The Process and Value of Reprogramming a Legacy Global Hydrological Model

Emmanuel Nyenah[1, 2], Petra Döll[1, 2], Martina Flörke[3], Leon Mühlenbruch[3], Lasse Nissen[1], Robert Reinecke[4]

[1]Institute of Physical Geography, Goethe University Frankfurt, 60438 Frankfurt am Main, Germany
[2]Senckenberg Biodiversity and Climate Research Centre (SBiK-F), 60325 Frankfurt am Main, Germany
[3]Institute of Engineering Hydrology and Water Resources Management, Ruhr University Bochum, 44801 Bochum, Germany
[4]Institute of Geography, Johannes Gutenberg-University Mainz, 55128 Mainz, Germany

Correspondence to: Emmanuel Nyenah (Nyenah@em.uni-frankfurt.de)

Abstract

Global hydrological models (GHMs) improve our understanding of water flows and storage on the continents and have undergone significant advancements in process representation over the past four decades. However, as research questions and GHMs become increasingly complex, maintaining and enhancing existing model codes efficiently has become challenging. Issues such as non-modular design, inconsistent variable naming, insufficient documentation, lack of automated software testing suites, and containerization hinder the sustainability of GHM research software as well as the reproducibility of study results obtained with the help of GHMs. Although some GHMs have been reprogrammed to address these challenges, existing literature focuses on evaluating the quality of model output rather than the quality of the reprogrammed software. To address this research gap and guide other researchers who wish to implement their existing models as sustainable research software, we describe how the most recent version of the GHM WaterGAP was reprogrammed. The success of reprogramming is assessed based on various software sustainability criteria and the FAIR4RS principles—findability, accessibility, interoperability, and reusability—since the primary goal was to improve software sustainability and research reproducibility, rather than model output quality. Following an agile project management approach, WaterGAP was rewritten from scratch in Python with a modular Model-View-Controller architecture. Due to the switch from C/C++ in the legacy code to Python, execution time doubled. Our evaluation indicates that the reprogramming substantially improved the software's usability, maintainability, and extensibility, making the reprogrammed WaterGAP software much more sustainable than its predecessor. The reprogrammed WaterGAP software can be easily understood, applied, and enhanced by novice and experienced modellers and is suited for collaborative code development across diverse teams and locations, fostering the establishment of a community GHM. We outline four lessons learned from the reprogramming process concerning the sustainability-runtime trade-off, the applicability of the agile approach, software design patterns, variable naming, external documentation, and automation.

1 Introduction

Over the past four decades, global hydrological models (GHMs) have made remarkable progress in process representation, such as the incorporation of artificial reservoirs and the differentiation of groundwater and surface water use (Telteu et al., 2021). While the most widely used spatial resolution of GHMs is still 30 arc-minutes (0.5°), the demand for more spatially resolved information has led to GHMs running at 5 arc-minutes (Eisner, 2016; Flörke et al., 2018; Sutanudjaja et al., 2018), 3 arc-minutes (Choulga et al., 2024) or even 30 arc-seconds (Hoch et al., 2023). Still, further progress is required to improve GHMs, e.g., to better represent human-environment interactions and reduce model uncertainties by improved integration of model output observations (Burt and McDonnell, 2015; Döll et al., 2024).

As research questions and, thus, GHMs become more complex, maintaining and further developing an existing model code in an efficient manner becomes increasingly challenging. Similar to other research software, GHMs are developed by scientists with limited software development training, time, and funding, and thus lack the software quality that is required for sustainable research software (Döll et al., 2023). A recent assessment of the software sustainability of global impact models, including nine GHMs, revealed limited accessibility, low adoption of containerized solutions, non-modular design, suboptimal comment density (defined as the number of lines of comment per total lines of code), and the absence of software testing (Nyenah et al., 2024). In addition, poor software quality also hinders the reproducibility of computational research (Döll et al., 2023; Reinecke et al., 2022).

While there are various definitions of sustainable research software (e.g., Anzt et al., 2021; Katz, 2022; Venters et al., 2018), the definition by Anzt et al. (2021) provides clear and measurable qualities that are suitable for evaluating the sustainability of complex research software such as a GHM. Anzt et al. (2021) define sustainable research software as software that 1) is maintainable, extensible, and flexible, i.e., adapts to user requirements, 2) has a defined software architecture, 3) is testable thus ensuring software components function as intended through practices like unit testing, 4) has comprehensive in-code and external documentation, and 5) is freely accessible, i.e., licensed as open source with a digital object identifier (DOI) for proper attribution. In the following, the term "research software" includes the algorithms, source code files, computational workflows, and executables developed during the research process or for a research objective (Barker et al., 2022).

The sustainability of GHMs and the reproducibility of GHM-based research could be significantly enhanced by reprogramming GHMs using modern best practices (Nyenah et al., 2024). These include adopting an open-source license, containerization, implementing a modular architecture, selecting informative variable names, and improving both the density of comments within the code and external documentation (refers to manuals, guides, tutorials, and any materials that provide information about your software to users and developers) (Nyenah et al., 2024). Additionally, applying FAIR (Findable, Accessible, Interoperable, and Reusable) principles for research software (FAIR4RS) improves research software reusability,

reproducibility, as well as transparency (Barker et al., 2022; Wilkinson et al., 2016). For instance, the eWaterCycle (Hut et al., 2022) platform has taken initial steps toward implementing FAIR principles for hydrological models, enabling other researchers to use these models without significant support from the original authors.

Efforts to improve comprehension, usage, maintenance, extension, and collaborative development have led to the reprogramming of several models, including the global land surface model CLASSIC (Melton et al., 2020) and GHMs such as HydroPy (Stacke and Hagemann, 2021) and PCR-GLOBWB (Sutanudjaja et al., 2018). However, the publications on these reprogrammed software focus on evaluating the performance of the model output and lack a detailed account of the reprogramming process and an evaluation of the success of the reprogramming effort.

To address this research gap and support the reprogramming of other legacy software, this paper provides a detailed account of the reprogramming process of GHM WaterGAP (Döll et al., 2003; Müller Schmied et al., 2024) and the characteristics of the new software. Reprogramming aimed to enhance the software's sustainability for long-term research use by a broad community and to increase the reproducibility of the computational research performed with this model. The success of the reprogramming was assessed by comparing the legacy code to the reprogrammed version according to numerous specific sustainability criteria and FAIR4RS principles. It is important to note that our goal in reprogramming WaterGAP was not to improve the model output; the reprogrammed software was to result in the same model output as the latest WaterGAP version 2.2e (Müller Schmied et al., 2024).

The paper is structured as follows: Section 2 introduces the WaterGAP model and the legacy software. Sustainability criteria for research software and methods relevant to this study are presented in Section 3. After describing the reprogramming process in Section 4, we present the architecture and new features of the reprogrammed software in Section 5. In Section 6, we evaluate the new WaterGAP software against selected sustainability criteria and the FAIR4RS principles and share lessons learned for others undertaking similar efforts (Section 7). Our conclusions follow in Section 8.

2 The WaterGAP model

2.1 Model description

WaterGAP is a global-scale water resources and use simulation model that has been developed since 1996 (Müller Schmied et al., 2024). Covering all land areas of the globe except Antarctica, it has been widely used in studies of water scarcity, drought, and ecologically relevant streamflow characteristics, considering the impacts of human water use, reservoirs, and climate change (Müller Schmied et al., 2021) as well as inter-basin transfers (Flörke et al., 2018). Model results have contributed to various reports by the Intergovernmental Panel on Climate Change (IPCC) (Jiménez Cisneros et al., 2014) and the State of Global Water Resources Report by the World Meteorological Organization (WMO) (WMO, 2024). WaterGAP is

also a key participant in the Inter-Sectoral Impact Model Intercomparison Project (ISIMIP) where the focus is on both model evaluation (and or improvement) and the impact assessment of anthropogenic changes such as human water use or climate change (Frieler and Vega, 2019; Heinicke et al., 2024; Warszawski et al., 2014). WaterGAP output is utilized in diverse research fields, e.g., life-cycle assessment (Boulay et al., 2015; Schomberg et al., 2021) or freshwater ecology (Datry et al., 2021; Domisch et al., 2017; Schneider et al., 2017). Furthermore, simulated water storage anomalies were used by geodesists to evaluate GRACE (Gravity Recovery and Climate Experiment) satellite observations of Earth's dynamic gravity field (Kusche et al., 2009; Schmidt et al., 2006), and streamflow estimates were used to analyze thermal and hydropower production (van Vliet et al., 2016; Wan et al., 2022).

WaterGAP exists as two main model families distinguished by their spatial resolutions. WaterGAP 2 operates at a 30 arc-minute resolution, with the latest version being 2.2e (Müller Schmied et al., 2024), while WaterGAP 3 uses a finer 5 arc-minute resolution (Flörke et al., 2018). WaterGAP 2 and 3 also differ with respect to some algorithms and input data. WaterGAP consists of five sectoral water use models for irrigation, livestock, domestic, manufacturing, and cooling of thermal power plants. These water use models are interlinked through the Groundwater Surface Water Use (GWSWUSE), which computes potential net abstractions from groundwater and surface water based on the output of the water use models. These net abstractions are an input to the WaterGAP Global Hydrology Model (WGHM) (Müller Schmied et al., 2021, 2024). All models combined are referred to as the WaterGAP model. WGHM computes both vertical water balance, encompassing the canopy, snow, and soil components, and lateral water balance, which includes groundwater, lakes, artificial reservoirs, wetlands, and rivers. The basin-specific calibration of WGHM distinguishes WaterGAP from other global hydrological models (Müller Schmied et al., 2021). It aims at reducing the bias of simulated streamflow by using a simple method (see Section 5.2 of Müller Schmied et al., 2024) to match the long-term mean annual observed streamflow at 1509 basin outlets, covering 55% of the global drainage area (excluding Antarctica and Greenland). This paper only concerns the reprogramming of the GWSWUSE and WGHM models operating at the 30 arc-minute resolution. The reprogrammed code, however, is flexible enough to also run at higher spatial resolutions if the appropriate inputs are supplied and processes specially tailored to the 30-arcminute resolutions are adapted (e.g., the snow processing routine and the water usage distribution to neighbouring cells).

2.2 Characteristics of the legacy software

The legacy software of WaterGAP was primarily written in C and C++ by PhD students and postdoctoral researchers with diverse programming backgrounds. The software is hosted on a private GitHub repository under the GNU Lesser General Public License (LGPL v3.0). The available model documentation includes two model description papers (Müller Schmied et al., 2021, 2024), as well as a number of documents not available to the public. The latest WGHM version (WaterGAP 2.2e) is archived on Zenodo (https://zenodo.org/records/10026943). However, this version is limited to review only and cannot be executed by external users. The source code of the latest WGHM version contains 25,204 lines of code across 85 files. The linking model GWSWUSE contains 3550 lines of code across 14 files.

