A Decade of RAPID-Reflections on the Development of an Open Source Geoscience Code

Cédric H. David, James S. Famiglietti, Zong-Liang Yang, Florence Habets, David R. Maidment

AGU PUBLICATIONS

Earth and Space Science

RESEARCH ARTICLE
10.1002/2015EA000142

Special Section:
Geoscience Papers of the Future

A decade of RAPID—Reflections on the development of an open source geoscience code

Cédric H. David[1,2], James S. Famiglietti[1,2,3], Zong-Liang Yang[4], Florence Habets[5], and David R. Maidment[6]

[1]Jet Propulsion Laboratory, California Institute of Technology, Pasadena, California, USA, [2]Center for Hydrologic Modeling, University of California, Irvine, California, USA, [3]Department of Earth System Science, University of California, Irvine, California, USA, [4]Department of Geological Sciences, Jackson School of Geosciences, University of Texas at Austin, Austin, Texas, USA, [5]UMR 7619 METIS, CNRS, UPMC, Paris, France, [6]Center for Research in Water Resources, University of Texas at Austin, Austin, Texas, USA

Abstract Earth science increasingly relies on computer-based methods and many government agencies now require further sharing of the digital products they helped fund. Earth scientists, while often supportive of more transparency in the methods they develop, are concerned by this recent requirement and puzzled by its multiple implications. This paper therefore presents a reflection on the numerous aspects of sharing code and data in the general field of computer modeling of dynamic Earth processes. Our reflection is based on 10 years of development of an open source model called the Routing Application for Parallel Computation of Discharge (RAPID) that simulates the propagation of water flow waves in river networks. Three consecutive but distinct phases of the sharing process are highlighted here: opening, exposing, and consolidating. Each one of these phases is presented as an independent and tractable increment aligned with the various stages of code development and justified based on the size of the users community. Several aspects of digital scholarship are presented here including licenses, documentation, websites, citable code and data repositories, and testing. While the many existing services facilitate the sharing of digital research products, digital scholarship also raises community challenges related to technical training, self-perceived inadequacy, community contribution, acknowledgment and performance assessment, and sustainable sharing.

Key Points:
- A reflection on the open source development of geoscience codes is presented
- Sharing can be broken down into three phases: opening, exposing, consolidating
- Free online services facilitate sharing and allow for further academic credits

Correspondence to:
C. H. David,
cedric.david@jpl.nasa.gov

Citation:
David, C. H., J. S. Famiglietti, Z.-L. Yang, F. Habets, and D. R. Maidment (2016), A decade of RAPID—Reflections on the development of an open source geoscience code, *Earth and Space Science*, *3*, doi:10.1002/2015EA000142.

Received 13 OCT 2015
Accepted 22 MAR 2016
Accepted article online 7 APR 2016

1. Introduction

Driven by the need to understand Earth's dynamic climate, geoscientists have dedicated much effort to creating numerical models of the major components of the climate system and to analyzing their outputs. Early modeling studies date back to the 1950s and include simulations of the Earth's atmosphere [*Phillips*, 1956], oceans [*Bryan and Cox*, 1967], land [*Manabe*, 1969], and rivers [*Miller et al*., 1994]. Decades later, computer modeling and data-intensive analysis have become key elements upon which modern climate science has been built [e.g., *Intergovernmental Panel on Climate Change*, 2013], and numerous geoscientists therefore dedicate considerable research energy to such endeavors. Computer-assisted research is equally ubiquitous in the broad scientific community, such that some have argued that computer modeling and data-intensive science be considered legitimate pillars of science, hence joining experimental science and theoretical science [*Bell*, 1987; *Bell et al*., 2009; *Hey et al*., 2009; *Hey*, 2010; *Hey and Payne*, 2015], although such a view is not without its critics [*Vardi*, 2010a, 2010b]. Nevertheless, computer modeling and analysis are now integral parts of many geoscience investigations.

