Programming Protocol and Exit Strategy

This protocol regulates the essential steps to ensure the reproducibility and sustainability of programming efforts in the NCCR Evolving Language. This also facilitates the publication of research data, as required by the SNSF, where researchers should comply with the FAIR principles.

1. Project Initialization

1.1. Repository Setup

- Use version control (e.g., Git) from the start.
- Store code in a centralized team GitHub/GitLab, if available.
- Follow a standardized directory structure, e.g.:

```
├── data/           # Raw and processed data
├── src/            # Source code
├── notebooks/      # (Jupyter or quarto) notebook or exploratory scripts
├── outputs/        # Generated data and results (e.g., figures, tables, models, static documents)
├── environment/    # Environment configuration (e.g., requirements, .yml, Dockerfile)
├── README.md
└── LICENSE
```

1.2. Write an informative README.md

- Project overview and purpose, research questions, etc, or an abstract if it is published.
- Environment prerequisites and installation.
- Reproduction steps for key results.
- Brief explanation of the repository structure and expected inputs/outputs.

The National Centres of Competence in Research (NCCRs) are a funding scheme of the Swiss National Science Foundation.

2. Coding Standards & Best Practices

2.1. General principles

- Follow a coding style guide (e.g., PEP8 for Python, Tidyverse Style for R).
- Maximally modularize codes to enhance readability and reusability.
- Use meaningful naming for modules/functions/parameters and add essential comments (where non-obvious). Avoid excessive abbreviations of names.
- Use branching to facilitate collaboration and code reviews/testing

2.2. Testing & Validation (optional; recommended for large development projects)

- Include unit tests for core functions and integration tests for pipelines.
- Document test scripts and test data
- Ensure the codes pass tests before any major commits or handoffs.

2.3. Handling Large Data

- Large data can be included in the same git repository of a project using Large File Storage (LFS) to facilitate reproducibility.
- If the data size exceeds the limit of Git LFS, the dataset should be stored in a open repository, with the link or digital object identifier included in the README.
- If a module or an API is available for fetching the large data (e.g., HuggingFace, Zenodo), it is recommended to include a pipeline to automate data fetching.

3. Environment Management

- Use environment management tools, e.g.:
    - conda + environment.yml
    - pip + requirements.txt
    - renv or targets
    - Dockerfile + docker-compose.yml
- Save exact versions of dependencies and make sure updated environments are synced (e.g. run pip freeze > requirements.txt upon changes)

4. Reproducibility Checklist

☐ All key analyses can be rerun following instructions in README (ideally with concrete commands) or with a top-level script (e.g. Makefile).
☐ Dependencies/Environments are documented and locked.
☐ Only relative paths are used (i.e., avoid using absolute paths with personal directories).
☐ Each script/module has sufficient inline documentation (e.g. comments, docstrings for large/core modules) explaining coding strategies and decisions (literate programming)
☐ External data sources (if used) are cited.
☐ Any data restrictions affecting reproducibility should be clarified (in LICENSE or README).

5. Collaboration & Review

- If there are multiple team members in a programming project, pull requests are encouraged to enable reviews between team members.
- Use issues and milestones in the repo to track bugs/features.
- If necessary, contact the TTF DataScience for code reviews. This can take place at any stage of the project.

6. Exit Strategy & Sustainability Plan

6.1. Minimal checklist before handing off or archiving a project

☐ All results are reproducible from instructions in README.
☐ Environment is documented and installable.
☐ Code has comments and inline documentation.
☐ Final outputs (plots, tables, etc.) are stored and labeled, ideally in line with the labels in the corresponding publication.
☐ All scripts and exploratory notebooks are cleaned (e.g., clear unused codes/cells).
☐ All restricted data (e.g., sensitive/personal data, data with restricted copyright) are removed.
☐ All credentials (e.g., passwords, secrets) are removed.

6.2. Exit Options

- Option 1: Submit project to a final audit with the TTF DataScience and, when necessary, schedule a walkthrough session (30–60 min) to explain details.
- Option 2: Create a short “handoff video” or demo notebook explaining usage and have another NCCR member check and approve the contents.

6.3. Archiving Protocol

- Download the repository (e.g. in .zip) and include it in the dataset
    - For Zenodo: tag the final commit, release the project and link it to a Zenodo record
- For large development projects: store the archived environment and the runnable codes/software by exporting to a Docker image.
- Publish the codes/image and data to a recommended open repository (see ORD guidelines of NCCR)

6.4. Special central service hosting

- The TTF DataScience offers to host services centrally (e.g., under a sub-domain of the NCCR Evolving Language website).
- The TTF DataScience fully governs and maintains the services with their contents to ensure the sustainability of the project.
- The archiving protocol as outlined in 6.3 should be strictly adhered to, regardless of whether the service of the project is centrally hosted.

  All services hosted by the TTF DataScience will be turned into nisle.ch after the end of the NCCR funding period to ensure long-term preservation.