In addition to lacking comprehensive and easily accessible external documentation, the sustainability of the legacy software is constrained by several software characteristics. The software has a limited modular structure, consisting of a collection of "script-like" files, with some having up to 6,000 total lines of code (Nyenah et al., 2024). The WGHM code has a non-optimal comment density (approximately 25%) compared to the recommended 30–60% (Nyenah et al., 2024). This makes the model code challenging to read and maintain. In addition, the WaterGAP software uses the non-standard binary file format called UNF for input and output data instead of the now widely used NetCDF format. Climate forcing data, for example, must first be converted to UNF before use. This introduces additional complexity, potentially creating barriers and susceptibility to errors for external users unfamiliar with the format or conversion tools. No unit tests (verifying that individual code components work correctly) exist to check if algorithms produce outputs within acceptable ranges. Furthermore, WaterGAP does not utilize containerization technology, which makes the reproduction of research results more difficult.

3 Methods

3.1 Software evaluation against sustainability criteria and the principles of findability, accessibility, interoperability, and reusability for research software

We assessed research software sustainability using nine indicators from Nyenah et al. (2024; their Table 1), including five indicators of best practices in software engineering and four indicators of source code quality. Table 1 describes each indicator, its rationale, and how we evaluated both the legacy and reprogrammed models against it.

Table 1: Sustainability indicators used for the assessment of the legacy and reprogrammed research software.

No	Indicators	Description
Best practices in software engineering		
1	External documentation	Effective use and ease of software maintainability rely on clear and extensive external documentation (Nyenah et al., 2024; Wilson et al., 2014). We evaluate the availability and extensiveness of external documentation by analyzing the following components: installation guide, tutorials, user guide, reference guide (in-depth descriptions of the model processes and the governing equations), glossary, contributor guide, and frequently asked questions (FAQs).
2	Version control and automation	Version control facilitates change tracking and supports collaboration (Wilson et al., 2014). We evaluate the use of version control considering the choice between public and private repositories, which significantly affects the repository's transparency and accessibility. We also checked the automation

		practices, focusing on automated testing, linting, and documentation to ensure consistent quality and maintainability.
3	Use of an open-source license	We determine the presence of open-source licenses by reviewing license files within repositories and comparing them with licenses approved by the Open Source Initiative (OSI) (https://opensource.org/licenses) (Nyenah et al., 2024).
4	Number of active developers	This indicates the capacity for ongoing software development and maintenance (Nyenah et al., 2024). We measured this by counting individuals who made commits to the codebase of the legacy and the reprogrammed code within the past two years (2023–2024).
5	Containerization	Containerization packages software with its full runtime environment, ensuring consistent execution across different systems (Nüst et al., 2020). This helps overcome reproducibility issues caused by variations in operating systems or dependencies. We simply check whether a containerization solution is provided.
Source code quality		
6	Public availability of an (automated) testing suite	We adopted the approach proposed by Nyenah et al. (2024), in using the public availability of an (automated) testing suite as a proxy for the ability to test software functionality. While test coverage is the ideal metric, current coverage tools do not support Python functions with Numba decorators, which compile Python functions into machine code for performance (GitHub issues, 2025; Lam et al., 2015; Stack Overflow, 2025).
7	Compliance with coding standards	Coding standards are industry best practices that guide software development for consistency and quality (Wang et al., 2008). To assess compliance, we used CLion static analysis for the legacy C/C++ code, which flags issues (including errors, typos, and warnings) based on the C/C++ Core Guidelines but does not provide a score to interpret results. A higher issue count generally indicates lower reliability or maintainability. For the reprogrammed code, we used Pylint to check compliance with PEP-8 conventions. Pylint assigns a score up to 10 for perfect compliance, with no lower bound (Molnar et al., 2020; Nyenah et al., 2024).
8	Comment density	We compute comment density as the ratio of the number of lines of comments to the total lines of code (TLOC). TLOC refers to the sum of source lines of code (SLOC) and comment lines. SLOC, in turn, represents the non-blank, non-

		comment lines within a source file. We regard a comment density of 30% to 60% as optimal (Arafat and Riehle, 2009; He, 2019; Nyenah et al., 2024).
9	Modularity	We evaluate the modularity of the software by the TLOC per file metric, with an ideal range of 10 to 1,000 TLOC per file (Nyenah et al., 2024). This metric reflects the organization of source codes into manageable modules, each focusing on a specific functionality. Modules within this range are typically easier to read, modify, and reuse.

For the FAIR4RS principles (Barker et al., 2022), we evaluated only the reprogrammed model against the eleven core principles. For findability (F), we verified whether the software is assigned a globally unique and persistent identifier (F1) and described with rich metadata (F2). We aimed at ensuring that the metadata included the software identifier (F3) and was searchable and indexable (F4). Regarding accessibility (A), we checked whether the software is retrievable via a standardized protocol (A1) and the metadata remain accessible even if the software were to become unavailable (A2). To evaluate interoperability (I), we examined whether the software reads, writes, and exchanges data following domain-relevant standards (I1) and includes qualified references to other objects (I2). For reusability (R), we checked whether the software is given a clear and accessible license (R1), includes qualified references to other software (R2), and meets community standards (R3).

3.2 Differences between the outputs of the reprogrammed and legacy software

To verify that the reprogrammed software computes the same model output as the legacy software unless an algorithm was changed to improve the computation, we compared the globally averaged water balance components and the global maps of renewable water resources, i.e., the long-term differences between precipitation and actual evapotranspiration in the 30 arc-minute grid cells (Müller Schmied et al., 2021). Both were calculated using the WGHM output of the legacy and the reprogrammed software. It is important to note that WGHM output also reflects the difference in GWSWUSE output as the potential net abstractions computed by GWSWUSE are incorporated into WGHM. The model setup for calculating the water balance components was “anthropogenic” (i.e., considering human water use and artificial reservoir management), with a 5-year spin-up and the simulation period 1901-2019. The water balance analysis focuses on key water balance components and the long-term average volume balance error for five distinct periods: 1961–1990, 1971–2000, 1981–2010, 1991–2019, and 2001–2019. Results for the legacy code are provided in Table 4 of Müller Schmied et al. (2024).

Renewable water resources were calculated over the period 1981-2010 by running the model in the naturalized mode, i.e., without considering human water use and reservoir operations, as detailed in sections 4.7.3 and 7.2.1 of Müller Schmied et al. (2021). It is worth noting that the total water resources value can be negative if the evapotranspiration value in a cell is greater

than the precipitation value, due to inflow from upstream cells. Output differences for many other model output variables were checked during the reprogramming process but are omitted here for clarity.

4 The reprogramming process

The reprogramming process for WaterGAP was enabled by a grant that financed one full-time PhD and a student assistant over three years, in the framework of the ReWaterGAP project. Figure 1 shows the timeline of the reprogramming process, emphasizing the important stages from the project proposal to the release of the reprogrammed software. It also presents the agile project management method we applied in the reprogramming process. Subsections 4.1 to 4.6 provide further details on how these tasks were implemented and managed throughout the project.

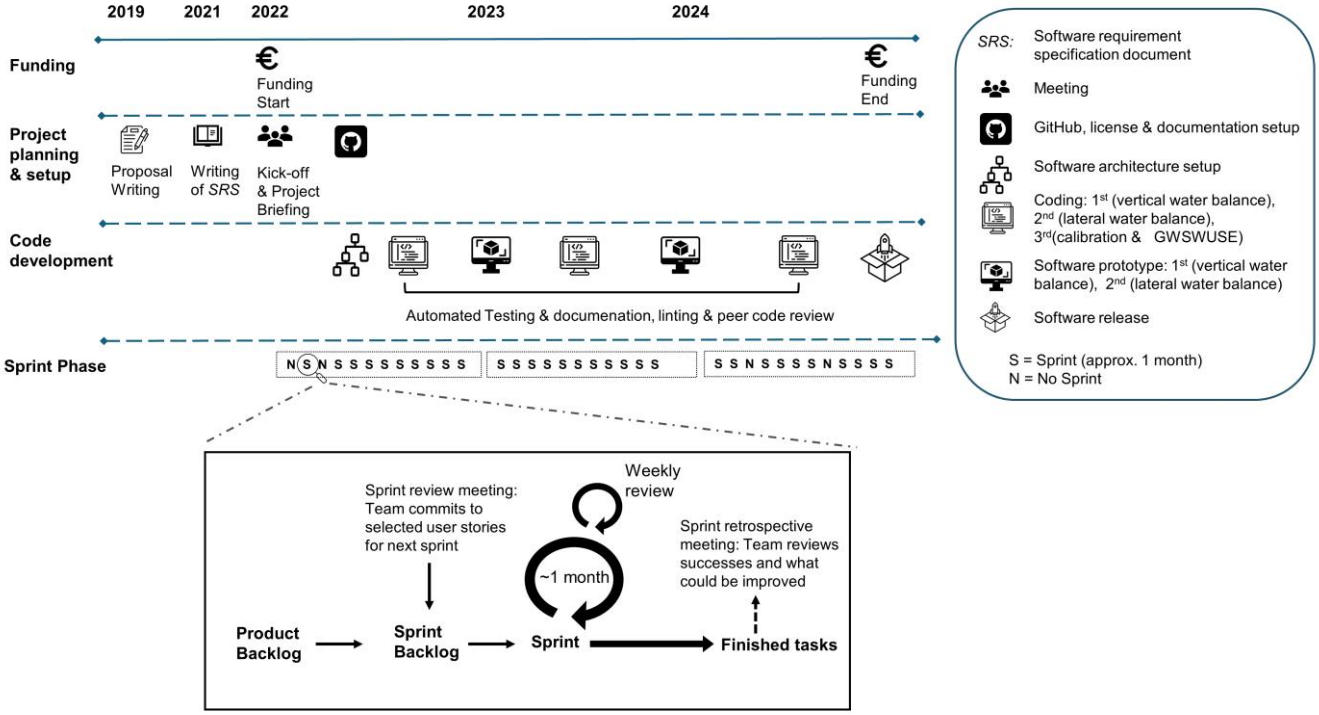


Figure 1: Timeline and agile project management setup for reprogramming of WaterGAP in the ReWaterGAP project.

4.1 Project planning and setup

After the writing and approval of the project proposal, a preliminary meeting was held in November 2021 among six senior developers. These are late-stage PhDs, PostDocs, and Professors with extensive expertise in the WaterGAP model and are also

actively involved in developing and maintaining the software. The goal of the meeting was to draft the software requirement specification document (see software specification document in the Supplement) (Fig. 1), which outlined the technical and functional goals of the project (see section 4.4). This was followed by a kickoff meeting to launch the project officially and a project briefing to align the product owners and development team (see section 4.2) on the project's purpose (comprehensively transforming the GHM WaterGAP into sustainable research software) (Fig. 1). As part of the planning and setup phase, a GitHub repository was established for version control, and a software documentation webpage was created.