The recent mandate [*Holdren*, 2013] requesting that the direct results of federally funded scientific research in the U.S. be made further accessible—including availability of digital data—has spurred much discussion in the scientific community. *Kattge et al.* [2014] argued that while data sharing is necessary, associated hurdles subsist, and proper means of acknowledgment (i.e., citations) are needed so that scientists can benefit from the added burden. This argument was further supported by the survey of *Kratz and Strasser* [2015]. Others have also suggested that the computer codes used to generate or to analyze data are equally important and should hence be made similarly accessible [*Nature*, 2014; *Nature Geoscience*, 2014]. Prior to the recent mandate, *Barnes* [2010] had already advocated for sharing computer code so that—like any other scientific method—code development could benefit from the peer review process. Additionally, the description of computations using only natural language or equations has inherent ambiguities that have unpredictable effects on results; hence, access to the source code is essential to reproducing the central findings of studies

[*Peng*, 2011; *Ince et al*., 2012]. Some of the largest geoscience modeling centers have been openly sharing their software for three decades [e.g., *Williamson*, 1983; *Anthes*, 1986; *Hurrell et al*., 2013] and have already embraced state-of-the-art software sharing practices [e.g., *Rew et al*., 2013]. The lack of training in software engineering and development, however, make such endeavor more challenging for most scientists [*Hey and Payne*, 2015].

River modeling—an area of Earth system modeling concerned with the numerical simulation of water flow waves in surface river networks—does not have a strong tradition of open source codes (as discussed below) and can benefit from further sharing. At scales ranging from continental to global, the river models that are the most widely used in existing literature are perhaps those of *Lohmann et al.* [1996], LISFLOOD-FP [*Bates and De Roo*, 2000], RTM [*Branstetter*, 2001], HRR [*Beighley et al*., 2009], and CaMa-FLOOD [*Yamazaki et al*., 2011]. The degrees at which these software are shared vary widely (see Appendix A for further details). LISFLOOD-FP can be obtained upon request to the developers and is only available in the form of an executable (i.e., not the source code) that is limited to noncommercial use. HRR and CaMa-FLOOD offer a greater degree of openness than LISFLOOD-FP because their respective source codes can be obtained, although this also requires contacting the developers. The code for RTM can be downloaded upon open registration, and that of Lohmann is openly available. Perhaps because their source codes are readily accessible, these two latter river models appear to be the only ones that are currently used by large modeling centers: RTM is an integral part of the Community Land Model [*Oleson et al*., 2013], and the code of *Lohmann et al.* [1996] is used along with the North American Land Data Assimilation System [*Lohmann et al*., 2004; *Xia et al*., 2012].

The generally limited tradition of sharing within the continental to global scale river modeling community makes this field of Earth system modeling a suitable candidate—out of potentially many—for a case study on sharing source code and data, which is the purpose of this paper. We here reflect on the benefits and challenges that have been associated with the open source development of an alternative river model (Routing Application for Parallel Computation of Discharge (RAPID) [*David et al*., 2011d]) based on 10 years of experience since its inception. This paper starts with a short background on RAPID (section 2). We then highlight three consecutive but distinct phases of the sharing process: opening (section 3), exposing (section 4), and consolidating (section 5). The implications for Earth science are then presented (section 6), followed by our conclusions. This paper is intended as a reflective perspective in line with this special issue on Geoscience Papers of the Future.

Our reflection based on the decade-long history of RAPID includes answers to questions that are relevant to sharing any other type of Earth science model: How to share a geoscience model? What are the minimum sharing steps? How to share data? How to make both model and data citable? How to programmatically reproduce an entire study? How to test a model and how to do so automatically? What are the limits of sharing?

Such questions shall be answered if, as a whole, the geoscience community yearns to abide AGU's motto of "unselfish cooperation in research."

2. Background on RAPID

The Routing Application for Parallel Computation of Discharge (RAPID) [*David et al*., 2011d] is a numerical model that simulates the propagation of water flow waves in networks of rivers composed of tens to hundreds of thousands of river reaches. River routing in RAPID is based on the traditional Muskingum method [*McCarthy*, 1938] adapted to a matrix-vector notation [*David et al*., 2011d]. Given the inflow of water from land and aquifers—as commonly computed by land surface models—RAPID can simulate river discharge at the outlet of each river reach of large river networks. River routing with the traditional Muskingum method is performed using two parameters: a temporal constant (k) characteristic of flow wave propagation and a nondimensional constant (x) representative of diffusive processes. Each river reach j of a river network can be prescribed its own set of parameters (k_j, x_j). RAPID includes an automated parameter estimation procedure based on an inverse method that searches for an optimal set of model parameters by comparing multiple model simulations with available observations located across the river basins of interest while varying the Muskingum parameters. Different types of river networks can be used including those based on traditional gridded representations [*David et al*., 2011a] and those from vector-based

