Supplement of Hydrol. Earth Syst. Sci., 26, 4773–4800, 2022
https://doi.org/10.5194/hess-26-4773-2022-supplement

Hydrology and Earth System Sciences
Open Access
EGU

Supplement of

HESS Opinions: Participatory Digital eARth Twin Hydrology systems (DARTHs) for everyone – a blueprint for hydrologists

Riccardo Rigon et al.

Correspondence to: Riccardo Rigon (riccardo.rigon@unitn.it)

Index

This supplemental material includes:

- a DARTHs CHEAT SHEET
- a Glossary
- an extensive definition of Model as an Application, Model as a Tool, Model as a Service, Model as a Resource, Model as a Commodity

DARTHs CHEAT SHEET

This is a companion to the paper:HESS Opinions. Participatory Digital Earth Twin Hydrology systems (DARTHs) for everyone: a blueprint for hydrologists, by Rigon et al. , 2022 This is a summary of the more extensive and articulated manuscript.
DARTHs are composable infrastructures with parts that are loosely connected, communicate with standard protocols, and can easily be substituted. There is not such a thing as a DARTH but rather there are **DARTH solutions. They constitute an ecosystem of tools** with parts that can be separated and recomposed into new solutions, exactly as happens for Linux distributions (distros). Just as distros usually need an integrator of resources, this constitutes a a further component of DARTHs. Below we subdivide the main requirements necessary to make the DARTHs work effectively.

Data

- Must be Open
- Provided on demand over the cloud
- Discoverable on the web
- Provided in standardized self-explanatory formats
- Retrievable calling open APIs from the most common computer languages

DARTHs Architecture requirements

DARTHs serve different users and roles
Therefore they are agnostic with with respect to::

- the science

- the programming language
- the operating system
- modelling styles and paradigms

Furthermore:

- they are lightweight with respect to programming habits, meaning they should be minimalist in adding programming rules and aim to maintain the code short
- variables names should be mapped into standard names that provide a unique identification

In Open DARTHs:

- Programmes must be open source and developable with open tools
- Simulations and operations must be traceable and replicable by construction
- Chunks of data, codes and modelling solutions should be organized in standard ways and be deployable over the web foresting and reuse by third parties

Hydrological models for DARTHs:

- need to be able to facilitate any type of modelling effort, such as models based on ODEs (systems of ordinary differential equations), PDEs (system of partial differential equations), MS (Statistical Modelling), ML (Machine Learning) and any other tool that science will invent.
- should be deployed in reusable and interoperable components (A component model is a model that is used to encapsulate equations or specific tasks into a reusable form. Components can be connected at runtime and they communicate by exchanging data in RAM memory)
- reusable components should obey standard rules for the appropriate identification of inputs and outputs
- need to be deployable overthe web on all the available platforms.
- model-to-model (components) communication should be allowed through API

DARTHs programming practices

DARTHs must:

- adopt separate software quality testing for any component
- adopt good programming practices
- have literate computing tools

- promote clean code and literate programming

DARTHs and Computing facilities

- DARTHs use various forms of parallelism but overall, parallelism should be provided as a service without having to change the programming habits of model developers.
- Workflows of tasks that can be schematized on graphs should be parallelized internally by using piping methods.
- Gridded (physically based) hydrological models should use middleware that separates the parallelization from the "physics" of implementation. The middleware should be non invasive.
- DARTHs can use multicores, multiprocessors, high end computers, distributed computing without necessarily having the direct intervention of the programmer
- DARTHs should use the natural partition of the Earth's surface into river catchments and subcatchments to split the computational work.

DARTHs and EO

- The complex retrieval chain from observed quantities to hydrological quantities has to become explicit and integrated in the modelling
- Data Assimilation of any type must become part of the modelling chain

DARTHs and Science Reliability and practices

- DARTH must make hypothesis testing very smooth (in the sense of allowing alternative modelling structures)
- The quantification of uncertainty must become inherently part of an open DARTH.
- DARTHs must support open science practices by construction

Overall

- To accomplish all the above tasks, DARTHs requires an appropriate, lightweight, noninvasive infrastructure (framework) that:
    - supports the features required
    - allows the development of the use and the continuous evolution of the DARTHs
    - separates the programming and the use of tools from the connection to data resources (EO, IoT, more traditional datasets),
    - connects to web services,
    - provides parallelism of computation
    - accesses High Performance Computing or HPC cloud services,

- allows communication between components,
- and components dispatching over the web,
- manages the potential security issues,

DARTHs Glossary

API (Application Programming Interfaces): a software intermediary that allows to connect computers or pieces of software to each other (https://en.wikipedia.org/wiki/API)

Bridges**:** Software libraries that connect computer languages to computer framework like, for instance, FORTRAN to a machine learning framework.