4.2 Agile project management

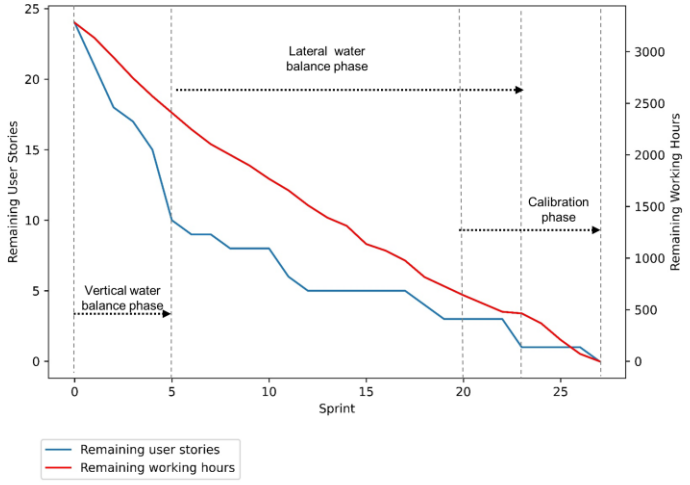
An agile methodology was adopted for software development, dividing the work into iterative sprints (Hema et al., 2020) (Fig. 1). The project spanned 31 sprints, 27 of which were focused on code development (see section 4.5), with the remainder allocated for reading project materials. Each sprint lasted approximately one month, except for one instance in 2023 when a sprint was extended to two months due to task complexity (Fig. 1). We adopted an agile process inspired by the well-known SCRUM method (Hema et al., 2020), which was tailored to the available resources.

The agile team was comprised of two product owners (the two professors leading the WaterGAP model development, Petra Döll and Martina Flörke), three developers (PhD student Emmanuel Nyenah reprogramming WGHM, Master student Lasse Nissen reprogramming GWSWUSE, and student assistant Leon Mühlenbruch writing the external documentation), and the software development advisor (Robert Reinecke). The software development advisor guides the developers on best practices, architecture, and code quality to ensure robust and sustainable software. At the beginning of each sprint, the agile team met to review past progress, with presentation and discussion of completed tasks, and review selected or newly defined user stories (software functionality from the user perspective) which served together with the uncompleted tasks as the basis for the sprint backlog for the next sprint (Fig. 1). User stories were selected from a comprehensive list of user stories and features (product backlog) outlined in the software requirement specification document, which the senior developers wrote before the project started. These user stories were assigned sprint points based on estimated difficulty and time required. A selection of user stories with a combined total of 10 sprint points was then selected to form a sprint backlog. Progress during each sprint was monitored through weekly meetings between the PhD student and the software development advisor, which provided an opportunity to address challenges encountered during the sprint (e.g., improving runtime of snow module). The agile process allowed the team to maintain steady progress toward the project goals and adapt to new requirements and challenges.

4.3 Tracking programming effort and progress

The use of agile project management facilitated the tracking of progress on user stories implementation and the corresponding effort (working hours) during code development. While the reprogramming of GWSWUSE was not included in the initial reprogramming scope, it was later included without the setup of user stories. As a result, progress tracking and effort measurement are only limited to the reprogramming of WHGM (see Excel file in the Supplement). 24 major user stories for

the WGHM were implemented across 27 sprints and organized into three main phases, the programming of the vertical water balance, the lateral water balance and the calibration routine (Fig. 2). The vertical water balance phase involved implementing key WGHM algorithms such as net radiation, Priestley-Taylor evapotranspiration (PET), and processes related to canopy, snow, and soil. A total of 14 user stories were completed over five sprints, requiring 872 working hours (Fig. 2). The lateral water balance phase focused on the implementation of groundwater, lake, and wetland algorithms, among many others. In total, eight user stories were completed during 18 sprints, requiring 1,800 working hours. A substantial portion of this effort (816 working hours across seven sprints) was dedicated to developing the surface and groundwater abstraction algorithm (see section 5). The Calibration phase focused on assessing the global water balance and implementing the calibration scheme. This phase was completed within seven sprints, requiring 608 working hours.



Phase	User stories
Vertical water balance	Create configuration module
	Create climate forcing and static data handling module
	Update configuration module to run different climate change scenarios as input
	Implement net radiation algorithm
	Implement potential evapotranspiration (Priestley-Taylor) algorithm
	Implement canopy algorithm
	Implement snow algorithm
	Implement soil algorithm
	Unit testing
	Code quality check (bugs, code format, variable names, units)
	Peer code review
	Read through ISIMIP quality control tool as a benchmark for the spatial testing tool
	Improve runtime of snow module algorithm
	NetCDF output chunking for large runs
Lateral water balance	Implement groundwater algorithm
	Implement local lakes and wetlands algorithm
	Implement global lakes and wetlands and river algorithm
	Develop a daily snapshot restart feature
	ReWaterGAP-ISIMIP QA tool integration and peer code review
	Implement surface and groundwater abstraction algorithm
	Improve ReWaterGAP runtime, unit testing
	Water use balance assessment
Calibration	Global water balance assessment
	Implement Calibration scheme, unit testing , code review

Figure 2: Cumulative plot of remaining user stories and corresponding working hours over 27 sprints (left), and a table of 24 major user stories (right, see Excel file in the Supplement), grouped by their corresponding phases: vertical water balance, lateral water balance, and calibration.

4.4 Software requirement specification

The software requirement specification document outlines the intended purpose, features, and functionality of the software, providing guidance for the development process (see software specification document in the Supplement). Key elements of the document are described below.

Software architecture and programming language: The selected architectural framework is the Model-View-Controller pattern, which organizes software components into coherent and modular structures with well-defined functionalities (Guaman et al., 2021). In this architecture, the Model component manages the core logic, encompassing hydrological equations and assumptions, and manages data for computed results. The Controller component controls the data flow into the Model and facilitates user interactions (e.g., by a configuration file). The View component presents outputs in various formats tailored to user needs, such as saving output in NetCDF format. Section 5 shows a detailed Model–View-Controller diagram of WGHM, illustrating the class, function, and package interactions. The reprogrammed software is written in Python, chosen for its readability, extensive community support, and a rich ecosystem of libraries (Oliphant, 2007). The Python ecosystem includes packages such as Xarray for handling NetCDF files and NumPy for efficient array computations, as well as tools that enable parallel computing (Harris et al., 2020; Hoyer and Hamman, 2017; Virtanen et al., 2020).

Flexible spatial resolution and restart at prescribed initial state: The new software should allow flexible changes in spatial resolution (30 arc-minutes or 5 arc-minutes). The software should be programmed in such a way that model states, such as storages and parameters, can be saved to disk. A new model run can then be started from this prescribed initial state. This feature has applications for near-real-time monitoring, ensemble forecasts, and data assimilation (e.g., with Parallel Data Assimilation Framework (PDAF))(Müller Schmied et al., 2024).

Data formats and user interfaces: The reprogrammed software should have a configuration file in JSON format, while climate forcing and model outputs are stored in NetCDF format. Using the NetCDF format avoids the need for conversion tools, making the software easier to use and reducing complications. Users interact with the new software through a command-line interface (CLI) and a configuration file that can be used to change the model inputs and outputs.

User stories: The software requirement specification document also outlines what the software should do from the user's perspective, known as the user story (Curcio et al., 2018; Dimitrijević et al., 2015). The use of user stories ensures that the needs and expectations of users are met. An example user story is "*As a user, I want the canopy storage and related fluxes to be functionally implemented, based on the model's description paper of WaterGAP2.2d and 2.2e.*" The requirement specification document includes multiple user stories that capture various functional and non-functional requirements. Functional requirements specify what the software should accomplish (e.g., compute canopy algorithm) (Curcio et al., 2018),

while non-functional requirements are quality constraints such as correctness, reusability, and maintainability that the software must meet (Curcio et al., 2018; Muhammad et al., 2023).

4.5 Best practices for code development

The following best practices were implemented during the code development process in order to ensure software sustainability.

Meaningful and consistent variable names. A critical aspect of the code development process was the establishment of consistent variable names. This was achieved through collaboration between developers, product owners, and the software development advisor. The resulting variable names are descriptive, logical, and uniform across the entire code, significantly enhancing readability and facilitating future maintenance. Examples of variable names can be found on the ReWaterGAP project documentation (Nyenah, 2025c).

Version control and automation. The project uses Git for version control, with GitHub as the platform for hosting the codebase. The source code of the reprogrammed software is open source and licensed under LGPL v3.0. Throughout the development process, bugs identified in the codes through peer code reviews were tracked using GitHub's issue tracking tool, which facilitates transparent tracking of progress and resolution. GitHub Actions were implemented to automate documentation updates, linting, and testing processes. This automation ensures that documentation remains current and maintains code quality through automated quality checks and testing prior to commits, significantly reducing the likelihood of errors in the main codebase.

In-code documentation. Both inline comments and docstrings were used for in-code documentation. Algorithms within the source code were carefully documented with inline comments, explaining the steps and assumptions underlying key processes. Docstrings, on the other hand, are structured comments at the function, class, and module levels that provide a general description of the code's purpose, parameters, and return values (Wiggins et al., 2023). They offer a quick reference for developers interacting with the code and can be extracted by documentation generation tools like Sphinx to create external documentation. The in-code documentation was done with the aim of making the code comprehensible for new developers and easy to maintain over time.

External documentation. In addition to in-code documentation, a comprehensive web-based documentation (Nyenah, 2025c) was generated using GitHub Actions, which facilitates the creation of automated workflows, and the Sphinx library (Sphinx Project, 2025), which is designed for creating well-structured documentation. The automated workflow is set up through a YAML script that utilizes Sphinx library to automate the documentation process. A key feature of Sphinx is its ability to extract docstrings from classes and functions, enabling developers to expand on these docstrings with additional content such as

figures, equations, underlying assumptions, and explanations of solution methods (both analytical and numerical). The Sphinx-generated documentation for the reprogrammed software includes an installation guide, tutorials, a user guide, a reference guide, a contributor's guide, frequently asked questions (FAQs), and a glossary. This web-based documentation is automatically updated in tandem with source code modifications, ensuring that the documentation consistently reflects the current state of the software. Automation for external documentation is currently only available for WGHM and that of GWSWUSE will be added later.

Logging. A logging system was added to the project to help with error handling and debugging. Errors and warnings are recorded in a log file, making it easier to troubleshoot issues. The logging system can be adjusted to different user needs, controlling the amount of detail saved. Additionally, the reprogrammed software can be run in debug mode, showing detailed information on the screen and in the log file. This helps users easily spot and fix problems during use.

Specialized library. Given the computationally intensive nature of WGHM software, minimizing its run time was crucial. To achieve this, the Numba library was used. Numba is a Just-in-Time (JIT) compiler for Python that can significantly enhance performance by converting Python functions into machine code at runtime (Lam et al., 2015). Numba operates using function decorators defined as wrapper functions that inform Numba regarding the specific functions that should be compiled into machine code (Lam et al., 2015).

Containerized solution for WGHM. A Dockerfile following best practices for writing Dockerfile (Nüst et al., 2020) is available to create a containerized environment for running WGHM. This Dockerfile sets up a Python environment with the required packages and clones the source code of the reprogrammed software during the build of a Docker image (executable file). Once the image is built, it can be run (the running instance of the image is known as a container (Nüst et al., 2020)). A simple tutorial on running the WGHM Docker container can be found in the ReWaterGAP project documentation (Nyenah, 2025c).

4.6 Quality assurance

Automated unit testing. Unit testing is a software development process in which individual pieces of code (units) are tested to ensure they function as expected (Pajankar, 2022). If the code is changed at a later stage, e.g., as new features are added, the automated execution of the test reduces the likelihood that something is “broken” without the developers noticing it. Furthermore, tests can help to develop new software features according to a specification, also called test-driven development, in which tests are written before implementing the software component (George and Williams, 2004). The code to test a function is called a unit test or test case, while a collection of test cases forms a test suite. Unit testing was conducted for the reprogrammed WGHM and GWSWUSE components using the Python unittest framework (Pajankar, 2022; Python, 2025). An example of unit tests written for the canopy storage module (Nyenah, 2025e) contains

1. a setup function (lines 25-44) that generates randomly plausible input data for testing and stores plausible minimum and maximum daily benchmark values (Müller Schmied et al., 2021),
2. a first unit test (Lines 47–76) which runs the canopy storage module for one day and compares this result against the benchmark, and
3. a second unit test (79-120) to verify whether the canopy storage module raises an error message when it encounters negative precipitation values in a grid cell.

