**“RIMBE/PUMP Data Team” Collaborative Project Constitution**

Authors: Ellie White, Julie Padilla, Lauren Koenig, Amelia Snyder, Ted Thompson

**January 2023**

# Scope

* This document establishes guidelines to standardize collaborative development within the RIMBE/Pump Data Team. The goal is to maximize cohesion and minimize distinctions without differences.
* This document does not establish coding standards. Those are specific to the coding languages used, and it does not explain how to use the tools outlined, but links to trainings are available.
* By sharing this document, we do not intend to overwhelm collaborators, but to facilitate better practices.

Contents

[Scope 1](#_Toc137125269)

[Section 1 – Organizational Levels 2](#_Toc137125270)

[Group 2](#_Toc137125271)

[Subgroups 2](#_Toc137125272)

[Projects 2](#_Toc137125273)

[Epics 2](#_Toc137125274)

[Milestones 2](#_Toc137125275)

[Section 2 – Defining Roles and Responsibilities 2](#_Toc137125276)

[Section 3 – Tools 3](#_Toc137125277)

[Task Management 3](#_Toc137125278)

[Coding Languages & Tasks 3](#_Toc137125279)

[Version Control & Repository Management 3](#_Toc137125280)

[Pipelining 5](#_Toc137125281)

[Other tools 5](#_Toc137125282)

[Section 4 – Data Discovery 5](#_Toc137125283)

[Section 5 - Data Creation 5](#_Toc137125284)

[Data Dictionaries 6](#_Toc137125285)

[Section 6 - Workflow Design 7](#_Toc137125286)

# Section 1 – Organizational Levels

In this project, we anticipate multiple code repositories being developed collaboratively. To prevent future confusion, we suggest using the following standards to organize code on GitLab (code.usgs.gov).

## Group

The top organizational level available on GitLab. We recommend that ALL our repositories are in the same group.

* WMA: <https://code.usgs.gov/wma/>

## Subgroups

Decide on a per repo basis which subgroup is a best fit. Some projects have their origins in Water Prediction (wp) and it would make sense that extensions of their work stays close to where it was originated.

* Water Prediction: <https://code.usgs.gov/wma/wp/>
* RIMBE: <https://code.usgs.gov/wma/rimbe/>

## Projects

When making a new repo, spell out any acronyms in the repo name (for searchability later) and add to the list below:

* <https://code.usgs.gov/wma/wp/national-geospatial-attributes>
* <https://code.usgs.gov/wma/rimbe/national-land-cover-database>
* CONUS 404 notebook
* Water Quality Processing pipeline

## Epics

This is a feature on GitLab for organizing issues associated with a code repository. We will use the following terms to tag project tasks.

* Sourcing: find the best place to pull data from
* Extracting and Validating: check for data integrity, enforce data types
* Processing/Transforming: clean, create aggregates or disaggregates
* Visualizations: figures for internal communications and publishing
* Staging: ensure data conforms structurally to the requirements of the output
* Publishing/Archiving/Releasing

## Milestones

* Bi-weekly sprint goals on Microsoft Teams
* Quarterly milestones

# Section 2 – Defining Roles and Responsibilities

Add your name and expertise below. These descriptions will be helpful when seeking expertise for problem-solving or code reviews.

* Ellie White: R programmer, pipelining, statistical analysis, data visualization
* Lauren Koenig-Snyder: R programmer, pipelining, statistical analysis
* Ted Thompson: Python programmer, pipelining, data engineering
* Margaux Sleckman: Python & R programmer, pipelining, data engineering
* Julie Padilla: Project Lead

# Section 3 – Tools

## Task Management

* Microsoft Planner:
  + Mark sub tasks complete as you complete them.
  + Show your work: include links to where the tasks have been completed. Ideally GitLab issues.
  + Project Manager or team lead will mark the task complete after review.
  + Determine priority of tasks, their dependencies, and set deadlines.
* GitLab Project Management Features: since we don’t have licenses, use these …
  + Labels: P1 (for priority), Bug, Discussion, Documentation, Enhancement or P2.
  + Issues: for documentation of ideas
  + CI/CD: how do we tell GitLab when are using targets?
* Microsoft Teams:
  + [Channel](https://teams.microsoft.com/l/team/19%3a6kpAFs3qp9vcVCWK34KiBJnaBd2bRVo7MuzGPdYr0_U1%40thread.tacv2/conversations?groupId=a3723e2f-342b-4d14-8189-0f4452fe339d&tenantId=0693b5ba-4b18-4d7b-9341-f32f400a5494)
  + Pin the “RIMBE/PUMP Data Team” chat in Teams.
  + Post questions to channel or ping the chat.
* Roadmap:
  + Link to project planning, Gantt charts?
  + How are we communicating shifting project priorities and asset allocation?
    - Development team should raise recommendations to task lead that then communicates it to the steering team
    - How is the steering team communicating to the development team?
  + How and when are we going to review long-term goals, big picture stuff, and major changes?
    - Monthly/Quarterly meetings with the steering team and everyone else.