Building tools*:* programmes that automate the creation of executable applications from source code. Building incorporates compiling, linking and packaging the code into a usable or executable form. (https://en.wikipedia.org/wiki/Software_build)

Component*:* self-contained building blocks, modules or units of code. Each well-designed component usually implements a single modelling concept. A component can be made of multiple algorithms still remaining limited in its scope. Components can be connected each other to obtain a modelling solution, and eventually replaced to get alternatives. This opens the way to compare different approaches within the same chain of tools, and inserted into modelling solutions as alternatives, thus opening the way to compare different approaches within the same chain of tools.

Coders, Hard - those who actually design and write code for models.

Coders, Hidden **-** those who program the basic infrastructure that the hard coders use as basis for their coding. For instance, in the case of the GEOframe system (e.g. Formetta et al. 2014), Formetta et al are the hard coders, the group of OMS3 developers (David et al, 2013) are the hidden coders.

Coders, Soft **-** those who just modify existing codes and, if possible, develop plug-ins using public Application Programming Interfaces (API).

Declarative (programming): Declarative programming is a high-level programming concept, which is the opposite of imperative programming. It is typically found in databases and configuration management software, paired with a domain-specific language (DSL). Declarative models rely on preconfigured capabilities in the language to accomplish a task without explicit case-by-case instructions on what steps to take (by Bertham). In declarative programming the task to obtain is “declared” instead of giving a list of instructions to obtain the task.

Dependencies: programmes required to run or provide extra features, as well as to test or build/compile other programmes.

Design pattern*:* a reusable solution to a commonly occurring problem within a given context in software design (https://en.wikipedia.org/wiki/Software_design_pattern)

Domain Specific Language (DSL): a computer language specialized to a particular application domain, e.g., HTML (https://en.wikipedia.org/wiki/Domain-specific_language)

Encapsulation*:* in object-oriented programming, it prevents direct access to objects by clients to avoid exposing hidden implementation details or violate state invariance (https://en.wikipedia.org/wiki/Encapsulation_(computer_programming)).

Imperative: This programming paradigm adopts statements to change the program's state (https://en.wikipedia.org/wiki/Imperative_programming).

Implicit parallelism*:* characteristic of a computer infrastructure (or a programming language) that allows a compiler or interpreter to automatically exploit, where possible, the parallel execution inherent to the computations (https://en.wikipedia.org/wiki/Implicit_parallelism).

Information hiding: Encapsulation is used to hide (insulate) the values or state of a structured data object inside a class (or a chunk of computer code), preventing direct access to them by clients in a way that could expose hidden implementations that can be changed in future versions of the software or that could be dangerous to change because of the creation of undesired side-effects on other parts of the software.

Integrated Modelling Framework **-** Same as MBC - The informatics that allows for the break-down of a complex model in such a way that it is broken down into smaller re-usable sub-components. The component domain of interest can span different disciplines, as happens in *Integrated assessment modelling. (*https://en.wikipedia.org/wiki/Integrated_assessment_modelling)

Internals*:* a model's internals means the implemented equations that constitute the model itself, internal parameters, or ancillary parts of the model used as intermediate steps for calculation.

Interoperable: characteristic of a product or system to work with other products or systems (https://en.wikipedia.org/wiki/Interoperability).

Non-invasive: in relation to the OMS3 framework, non-invasive means that it does not change the habits of a good programmer.

Libraries*:* data and/or programming code, used to develop other software programs and applications.

