# Software development blueprints for *Epiverse-TRACE*

## Context

This blueprint document lays out guidelines for the development of *Epiverse-TRACE*, a free, open-source ecosystem of R packages for outbreak analytics. Rather than focussing only on coding aspects, we adopt the view that software development is a wider, more complex experience relying first and foremost on human interactions between and within developer and user groups. For this reason, the present document provides guidelines on coding practices as well as collaboration workflows, documentation, training, and community health monitoring.

This document was originally written as an attempt to summarize group discussions which took place at *R YouReady?*, a workshop co-organized by data.org and the London School of Hygiene and Tropical Medicine (LSHTM), which gathered experts in R package development as well as from the wider open-source software community in London, on the 7th June 2022. The document therefore reflects inputs from many people, including the workshop's participants, but also people who provided inputs at a later stage. None of the guidelines provided below are set in stone: this blueprint will evolve as *Epiverse-TRACE* grows, and should be taken as a living and evolving document.

## Contributors

*Contributions in alphabetic order:*

Anna Carnegie, Rosalind Eggo, Rich FitzJohn, Dawn Foster, Hannah Frick, Cyril Geismar, Sarah Gibson, Geraldine Gómez Millan, Ernest Guevarra, Hugo Gruson, Thibaut Jombart, Emma Marty, Ibrahim Mahgoub, Nuredin Mohammed, Rebecca Nash, Jaime Pavlich-Mariscal, Maëlle Salmon, Malvika Sharan, Janetta Skarp, Tim Taylor, Heather Turner, Chantal Wood

*Maintainer*: Thibaut Jombart (thibaut@data.org)

## Abbreviations

- CHAOSS: Community Health Analytics Open Source Software
- DPG: Digital Public Good
- OSS: Open Source Software
- LMIC: Low and Middle Income Country
- MVP: Minimal Viable Product
- PR: Pull Request
- WIP: Work In Progress

## Version

The current version of this document is 0.0.2, published on the 2nd Aug 2022.

---

## Guidelines

The recommendations below aim to cover the most important aspects of the development of *Epiverse-TRACE*. For simplicity, we distinguish *users* from *developers* according to the type of contributions they make to a software project: *developers* create content (code and documentation), while *users* advise, test, and provide feedback on content. User-bases will be project-dependent but would typically include field epidemiologists or public health officers.

Our blueprints is defined by 7 key aspects/principles:

- Software development as co-creation
- Lean and agile collaboration framework
- Decentralizing code ownership
- Code reviews
- Documentation
- Being part of the OSS landscape
- Community health

The mindmap below provides an overview of the blueprints content. Please refer to the respective sections for details and explanations.

### Software development as co-creation

Useful software development is unlikely to be achieved by developers alone: rather, it needs to be co-created by users and developers. Some key principles of this co-creation include:

- Users provide key inputs on the identification and prioritization of minimal viable products (MVPs), which can be defined as the smallest increment of the project delivering value for the user
- Users should be involved from the start of a project, and provide continuous feedback as the project develops
- User contributions need to be valued and acknowledged clearly
- User contributions need to be an intrinsic part of the collaboration framework (see section below), with dedicated events such as brainstorming sessions or product reviews
- Co-creation can also take the form of pair-programming to combine domain experts and developers

### Lean and agile collaboration framework

The main objective of our collaboration framework will be to optimize interactions between users and developers. We thus naturally embrace the agile philosophy in which projects progress iteratively through short cycles focussing on delivering and reviewing MVPs. We foresee that each cycle should have three main phases:

1. *Planning*: users and developers write a backlog of tasks to complete, identify top priorities, and define which MVPs will be worked on in the production phase
2. *Production*: developers produce MVPs; if pair-programming is used, users could be involved at this stage too
3. *Review*: presentation of MVPs and collection of feedback; the backlog is updated accordingly; this is also the opportunity for a retrospective assessment of the process: What was done well? What caused problems? How can the team improve the work process?

To maximize agility, we recommend also adopting the lean principle in which the amount of work in progress (WIP) at any time is minimized. In other words, it is more efficient to work on a few MVPs and deliver value

quickly, rather than working on many MVPs at the same time, resulting in a lot of WIP and low completion rates. We recommend using Kanban boards to keep track of progress, and as a tool for identifying long-lasting WIP, likely indicative of blockage or issues which need addressing. These boards should be fully public to encourage and facilitate external visibility and contributions.

