethnicity, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

Our standards

Examples of behavior that contributes to creating a positive environment include:

- Using welcoming and inclusive language
- Being respectful of differing viewpoints and experiences
- Gracefully accepting constructive criticism
- Focusing on what is best for the community
- Showing empathy towards other community members

Examples of unacceptable behavior by participants include:

- The use of sexualized language or imagery and unwelcome sexual attention or advances
- Trolling, insulting/derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or electronic address, without explicit permission

- Other conduct which could reasonably be considered inappropriate in a professional setting

Our responsibilities

Project maintainers are responsible for clarifying the standards of acceptable behavior and are expected to take appropriate and fair corrective action in response to any instances of unacceptable behavior.

Project maintainers have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, or to ban temporarily or permanently any contributor for other behaviors that they deem inappropriate, threatening, offensive, or harmful.

Scope

This Code of Conduct applies both within project spaces and in public spaces when an individual is representing the project or its community. Examples of representing a project or community include using an official project e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event. Representation of a project may be further defined and clarified by project maintainers.

Enforcement

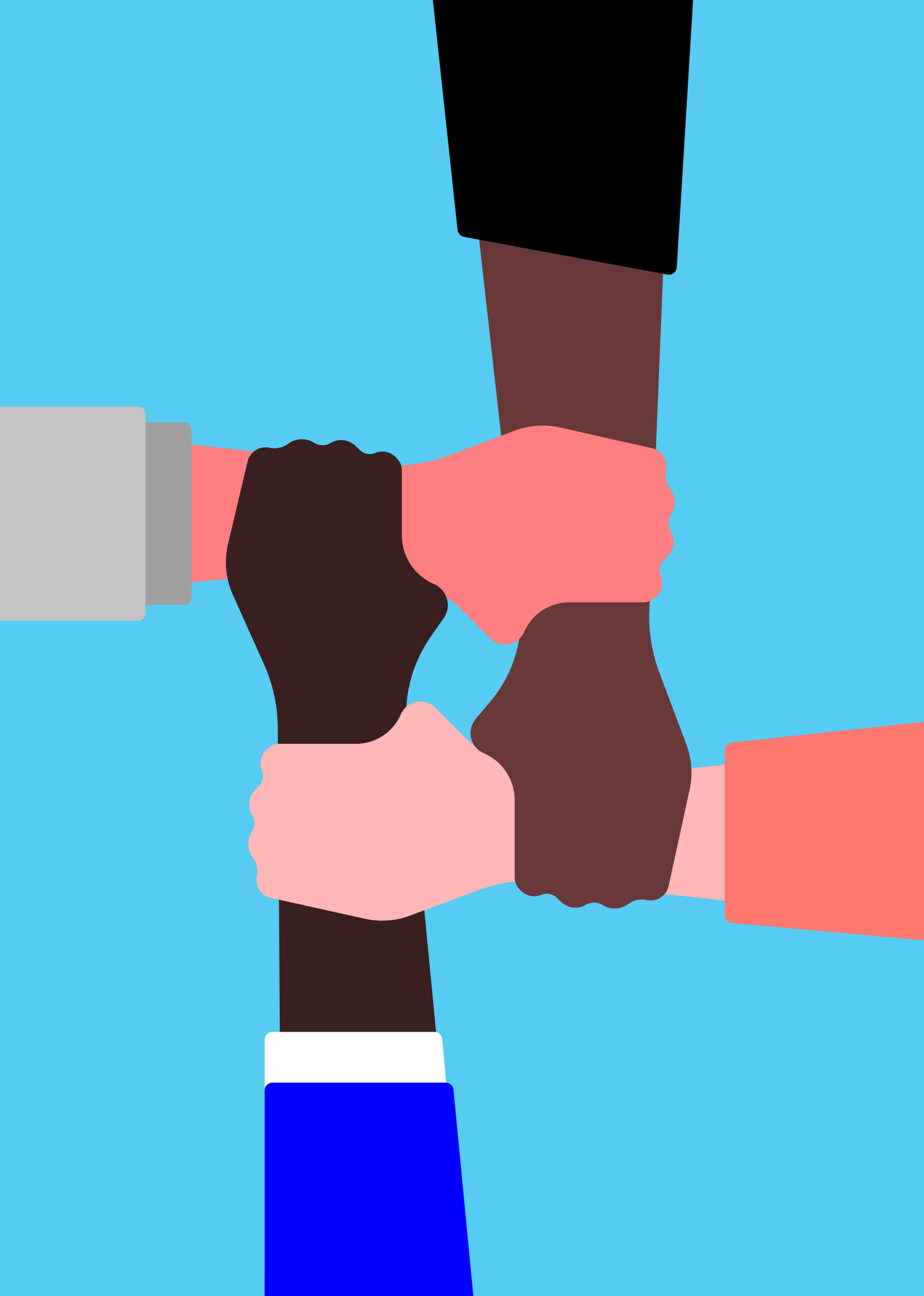
Instances of abusive, harassing, or otherwise unacceptable behavior may be reported by contacting the project team at directors@publiccode.net. All complaints will be reviewed and investigated and will result in a response that is deemed necessary and appropriate to the circumstances. The project team is obligated to maintain confidentiality with regard to the reporter of an incident. Further details of specific enforcement policies may be posted separately.