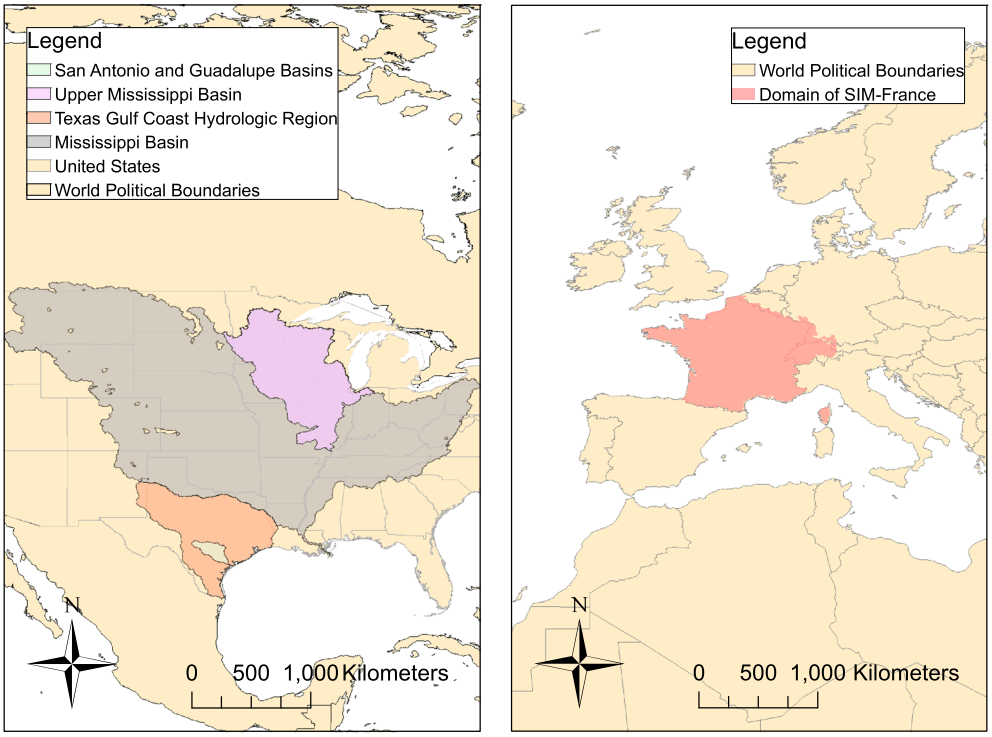

Figure 1. Selected applications of RAPID. (left) The San Antonio and Guadalupe Basins were used in *David et al.* [2011d]. The Upper Mississippi Basin was used in *David et al.* [2011d, 2013a]. The Texas Gulf Coast Region was used in *David et al.* [2013b]. The Mississippi Basin was used in *David et al.* [2015]. (right) The domain of SIM-France was used in *David et al.* [2011a].

hydrographic data sets (the "blue lines" on maps) [*David et al.*, 2011d, 2013a, 2013b, 2015]. River routing and parameter estimation can both be performed on a subset of a large domain in order to study local processes. RAPID has been applied to areas ranging from 10^3 km^2 to 10^6 km^2 (Figure 1) and can be run on personal computers as well as on larger parallel computing machines with demonstrated fixed-size parallel speedup [*David et al.*, 2011d, 2013a, 2015].

The idea of building a river routing model capable of simulating river discharge in large vector-based hydrographic data sets was first discussed with the author in January 2006 (Figure 2; this idea was discussed in a meeting between Cédric H. David and David R. Maidment held on Thursday, 26 January 2006 at the Center for Research in Water Resources of the University of Texas at Austin). The first lines of the RAPID source code were written in September 2007 after a year of initial research. The development of RAPID has been ongoing since then, with a source code now containing over 5000 lines. The code is written using the FORTRAN programming language and three scientific libraries: the Network Common Data Form (netCDF) [*Rew and Davis*, 1990] for the largest input and output files, the Portable, Extensible Toolkit for Scientific Computation (PETSc) [*Balay et al.*, 1997, 2013] for linear algebra, and the Toolkit for Advanced Optimization (TAO) [*Munson et al.*, 2012] for automatic parameter estimation. PETSc and TAO both use a standard for parallel computing called the Message Passing Interface (MPI) [*Dongarra et al.*, 1994].