## Coding Languages & Tasks

* Python
  + NOAA GEFS Processing
  + CONUS 404
* R
  + [Geospatial Attributes Processing](https://code.usgs.gov/wma/wp/national-geospatial-attributes)
  + [National Land Cover Database](https://code.usgs.gov/wma/rimbe/national-land-cover-database)
  + WQP

## Version Control & Repository Management

* GitHub/GitLab Repo:
  + Use the [standard template repository](https://code.usgs.gov/wma/rimbe/standard-template-repository) when starting a repo.
  + Separate folders and code into your relevant phases (e.g., 1\_fetch/, 2\_process/, 3\_visualizations/).
    - Code under each phase will live in src/ (e.g., 2\_process/src/).
    - Outputs of each phase will live in out/ (e.g., 2\_process/out/).
* Description:
  + Use a README with [changelogs](https://keepachangelog.com/en/1.0.0/) to document changes in code that may impact results. This allows a user to understand the changes that happened and revert to previous version of the workflow, if needed.
  + Document your repo’s structure and mention what you will find in each folder. If you have visualized the workflow (like with this Mural Template), include that in the README.
* Issues:
  + Ideas and discussions will be documented in issues. Not all issues will become a branch, but a branch should have a corresponding issue.
* Branches:
  + Put the issue number and your initials in the beginning of a branch name, and use “–“ for spaces (e.g., 8-ew-model-eval).
  + Main branch will be protected and only working staged branches will get merged into main.
  + Branch and pull when you are a close group of developers working rapidly. Fork and pull when you don’t want to manage access to your repository and want to discard experiments easily. I think we mostly fall into the branch and pull category.
* Pull Requests (AKA Merge Requests):
  + Pull requests and major code contributions will have reviews([best practices](https://dsp-manual.wma.chs.usgs.gov/docs/programming/peer_code_review/)). Identify who is doing code review and approving pull requests. When possible, rotate reviewers so others on the team stay abreast of changes.
  + Per USGS [Policy](https://www.usgs.gov/products/software/software-management/types-software-review), no repos will have sensitive information including: Personally Identifiable Information (PII), absolute file system paths, internal server host names or IP addresses, Usernames/passwords.
    - You can obscure absolute file system paths yourself or use R Projects if working in R.
* Containers:
  + When possible, use a Docker container to standardize the project’s working environment. If we create an image, we should apply it to the larger RIMBE project.
  + If you do not want to use Docker, report out the language version, platforms, and package versions in the README.
    - In R you can find this info by typing: SessionInfo() in the console.
* .gitignore:
  + Ignore your input data files (unless they are small) and your out/ folders. You can commit a .empty file in those folders so that the folder structure of the repo is preserved on the remote and for anyone that clones the repository.
  + Example in python, example in R.

## Pipelining

Pipelining should always be considered and planned for, but depending on the nature of the repo, it may not be appropriate, or you might choose to build the pipeline once you have done some early days exploratory work.

* Python: [Snakemake — Snakemake 7.20.0 documentation,](https://snakemake.readthedocs.io/en/stable/)
* R: [The {targets} R package user manual (ropensci.org)](https://books.ropensci.org/targets/)
  + Name targets with prefixes (e.g., p2\_droughts).
  + Name file targets with file-type suffixes (e.g., p1\_nlcd\_drb\_csv).
  + .gitignore the \_targets/ folder.
* [Training materials](https://dsp-manual.wma.chs.usgs.gov/docs/reference/pipeline-training-summary/).

## Other tools

The NHGF Catalog also has a tools section including useful packages, data services, Web Feature Services, and notebooks: <https://water.usgs.gov/usgs/nhgf_catalog/#/tools>

# Section 4 – Data Discovery

Look for data sources that are:

* Efficient (i.e., downloadable and usable at CONUS scale)
* Accessible remotely to us on the project (bonus if publicly accessible)
* Able to take multiple requests to download full set (able to accommodate a reusable pipeline)
* Stable and persistent (for when new drops come out every couple years, new drops update old data)

Good starting points are:

* NHGF Catalog: [nhgf\_catalog (usgs.gov)](https://water.usgs.gov/usgs/nhgf_catalog/#/data)
* Geo Data Portal: [USGS Geo Data Portal](https://labs.waterdata.usgs.gov/gdp_web/)
* OPeNDAP Catalog: [R OpenDap Client • opendap.catalog (mikejohnson51.github.io)](https://mikejohnson51.github.io/opendap.catalog/)
* NHGF STAC – in dev currently, will have almost the same content as catalogs above, but more machine-readable