Project maintainers who do not follow or enforce the Code of Conduct in good faith may face temporary or permanent repercussions as determined by other members of the project's leadership.

Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 1.4, available at https://www.contributor-covenant.org/version/1/4/code-of-conduct.html

https://www.contributor-covenant.org



Governance

This standard lies at the core of the codebase stewardship provided by the Foundation for Public Code. We decide if a codebase is ready for community co-development based on this document.

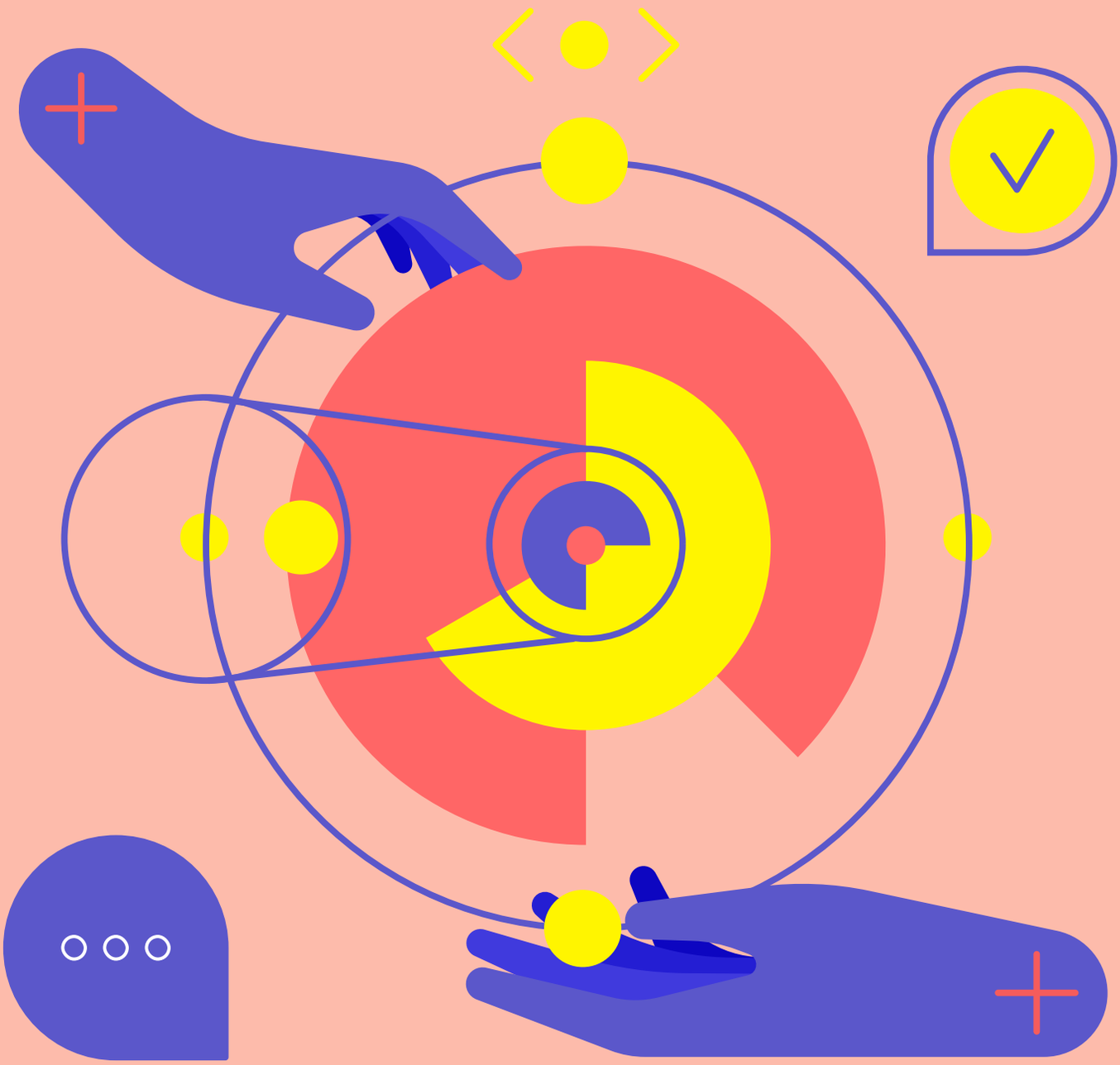
https://publiccode.net/

The standard is maintained by Foundation for Public Code staff.

We welcome contributions – such as suggestions for changes or general feedback – from anyone.

Because of the important role that the Standard for Public Code has in our core process we require the highest standards from the Standard.

We will try to respond promptly to all pull requests. The pull request is an opportunity to work together to improve our methods and the Standard. We may not accept all changes, but we will explain our logic.



Version history

Version 0.2.2

November 29th 2021: 🏛 the eighth draft recognizes that policy which executes as code may not be in English.

- Document exception to “All code MUST be in English” where policy is interpreted as code.
- Add MAY requirement regarding committer email addresses in Maintain version control.
- Expand guidance to Policy Makers in Bundle policy and code.
- Expand guidance to Developers and designers in Use a coherent style.