Linkers - power users who assemble existing codes through scripting languages or procedures in a DSL

MaaA - Model as an Application - See the dedicated supplementary document below

MaaC - Model as a Commodity - See the dedicated supplementary document below

MaaT - Model as a Tool - See the dedicated supplementary document below

MaaS - Model as a Service - See the dedicated supplementary document below

MaaR - Model as a Resource - See the dedicated supplementary document

Modelling by Components (MBC) - The strategy of modelling used by the Integrated Modelling Frameworks (see above).

Modelling framework*:* It is analogous to a software framework, with the specialization in providing reusable components for building mathematical models. There are many modelling frameworks on the market, examples are MATLAB, Modelica, and so on (see Chen 2020 for a comprehensive list)

Modelling solution**:** components can be joined together to obtain a modelling solution to accomplish a complicated task, such as simulating the water budget storages and fluxes.

Open source: software with a codebase that is freely available for possible modification and redistribution, e.g., open source code/ data. Open source code has an open source license of use (https://opensource.org/licenses/category)

Players - those who use the models for producing scenarios or specific studies for some purpose, without modifying the code base but characterizing the parameters of the model and exploiting their capabilities. They can be researchers or technical staff.

Providers - those who provide the data that are essential for running the models.

Refactoring: re-structuring the code or parts of code without changing its overall behaviour/ scope.

Realism (of models) - It is mainly concerned with reproducing natural behaviour (though some philosophical arguments can be made, see, for instance, Dietrich et al., 2003). A realistic model reproduces the variety and behavioural (dynamical) complexity of the (hydrological) system it is intended to describe. In evaluating this characteristic, one cannot limit the analysis to the model core itself, but must also look at other aspects such as the calibration tools or if the model is integrated with data assimilations techniques. (See also reliability)

Reliability (of models) - A model is only ever relatively reliable. Therefore, in the present context we define a model as reliable if it is possible to give an error of the estimated quantities in any of the circumstances it can be used. According to the scope of a simulation, the reliability is sufficient if the error (estimate) is acceptable for the scope itself. Reliability, from a more philosophical point of view, is related to the concept of Popper's falsification process (e.g. Nearing et al., 2021). However, the current shortcut to assess reliability of models is goodness of fit (GOF) indexes, such as the Nash-Sutcliffe (e.g. McCuen et al., 2006) or the Kling-Gupta Efficiency (Kling and Gupta, 2011). These indicators are clearly a good thing, but they are barely sufficient in getting a clear picture of the reliability of models; therefore, the conjoint use of further indicators has been invoked (Addor, 2017) to this end. Even these efforts are probably insufficient, and confidence intervals and more accurate analyses of uncertainty should be produced.

Replicability (of model behaviour) - Replicability refers to the fact that multiple runs of the same model with the same inputs and the same setup must always produce the same results. A model's behaviour is replicable if its workflow is recorded or appropriately documented and the workflow deployed verbatim (not forgetting the observations made in Ceola et al., 2015). If the model is stochastic, however, the replicability concept is transferred to the statistics of the model's output. A special case is the one of models that depend on parameter calibration: because parameter fitness is usually established with stochastic searches, in this case, the replicability of the whole running actions is impossible. However, with fixed parameters, simulations must be replicable.

Reproducibility (of models) - Results presented in a science context, either from a real or a virtual, computer-based, experiment, must be autonomously reproducible by third parties by following the same procedures used in the experiments. This is one of the pillars of science. Being specific, a model's behaviour should be reproducible by other codes whose implementation follows the information contained in the documentation of the original

software. As a matter of fact, however, the precise reproducibility of a model is often difficult because of hidden implementation details or behaviour of some internally used algorithms.

Robustness (of models) - It is a property of the informatics and numerics of a model. A robust model works for the largest foreseen set of use cases without issuing exceptions and being able to manage them when they are unavoidable. Model algorithms and design should be accurately planned for this purpose (to obtain what is called a software system product, Frederick, 1995). Besides, specifically in the DARTHs context, we expect that the model can be run on different operating systems without taking care of the different platform details.

Reusability (of models) - A model is maximally reusable if any of its parts can be reused effortlessly withinother models that share the appropriate characteristics. The Modelling By Components (MBC) paradigm aims also to enhance this property. Considered together with robustness, the reusability quality allows the simulation of a large set of physical (hydrological) situations, including those for which the model was not initially conceived.