General notes:

* Avoid data download - use an existing data source co-located with your computer whenever possible
* Prioritize "chunked" data (data that has or can have natural splits) that is conducive to Parallelized workflows if you plan on doing large scale analysis
* add in link to baby STAC: <https://code.usgs.gov/wma/nhgf/stac>

# Section 5 - Data Creation

* Agree upon output formats before starting work. Keep end-user (interested parties) and their prefer censes in mind when developing outputs.
* Recommendations
* Naming conventions:
  + Date: use “date” variable name when there is no time component and format it YYYY-MM-DD
  + Time: use “time” variable name and format HH:MM:SS
  + Datetime: use “datetime” and format YYYY-MM-DD HH:MM:SS (with POSIX you can define time zones, with POSIXlt you can extract different components)
  + Stations/Basins: use “staid” (or “site\_no” or “StaID”?) character vector and make sure leading zeros are not dropped
* Naming conventions:
  + All upper case, Title case?
  + Use “\_” for spaces?
  + E.g., PARENT\_AGENCY, ParentAgency, parent\_agency, parent\_agency\_units?
  + [Best practices](https://prod-is-cms-assets.s3.us-west-2.amazonaws.com/pnamp/prod/d57bac50-7caa-11ed-9fb9-0d1603bac9b4-best_practices_for_data_dictionary_definitions_and_usage_version_1.1_2006-11-14.pdf)

## Data Dictionaries

Having a shared/precise vocabulary is essential for accurately discussing data elements.

* If it exists, we will adopt a documented data dictionary for existing data set as it is not necessary to produce separate documentation and doing so would add confusion.
* We will create and update a data dictionary (in spreadsheet format?) for data elements that are created in the project.
* Where should we put this? Is there a need to compile all data sources? Is there a need to standardize dictionary across data sources? Need to keep this close to the data, but also realize that we may need to combine data sources.
* Here are some possible elements to get us started on a dictionary:
  + Name
  + Description
    - include these elements where applicable: 1) definition, 2) source, calculations, and defaults, 3) type and validation (e.g., allow nulls? T/F), 4) units and precision (e.g., max length), 5) list of values if limited set (e.g., levels on a factor), 6) foreign keys, 7) life cycle (e.g., Do values get updated? How?), 8) use, and 9) comments
    - QC tests that have been performed
  + Source
  + Restrictions
    - Use Limitations
    - Legal Constraints
  + Fees
  + Spatial Reference System
  + Spatial Scope and Scale
  + Temporal Scope and Scale
  + Volume Estimate
  + Acquisition Date
  + Citation
  + Digital Object Identifier

# Section 6 - Workflow Design

* Design workflows to be modular – each piece does a small task (e.g. one task for fetch, one for cleaning, one for aggregation, etc.) so that different components can be stitched together.
  + E.g., [forecasted-met-drivers](https://code.usgs.gov/wma/wp/forecasted-met-drivers)
  + E.g., [HyTEST workflows](https://github.com/hytest-org/hytest) for demonstration of distributed computation on large datasets (without the need for download).
* Draw out your workflow first: You can use this handy Mural template.
  + E.g., [geospatial-attributes-processing](https://app.mural.co/t/gswocooeto6166/m/gswocooeto6166/1674664777393/0c9d8beacaa9c442e27bc5fe8112f05e6deaa68b?sender=uc2098797df19e98c2b2f4081)
* Define processing decisions outside of actual scripts-in a config file-because:
  + It easily surfaces decisions to users and the configuration can be saved along with outputs, so you know how to regenerate the results.
  + Allows for easier adaptation of workflows to make different processing decisions.
  + This could be a .yml file, or at the top of your \_targets.R file.
  + E.g., where we defined several parameters needed for the data pull in a config file, like start/end date, the datum and unit type for the data: [drb-estuary-salinity-ml/params\_config\_fetch\_noaa\_nos.yaml at main · USGS-R/drb-estuary-salinity-ml (github.com)](https://github.com/USGS-R/drb-estuary-salinity-ml/blob/main/01_fetch/params_config_fetch_noaa_nos.yaml)
  + E.g., for data processing: [drb-estuary-salinity-ml/params\_config\_munge\_noaa\_nos.yaml at main · USGS-R/drb-estuary-salinity-ml (github.com)](https://github.com/USGS-R/drb-estuary-salinity-ml/blob/main/02_munge/params_config_munge_noaa_nos.yaml)
* If you are releasing data or software:
  + Use Git releases which zips up your repo and ties it to a certain commit
  + [Guidance](https://dsp-manual.wma.chs.usgs.gov/docs/data_releases/)
  + [Checklist](https://code.chs.usgs.gov/software/software-management/-/raw/main/software-release-checklist.pdf)