The complete suite of tests for the reprogrammed WGHM software and that of the reprogrammed GWSWUSE are publicly available on GitHub (Nissen, 2025b; Nyenah, 2025d). To automate the execution of test suites, a GitHub Actions workflow has been created using a YAML script. An example script is found on Github (Nyenah, 2025b) . Automation for testing is currently only available for WGHM, while automated testing of GWSWUSE will be added later. To ensure the testing of model functionality, new tests must be added to the existing test suites whenever new process algorithms are introduced. Expanding on existing tests, for example, to test additional edge cases, is the first task we recommend when onboarding new developers into an existing project.

Peer code review. To enhance code quality, three hackathon-style peer code review sessions were organized. During these events, eight WaterGAP developers examined the WGHM codebase, executed the software, and actively sought out bugs. These sessions also evaluated the clarity of external documentation, ensuring that it was comprehensible and user-friendly. In addition, several weekly meetings involving developers and the software development advisor were dedicated to code reviews.

Linting. To maintain consistency and readability across the code, the reprogrammed software code was checked against PEP-8 conventions, which define the style guidelines for Python code (van Rossum et al., 2001) . The Python library Pylint was employed to assess the code for potential bugs and deviations from these conventions. Linting is also automated and an example script is available on GitHub (Nyenah, 2025a). Automation for linting for is currently only available for WGHM.

Comparison of legacy and reprogrammed software output. We verified that the reprogrammed software produces similar outputs to the legacy software, as demonstrated in the global water balance and analysis of renewable water resources (see Appendix).

5 Architecture and new features of the reprogrammed software

Software architecture defines how different components of the software interact with each other (McConnell, 2004). When components are designed to be modular, they form a coherent structure with well-defined functionality. This makes it easier to extend, modify, and test individual components. As a result, good architecture helps improve software quality and long-term maintainability.

In Figure 3, the Model-View-Controller (MVC) architectural pattern for the reprogrammed WGHM software is shown in its low-level implementation. (Gamma et al., 1994; Guaman et al., 2021). The run_watergap module coordinates the entire model workflow. It manages process initialization and daily time-stepping to perform computations. The Controller package is responsible for handling configuration-related tasks. The *Config_handler()* function processes information from configuration files, which includes paths to input data such as climate forcing and static datasets, as well as runtime settings like the simulation period and the type of run (e.g., naturalized or anthropogenic). This information is passed into classes such as *ClimateForcing* and *StaticData*, which read the specified input files and prepare the data for use in the model. For instance, the *ClimateForcing* class accesses and processes precipitation and temperature data, while *StaticData* processes data for land cover and other static variables.

The Model package implements all hydrological processes organized into vertical and lateral water balance components. In the reprogrammed WGHM, each storage compartment is designed as a separate Python module. The *VerticalWaterBalance* class coordinates hydrological processes such as the calculation of net radiation and potential evapotranspiration (PET) using the Priestley-Taylor algorithm (although other PET schemes can be easily incorporated), canopy, snow, and the soil water balance. The class uses its *calculate()* function to manage these computations and obtain the resulting storages and fluxes through the *get_storages_and_fluxes()* function. Also, the *LateralWaterBalance* class addresses horizontal water movements via *calculate()* function which further calls the *river_routing()* function. This involves the simulation of storage compartments like groundwater, lakes and wetlands, reservoir-regulated water bodies, and rivers. It similarly retrieves all associated storages and fluxes through its *get_storages_and_fluxes()* functions. Model parameters in NetCDF format are also processed here. The NetCDF format not only facilitates easy visualization of parameter distribution but also enables convenient parameter modification using libraries like Xarray (Hoyer and Hamman, 2017).

The View package processes the outputs generated by the Model. It extracts storages and fluxes through the *get_storages_and_fluxes()* functions of the *VerticalWaterBalance* and *LateralWaterBalance* classes. Outputs are then converted to base units and saved as NetCDF files. NetCDFs are enriched with metadata, which comply with ISIMIP conventions (ISIMIP, 2025).

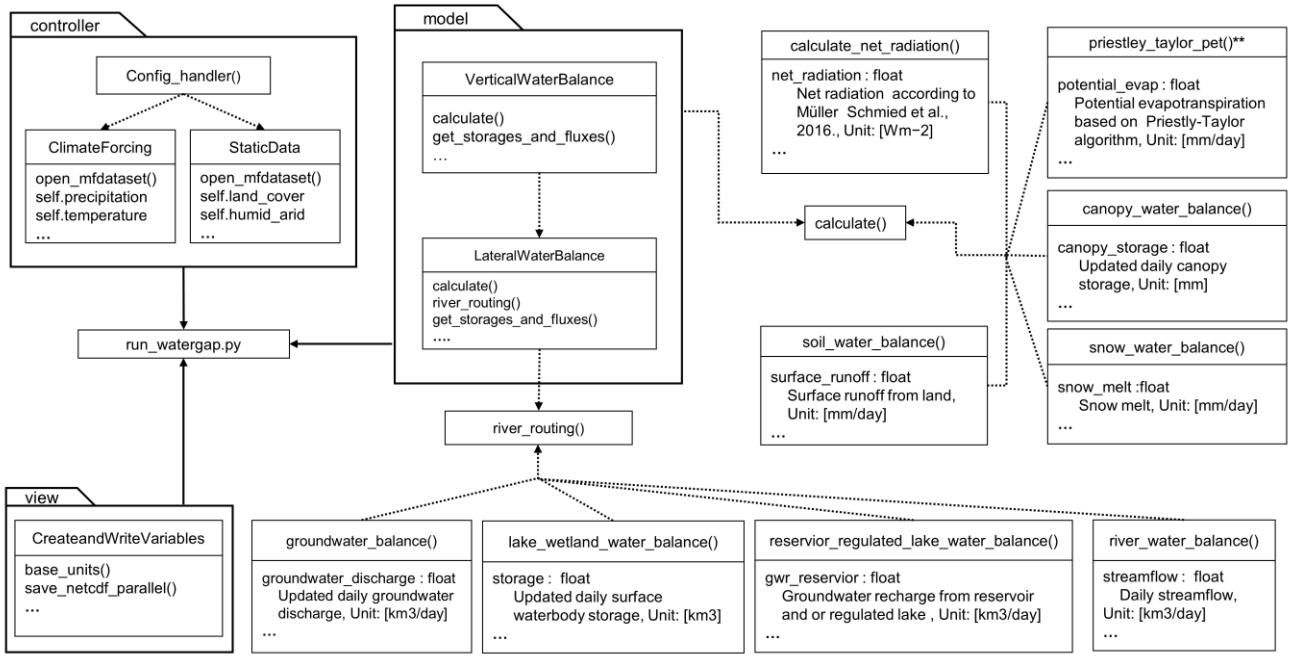


Figure 3: Model-View–Controller (MVC) architectural pattern of the reprogrammed WGHM software at the package, class, and function levels. The Controller package manages the configuration and input data (e.g., climate time series and static data), the Model package contains core hydrological processes, and the View package handles the saving and presentation of model outputs in NetCDF format. Classes are represented with capitalized names, and functions are denoted by lower-case names ending with parentheses.

As a new feature in the reprogrammed WGHM, we revised the algorithm governing surface water demand satisfaction and its impact on return flows to groundwater. The legacy code lacked sufficient in-code and external documentation to enable code comprehension. After discussing the underlying conceptual model with the product owners, we developed an improved and consistent algorithm. For more details about the new abstraction algorithm, readers can refer to WGHM model documentation (Nyenah, 2025c).

The reprogrammed GWSWSUE component follows a similar MVC architecture (see Fig. S1). The source code is available on GitHub (Nissen, 2025a). As part of the reprogramming process, several new features were added to the GWSWUSE. The reprogrammed software includes the modification of model equations to enable the calculation and write-out of potential sectoral net abstractions from groundwater and surface water, and the sectoral return flows to groundwater and surface water,

while the legacy GWSWUSE code only provided total net abstractions from groundwater and surface water. Moreover, additional optional model settings were added in the reprogrammed GWSWUSE that enable an improved and updated modelling of irrigation water use, including both water abstractions and consumptive use (i.e. abstracted water that evapotranspirates during use (Müller Schmied et al., 2021).

Most importantly, the reprogrammed GWSWUSE can now optionally handle the input of a new variant of the irrigation water use model GIM (Müller Schmied et al., 2021) that uses an updated dataset of the time series of area equipped for irrigation (AEI) in each grid cell to compute the consumptive irrigation water use on the AEI. It then implements recent information on country values of area actually irrigated (AAI) and AEI available from AQUASTAT (https://www.fao.org/aquastat/) for the time period 1964-2020 to compute consumptive irrigation water use on the AAI since 1901. While the old gridded AEI dataset covered the period from 1900 to 2005 (Siebert et al., 2015), the new gridded data incorporates new data for 2000 to 2015 (Mehta et al., 2024). Consumptive irrigation water use on AAI in the period 1901-2015 is computed by multiplying the gridded output of GIM by the country-specific ratio of AAI-to-AEI, while results for the period 2016-2020 are computed in reprogrammed GWSWUSE by multiplying the AAI-to-AEI ratio for 2015 by the ratio of AAI in the specific year to the AAI in 2015. Irrigation after 2020 is handled in the new GWSWUSE like 2020. (Müller Schmied et al., 2021). Another newly included option is an alternative computation of irrigation water abstractions from groundwater. Instead of a globally valid water use efficiency of 0.7, the user can select that the water use efficiency for irrigation with groundwater is not less than the country-specific water use efficiency for irrigation with surface water. For more details on these new features and the overall functionality of the new GWSWUSE software, please refer to the external documentation (Nyenah, 2025c) and Fig. S2.

6 Evaluation against sustainability criteria and the principles of findability, accessibility, interoperability, and reusability for research software

The reprogrammed WGHM and GWSWUSE software demonstrate significant improvements in software engineering practices and source code quality compared to the legacy software. They include a more comprehensive external documentation, which was absent in the legacy software (Table 2). 46 out of 64 participants of a user survey on the reprogrammed WGHM agreed (with 32 out of 64 strongly agreeing) that the provided external documentation clearly explained the code (Fig. S7). While the legacy and reprogrammed software use GitHub for version control, the latter is publicly accessible and includes automation (currently for WGHM) for documentation, testing, and linting (Table 2). The reprogrammed software provides containerization (currently for WGHM), which was unavailable for the legacy software (Table 2). Regarding active development over the past two years, both the reprogrammed and legacy WGHM software have seen ongoing development, with three active developers working on the reprogrammed version and four developers maintaining the legacy version. Research projects continue to rely on the legacy WGHM codebase; thus, development activities are expected to continue until

a smooth transition to the reprogrammed software is achieved. In contrast, no active development has occurred for the legacy GWSWUSE model in the past two years.

