Software Development Best Practices of the AI-READI Project

Version history

Version number	Change type (Approval or Suggestions)	Implemented by	Date	Description of changes
0.1	-	Bhavesh Patel, Sanjay Soundarajan	09/30/2022	First draft
0.1	Suggestions	Shannon McWeeney	10/12/2022	Edits from Shannon
0.1	Suggestions	Erik Benton; Liz Moyer; Aaron Coyner; Ben Cordier; Wei-Chun Lin; Aaron Cohen; Julie Mitchell	10/13/2022	Add'l edits and comments from OHSU
0.2	Approval	Bhavesh Patel	10/17/2022	Approved above listed suggestions for v0.1 when agreed with them
0.2	Suggestions	Bhavesh Patel, Sanjay Soundarajan	10/17/2022	Added comments and suggestions on the above listed suggestions for v0.1. Also made some edits to improve other elements (e.g., weblink to suggested automation tools)
0.3	Approval	Bhavesh Patel	10/21/2022	Approved additional elements from 0.1 and 0.2 when agreement from original initiator
0.4	Approval	Shannon McWeeney	10/25/2022	Accepted changes and cleaned up document for review by other modules
0.4	Approval	SKILLS (Sally	10/27/22	Not currently planning to

		Baxter)		write any code for the project but reviewed the document in its entirety and no comments. Looks great!
0.4	Approval	TEAMING (Sara Singer)	10/27/2022	No comments, Not planning to write any code
0.4	Approval	DATA (Cecilia Lee):	11/01/2022	Not currently planning to write any code but reviewed the document and no comments. Thank you!
0.4	Approval	ETAI (Camille Nebeker)	11/15/22	Not planning to write code. Reviewed document.Please check in with ETAI as needed.
0.4	Approval	STANDARDS (Aaron Lee)	11/18/22	Accepts changes to previous comments
0.5	Approval	Bhavesh Patel	11/19/22	Approved changes made based on comments from Aaron Lee
0.5	Suggestions	Ben Cordier	11/21/22	Added suggestions re. Browser support and testing (Section 2, clause i).
1.0.0	Approval	Bhavesh Patel	11/23/2022	Accepted changes suggested by Erik, Ben and Sanjay about best practices for testing. Accepted section E for monitoring best practice.Implement and accepted changes authorship and dependencies. Re-numbered the different sections including the best practices for better conversion to md format.

A. Overview

Given the complexity of the Bridge2AI projects, it is critical to establish software development best practices for both the Tools module and the other stakeholders. This is reflected in Milestone 1.1.1 of the Tools Module of our Bridge2AI data generation project, AI-READI: "Establish software development practices to follow for the project." Briefly, the goal of this milestone is to establish project-wide best practices to be followed by anyone developing code for the AI-READI project. The rationales for establishing these best practices are to:

- Help the standardization of the AI-READI software to facilitate future reuse, *i.e.* make the AI-READI software Findable, Accessible, Interoperable, and Reusable (FAIR)
- Ensure the efficient and effective development of software across the project, especially when contributions from several teams are required
- Facilitate on-boarding of new developers to the project

Our focus here is on establishing actionable best practices that can be directly implemented by developers of the project. These best practices are intended to remain general enough such that they can apply to any software development project within AI-READI.

B. Maintaining the best practices

This document will be used to establish the first version of the best practices (v1.0), since Google Docs facilitates collaborative inputs. Once v1.0 is finalized, we will publish v1.0 of the best practices in the "software-dev-best-practices" repository in the AI-READI GitHub organization and maintain all future versions from there. The best practices will be shared openly under the Creative Commons Attribution 4.0 International (CC-BY) license. The README of that repository will contain elements of Sections A and B of this document. For v1.0, we will include a PDF version of this document as well as a markdown version of the best practices (Section C), a markdown version of the design rationale for the best practices (Section D), and a markdown version our approach for monitoring the best practices (Section E). Future modifications and suggestions can be made by submitting a pull request with edits on the markdown versions. Such edits could be made, for instance, to improve v1.0 or align with potential best practices agreed upon by the Bridge2AI consortium. These pull requests will be reviewed and approved by the Tools module PIs with inputs from other modules PIs or team members if required. A dedicated folder will be created for each version in the "software-dev-best-practices" repository such that they are all preserved. A "CHANGELOG.md" will be maintained to document changes between versions.