The community of RAPID users has grown steadily since the software inception a decade ago and currently includes researchers in universities, government agencies, and industry. Notable applications of RAPID include studies of river aquifer/interactions [*Saleh et al.*, 2011; *Flipo et al.*, 2012; *Thierion et al.*, 2012], continental-scale high-resolution flow modeling [*Tavakoly et al.*, 2012], decision support during droughts [*Zhao et al.*, 2014], computation of river height at the regional scale in preparation for an expected satellite mission [*Häfliger et al.*, 2015], nitrogen transport modeling [*Tavakoly et al.*, 2015], reservoir storage simulations [*Lin et al.*, 2015], and operational flood forecasting for the United States [*Maidment*, 2015].

The breadth of the current RAPID users community can be explained—at least in part—by a conscious effort to share the software and related material in an open manner.

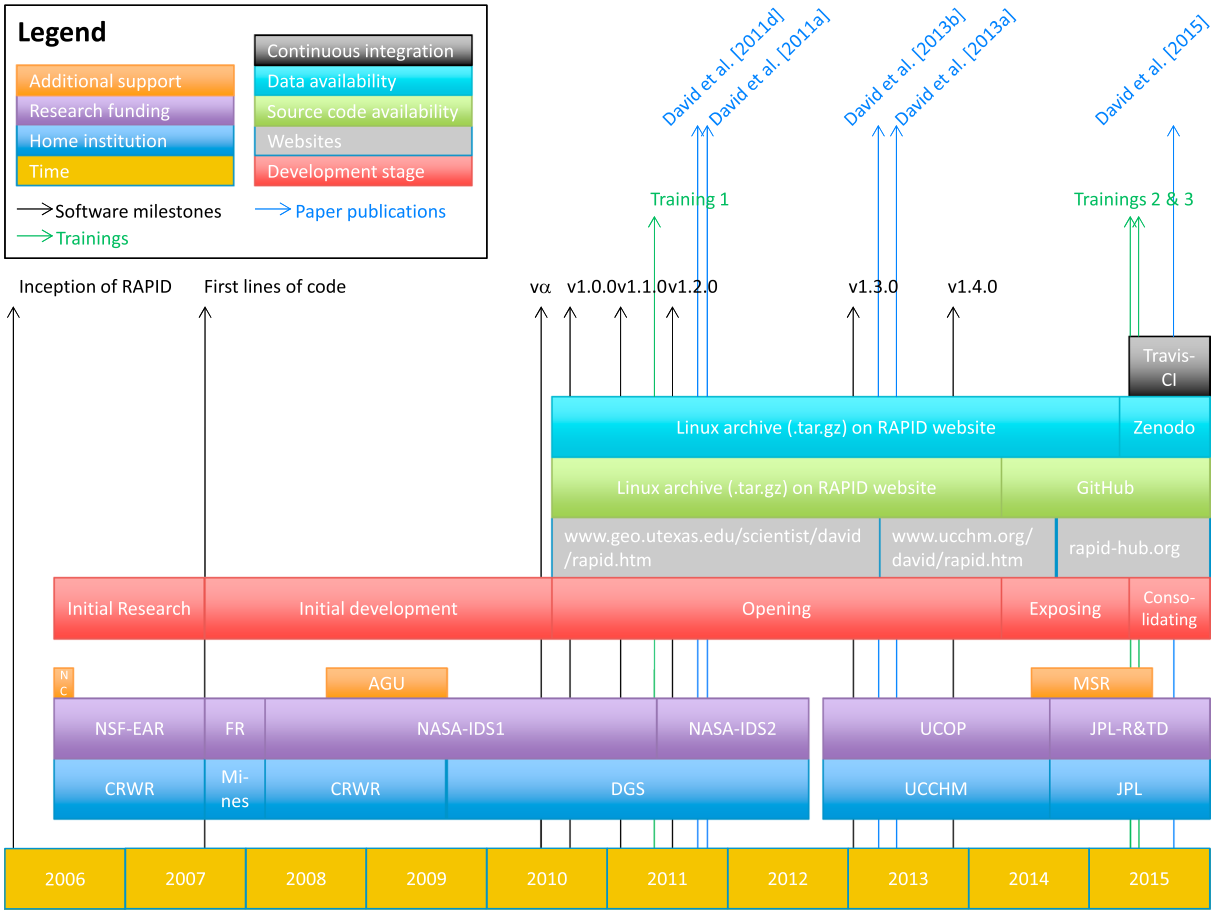

Figure 2. Timeline of the development of RAPID. Note that—with the exception of v1.0.0—the official version numbers all correspond to exact snapshots of the source code used in the writing of published RAPID articles. The v1.0.0 benefited from data model enhancements compared to the alpha version which had been used in *David et al.* [2011d]. The list of acronyms used is the following: Center for Research in Water Resources, the University of Texas at Austin (CRWR), Mines-Paristech (Mines), Department of Geological Sciences, Jackson School of Geosciences, the University of Texas at Austin (DGS), University of California Center for Hydrologic Modeling (UCCHM), Jet Propulsion Laboratory, California Institute of Technology (JPL), U.S. National Science Foundation Project EAR-0413265 (NSF-EAR), French projects VulNAR/PIREN-Seine and Mines-Paristech (FR), U.S. National Aeronautics and Space Administration Projects NNX07AL79G (NASA-IDS1) and NNX11AE42G (NASA-IDS2), University of California Office of the President (UCOP), Jet Propulsion Laboratory Strategic Research and Technology Development (JPL-R&TD), National Center for Atmospheric Research Advanced Study Program (NC), American Geophysical Union Horton Research Grant (AGU), and Microsoft Azure for Research (MSR).

3. Initial Steps of Sharing Geoscience Software—Opening RAPID

While many geoscientists are seduced by the concept of open source software, the lack of knowledge regarding where to initiate the process of sharing seems to be the most commonly acknowledged impediment. The purpose of this section is therefore to expose the three most basic steps involved in open sourcing in order to overcome this initial roadblock.

3.1. Authorship and Licensing of the Source Code

Perhaps the two most important steps prior to actually sharing software are to determine the authorship and the type of license for use. Authorship is key because publication is a prime metric for the scientific community. Software licenses are essential to specify what type of usage the authors allow and to avoid any ambiguity with regards to potential restrictions intended by them. While authorship is commonly valued by academics, license specification in geoscience software is relatively infrequent, either because of lack of computer science training or because the consequences of missing licenses are underappreciated. Yet sharing software without a license is unwise because it implies that software developers leave copyright details unspecified and therefore subject to default applicable laws. Such details include questions related to reproducing, creating derived products, distributing, displaying, and performing of the work [e.g., *US Congress*, 1976].