Regarding source code quality, the reprogrammed WGHM includes a publicly available automated testing suite, which ensures components of the software function as intended (Table 2). The reprogrammed software programs comply with Python PEP-8 coding standards, with a Pylint score of 9.40 (out of 10) for WGHM and 9.65 for GWSWUSE. In contrast, the legacy software contains several warnings, typos, and errors when evaluated against C/C++ Core guidelines, leading to potential issues like poor code readability and difficulty in maintenance (Table 2). Comment density has improved for WGHM from 21% in the legacy software to 47% in the reprogrammed software, improving readability and enabling easier maintenance (Table 2, Fig. S3a). This aligns with the user survey evaluating the reprogrammed WGHM's Priestley-Taylor PET code snippet, which indicated high levels of code readability and modifiability (see section S2 in the Supplement)). However, the comment density in GWSWUSE decreased from 50% to 26%, even though it was sufficient for code comprehension. This decline is partly because developers in the legacy version of GWSWUSE recorded file history in the headers, which increased the number of comment lines. Based on the modularity metric, the legacy software programs include several files that exceed the recommended range of 10–1,000 TLOC per file (see Fig. S3b). The reprogrammed software has a modular structure, keeping TLOC per file within the recommended limits (see Fig. S3b).

Table 2: Sustainability indicators for the legacy and reprogrammed WGHM Software

No.	Indicator	Legacy WGHM & GWSWUSE	Reprogrammed WGHM & GWSWUSE
Best practice in software engineering			
1	External documentation	No	Yes
2	Version control and automation	Yes, GitHub (private), No automations available	Yes, GitHub (public), Automation for documentation, testing, and linting (currently for WGHM)
3	Open-source license	LGPLv3	LGPLv3
4	Number of active developers	WGHM = 4, GWSWUSE = 0	WGHM = 3, GWSWUSE =1
5	Containerization	No	Yes (currently for WHGM)
Source code quality			
6	Public availability of an automated testing suite	No	Yes (currently only for WGHM)

7	Compliance with coding standards	Several code violations: WGHM (~280 warnings, ~3600 typos, 140 errors), GWSWUSE (~70 warnings, ~860 typos, 5 errors)	Yes, WGHM Pylint score = 9.40/10 , GWSWUSE Pylint score = 9.65/10
8	Comment density	WGHM = 21%, GWSWUSE = 50%	WGHM = 47%, GWSWUSE = 26%
9	Modularity	No	Yes

The reprogrammed software aligns with the eleven main FAIR4RS principles. It has a versioned DOI from Zenodo (FAIR4RS principle F1, Barker et al., 2022), along with rich metadata such as web-based documentation (F2) that includes the DOI (F3). Metadata is searchable and indexable (e.g., via a search engine) (F4). The software can be downloaded from both the GitHub repository and Zenodo (A1), and the metadata will remain accessible even if the software becomes unavailable on Zenodo (A2). The software uses data types (NetCDF) for input, output, and data exchange that are widely used in the global hydrological and impact model community (I1). The software includes qualified references to other objects (e.g., climate forcing data) (I2). It is published under a Lesser General Public License v3.0 (R1). The code also includes qualified references to other software, such as various Python libraries (e.g., Xarray, Numpy, Numba) (R2), and the software meets domain-relevant community standards (e.g., variable naming convention from ISIMIP) (R3).

7 Lesson learned

When reprogramming WaterGAP, we made four key observations that we hope can guide others in their efforts to improve the sustainability of their research software.

7.1 Software sustainability and runtime trade-off

A more sustainable software may have negative runtime consequences in the short term. Considering sustainable research software indicators and the FAIR4RS principles in the reprogramming of the legacy code has enhanced the software quality, extensibility, reproducibility, and long-term sustainability of WaterGAP. Unfortunately, the transition from C/C++ in the legacy software to Python, an interpreted language, has approximately doubled the WGHM runtime. This is to be expected as numerical computations in Python can be 3-10 times slower compared to C/C++ (Cai et al., 2005). The average runtime for a standard run on an AMD EPYC 7543 processor with 3.7 GHz is about 7-8 minutes per simulated year for the reprogrammed software, compared to 3-4 minutes for the legacy software. Considering that this may lead to critical runtime-related constraints for model calibration and ensemble methods, such as those used for sensitivity analysis and ensemble forecasts, is the choice of Python justifiable?

To reach this runtime, we already utilized the optimization library Numba, which compiles parts of the Python code. Python is generally slower in terms of runtime performance compared to C/C++ since it uses interpretation instead of compilation (Cai et al., 2005). Compiled code is translated into machine code by a compiler before execution, resulting in a standalone executable file that can be run directly by the processor. On the other hand, interpreted code is executed line by line by an interpreter during runtime, meaning the code must be interpreted every time it is run. Compiled code generally executes faster but often requires a separate compilation step and may be less portable. In contrast, interpreted code is typically more portable but executes more slowly. The pure Python implementation of the GHM HydroPy model is three times slower than the version with a routing scheme written in Fortran (Stacke and Hagemann, 2021).

However, Python generally produces more readable, less error-prone, and more maintainable code than C++, primarily due to its simpler syntax, dynamic typing, automatic memory management, and higher-level abstractions (Balreira et al., 2023; Johnson, 2025; Prechelt, 2000). These features reduce the likelihood of errors and allow developers to express complex ideas more concisely. Python's extensive standard library and ecosystem further enhance maintainability by reducing the need for custom code. In contrast, C++'s more complex syntax and manual memory management can lead to more errors and harder-to-maintain code. Most scientists lack the necessary skills to produce high-quality C++ code and are unlikely to follow any best practices (Reinecke et al., 2022). We believe that the benefits of Python regarding code quality outweigh the runtime increase. The switch from C/C++ to Python makes it easier for scientists, particularly those with restricted programming experience, to understand, modify, extend, and maintain a complex model. Slow code can always be made fast with better hardware, but hardware cannot fix bad code and unsustainable software.

7.2 Agile process benefits in academic settings

Agile principles offer significant benefits in academic software development. The agile development process, along with the use of user stories, was essential to our reprogramming effort, enabling iterative improvements through continuous feedback. Specifically, Agile supports flexibility in incorporating evolving research questions and enables effective progress tracking. For example, tracking progress allowed us to monitor the number of user stories completed within each sprint, the time invested, and the remaining tasks to be tackled in the upcoming sprint. Such tracking helped us assess whether we were on track to complete the overall project within the required timeframe. Tools such as task boards and backlogs provide transparency and help manage workflows efficiently. This is particularly important in academic settings where timelines are often constrained and team composition can change frequently.

Agile's emphasis on regular communication helps align the efforts of diverse contributors, including students, researchers, and supervisors (the "project owners"), ensuring everyone stays informed and coordinated throughout the project. User stories helped ensure that the software features matched the scientific requirements of WaterGAP. Through sprint reviews and retrospective meetings (see Fig. 1), we collaboratively reviewed various user stories to assess whether changes in

conceptualization and hence algorithms are needed. For example, we revised the algorithm governing surface water demand satisfaction due to the limited documentation available. During these meetings, we also discuss efficient technical solutions for some user stories, such as improving the runtime of the snow module and enhancing the overall runtime of the WGHM software. These discussions not only enabled conceptual alignment but also allowed us to find efficient technical solutions, incorporating input from product owners and the software development advisor. Despite these advantages, we faced several challenges. Estimating the time needed to complete user stories proved difficult, and coordinating an agile process with only a few developers in an academic setting was somewhat challenging. Nevertheless, we recommend this approach for other reprogramming projects, as it supports timely and user-focused development and helps the team stay updated on progress.

7.3 Code architecture and design practices

Following best practices in software design to develop a software architecture is pivotal for sustainable software. Defining software architecture and its modular design is an iterative process that benefits greatly from the input of software experts. Architectural decisions play a critical role in determining how easily a model can be extended or modified without affecting other software components. For example, implementing each storage compartment as an independent Python module enabled targeted test development and comprehensive testing before integration. Guidance on software design patterns can be found, for example, in Gamma et al. (1994). A modular design also leads to improved readability, as single components of a project (e.g., code files) are more concise in their purpose.

Furthermore, good software engineering practice can further improve readability and maintainability, such as establishing meaningful and consistent variable names, which is also an iterative process that requires collaborative effort among developers and domain experts. For large projects, it is common for different developers to use various names for the same underlying concept, which can lead to confusion and subtle bugs if only one instance of a variable is updated (McConnell, 2004). For example, suppose a variable is referred to as both “storage_canopy” and “canopy_storage” in different parts of the code. In that case, an update to one variable may not automatically propagate to the other, resulting in inconsistencies in model output. Importantly, naming conventions need to be documented and enforced through the product owners and the active development team to guide future model development and reduce the risk of errors associated with ambiguous or inconsistent variable names.

7.4 Documentation and automation during development

Documentation and process automation during software development lead to more sustainable software. Without sufficient documentation, it is challenging to comprehend the underlying concepts of algorithms and to modify, extend, or utilize the resulting software. An example is highlighted in Section 5, where we revised the algorithm governing surface water demand satisfaction and its impact on return flows to groundwater. The legacy code lacked sufficient in-code and external documentation, making it hard to understand and to re-implement. As a result, we developed an improved and consistent

algorithm, accompanied by in-depth documentation. Throughout the project, we did not copy old documentation from the previous model; instead, we wrote it from scratch, keeping in mind the sustainability of the new software. We strongly recommend writing model documentation alongside code development rather than leaving it until the end. This approach helps capture critical assumptions, such as those embedded in algorithms, while they are still fresh in the developers' minds. Peer review of documentation improves its quality and clarity.

Manually updating documentation, running tests, and linting can be time-consuming, especially for large software projects. Developers may even forget to perform these tasks after modifying or extending the software, which can lead to buggy or broken code. Automating documentation generation reduces manual effort and helps keep the documentation up to date. Similarly, automating linting and testing ensures that the code functions correctly without the need for constant manual checks. Automation is key to efficient development and high software quality, and we strongly recommend adopting this practice.

8 Conclusion

This study details how the legacy software of the state-of-the-art global hydrological model, WaterGAP, was reprogrammed to create a sustainable research software that can be efficiently applied and enhanced within the original developer group and a broader research community. The reprogrammed software has undergone extensive quality control, improving its reliability and the transparency of model assumptions. A new modular structure, combined with the use of the Python programming language, has significantly enhanced readability, modifiability, and extensibility. In addition to the improved software quality, comprehensive documentation, containerization, and the use of standardized input and output formats make it more accessible to users with different levels of expertise. The open-source nature of the software also facilitates comparison of algorithms, consistency checks, and error detection, ultimately supporting the advancement of hydrological sciences. Ultimately, the reprogrammed WaterGAP software is expected to facilitate scientific studies that are more reproducible than those conducted with the legacy software.

With the reprogrammed WaterGAP software, interested researchers can now be taught at conferences or summer schools about how to apply and improve the software, thus expanding the WaterGAP community and advancing global hydrological modelling. Moreover, the reprogrammed software, with its improved modularity, can serve as a teaching tool for Bachelor's and Master's students, helping them learn how to write and modify algorithms or incorporate new data, thereby enhancing their understanding of the global hydrological cycle.