- Add “Different contexts” to glossary.
- Add Mauko Quiroga and Charlotte Heikendorf to AUTHORS.
- Add Digital Public Goods approval badge.
- Added “next” and “previous” links to criteria pages of web version.
- Add Open Standards principles to further reading.
- Add Definition of plain language to further reading.
- Move the Semantic Versioning Specification further reading reference.
- Clarify that publiccode.yml is one example of a machine readable metadata description.
- Changed “your codebase” and “your organization” to be less possessive.
- Made additional minor changes to text for clarity.
- Add instructions for creating a print version.

Version 0.2.1

March 1st 2021: 🧀 the seventh draft has minor cleaning up after version 0.2.0.

- New SHOULD requirement on using a distributed version control system and why distributed is important.
- Feedback requirements for rejected contributions are more strict than accepted ones.
- Specify that copyright and license notices should also be machine readable.
- Advice on how to test that notices be machine readable.
- Clarify guidance for rolling releases.
- Clear up definition of version control in glossary.
- Add further reading encouraging contribution, SPDX, Git and reviewing contributions.
- Add links to videos about the concept of public code.
- Update BPMN link.
- Reduce link duplication.
- Add Alba Roza and Ngô Ngọc Đức Huy to authors.
- Made additional minor changes to text for clarity.

Version 0.2.0

October 26th 2020: 🎊 the sixth draft splits a requirement and adds clarity.