In the case of RAPID the determination of authorship was fairly straightforward because a unique contributor had been writing the source code. However, some licenses allow for an organization to be named in the license. At the time of the initial release of RAPID, its development had taken place while the author had been in three different departments of two institutions located in two countries; hence, the selection of the organization was a challenge. Fortunately, some licenses allow the author name to be used as the organization name as well, which was the option chosen for simplicity. Decisions concerning authorship and organization were both made with agreement from all scientists involved in guidance and funding during the source code development.

The will to foster collaborations among the institutions that had contributed to the development of RAPID and the desire to encourage broader community use both justified the release of the code in an "open source" manner. Three of the commonly used license types for open source software were considered in the selection: the GNU's Not Unix (GNU) General Public License (GNU GPL) in its version 2.0, the Massachusetts Institute of Technology (MIT) license, and the Berkeley Software Distribution (BSD) 3-Clause license; all of which being accessible from the Open Source Initiative (https://opensource.org/). These licenses all permit private use, commercial use, modifications, and distribution. Additionally, these licenses all include a requirement for inclusion of the license and copyright notice when sharing the source code, and a statement that neither the software author nor the license owner can be held liable for any potential damages. The selection of the license to use for RAPID was made based on the few aspects in which these common licenses differ. The GNU GPL was not chosen because of a requirement to keep the same license when distributing derived works (i.e., copyleft), which was found too restrictive and could have limited potential collaborations with industrial partners. The MIT license does not specifically state that the developers' names cannot be used in potential advertisements over derived products, which made the author uncomfortable. The BSD 3-Clause license was therefore selected for RAPID with agreement from all scientists involved in guidance and funding during the source code development. Note that a full in-depth review of the details associated to available open source licenses is beyond the scope of this paper. However, online services such as http://choosealicense.com/licenses/ or http://oss-watch.ac.uk/ provide helpful information regarding open source licenses and may be valuable to readers when deciding on a license for their own software. Additionally, the book of *Rosen* [2005] offers an excellent in-depth analysis of open source licenses and presents many important aspects that are not mentioned here, including: collective works, work made for hire, and dual licensing.