Appendix

Differences between the outputs of the reprogrammed and legacy software

Globally aggregated water balance components ($km^3\,yr^{-1}$) are shown for the reprogrammed version and the legacy code in Table A1. As expected, the output of both WGHM versions is very similar. The most notable changes are in the actual net abstraction from surface water and groundwater, which can be attributed to the implementation of a consistent water abstraction algorithm in WGHM and small variations in the outputs of the reprogrammed GWSWUSE compared to the legacy version. Additionally, the use of a new minimization algorithm for parameter calibration, the "Powell method" from SciPy (Virtanen et al., 2020), has contributed to the overall variations in the water balance components (see Fig. S4 for the variations in calibrated parameters contributing to variation in water balance components).

Table A1: Global-scale (excluding Antarctica and Greenland) water balance components ($km^3\,yr^{-1}$) for the reprogrammed and legacy WaterGAP global hydrological models, driven by the climate forcing data from gswp3-w5e5. Long-term average volume balance error is calculated as the difference between component 1 and the sum of components 2, 3, and 8. Values without parentheses correspond to the reprogrammed WGHM, while values in parentheses refer to the legacy WGHM.

No.	Component	1961-1990	1971–2000	1981–2010	1991–2019	2001–2019
1	Precipitation	110637 (110637)	111279 (111279)	111350 (111350)	111574 (111574)	111655 (111655)
2	Actual evapotranspiration	71404 (71325)	71839 (71755)	71904 (71816)	72091 (71998)	72158 (72063)
3	Streamflow into oceans	39222 (39295)	39454 (39530)	39506 (39584)	39584 (39666)	39614 (39697)
4	Inflow into inland sinks	774 (776)	793 (794)	794 (795)	840 (841)	845 (846)
5	Actual consumptive water use	909 (904)	1055 (1049)	1203 (1195)	1316 (1307)	1379 (1369)
6	Actual net abstraction from surface water	1000 (1036)	1140 (1186)	1282 (1338)	1386 (1448)	1435 (1501)
7	Actual net abstraction from groundwater	-91 (−132)	-85 (−137)	-79 (−143)	-70 (−141)	-56 (−132)
8	Change in total water storage	11 (17)	-14 (−6)	-59 (−49)	-102 (−91)	-117 (−105)
9	Long-term average volume balance error	0.09 (−0.46)	0.10 (−0.34)	0.11 (−0.20)	0.10 (−0.08)	0.10 (−0.07)

The differences between the grid cell values of renewable water resources between the reprogrammed software (Fig. A1, a) and the legacy software are small. For 98 % of the global land area, the difference remains within ±10% (Fig. A1, b). More specifically, 72.3% of the global land area has renewable water resources that differ within ±1 % while 25.7% of the global land area falls within the range of ±1% and ±10% (Fig. A1, b). Only 0.09% of the global land area shows relative difference exceeding ±100%. The differences between the legacy and reprogrammed versions for renewable water resources are only due to variations in calibration parameters

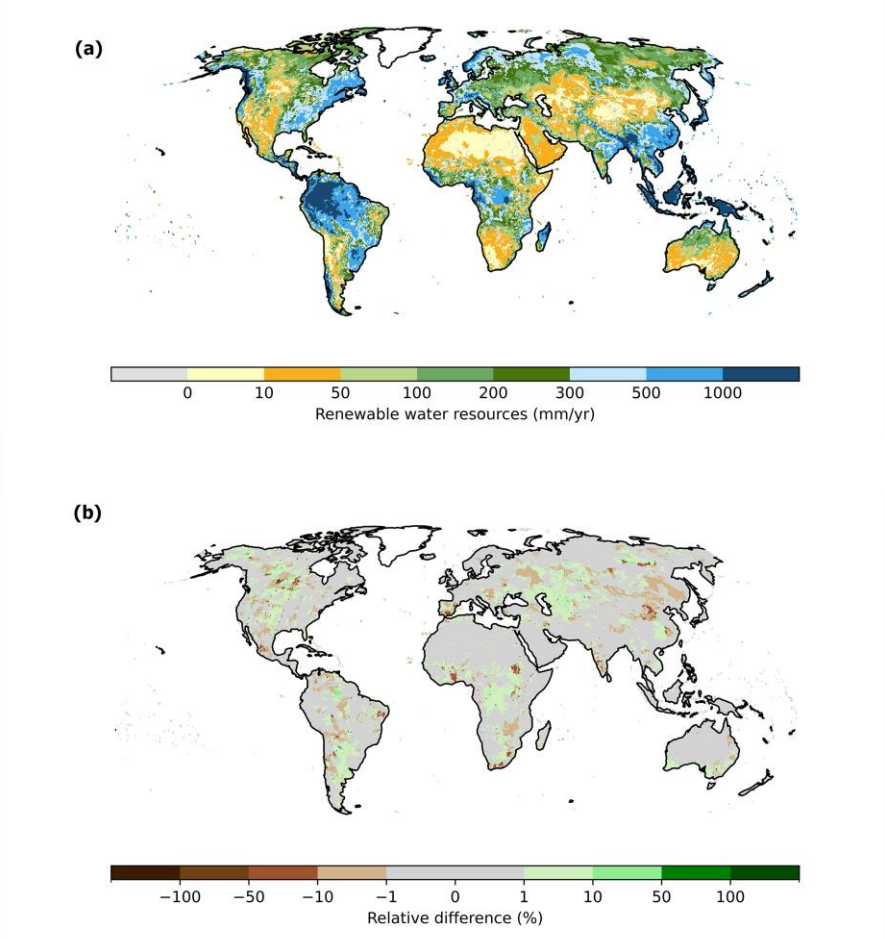


Figure A1: Total renewable water resources [mm yr^{-1}] for the period 1981–2010 computed by the reprogrammed WaterGAP model (a), percent differences of computed total renewable water resources between the reprogrammed and legacy WaterGAP model for the period 1981–2010. Positive values in b) indicate that the legacy WGHM estimates higher renewable water resources than the reprogrammed WGHM.

Code and data availability

The reprogrammed WGHM model source code as well as Docker file can be found at https://github.com/HydrologyFrankfurt/ReWaterGAP. An archived release of the reprogrammed WGHM is also made available on Zenodo (https://doi.org/10.5281/zenodo.14988011) (Nyenah et al., 2025a). The source code of the new GWSWUSE is available on GitHub (https://github.com/HydrologyFrankfurt/ReGWSWUSE.git) as well as Zenodo (https://doi.org/10.5281/zenodo.14988011) (Nyenah et al., 2025a). External documentation for both source codes can be accessed via (https://hydrologyfrankfurt.github.io/ReWaterGAP/). The Python scripts utilized for analysis are available on Zenodo (https://doi.org/10.5281/zenodo.14988257) (Nyenah et al., 2025b).

Author contributions

EN, PD, and RR designed the study. EN performed the analysis and wrote the paper with significant contributions from PD, MF, LM, LN and RR. RR and PD supervised EN.

Acknowledgments

The study was supported by a grant of the Deutsche Forschungsgemeinschaft (DFG) (grant no. 443183317). We thank Linda Söller and Laura Müller for their valuable advice on the development of the user survey

References

Anzt, H., Bach, F., Druskat, S., Löffler, F., Loewe, A., Renard, B., Seemann, G., Struck, A., Achhammer, E., Aggarwal, P., Appel, F., Bader, M., Brusch, L., Busse, C., Chourdakis, G., Dabrowski, P., Ebert, P., Flemisch, B., Friedl, S., Fritzsch, B., Funk, M., Gast, V., Goth, F., Grad, J., Hegewald, J., Hermann, S., Hohmann, F., Janosch, S., Kutra, D., Linxweiler, J., Muth, T., Peters-Kottig, W., Rack, F., Raters, F., Rave, S., Reina, G., Reißig, M., Ropinski, T., Schaarschmidt, J., Seibold, H., Thiele, J., Uekermann, B., Unger, S., and Weeber, R.: An environment for sustainable research software in Germany and beyond: current state, open challenges, and call for action [version 2; peer review: 2 approved], F1000Research, 9, https://doi.org/10.12688/f1000research.23224.2, 2021.

Arafat, O. and Riehle, D.: The comment density of open source software code, in: 2009 31st International Conference on Software Engineering - Companion Volume, 195–198, https://doi.org/10.1109/ICSE-COMPANION.2009.5070980, 2009.

Balreira, D. G., Silveira, T. L. T. da, and Wickboldt, J. A.: Investigating the impact of adopting Python and C languages for introductory engineering programming courses, Computer Applications in Engineering Education, 31, 47–62, https://doi.org/10.1002/cae.22570, 2023.

Barker, M., Chue Hong, N. P., Katz, D. S., Lamprecht, A.-L., Martinez-Ortiz, C., Psomopoulos, F., Harrow, J., Castro, L. J., Gruenpeter, M., Martinez, P. A., and Honeyman, T.: Introducing the FAIR Principles for research software, Scientific Data, 9, 622, https://doi.org/10.1038/s41597-022-01710-x, 2022.

Bierkens, M. F. P.: Global hydrology 2015: State, trends, and directions, Water Resources Research, 51, 4923–4947, https://doi.org/10.1002/2015WR017173, 2015.

Boulay, A.-M., Bare, J., De Camillis, C., Döll, P., Gassert, F., Gerten, D., Humbert, S., Inaba, A., Itsubo, N., Lemoine, Y., Margni, M., Motoshita, M., Núñez, M., Pastor, A. V., Ridoutt, B., Schencker, U., Shirakawa, N., Vionnet, S., Worbe, S., Yoshikawa, S., and Pfister, S.: Consensus building on the development of a stress-based indicator for LCA-based impact assessment of water consumption: outcome of the expert workshops, Int J Life Cycle Assess, 20, 577–583, https://doi.org/10.1007/s11367-015-0869-8, 2015.

Burt, T. P. and McDonnell, J. J.: Whither field hydrology? The need for discovery science and outrageous hydrological hypotheses, Water Resources Research, 51, 5919–5928, https://doi.org/10.1002/2014WR016839, 2015.

Cai, X., Langtangen, H. P., and Moe, H.: On the Performance of the Python Programming Language for Serial and Parallel Scientific Computations, Scientific Programming, 13, 619804, https://doi.org/10.1155/2005/619804, 2005.

Choulga, M., Moschini, F., Mazzetti, C., Grimaldi, S., Disperati, J., Beck, H., Salamon, P., and Prudhomme, C.: Technical note: Surface fields for global environmental modelling, Hydrology and Earth System Sciences, 28, 2991–3036, https://doi.org/10.5194/hess-28-2991-2024, 2024.

Curcio, K., Navarro, T., Malucelli, A., and Reinehr, S.: Requirements engineering: A systematic mapping study in agile software development, Journal of Systems and Software, 139, 32–50, https://doi.org/10.1016/j.jss.2018.01.036, 2018.