### Decentralizing code ownership

The sustainability of open source software (OSS) projects increases with the number of developers who understand the code base enough to make contributions. While there is always a need for a single official maintainer, we will aim at decentralizing code ownership as much as possible, using a series of practices:

- Avoid siloed codebase: have team members move across different projects
- Aim for code clarity by following coding standards established by Rstudio; consider using static code analysis *e.g. lintr* or *styler* to ensure adherence with standards
- Use modular, reusable code, with small, well-documented functions having a clear purpose
- Use canonical, widely used data structures and tools where possible
- Write tests for every function so that other developers can identify easily what it is expected to do, and where/how it is expected to fail
- Implement extensive automated testing, aiming for maximum code coverage, to help identify potential issues in new contributions; we recommend using existing R standards for testing (*testthat*), coverage (*covr*) and continuous integration using github actions (*usethis*)
- Contribute to the code base by PRs with well-identified processes for code reviews (see dedicated section below)
- Conduct occasional audits of the entire code base by other team members

### Code reviews

Code reviews are not just a way to improve overall code quality: they are a key mechanism through which team members can share programming knowledge, better understand existing code bases, and standardize coding practices. Key principles include:

- Consider code reviews are not a hierarchical exercise: they are performed by all members of the developers team.
- Understand that code reviews are a great opportunity for training and collaboration
- Consider Alex Hill'squadrant of code reviews when conducting reviews and treat separately:
  - Trivial issues which can be solved by improving automation (*e.g.* styling using *lintr* or *styler*)
  - Non-conflictual, factual issues which can be quickly agreed on (*e.g.* missing tests, typos in the documentation)
  - Spend time to discuss high-conflict, high-reward items, and take a collaborative approach to identifying solutions
- Use pair-programming sessions when developing features to foster a sense of code co-ownership and peer learning
- Make code reviews an intrinsic part of the contribution workflow: they are typically performed as part of the PR review process.
- Use code walk-throughs ahead of reviews to facilitate the review process.
- Prefer small reviews to large ones
- Conduct occasional audits of the entire code base

### Documentation

Consistent, high quality, and accessible documentation is key to adopting software tools. Key consideration for delivering good documentation include:

- Consider documentation on the same level as code, rather than as an afterthought
- Use the Diataxis framework to distinguish: tutorials, how-to guides, reference manuals, and explanatory material
- Generate documentation dynamically using standard tools: *roxygen2* for basic documentation, *rmarkdown* for vignettes, *pkgdown* for websites
- Provide user-friendly cheatsheets in html and pdf
- Provide interactive content (tutorials, how-to guides) with no need for installing dependencies locally using mybinder.org
- Write documentation assuming minimum R literacy and prior knowledge
- Content should clearly state requirements, objectives, and value for the user
- Aim to provide translations for the most common languages

**Being part of the OSS landscape**

As an open-source initiative, Epiverse-TRACE will benefit from numerous other OSS projects. In becoming part of the OSS landscape, we will observe a few principles:

- We use open-source, liberal licenses (*e.g.* MIT) for our tools
- We build upon open data standards (*e.g.* FHIR, HDX) and develop interoperability with existing data resources and Health Information Management Systems (HIMS)
- We develop tools for social impact, in compliance with Digital Public Goods standard
- We take, but we give back: we use and build upon OSS projects, but we give back to the community by developing new OSS tools, and contributing to existing ones
- We prioritize contributions to existing projects over forks, and forks over project duplications
- When code re-use is needed, we duly credit authors and point to the source, and invite authors as contributors on our project. We give credits by prefacing reused code by a comment e.g.: `// credits: this code was adapted from SOURCE_URL`, where `SOURCE_URL` is the URL we obtained the code from (*e.g.* stackoverflow, etc.). Using a fixed syntax for credits can be used to summarize such contributions programmatically.
- When reusing licensed code, we ensure that third-party licenses are compatible with our project's license. For instance, we may want to avoid reusing GPL code if our project uses MIT, since we would then have to change our license to GPL. See this article by the Free Software Foundation for more information on free software licenses.
- We prioritize the integration of existing tools over development from scratch of new tools
- We welcome external contributions and promote transparency on all our projects
- When contributing to external projects, we observe their contributor code of conduct

**Community health**

The success of an OSS project like epiverse-trace relies heavily on the quality and frequency of interactions between members of our community of users and developers. We will ensure a healthy community through the following principles:

- Foster a welcoming environment which promotes diversity and inclusiveness
- Provide a code of conduct for our community members to highlight and support our values
- Give proper recognition to all types of contributions, which go beyond coding and can take diverse forms *e.g.* testing, input on tool design, training
- Use CHAOSS metrics to monitor community health and reflect our values
- Use occasional surveys to monitor aspects of community health which cannot be captured automatically
- Invest in mentoring programs to strengthen capacity where it is most needed