The specification of both authorship and license can therefore be seen as a key endeavor prior to sharing software because such allows to engage in discussions that help clarifying the intentions and concerns of all parties involved. The clarifications provided by licenses then rule any potential utilization of the software that might otherwise have been unforeseen. The inclusion of the license in the source code then allows for clearer and safer sharing outside of the circle of developers than would otherwise be possible.

3.2. Software Documentation

An important practice before sharing is to document the software. At the most basic level, documenting involves describing the various tasks performed by the program with comments in the source code. Readability is also appreciated by users; hence, some cleaning and formatting of the code can be beneficial. An additional, valuable step in documenting software is to describe the platforms supported (operating systems), programming language, dependencies on other software, steps of installation, and commands (or options) for execution. Such can be done in a users' manual or a short tutorial.

However, writing full software documentation can be very time consuming, and the associated academic credits are currently relatively limited compared to the preparation of a scientific manuscript. In the case of RAPID, software documentation therefore initially focused on commenting, cleaning, and formatting the source code and on preparing a basic tutorial. Further documentation was later prepared on an as-needed basis and improved over time. The "metric" used for determining the need for additional documentation is the frequency of requests for help from different users on any given topic. Current RAPID documentation includes a series of Portable Document Format (.pdf) files with information related to the basic functioning of the model, installation procedures, and example tutorials.

Therefore, despite the widely accepted value of users' manuals, an alternative approach has been taken for RAPID, in which a basic tutorial was initially produced and additional documents have been prepared when needed and improved iteratively.

3.3. Sharing the Software

Once authorship and license are determined, basic formatting of the source code is performed, and minimal supporting documentation is prepared; the source code can be deemed ready to be distributed. At this stage researchers may consider sharing their software with collaborators using the simplest modes of transmission including email or portable storage devices.

Additionally, geoscience researchers might wish to share their software beyond their group of direct collaborators. The most direct way to enable discovery and access to the software (or the supporting information) is to create a website. The initial benefits of sharing the source code and its supporting documents online are twofold. First, a website provides a way for collaborators to access the software and material at any time even if the developers are not available. Second, many potential collaborators might want to assess the capabilities of a piece of software and gauge whether or not it could help their needs prior to contacting the authors.

The first RAPID website was published in July 2010 to promote community usage and facilitate ongoing collaborations and was at the time hosted by the software author's institution. At that point in the development, RAPID had reached a certain level of maturity, demonstrated by its ability to run on two conceptually different types of river networks: a vector-based hydrographic data set [*David et al*., 2011d] and a grid-based network [*David et al*., 2011a]. Note, however, that the corresponding manuscripts had yet to be accepted for publication (Figure 2). The website creation and online sharing of the open source software therefore occurred approximately 3 years after the first lines of code were written but a year before the first papers were published. Various updates to the RAPID code were then routinely made available as compressed Linux archives (.tar.gz) in the download section of the website. Over the years, the site was migrated twice as the employment of the author evolved. The URLs of each website were all published in a series of peer-reviewed papers; therefore, continuity among websites had to be ensured. Such was made possible through automatic HTML redirection of older URLs to the newer ones. The RAPID website (now at http://rapid-hub.org) is still regularly updated and contains a basic introduction, a download page, links to publications, supporting documents, animations of model results, information on training, and contact details.

Unexpectedly to the developer, the most immediate benefits of sharing the RAPID source code and documentation online have been community feedbacks and contributions. Bug reporting from users on applications to various computing environments (e.g., operating system and compiler collection) has enabled the source code to be strengthened. Requests for clarification and reports of mistakes in the tutorials have both improved the documentation. The most gratifying feedback, however, has been when users write their own documents or programming scripts to be shared back to the community of RAPID users. Such contributions are also available on the RAPID website. Note that while focus is made here on classic dedicated websites, online software communities can also provide valuable hosting services [e.g., *Horsburgh et al*., 2015] and sometimes even include the ability to run software online [e.g., *Peckham et al*., 2013]. Finally, this study presents an approach in which the source code is shared only after reaching a certain level of maturity; partly because scientists tend to refrain from sharing raw software [*Barnes*, 2010]. An alternative approach where software is shared from the onset could be equally valuable and potentially lead to similar but earlier benefits.

4. Fostering Sustainable Use and Development of a Geoscience Model—Exposing RAPID

The first few years following the initial sharing of RAPID online were accompanied by a slow but steady growth of its user base. The increasing size of the users community exposed a series of challenges that hindered sustainability of the sharing endeavor. The following describes the difficulties encountered and the steps taken to empower users by increasing transparency. These steps also allowed saving developer time while increasing scientific outreach and furthering academic credit.