Datry, T., Allen, D., Argelich, R., Barquin, J., Bonada, N., Boulton, A., Branger, F., Cai, Y., Cañedo-Argüelles, M., Cid, N., Csabai, Z., Dallimer, M., Araújo, J. C. de, Declerck, S., Dekker, T., Döll, P., Encalada, A., Forcellini, M., Foulquier, A., Heino, J., Jabot, F., Keszler, P., Kopperoinen, L., Kralisch, S., Künne, A., Lamouroux, N., Lauvernet, C., Lehtoranta, V., Loskotová, B., Marcé, R., Ortega, J. M., Matauschek, C., Miliša, M., Mogyorósi, S., Moya, N., Schmied, H. M., Munné, A., Munoz, F., Mykrä, H., Pal, I., Paloniemi, R., Pařil, P., Pengal, P., Pernecker, B., Polášek, M., Rezende, C., Sabater, S., Sarremejane, R., Schmidt, G., Domis, L. S., Singer, G., Suárez, E., Talluto, M., Teurlincx, S., Trautmann, T., Truchy, A., Tyllianakis, E., Väisänen, S., Varumo, L., Vidal, J.-P., Vilmi, A., and Vinyoles, D.: Securing Biodiversity, Functional Integrity, and Ecosystem Services in Drying River Networks (DRYvER), Research Ideas and Outcomes, 7, e77750, https://doi.org/10.3897/rio.7.e77750, 2021.

Dimitrijević, S., Jovanović, J., and Devedžić, V.: A comparative study of software tools for user story management, Information and Software Technology, 57, 352–368, https://doi.org/10.1016/j.infsof.2014.05.012, 2015.

Döll, P., Kaspar, F., Lehner, B.: A global hydrological model for deriving water availability indicators: model tuning and validation. Journal of Hydrology, 270 (1-2), 105-134. https://doi.org/10.1016/S0022-1694(02)00283-4, 2003.

Döll, P., Sester, M., Feuerhake, U., Frahm, H., Fritzsch, B., Hezel, D. C., Kaus, B., Kolditz, O., Linxweiler, J., Müller Schmied, H., Nyenah, E., Risse, B., Schielein, U., Schlauch, T., Streck, T., and van den Oord, G.: Sustainable research software for high-quality computational research in the Earth System Sciences: Recommendations for universities, funders and the scientific community in Germany, https://doi.org/10.23689/fidgeo-5805, 2023.

Döll, P., Hasan, H. M. M., Schulze, K., Gerdener, H., Börger, L., Shadkam, S., Ackermann, S., Hosseini-Moghari, S.-M., Müller Schmied, H., Güntner, A., and Kusche, J.: Leveraging multi-variable observations to reduce and quantify the output uncertainty of a global hydrological model: evaluation of three ensemble-based approaches for the Mississippi River basin, Hydrology and Earth System Sciences, 28, 2259–2295, https://doi.org/10.5194/hess-28-2259-2024, 2024.

Domisch, S., Portmann, F. T., Kuemmerlen, M., O'Hara, R. B., Johnson, R. K., Davy-Bowker, J., Bækken, T., Zamora-Muñoz, C., Sáinz-Bariáin, M., Bonada, N., Haase, P., Döll, P., and Jähnig, S. C.: Using streamflow observations to estimate the impact

of hydrological regimes and anthropogenic water use on European stream macroinvertebrate occurrences, Ecohydrology, 10, e1895, https://doi.org/10.1002/eco.1895, 2017.

Eisner, S.: Comprehensive evaluation of the WaterGAP3 model across climatic, physiographic, and anthropogenic gradients, 2016, PhD thesis, Kassel University, 2016, https://kobra.uni-kassel.de/bitstreams/99b4fc59-8807-40a9-82a5-d12a7f1a0788/download, last access: 11th March 2025

Flörke, M., Schneider, C., and McDonald, R. I.: Water competition between cities and agriculture driven by climate change and urban growth, Nat Sustain, 1, 51–58, https://doi.org/10.1038/s41893-017-0006-8, 2018.

Frieler, K. and Vega, I.: ISIMIP & ISIpedia - Inter-sectoral impact modeling and communication of national impact assessments, 2019, Bonn Climate Change Conference, session SBSTA 50, https://unfccc.int/documents/197148, last access: 20th February 2025.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J.: Design patterns, Addison Wesley, Boston, MA, 1994.

George, B. and Williams, L.: A structured experiment of test-driven development, Information and Software Technology, 46, 337–342, https://doi.org/10.1016/j.infsof.2003.09.011, 2004.

GitHub issues: coveragepy Issue #849, https://github.com/nedbat/coveragepy/issues/849, last access: 1st July 2025.

Guaman, D., Delgado, S., and Perez, J.: Classifying Model-View-Controller Software Applications Using Self-Organizing Maps, IEEE Access, 9, 45201–45229, https://doi.org/10.1109/ACCESS.2021.3066348, 2021.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E.: Array programming with NumPy, Nature, 585, 357–362, https://doi.org/10.1038/s41586-020-2649-2, 2020.

He, H.: Understanding Source Code Comments at Large-Scale, in: Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, New York, NY, USA, event-place: Tallinn, Estonia, 1217–1219, https://doi.org/10.1145/3338906.3342494, 2019.

Heinicke, S., Volkholz, J., Schewe, J., Gosling, S. N., Müller Schmied, H., Zimmermann, S., Mengel, M., Sauer, I. J., Burek, P., Chang, J., Kou-Giesbrecht, S., Grillakis, M., Guillaumot, L., Hanasaki, N., Koutroulis, A., Otta, K., Qi, W., Satoh, Y., Stacke, T., Yokohata, T., and Frieler, K.: Global hydrological models continue to overestimate river discharge, Environ. Res. Lett., 19, 074005, https://doi.org/10.1088/1748-9326/ad52b0, 2024.

Hema, V., Thota, S., Naresh Kumar, S., Padmaja, C., Rama Krishna, C. B., and Mahender, K.: Scrum: An Effective Software Development Agile Tool, IOP Conf. Ser.: Mater. Sci. Eng., 981, 022060, https://doi.org/10.1088/1757-899X/981/2/022060, 2020.

Hoch, J. M., Sutanudjaja, E. H., Wanders, N., van Beek, R. L. P. H., and Bierkens, M. F. P.: Hyper-resolution PCR-GLOBWB: opportunities and challenges from refining model spatial resolution to 1 km over the European continent, Hydrology and Earth System Sciences, 27, 1383–1401, https://doi.org/10.5194/hess-27-1383-2023, 2023.

Hoyer, S. and Hamman, J.: xarray: N-D labeled Arrays and Datasets in Python, Journal of Open Research Software, 5, https://doi.org/10.5334/jors.148, 2017.

Hut, R., Drost, N., van de Giesen, N., van Werkhoven, B., Abdollahi, B., Aerts, J., Albers, T., Alidoost, F., Andela, B., Camphuijsen, J., Dzigan, Y., van Haren, R., Hutton, E., Kalverla, P., van Meersbergen, M., van den Oord, G., Pelupessy, I., Smeets, S., Verhoeven, S., de Vos, M., and Weel, B.: The eWaterCycle platform for open and FAIR hydrological collaboration, Geoscientific Model Development, 15, 5371–5390, https://doi.org/10.5194/gmd-15-5371-2022, 2022.

Hutton, C., Wagener, T., Freer, J., Han, D., Duffy, C., and Arheimer, B.: Most computational hydrology is not reproducible, so is it really science?, Water Resources Research, 52, 7548–7555, https://doi.org/10.1002/2016WR019285, 2016.

ISIMIP: ISIMIP protocol: Preparing simulation files, https://www.isimip.org/protocol/preparing-simulation-files/, last access: 1st July 2025.

Jiménez Cisneros, B. E., Oki, T., Arnell, N. W., Benito, G., Döll, P., Jiang, T., Cogley, J. G., and Mwakalila, S. S.: Freshwater resources, in: Climate Change 2014: Impacts, Adaptation, and Vulnerability. Part A: Global and Sectoral Aspects, Cambridge University Press, Cambridge, United Kingdom and New York, NY, USA, 229–269, 2014.

Johnson, S.: Python vs. C++: A Comparison of Key Features and Differences, https://www.stxnext.com/blog/python-vs-cplus-plus-comparison, last access: 20th February 2025.

Katz, D. S.: Research Software: Challenges & Actions. The Future of Research Software: International Funders Workshop, Amsterdam, Netherlands., https://doi.org/10.5281/zenodo.7295423, 2022.

Kusche, J., Schmidt, R., Petrovic, S., and Rietbroek, R.: Decorrelated GRACE time-variable gravity solutions by GFZ, and their validation using a hydrological model, J Geod, 83, 903–913, https://doi.org/10.1007/s00190-009-0308-3, 2009.

Lam, S. K., Pitrou, A., and Seibert, S.: Numba: a LLVM-based Python JIT compiler, in: Proceedings of the Second Workshop on the LLVM Compiler Infrastructure in HPC, SC15: The International Conference for High Performance Computing, Networking, Storage and Analysis, Austin Texas, 1–6, https://doi.org/10.1145/2833157.2833162, 2015.

McConnell, S.: in: Code Complete, Second Edition, Microsoft Press, USA, 565–596, 2004.

Mehta, P., Siebert, S., Kummu, M., Deng, Q., Ali, T., Marston, L., Xie, W., and Davis, K. F.: Half of twenty-first century global irrigation expansion has been in water-stressed regions, Nat Water, 2, 254–261, https://doi.org/10.1038/s44221-024-00206-9, 2024.

Melton, J. R., Arora, V. K., Wisernig-Cojoc, E., Seiler, C., Fortier, M., Chan, E., and Teckentrup, L.: CLASSIC v1.0: the open-source community successor to the Canadian Land Surface Scheme (CLASS) and the Canadian Terrestrial Ecosystem Model (CTEM) – Part 1: Model framework and site-level performance, Geoscientific Model Development, 13, 2825–2850, https://doi.org/10.5194/gmd-13-2825-2020, 2020.

Molnar, A.-J., Motogna, S., and Vlad, C.: Using static analysis tools to assist student project evaluation, in: Proceedings of the 2nd ACM SIGSOFT International Workshop on Education through Advanced Software Engineering and Artificial Intelligence, ESEC/FSE ’20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual USA, 7–12, https://doi.org/10.1145/3412453.3423195, 2020.

Muhammad, A., Siddique, A., Mubasher, M., Aldweesh, A., and Naveed, Q. N.: Prioritizing Non-Functional Requirements in Agile Process Using Multi Criteria Decision Making Analysis, IEEE Access, 11, 24631–24654, https://doi.org/10.1109/ACCESS.2023.3253771, 2023.

Müller Schmied, H., Cáceres, D., Eisner, S., Flörke, M., Herbert, C., Niemann, C., Peiris, T. A., Popat, E., Portmann, F. T., Reinecke, R., Schumacher, M., Shadkam, S., Telteu, C.-E., Trautmann, T., and Döll, P.: The global water resources and use

model WaterGAP v2.2d: model description and evaluation, Geoscientific Model Development, 14, 1037–1079, https://doi.org/10.5194/gmd-14-1037-2021, 2021.

Müller Schmied, H., Trautmann, T., Ackermann, S., Cáceres, D., Flörke, M., Gerdener, H., Kynast, E., Peiris, T. A., Schiebener, L., Schumacher, M., and Döll, P.: The global water resources and use model WaterGAP v2.2e: description and evaluation of modifications and new features, Geoscientific Model Development Discussions, 1–46, https://doi.org/10.5194/gmd-2023-213, 2023.