A GitHub release will be created for each new version of the best practices. The GitHub repository will also be archived on Zenodo such that each version of the best practices can be referenced with a unique digital object identifier (DOI). Additionally, the latest version of best practices will be visible on dev.aireadi.org, the developer documentation website for the AI-READI project. This documentation will display the best practices in a user-friendly manner and will contain additional elements (*e.g.*, code snippets) such that the best practices can be easily implemented by developers.

C. Software development best practices for AI-READI

Our best practices are designed to A) Comply with the FAIR4RS principles, B) Promote open-source practices, C) Facilitate collaborations and inclusivity (aligning with inclusive development and innovation which are part of the CARE principles), and D) Provide convenience to developers. The rationale for such a design is provided in Section D of this document. For each element of the best practices, we indicate which of the categories A-D they fulfill. Moreover, our best practices are based on the resources listed below. These resources were selected based on the authors' knowledge of their relevance to the categories A-D. For each element of the best practices, we indicate which resources it is based on.

Resources:

[1] AI-READI team members' knowledge and experience
[2] Making Biomedical Research Software FAIR: Actionable Step-by-step Guidelines with a User-support Tool, https://doi.org/10.1101/2022.04.18.488694
[3] General guidelines for biomedical software development, https://doi.org/10.12688%2Ff1000research.10750.2
[4] Good enough practices in scientific computing, https://doi.org/10.1371/journal.pcbi.1005510
[5] NIH Best Practices for Sharing Research Software https://datascience.nih.gov/tools-and-analytics/best-practices-for-sharing-research-software-faq
[6] Ten simple rules for documenting scientific software https://doi.org/10.1371/journal.pcbi.1006561
[7] 4 Simple recommendations for Open Source Software https://softdev4research.github.io/4OSS-lesson/01-introduction/index.html
[8] Reasons to use TypeScript over JavaScript https://www.typescriptlang.org/why-create-typescript
[9] Best practices for Github https://w3c.github.io/best-practices.html
[10] Elixir software management plan template https://elixir-europe.org/sites/default/files/documents/software-management-plan.pdf
[11] Top 10 metrics for life science software good practices [version 1; peer review: 2 approved] https://f1000research.com/articles/5-2000/v1
[12] Towards FAIR principles for research software https://content.iospress.com/articles/data-science/ds190026
[13] Care principles https://www.gida-global.org/care
[14] IEEE-CS Code of Ethics https://www.computer.org/education/code-of-ethics
[15] Understanding community participation and engagement in open source software Projects: A systematic mapping study (and references within) https://www.sciencedirect.com/science/article/pii/S1319157820305139
[16] Ethics in the Software Development Process: From Codes of Conduct to Ethical Deliberation https://arxiv.org/abs/2011.03016
[17] Ethics in computer software design and Development

https://www.srs.fs.usda.gov/pubs/VT_Publications/01t7.pdf
[18] Web accessibility in mind
https://webaim.org/
[19] Tools to benchmark software products (Android, Windows, Web) for accessibility requirements
https://accessibilityinsights.io/
[20] Responsive inclusive design (RiD): a new model for inclusive software development
https://link.springer.com/article/10.1007/s10209-022-00893-9#Sec13
[21] License Compatibility and Relicensing
https://www.gnu.org/licenses/license-compatibility.html

0. Prior to development

Prior to development, ensure an ethical and inclusive framework for deliberation and utilize a responsive inclusive design (RiD) [Category: C] [Ref: 13-17, 20] to ensure that users are active participants in the design and validation of the system from the beginning of the project and that the development project is based on iterative models in the design and verification phases that involves these users, communities and domain experts.

1. Project setup

1.1. Work from GitHub with a GitHub repository dedicated to your software in the AI-READI organization [Category: A, B, C] [Ref: 1-7]

1.2. Follow GitHub best practices [Category: C] [Ref: 9]

1.3. Make the GitHub repository public from day 1 [Category: B] [1, 7]

1.4. Choose a license and include a “LICENSE” file containing standard license terms in the root of your GitHub repository [Category: A, C] [Ref: 2-4, 7]

1.4.1. All software in the AI-READI project will be developed under the MIT license (unless exceptions) [Category: B] [Ref: 1]

1.5. Add a “README.md” file in the root of your GitHub repository and include at least the following information as they become available [Category: A, C] [Ref: 2, 4, 6]:

1.5.1. Brief description of the software [Ref: 2, 4, 6, 7,10-12]

1.5.2. How to run the software (from the source code, mention major dependencies required to setup the development environment such as Python or Node along with their versions) [Ref: 2, 4, 6, 10-12]