Runtime**:** running phase of a programme, when a programme is executed (including the preparatory actions, like setting the parameters required by the modelling solution and the workflow)

Simulation: within the OMS3 framework, the simulation file is composed of three main parts:

- component, where the executable components to use are specified,
- parameter, which specifies the parameters of each component, and
- connect, where the connections between the components are specified.

Connections are always made according to the out-to-in schema, i.e. the output of a component is the input on the following component

Service Oriented Architecture (SOA) - SOA is a software development model that allows services to communicate across different platforms and languages to form applications. In SOA, a service is a self-contained unit of software designed to complete a specific task.

Silos - MaaA are usually defined as “silos”, in the sense that they cannot exchange data and procedures with other MaaA and do not favour the exchange of knowledge among related disciplines.

Top-down and bottom-up approach*:* a top-down approach refers to a strategy in which decisions are made from the highest level down to the lowest. A bottom-up approach is the opposite: it starts from the smallest systems/subsystems, building up to more complex systems (https://en.wikipedia.org/wiki/Top-down_and_bottom-up_design)

Viewers - those who browse the results of the models for various purposes,for instance, for decision making, policies building, etc.

MaaA, MaaT, MaaS, MaaR, MaaC

MaaA - Model as an Application

MaaA - Models as Applications are fully fledged models that have an architecture that is inclusive of the data formats and visualization tools. What follows for MaaA is taken with little modification, and sometimes verbatim, from Rizzoli et al. (2006) and Knoben et al (2021).

1. **An MaaA bundles data, algorithms and the graphical user interface of a model in an application.** This makes the model very hard to re-use out of its original context. Most MaaTs are also "monoliths" composed of hundreds of thousands lines of code.
2. An MaaA works just on one operating system MS Windows, Mac OS or Linux.

On MaaA, despite being unaware of the nomenclature we introduced here, Knoben et al. (2021) provides the following description:

"*These tools are typically provided as self-contained packages. Packages tend to be easy to use for their intended purpose but take time to understand and do not necessarily provide much flexibility to deviate from their intended purpose. Layering additional functions on top of an existing package or modifying a package's source code is"* sometimes *"possible, but can be outside the comfort zone of many users.*"

Other usual characteristics of MaaA are:

- Their evolution is totally in the hands of the original developers. This is a good thing for intellectual property rights and in a commercial environment, but this is very much a bad thing for science and the way it is supposed to progress.
- MaaAs often do not come with associated datasets for testing. Moreover, the adoption of object-oriented programming, while it is a good thing for model reusability and portability, makes things more complex for testing, because of a number of problems such as "*observability in virtual method calls and state dependent behavior of objects"* .
- The way MaaAs are coded (as monolithic software entities) displays a strong level of internal cohesion, and, if a modeller is interested in reusing a particular function within a bigger model, they can find it very hard to isolate and extract it, given the strong dependencies existing in the source code parts.

- Their data formats do not come from a community agreement and their developers typically decide to have output data in a format relevant to their own application, which may not be a format that is widely used by others. It is cumbersome for developers to have their tools ingest multiple different data formats and such functionality is therefore somewhat rare (slightly modified from Knoben et al 2021).

It could be observed that an MaaA could be evolved to eliminate the various characteristics in the bullet list. In fact there exists a variety of MaaA that, especially recently, have pursued such achievements (modular code, open to common data formats, separation between the graphical model interface and the rest of the code) but still preserve the tight internal coupling of all the model specific parts with others that could be separated.

MaaT - Model as a Tool (Mainly From Nativi et al., 2021)

In MaaTs, as opposed to MaaAs, there are at least two levels of abstraction: the interface is abstracted from the model, in a client-server way, and the model is loosely coupled to the data. The interacting tools are distinct from the model itself and can eventually be changed in the sense clearly described in Knoben et al., (2021).