Müller Schmied, H., Trautmann, T., Ackermann, S., Cáceres, D., Flörke, M., Gerdener, H., Kynast, E., Peiris, T. A., Schiebener, L., Schumacher, M., and Döll, P.: The global water resources and use model WaterGAP v2.2e: description and evaluation of modifications and new features, Geoscientific Model Development, 17, 8817–8852, https://doi.org/10.5194/gmd-17-8817-2024, 2024.

Nissen, L.: Reprogrammed groundwater surface water use model, https://github.com/HydrologyFrankfurt/ReGWSWUSE.git last access: 1st July 2025a.

Nissen, L.: Unit tests for reprogrammed GWSWUSE software, https://github.com/HydrologyFrankfurt/ReGWSWUSE/tree/main/test, last access: 1st July 2025b.

Nüst, D., Sochat, V., Marwick, B., Eglen, S. J., Head, T., Hirst, T., and Evans, B. D.: Ten simple rules for writing Dockerfiles for reproducible data science, PLoS Comput Biol, 16, e1008316, https://doi.org/10.1371/journal.pcbi.1008316, 2020.

Nyenah, E.: Automated linting workflow for WGHM software, https://github.com/HydrologyFrankfurt/ReWaterGAP/blob/main/.github/workflows/lint.yaml, last access: 1st July 2025a.

Nyenah, E.: Automated unit test workflow for WGHM software, https://github.com/HydrologyFrankfurt/ReWaterGAP/blob/main/.github/workflows/unit_test.yaml, last access: 1st July2025b.

Nyenah, E.: ReWaterGAP Web-based Documentation, https://hydrologyfrankfurt.github.io/ReWaterGAP/, last access: 1st July 2025c.

Nyenah, E.: Unit tests for reprogrammed WGHM software, https://github.com/HydrologyFrankfurt/ReWaterGAP/tree/main/test, last access: 1st July 2025d.

Nyenah, E.: Unit tests for the canopy storage module, https://github.com/HydrologyFrankfurt/ReWaterGAP/blob/main/test/test_canopy.py, last access: 1st July2025e.

Nyenah, E., Döll, P., Katz, D. S., and Reinecke, R.: Software sustainability of global impact models, Geoscientific Model Development Discussions, 1–29, https://doi.org/10.5194/gmd-2024-97, 2024.

Nyenah, E., Döll, P., Floerke, M., Mühlenbruch, L., Nissen, L., and Reinecke, R.: Reprogrammed version of the WaterGAP V2.2e and Groundwater Surface Water Use (GWSWUSE) linking model.,l, https://doi.org/10.5281/zenodo.14988011, 2025a.

Nyenah, E., Döll, P., Floerke, M., Mühlenbruch, L., Nissen, L., and Reinecke, R.: The Process and Value of Reprogramming a Legacy Global Hydrological Model, https://doi.org/10.5281/zenodo.14988258, 2025b.

Oliphant, T. E.: Python for Scientific Computing, Computing in Science & Engineering, 9, 10–20, https://doi.org/10.1109/MCSE.2007.58, 2007.

Pajankar, A.: unittest, in: Python Unit Test Automation: Automate, Organize, and Execute Unit Tests in Python, edited by: Pajankar, A., Apress, Berkeley, CA, 43–90, https://doi.org/10.1007/978-1-4842-7854-3_3, 2022.

Prechelt, L.: An empirical comparison of C, C++, Java, Perl, Python, Rexx, and Tcl for a search/string-processing program, https://page.mi.fu-berlin.de/prechelt/Biblio/jccpprtTR.pdf, last access: 20th February 2025.

Python, S. F.: unittest — Unit testing framework, https://docs.python.org/3/library/unittest.html, last access: 20th February 2025.

Reinecke, R., Trautmann, T., Wagener, T., and Schüler, K.: The critical need to foster computational reproducibility, Environmental Research Letters, 17, https://doi.org/10.1088/1748-9326/ac5cf8, 2022.

van Rossum, G., Warsaw, B., and Coghlan, N.: PEP 8 – Style guide for Python code, 2001, https://docs.python.org/3/library/unittest.html, last access: 20th February 2025, last access: 1st July 2025.

Schmidt, R., Schwintzer, P., Flechtner, F., Reigber, Ch., Güntner, A., Döll, P., Ramillien, G., Cazenave, A., Petrovic, S., Jochmann, H., and Wünsch, J.: GRACE observations of changes in continental water storage, Global and Planetary Change, 50, 112–126, https://doi.org/10.1016/j.gloplacha.2004.11.018, 2006.

Schneider, C., Flörke, M., De Stefano, L., and Petersen-Perlman, J. D.: Hydrological threats to riparian wetlands of international importance – a global quantitative and qualitative analysis, Hydrology and Earth System Sciences, 21, 2799–2815, https://doi.org/10.5194/hess-21-2799-2017, 2017.

Schomberg, A. C., Bringezu, S., and Flörke, M.: Extended life cycle assessment reveals the spatially-explicit water scarcity footprint of a lithium-ion battery storage, Commun Earth Environ, 2, 1–10, https://doi.org/10.1038/s43247-020-00080-9, 2021.

Siebert, S., Kummu, M., Porkka, M., Döll, P., Ramankutty, N., and Scanlon, B. R.: A global data set of the extent of irrigated land from 1900 to 2005, Hydrology and Earth System Sciences, 19, 1521–1545, https://doi.org/10.5194/hess-19-1521-2015, 2015.

Sphinx Project: Sphinx Documentation, https://www.sphinx-doc.org/en/master/, last access: 1st July 2025.

Stack Overflow: Analyzing coverage of numba-wrapped functions, https://stackoverflow.com/questions/26875191/analyzing-coverage-of-numba-wrapped-functions, last access: 1st July 2025.

Stacke, T. and Hagemann, S.: HydroPy (v1.0): a new global hydrology model written in Python, Geoscientific Model Development, 14, 7795–7816, https://doi.org/10.5194/gmd-14-7795-2021, 2021.

Sutanudjaja, E. H., van Beek, R., Wanders, N., Wada, Y., Bosmans, J. H. C., Drost, N., van der Ent, R. J., de Graaf, I. E. M., Hoch, J. M., de Jong, K., Karssenberg, D., López López, P., Peßenteiner, S., Schmitz, O., Straatsma, M. W., Vannametee, E., Wisser, D., and Bierkens, M. F. P.: PCR-GLOBWB 2: a 5 arcmin global hydrological and water resources model, Geoscientific Model Development, 11, 2429–2453, https://doi.org/10.5194/gmd-11-2429-2018, 2018.

Telteu, C.-E., Müller Schmied, H., Thiery, W., Leng, G., Burek, P., Liu, X., Boulange, J. E. S., Andersen, L. S., Grillakis, M., Gosling, S. N., Satoh, Y., Rakovec, O., Stacke, T., Chang, J., Wanders, N., Shah, H. L., Trautmann, T., Mao, G., Hanasaki, N., Koutroulis, A., Pokhrel, Y., Samaniego, L., Wada, Y., Mishra, V., Liu, J., Döll, P., Zhao, F., Gädeke, A., Rabin, S. S., and Herz, F.: Understanding each other's models: an introduction and a standard representation of 16 global water models to support intercomparison, improvement, and communication, Geoscientific Model Development, 14, 3843–3878, https://doi.org/10.5194/gmd-14-3843-2021, 2021.

Venters, C. C., Capilla, R., Betz, S., Penzenstadler, B., Crick, T., Crouch, S., Nakagawa, E. Y., Becker, C., and Carrillo, C.: Software sustainability: Research and practice from a software architecture viewpoint, Journal of Systems and Software, 138, 174–188, https://doi.org/10.1016/j.jss.2017.12.026, 2018.

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., Carey, C. J., Polat, İ., Feng, Y., Moore, E. W., VanderPlas, J., Laxalde, D., Perktold, J., Cimrman, R., Henriksen, I., Quintero, E. A., Harris, C. R., Archibald, A. M., Ribeiro, A. H., Pedregosa, F., and van Mulbregt, P.: SciPy 1.0: fundamental algorithms for scientific computing in Python, Nat Methods, 17, 261–272, https://doi.org/10.1038/s41592-019-0686-2, 2020.

van Vliet, M. T. H., van Beek, L. P. H., Eisner, S., Flörke, M., Wada, Y., and Bierkens, M. F. P.: Multi-model assessment of global hydropower and cooling water discharge potential under climate change, Global Environmental Change, 40, 156–170, https://doi.org/10.1016/j.gloenvcha.2016.07.007, 2016.

Wan, W., Döll, P., and Zheng, H.: Risk of Climate Change for Hydroelectricity Production in China Is Small but Significant Reductions Cannot Be Precluded for More Than a Third of the Installed Capacity, Water Resources Research, 58, e2022WR032380, https://doi.org/10.1029/2022WR032380, 2022.

Wang, Y., Zheng, B., and Huang, H.: Complying with Coding Standards or Retaining Programming Style: A Quality Outlook at Source Code Level, Journal of Software Engineering and Applications, 1, 88–91, https://doi.org/10.4236/jsea.2008.11013, 2008.

Warszawski, L., Frieler, K., Huber, V., Piontek, F., Serdeczny, O., and Schewe, J.: The Inter-Sectoral Impact Model Intercomparison Project (ISI–MIP): Project framework, Proceedings of the National Academy of Sciences, 111, 3228–3232, https://doi.org/10.1073/pnas.1312330110, 2014.

Wiggins, G., Cage, G., Smith, R., Hitefield, S., McDonnell, M., Drane, L., McGaha, J., Brim, M., Abraham, M., Archibald, R., and Malviya-Thakur, A.: Best practices for documenting a scientific Python project, , https://doi.org/10.5281/zenodo.10426364, 2023.

Wilkinson, M. D., Dumontier, M., Aalbersberg, Ij. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., Bouwman, J., Brookes, A. J., Clark, T., Crosas, M., Dillo, I., Dumon, O., Edmunds, S., Evelo, C. T., Finkers, R., Gonzalez-Beltran, A., Gray, A. J. G., Groth, P., Goble, C., Grethe, J. S., Heringa, J., 't Hoen, P. A. C., Hooft, R., Kuhn, T., Kok, R., Kok, J., Lusher, S. J., Martone, M. E., Mons, A., Packer, A. L., Persson, B., Rocca-Serra, P., Roos, M., van Schaik, R., Sansone, S.-A., Schultes, E., Sengstag, T., Slater, T., Strawn, G., Swertz, M. A., Thompson, M., van der Lei, J., van Mulligen, E., Velterop, J., Waagmeester, A., Wittenburg, P., Wolstencroft, K., Zhao, J., and Mons, B.: The FAIR Guiding Principles for scientific data management and stewardship, Sci Data, 3, 160018, https://doi.org/10.1038/sdata.2016.18, 2016.

Wilson, G., Aruliah, D. A., Brown, C. T., Hong, N. P. C., Davis, M., Guy, R. T., Haddock, S. H. D., Huff, K. D., Mitchell, I. M., Plumbley, M. D., Waugh, B., White, E. P., and Wilson, P.: Best Practices for Scientific Computing, PLOS Biology, 12, e1001745, https://doi.org/10.1371/journal.pbio.1001745, 2014.

WMO: State of Global Water Resources report 2023, https://library.wmo.int/records/item/69033-state-ofglobalwaterresources-report-2023, last access: 20th February 2025.