1.5.3. Inputs and outputs of the software, if applicable [Ref: 2]

1.5.4. Parameters, data, and data formats required to run the software, if applicable [Ref: 2, 10-12]

1.5.5. The standards followed during development (c.f. section 2.1) [Ref: 2, 10-12]

1.5.6. How to report bugs/issues (c.f. section 4.5) [Ref: 2, 10-12]

1.5.7. How to cite the software (mention Zenodo citation as per 6.e and other publications if applicable). Use the APA citation format. [Ref: 2]

1.5.8. Acknowledgment of NIH funding support [Category: C] [Ref: 5]

1.5.9. Other acknowledgments, if applicable [Ref: 6]
1.5.10. Include shields.io badges at the top of your README to showcase the following: license, number of contributors, number of open issues, Zenodo DOI (c.f. section 6.5) [Ref: 1, 6]
1.5.11. Include the AI-READI logo available here at the bottom of your README [Category C][Ref: 1].

2. Code development

2.1. Follow language-specific standards and conventions when writing your code [Category: A, C] [Ref: 1, 2, 10-12]
2.1.1. Python: Python Developer's Guide
2.1.2. R: Google's R Style Guide
2.1.3. Java: Google's Java Style Guide
2.2. Use an appropriate linter for automated code quality improvements and formatting fixes (c.f. section 7)
2.3. Give functions and variables meaningful names [Category: C] [Ref: 3, 4]
2.4. Include comments within your code when deemed necessary for reuse. These comments need to be added using already accepted standards or conventions (e.g: JSDoc for JavaScript, Google style guide for docstrings for Python code) [Category: A, C] [Ref: 1, 2, 3, 4, 6, 10-12]
2.5. If using external libraries, favor well-maintained software libraries and make sure their license is compatible with your software license [Category: C] [Ref: 4, 7, 21]
2.6. Record software dependencies in a requirements.txt or similar file [Category: A, C] [Ref: 2, 4]
2.7. Add appropriate tests (unit, integration, and end-to-end) to all modules in your application. Try to achieve high coverage to ensure that future developments do not cause accidental regressions in your application. Continuous Integration can/should? be set up to handle automated testing between all developers in the team. Cross-browser accessibility can be handled by the end-to-end test runner [Category: B, D] [Ref: 3, 4, 11]
2.8. Identify and utilize tools to benchmark for accessibility such as Accessibility Insights for Web development etc. [Category: C] [Ref: 18-19]
2.9. Web-based software should support Modern Browsers developed with the Webkit, Blink, or Gecko rendering engines (e.g. Chrome, Safari, Firefox, MS Edge). Uncertainties around specific support for recent HTML/CSS/JavaScript features may be resolved via the CanIUse database (https://caniuse.com/). Testing of front-end layout rendering and cross-browser consistency can be performed locally via browser-specific development tools (e.g. Firebug, Chromium developer tools) or virtualization (e.g. https://www.browserstack.com/) if available.

3. Pushing changes to GitHub repository

3.1. Use two branches for your software: main and dev. The main branch will be the most stable. The dev branch contains new features, and more unstable branches may be pull requested to this branch. Only bug fix branches should be merged

directly into the main branch and these merges should be initiated by a Pull Request (PR) from the bug fix branch. All additional features/bug fixes should always be developed and merged via feature branches. [Category: C] [Ref: 3]

3.2. When committing a change to the repository, make sure the change reflects a single purpose (*e.g.* fixing a bug or adding a new feature) [Category: C] [Ref: 3]

3.3. Use conventional commits [Category: C] [Ref: 1]

3.4. Make sure your git commit hooks are always run before committing to the public repository. Avoid skipping these hooks as they might run important actions on your system.

3.5. When merging changes to the dev branch, create a PR from the feature or bug fix branch and invite at least one other collaborator to review the changes. Leave comments in the PR to clarify intent, decision points or particular sections to review to ensure the reviewer can focus on the areas of importance. Reviewers should use the PR to respectfully ask clarifying questions or make suggestions of the approach when appropriate.