However, a given implementation of the model runs on a specific server and the Runner interacts with the model through the user interface. In an MaaT the models are preloaded on a specific machine. Besides, **it is not possible to modify the interaction between the server and the client which is kept fixed by the user interface.**
Benefits of this informatics include again a strong control of the model use and execution (which could be useful to control what happens in an operational service). There are limitations on the usability and flexibility of the model, as well as its scalability due to the limitation of the specific server. Machine-to-machine interoperability (chaining capabilities) is not allowed. Knoben et al (2021), without knowing the acronym, defines MaaT well when talking of some web-based services writes: "*... several of these tools are provided as web-based services. This can be appealing because, for example, data can be pre-downloaded to speed up model configuration and model simulations can be easily shared. The advantage of such approaches is that they can be combined with some form of server-side data transformations (e.g., subsetting or averaging), which minimizes data transfers. Storing the inputs for and outputs of large-domain simulations can, however, be cumbersome, and keeping pre-downloaded data up-to-date and sufficient for all user needs takes sustained, long-term effort. **A further complication is that it is regrettably common that such web-based services require some form of manual interaction with the webpage, limiting opportunities to automate data acquisition tasks***".
However, it can be said that the models' core in a MaaT is agnostic with respect to the data source and formats (for a detailed explanation see Knoben et al., 2021).

In an open MaaT,

- model evolution should be in the hands of a community
- models should come with an appropriate set of tests both for the informatics and the physics.
- a modular structure for the code should be the rule in programming, even without particular encapsulation methodologies exploited.
- tools for data brokering could be available

MaaS - Model as a Service (Mainly From Nativi et al., 2021 and David et al, 2014)

A "Model-as-a-Service" provides the capability to execute simulation models as a service. As Wikipedia reports: "*In the contexts of software architecture, service-orientation and service-oriented architecture, the term service refers to a software functionality or a set of software functionalities (such as the retrieval of specified information or the execution of a set of operations) with a purpose* ***that different clients can reuse for different purposes****, together with the policies that should control its usage (based on the identity of the client requesting the service, for example).*"
As for the previous case of MaaT, a given implementation of the analytical model runs on a specific server, but this time, **APIs are exposed to interacting with the model.** Therefore, interoperability consists of machine-to-machine interaction **through a published API**, e.g., for a run configuration and execution. Nevertheless, it is not possible to move the model and make it run on a different machine (without having to "manually" install the model and its managing software on these machines). **Concerns deal with a still limited flexibility and possible scalability issues (depending on the server capacities).**

There are two main usage patterns for the use of these software architectures : (i) The model can be pre-deployed, has a well-known service endpoint, and is supported by supplemental data services. This is quite common for operational models (not in hydrology though) used in a production environment. Moreover, (ii) **the model can be dynamically deployed from the client before execution (implying that a MaaS is made up of a pool of modelling components that can be linked just before run time with a scripting language)**. Model service development for research purposes needs such a behaviour, since changing models is at the base of hypothesis testing. Both approaches address a different workflow and need for availability and security.

We expect that in a MaaS system, there is the maximum separation of parts between data access, models cores, and interfaces. The modelling structure is of MBC type and visualization of data, as well the execution of models, can be done in various ways through

appropriate APIs. By all means, the models run on a limited number of servers (or server types) that are known a-priori.

MaaR - Model as a Resource

The interoperability level resamples the same patterns used for any other shared digital resource, like a dataset.

- **This time, the model itself (and not a given implementation) is accessed through a resource-oriented interface, i.e., API, and**
- **a software infrastructure layer, uploadable on heterogeneous hardwares , manages (not the user directly) a set of compliant models.**
- **That allows one to move models and make them run on the machine that best performs for a specific use case among a large pool of computing facilities without the need of user intervention (at the model level).**

Cloud services can distribute the model runs on various architectures (such as: cloud services, high performance computing machines, multicore machines, clusters of computers), dynamically adapting the request of resources to the demand, independently of the background hardware, thanks to the appropriate software layers.
There are clear benefits in terms of flexibility, scalability, and interoperability. The main concerns, maybe, regard the sound utilization of the model.

MaaC - Models as a Commodity

They are MaaS or MaaR that in addition have controls on the Science and their explanation and tools that facilitate open and participatory workflows.