- Split “Welcome contributions” criterion into “Make contributing easy” and “Welcome contributors”.
- Rename criterion “Pay attention to codebase maturity” to “Document codebase maturity”.
- Changed MUST to SHOULD for requirement of codebase in use by multiple parties.
- Add MUST NOT requirement regarding copyright assignment.
- Clarify role of configuration in reusable code requirement.
- Glossary additions: continuous integration, policy, repository, and version control.

- Replace references to ‘cities’ with ‘public organizations’.
- Clarify aspects of sensitive code by separating contributor and reviewer requirements into separate items.
- Expand further reading, and guidance to policy makers, developers and designers.
- Add Felix Faassen and Arnout Engelen to authors.
- Made additional minor changes to text for clarity.

Version 0.1.4

November 27th 2019: 🧹 the fifth draft consists mostly of additional minor fixes.

- Linked License.md file.
- Add Sky Bristol, Marcus Klaas de Vries, and Jan Ainali to authors.
- Made punctuation more consistent, especially for bullet lists.
- Made some minor changes to text for clarity.

Version 0.1.3

October 8th 2019: 🍂 the fourth draft only patches and fixes minor things for the autumn cleaning

- Renamed continuous delivery to continuous integration.
- Referencing accessibility guidelines in the language standard.
- A bunch of style and consistency fixes.

Version 0.1.2

August 22th 2019: 🌠 the third draft focuses on better text and takes community input

- With some great new contributors comes a fresh author list.
- All links are now HTTPS.

- General proofreading, wording clarifications, and smashed typos.
- Updated criteria:
  - Requirement for reuse in different contexts
  - Recommendation for explicit versioning
  - Recommendation for multi party development
  - Recommendation for license headers in files
  - Recommendation for vulnerability reporting
  - Recommendation for explicit documentation of governance

Version 0.1.1

May 9th 2019: 🤔 the second draft fixes a few basic oversights and fixes a lot of typos

- Removed references to the Foundation for Public Code, we're going to have to change the name in becoming an association.
- Updated the introduction.
- Updated the glossary.
- Added the code of conduct.
- We've recommended using the publiccode.yml standard for easier reuse.

Version 0.1.0

April 16th 2019: 🎉 the first draft is ready, it is all brand new and has snazzy new ideas in it

- 14 criteria with their requirements and how to operationalize them.
- An introduction with a high level background, what this standard is, and how the Foundation for Public Code will use it.

This first version was produced together with the Amsterdam University of Applied Sciences and the City of Amsterdam as a part of the Smart Cities? Public Code! project.

This license is the legal contract that allows anyone to do anything they like with the content in this entire document.

CC0 1.0 Universal

Statement of purpose

The laws of most jurisdictions throughout the world automatically confer exclusive Copyright and Related Rights (defined below) upon the creator and subsequent owner(s) (each and all, an “owner”) of an original work of authorship and/or a database (each, a “Work”).

Certain owners wish to permanently relinquish those rights to a Work for the purpose of contributing to a commons of creative, cultural and scientific works (“Commons”) that the public can reliably and without fear of later claims of infringement build upon, modify, incorporate in other works, reuse and redistribute as freely as possible in any form whatsoever and for any purposes, including without limitation commercial purposes. These owners may contribute to the Commons to promote the ideal of a free culture and the further production of creative, cultural and scientific works, or to gain reputation or greater distribution for their Work in part through the use and efforts of others.

For these and/or other purposes and motivations, and without any expectation of additional consideration or compensation, the person associating CC0 with a Work (the “Affirmer”), to the extent that he or she is an owner of Copyright and Related Rights in the Work, voluntarily elects to apply CC0 to the Work and publicly distribute the Work under its terms, with knowledge of his or her Copyright and Related Rights in the Work and the meaning and intended legal effect of CC0 on those rights.