4. **Documenting**

4.1. If more details are required to run the software from the source code in addition to what is included in the "README.md" (c.f. section 1.4.2), include them in other markdown files and refer to them in the README. These other markdown files can be located in a "docs" folder at the root of the repository [Category: A, C] [Ref: 1]

4.2. If the software contains a user interface, maintain user documentation of the interface and identify in your README where the user documentation is located

4.2.1. Use the wiki of your GitHub repository for simple software [Category: A, C] [Ref: 1]

4.2.2. Build and maintain a dedicated webpage for the documentation for more complex software. Have a dedicated GitHub repository for your documentation code in this case and follow our best practices for that code as well. [Category: A, C] [Ref: 1]

4.3. If your software includes an application programming interface (API), it must be documented separately as per API best practices. In the AI-READI project, APIs will be documented using the OpenAPI Specification and hoppscotch. Mention in your README where the API documentation is located. [Category: A, C] [Ref: 1, 6, 7]

4.4. Include instructions for contributing to your software in a "CONTRIBUTING.md" file located at the root of your GitHub repository [Category: C] [Ref: 4, 7]

4.5. Use GitHub issues to manage bugs/issues [Category: D] [Ref: 1]

5. Adding metadata

5.1. Include metadata files:

5.1.1. Include a codemeta.json file in the root of your GitHub repository [Category: A] [Ref: 2, 7].

Provide at least the following information: Software name ("name"), Software description/abstract ("description"), Unique identifier

("identifier"), Authors (“givenName”, “familyName”) with their Organization name (“affiliation”), Keywords (“keywords”), Programming Language (“programmingLanguage”), First and current release date (“datePublished” and “dateModified”), License used (“license”)

5.1.2. Include a CITATION.cff file in the root of your GitHub repository [Category: A] [Ref: 2, 4, 5, 6]
Provide at least the following information: Authors ("given-names", “family-names") with their Organization name (“affiliation”), Software description/abstract (“abstract”), Unique identifier ("identifiers"), Keywords (“keywords”), License (“license”), Release date (“date-released”)

6. Version release

6.1. Document changes between versions of your software in a “CHANGELOG.md” file located at the root of your GitHub repository [Category: C] [Ref: 1]
- 6.1.1. Follow the Semantic Versioning 2.0 principles outlined here for version numbering [Category: C] [Ref: 1]
- 6.1.2. Follow the “keep a changelog” guidelines for structuring your CHANGELOG file [Category: C] [Ref: 1]

6.2. Make a GitHub release for each version of your software [Category: A] [Ref: 1]
- 6.2.1. Use an automated git changelog builder based on conventional commits to document the changes for each release [Category: D] [Ref: 1]

6.3. Deposit in a language-specific repository if available: [Category: A] [Ref: 2]
- 6.3.1. Python packages: PyPI [Category: A] [Ref: 2]
- 6.3.2. R packages: CRAN [Category: A] [Ref: 2]
- 6.3.3. Docker-based tools: Dockstore [Category: A] [Ref: 2, 5]
- 6.3.4. JavaScript packages: npm [Category: A] [Ref: 1]

6.4. Deposit in a domain-specific repository if available:
- 6.4.1. Computational neuroscience models: ModelDB [Category: A] [Ref: 2]
- 6.4.2. R-packages aimed at the analysis of genomics data: Bioconductor [Category: A] [Ref: 2, 5]

6.5. Deposit software on Zenodo: [Category: A] [Ref: 2, 4-7]
- 6.5.1. Include source code [Category: A] [Ref: 2, 5]
- 6.5.2. Include executable and other files necessary to run the software if applicable [Category: A] [Ref: 2, 5]

6.6. Share software on the bio.tools registry if the software could be of interest to the larger biomedical community outside the AI-READI project [Category: A] [Ref: 2, 7]

7. Suggested automation tools/Developer experience

In this section, we provide suggestions of automated tools and approaches to fullfill some of the items from sections 1-6. These are purely suggestions and developers are free to follow other approaches as long as they fullfill all the items from sections 1-6.

7.1. Unless you have a preferred code editor, use VS Code for developing your code and include the `.vscode` directory in your GitHub repository.