Differently from the other, previous, classifications, the Model as a Commodity definition **does not imply information technology issues** but rather questions that are strictly related to science. They can be used as a mass-produced unspecialized product (the meaning of commodity) because they are equipped with some special features described below.
They must be open source, carefully programmed according to clean code rules (e.g. Martin, 2009) and appropriately documented (as required, for instance, by the Journal of Open source Software, JOSS)
DARTHs must be provided with **error (uncertainty) estimations** for all the hindcasted and forecasted quantities. The presence of error estimates should warn about the quality of the checks that were made to assess the reliability of the model. The topic is a difficult one with a large amount of literature (e.g. Beven, 2016), itself often difficult and obscure (e.g. Nearing et al., 2016), and the requirement here of having a quantification of uncertainty does not enter

into the dispute of the origin of errors. In fact, in staying with Cox's (1946) statement that "*purely empirically, probability and statistics can, of course, describe anything from observations to model residuals regardless of the actual sources of uncertainty as an expression of our reasonable expectations*" (taken from Beven, 2016), then, at least an empirical estimation of the error on the base of recorded data is possible.
Because modelling requires a first phase of training/calibration on past datasets, the error of modelling must include an analytic performance over the past data of the model. **Therefore a MaaC is a MaaS or MaaR provided with error estimations on any of the hindcasted or forecasted quantities**, a warning for the use of any quantity (and a major effort for modellers).
A further science-side aspect would be that in a DARTH the forecasted quantities would also be given names that provide **some complete description of what the quantity is or how it was produced**. For instance a phrase like: "we are providing runoff over the catchment area" should not be used, but it should be specified, for instance: "with the SCS-CN method". A better description would be: "SCS-CN runoff over the catchment area". In this case, the specification would indicate a very approximate, yet common, methodology to get the runoff. The pedestrian user might not be aware of what the SCS-CN method is, but the presence of the specification should advertise that other methods exist and the results could be questionable.

The MaaCs inherit from MaaS and MaaR their **composable structure**, however **with a purpose**. Components are self-contained building blocks, modules or units of code. Each well-designed component usually implements a single modeling concept. Multiple algorithms can be implemented within the same component or in various components, and inserted into modelling solutions as alternatives, thus opening the way to compare, within the same chain of tools, different approaches. This responds also to a science requirement, i.e. the idea that models should be used in DARTHs as hypotheses to be tested among various possibilities (Clark et al, 2011). This flexibility will not usually be directly available to the less aware end-users, but it will certainly be useful to scientists in providing more reliable modelling.
Therefore, MaaC requires tools to support the workflow of hypothesis testing. These tools are usually provided with **"literate computing" workflows**, such as those explained in the Appendix to the paper. In Maac **replicability (and tracking) of the workflows** should be also guaranteed. Finally, MaaC are infrastructured (designed) in such a way to **allow, by construction, open and participatory science** (of which the MBC is one) and they provide tools for doing it easily. .

References

Addor, Nans, Andrew J. Newman, Naoki Mizukami, and Martyn P. Clark. 2017. "The CAMELS Data Set: Catchment Attributes and Meteorology for Large-Sample Studies." *Hydrology and Earth System Sciences* 21 (10): 5293–5313. https://doi.org/10.5194/hess-21-5293-2017.

Beven, Keith. 2016. "Facets of Uncertainty: Epistemic Uncertainty, Non-Stationarity, Likelihood, Hypothesis Testing, and Communication." Hydrological Sciences Journal 61 (9): 1652–65.

Brooks, Frederick P., Jr. 1995. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition*. Pearson Education. https://play.google.com/store/books/details?id=Yq35BY5Fk3gC.

Ceola, S., B. Arheimer, E. Baratti, G. Blöschl, R. Capell, A. Castellarin, J. Freer, et al. 2015. "Virtual Laboratories: New Opportunities for Collaborative Water Science." *Hydrology and Earth System Sciences* 19 (4): 2101–17. https://doi.org/10.5194/hess-19-2101-2015.

Clark, Martyn P., Dmitri Kavetski, and Fabrizio Fenicia. 2011. "Pursuing the Method of Multiple Working Hypotheses for Hydrological Modeling." Water Resources Research 47 (9). https://doi.org/10.1029/2010wr009827.