1. Copyright and Related Rights. A Work made available under CC0 may be protected by copyright and related or neighboring rights (“Copyright and Related

Rights"). Copyright and Related Rights include, but are not limited to, the following:

i. the right to reproduce, adapt, distribute, perform, display, communicate, and translate a Work;
ii. moral rights retained by the original author(s) and/or performer(s);
iii. publicity and privacy rights pertaining to a person's image or likeness depicted in a Work;
iv. rights protecting against unfair competition in regards to a Work, subject to the limitations in paragraph 4(a), below;
v. rights protecting the extraction, dissemination, use and reuse of data in a Work;
vi. database rights (such as those arising under Directive 96/9/EC of the European Parliament and of the Council of 11 March 1996 on the legal protection of databases, and under any national implementation thereof, including any amended or successor version of such directive); and
vii. other similar, equivalent or corresponding rights throughout the world based on applicable law or treaty, and any national implementations thereof.

2. Waiver. To the greatest extent permitted by, but not in contravention of, applicable law, Affirmer hereby overtly, fully, permanently, irrevocably and unconditionally waives, abandons, and surrenders all of Affirmer's Copyright and Related Rights and associated claims and causes of action, whether now known or unknown (including existing as well as future claims and causes of action), in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "Waiver"). Affirmer makes the Waiver for the benefit of each member of the public at large and to the detriment of Affirmer's heirs and successors, fully intending that such Waiver shall not be subject to revocation, rescission, cancellation, termination, or any other legal or equitable action to disrupt the quiet enjoyment of the Work by the public as contemplated by Affirmer's express Statement of Purpose.

3. Public License Fallback. Should any part of the Waiver for any reason be judged legally invalid or ineffective under applicable law, then the Waiver shall be preserved to the maximum extent permitted taking into account Affirmer's express Statement of Purpose. In addition, to the extent the Waiver is so judged Affirmer hereby grants to each affected person a royalty-free, non transferable, non sublicensable, non exclusive, irrevocable and unconditional license to exercise Affirmer's Copyright and Related Rights in the Work (i) in all territories worldwide, (ii) for the maximum duration provided by applicable law or treaty (including future time extensions), (iii) in any current or future medium and for any number of copies, and (iv) for any purpose whatsoever, including without limitation commercial, advertising or promotional purposes (the "License"). The License shall be deemed effective as of the date CC0 was applied by Affirmer to the Work. Should any part of the License for any reason be judged legally invalid or ineffective under applicable law, such partial invalidity or ineffectiveness shall not invalidate the remainder of the License, and in such case Affirmer hereby affirms that he or she will not (i) exercise any of his or her remaining Copyright and Related Rights in the Work or (ii) assert any associated claims and causes of action with respect to the Work, in either case contrary to Affirmer's express Statement of Purpose.
4. Limitations and Disclaimers.
    i. No trademark or patent rights held by Affirmer are waived, abandoned, surrendered, licensed or otherwise affected by this document.
    ii. Affirmer offers the Work as-is and makes no representations or warranties of any kind concerning the Work, express, implied, statutory or otherwise, including without limitation warranties of title, merchantability, fitness for a particular purpose, non infringement, or the absence of latent or other defects, accuracy, or the present or absence of errors, whether or not discoverable, all to the greatest extent permissible under applicable law.
    iii. Affirmer disclaims responsibility for clearing rights of other persons that may apply to the Work or

any use thereof, including without limitation any person's Copyright and Related Rights in the Work. Further, Affirmer disclaims responsibility for obtaining any necessary consents, permissions or other rights required for any use of the Work. d. Affirmer understands and acknowledges that Creative Commons is not a party to this document and has no duty or obligation with respect to this CC0 or use of the Work.

For more information, please see https://creativecommons.org/publicdomain/zero/1.0/

https://creativecommons.org/publicdomain/zero/1.0/

Contact

For questions and more information about the Foundation for Public Code you can find us at our website, email us at info@publiccode.net, or call us at +31 20 2 444 500

https://publiccode.net