7.2. Use these VS Code extensions (or their equivalent in your preferred code editor) as applicable to your project to automatically implement some elements of the best practices: [Category: D] [Ref: 1]
  7.2.1. Better Comments: Help you create more human-friendly comments and visualize them in your code.
  7.2.2. ESLint: Integrates ESLint code formatting into VS Code for Javascript projects.
  7.2.3. Isort: A Visual Studio Code extension that provides import sorting for Python using isort.
  7.2.4. Prettier - Code formatter: Prettier is an opinionated code formatter. It enforces a consistent style by parsing your code and re-printing it with its own rules that take the maximum line length into account, wrapping code when necessary.
  7.2.5. TypeScript Vue Plugin (Volar): A TypeScript server plugin to make the TypeScript server know *.vue files.
7.3. Use an appropriate linter for additional code quality checks before pushing a commit (eg: Flake/Black for Python, ESlint/Prettier for Javascript, etc.)
7.4. When adding automated testing to your application try to use the following testing frameworks if possible:
  7.4.1. Javascript:
    7.4.1.1. Unit/Integration - Vitest/Jest
    7.4.1.2. End-to-end - Cypress
  7.4.2. Python:
    7.4.2.1. Unit/Integration - pytest
7.5. Use GitHub actions to automate tasks such as fixing grammatical errors, formatting code, managing GitHub issue submission, stale issues and PRs, run unit/integration or end-to-end tests, build and release app/api/documentation versions on GitHub and Zenodo. Include GitHub action workflows in a ".github/workflows" folder in your repository. Standard workflow files are maintained for the AI-READI project in the "github-workflows" repository and can be copied into your software repository. [Category: D] [Ref: 1]
7.6. Use TypeScript for any web/Node.js applications instead of JavaScript where possible. TypeScript is a strongly typed programming language that extends JavaScript, for a better developer experience and more fault tolerant applications. [Category: D][Ref: 1]
7.7. Use Docusaurus for developing and maintaining simple hostable development and/or user documentation [Category: D] [Ref: 1]
7.8. Use FAIRshare to create metadata files easily, make GitHub releases, share software files on Zenodo, and register the software on bio.tools [Category: D] [Ref: 1-2]
7.9. Mirror repository on GitLab for backup. The AI-READI organization has GitLab sync automatically set up so no further actions are required [Category: D] [Ref: 1]

D. Rationales

We provide below the rationales behind the design of our best practices.

1. Comply with the FAIR4RS principles

The Findable, Accessible, Interoperable, Reusable (FAIR) Principles published in 2016 provide a framework for optimizing the reuse of all digital research objects, including research software. They are widely accepted by all research fields (7.5k+ citations) and are a key aspect of the Bridge2AI program. Our best practices thus include elements for making all research software developed in the AI-READI project FAIR. While postulated for all digital research objects, several research groups have shown that the FAIR principles as written do not directly apply to software because they do not capture the specific traits of research software. To address this shortcoming, the FAIR for Research Software (FAIR4RS) Working Group of the Research Data Alliance (RDA) established in 2021 reformulated FAIR principles tailored for research software called the FAIR4RS principles. Just like the original FAIR principles, the FAIR4RS principles are aspirational and aimed at providing a general framework. Our team has established actionable guidelines called the FAIR-BioRS guidelines for making biomedical research software compliant with the FAIR4RS principles. Our software development best practices are designed based on these FAIR-BioRS guidelines.

2. Promote open-source practices

FAIR is not equal to Open: the “A” in FAIR stands for “Accessible under well-defined conditions”, not “Accessible to all without restrictions”. In the AI-READI project, we will aim to promote open-source development practices to facilitate the reuse of our software further and encourage external contributions. Our best practices thus include elements for promoting open source practices in all software developed in the AI-READI project. These elements are based mainly on relevant literature such as the “General guidelines for biomedical software development”, the “Good enough practices in scientific computing”, NIH’s “Best Practices for Sharing Research Software “, and the “Ten simple rules for documenting scientific software”.

3. Facilitate collaborations and inclusivity

Since we expect software to typically involve multiple developers and stakeholders/communities, our best practices include elements to facilitate collaborations. These elements are based on the AI-READI team members' knowledge and experience, as well as relevant literature such as the “General guidelines for biomedical software development”, “Good enough practices in scientific computing”, aspects of the CARE principles (which are relevant not only to data but also software), published studies on community engagement in software development etc.

4. Provide convenience to developers

We recognize that implementing some of the elements suggested in our best practices could be time-consuming for developers. Our best practices thus include elements for developers to conveniently and efficiently implement these elements. These elements are based mainly on the AI-READI team members' knowledge and experience. For example, wherever possible, boilerplate and partially completed template files will be provided for configuration, documentation, meta-data and other support files to decrease the time and effort necessary to meet these requirements.

E. Monitoring compliance with the best practices

Compliance with these best practices will be self monitored by the developers of each code project. Each repository will be required to include the following self-assessment file at the root folder of the repository. This file will be updated progressively as the developers comply with the various items of the best practices. Anyone could then quickly assess compliance and open a GitHub issue or pull request if any anomalies are detected.