Cox, R. T.,1946. Probability, frequency and reasonable expectation. American Journal of Physics, 14, 1–13.doi:10.1119/1.1990764

David, O., J. C. Ascough II, W. Lloyd, T. R. Green, K. W. Rojas, G. H. Leavesley, and L. R. Ahuja. 2013. "A Software Engineering Perspective on Environmental Modeling Framework Design: The Object Modeling System." *Environmental Modelling & Software* 39 (c): 201–13. https://doi.org/10.1016/j.envsoft.2012.03.006.

David, Olaf, Wes Lloyd, Ken Rojas, Mazdak Arabi, Frank Geter, James Ascough, Tim Green, G. Leavesley, and Jack Carlson. 2014. "Modeling-as-a-Service (MaaS) Using the Cloud Services Innovation Platform (CSIP)." In International Congress on Environmental Modelling and Software. scholarsarchive.byu.edu. https://scholarsarchive.byu.edu/iemssconference/2014/Stream-A/30/.

Dietrich, W. E., D. G. Bellugi, L. S. Sklar, and Jonathan D. Stock, Arjun M. Heimsath, Joshua J. Roering. 2003. "Geomorphic Transport Laws for Predicting Landscape Form and Dynamics." In *Prediction in Gemorphology*, 135:103–32. Geophysical Monograph.

Formetta, G., A. Antonello, S. Franceschi, O. David, and R. Rigon. 2014. “Hydrological Modelling with Components: A GIS-Based Open-Source Framework.” *Environmental Modelling & Software* 55 (May): 190–200. https://doi.org/10.1016/j.envsoft.2014.01.019.

Gupta, Hoshin Vijai, and Harald Kling. 2011. “On Typical Range, Sensitivity, and Normalization of Mean Squared Error and Nash-Sutcliffe Efficiency Type Metrics.” *Water Resources Research* 47 (10). https://doi.org/10.1029/2011wr010962.

Knoben, Wouter Johannes Maria, Martyn P. Clark, Jerad Bales, Andrew Bennett, S. Gharari, Christopher B. Marsh, Bart Nijssen, et al. 2021. “Community Workflows to Advance Reproducibility in Hydrologic Modeling: Separating Model-Agnostic and Model-Specific Configuration Steps in Applications of Large-Domain Hydrologic Models.” Earth and Space Science Open Archive. https://doi.org/10.1002/essoar.10509195.1.

Martin, Robert C. 2009. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall. https://play.google.com/store/books/details?id=hjEFCAAAQBAJ.

McCuen, Richard H., Zachary Knight, and A. Gillian Cutter. 2006. “Evaluation of the Nash–Sutcliffe Efficiency Index.” *Journal of Hydrologic Engineering*, September, 1–6.

Nativi, Stefano, Paolo Mazzetti, and Max Craglia. 2021. “Digital Ecosystems for Developing Digital Twins of the Earth: The Destination Earth Case.” Remote Sensing 13 (11): 2119.

Nearing, Grey S., Yudong Tian, Hoshin V. Gupta, Martyn P. Clark, Kenneth W. Harrison, and Steven V. Weijs. 2016. “A Philosophical Basis for Hydrological Uncertainty.” Hydrological Sciences Journal 61 (9): 1666–78.

Nearing, Grey S., Frederik Kratzert, Alden Keefe Sampson, Craig S. Pelissier, Daniel Klotz, Jonathan M. Frame, Cristina Prieto, and Hoshin V. Gupta. 2021. “What Role Does Hydrological Science Play in the Age of Machine Learning?” *Water Resources Research* 57 (3). https://doi.org/10.1029/2020wr028091.

Rigon, Riccardo, Giuseppe Formetta, Marialaura Bancheri, Niccolò Tubini, Concetta D’Amato, Olaf David, and Christian Massari. 2022. “HESS Opinions: Participatory Digital Earth Twin Hydrology Systems (DARTHs) for Everyone: A Blueprint for Hydrologists.” Hydrology and Earth System Sciences Discussions, 1–38.

Rizzoli, A. E., M. G. E. Svensson, E. Rowe, M. Donatelli, R. M. Muetzelfeldt, T. van der Wal, F. K. van Evert, and F. Villa. 2006. “Modelling Framework (SeamFrame) Requirements.” SEAMLESS.