4.1. Version Control System and Online Code Repository

Between July 2010, when RAPID was first shared online, and March 2014, a total of 32 successive archives to the source code were made available on the website. For each revision, the entire source code was included in an archive file with a name that included the date of creation. A text file briefly describing the changes made to the code was then maintained (but not shared) by the author. The increasing number of versions of the source code associated with the growth of the RAPID users community made it difficult to support

users; particularly when their applications were based on outdated RAPID code. It became clear then that there was a need for a more advanced code management strategy, including mechanisms for fully documenting and exposing code changes, and for enabling automatic updates by users.

Documenting code changes has traditionally been done using a type of computer science tool called Version Control Systems (VCSs). Notable examples include the Concurrent Versions System (`cvs`), Subversion (`svn`), Mercurial (`hg`), and git (`git`). VCSs have many capabilities including: tracking, commenting, labeling, saving, comparing, and sometimes merging code changes. These systems can be deployed and used by a single person or by a team of developers to track code changes locally. Alternatively, VCSs are capable of interacting with online servers called code repositories, in which case the source code is hosted in a remote location. Examples of code repositories include SourceForge, CodePlex, Google Code, BitBucket, and GitHub.

The increasing notoriety and usage of both `git` and GitHub motivated their use for RAPID. These two tools are designed to work hand in hand and together allow tracking and fully exposing code changes online. The somewhat overwhelming tasks of tracking and documenting changes in all 32 previously released versions of the RAPID code using `git` and publishing the full repository on GitHub were therefore undertaken in March 2014. Despite a steep learning curve associated with understanding the functioning and usage of these tools, the effort turned out to be more natural and less time consuming (approximately 100 h) than initially anticipated. One of the capabilities of `git` that was valuable for this endeavor is the tagging of the source code at any given stage. Tags can then be used to navigate through various snapshots of the software. In the case of RAPID, tags consisting of the previous release dates were used and hence enabled instant retrieval of any of the past versions. Version control for all 32 previous releases was completed on 8 April 2014, date that also coincided with the first code update published on GitHub. The same process has continued since April 2014, and the repository now contains 53 tagged versions including five official releases (see below). The full downloadable repository including all previously released versions of RAPID with tracked changes is available at https://github.com/c-h-david/rapid (Figure 3). This GitHub repository allows obtaining all previous versions of the code, navigating among them, and retrieving any available updates; each of these actions using single-line commands.

Several unanticipated aspects of version control systems and code repositories have also been beneficial to RAPID. Among them is the ability to create official releases of the source code with tags using classic version numbers. To leverage this capability, it was decided to associate official release numbers to the snapshots of the code used in peer-reviewed papers written in the development of RAPID [i.e., *David et al*., 2011a, 2011d, 2013a, 2013b, 2015] with incremental version numbers from v1.0.0 to v1.4.0. A direct link to the latest official release of RAPID is at: https://github.com/c-h-david/rapid/releases/latest. This study will be a candidate for a new official version number. The GitHub official releases are also convenient because they can be assigned a unique citable Digital Object Identifier (DOI) through a data repository called Zenodo (see section 4.2) as was done in this study. Each one of the official release numbers hence now has a citable reference (respectively, *David* [2010, 2011a, 2011b, 2013a, 2013b]). Another advantage of code repositories is the ability to browse through the various versions of the source code online; which is helpful for training, for debugging, and for electronic communications. Also of note are the social media aspects of code repositories that allow users to be kept appraised of new software developments. Incidentally, industrial partners started manifesting their interest in RAPID after it was version controlled and published in a code repository, perhaps because these joint efforts contributed to making the software more professional.

The combined use of version control systems and code repositories for geoscience software therefore allows levels of transparency in code changes and accessibility to updates that are not permitted by the more common sharing of snapshots of the source code. Together, VCSs and code repositories can help foster further community use while lightening the user support load and increasing author acknowledgment through citable references.

4.2. Sharing the Data

Over the years of development of RAPID, the size of all input and output files used in associated peer-reviewed publications has grown. These data sets have been stored on various machines and their cumulative size approaches 200 GB. Such data can fill a large portion of the total storage capacity of current personal computers and therefore represent a potential issue for data storage and data preservation. Additionally, these files provide examples of the inputs necessary to run RAPID (and of the outputs it produces) and can

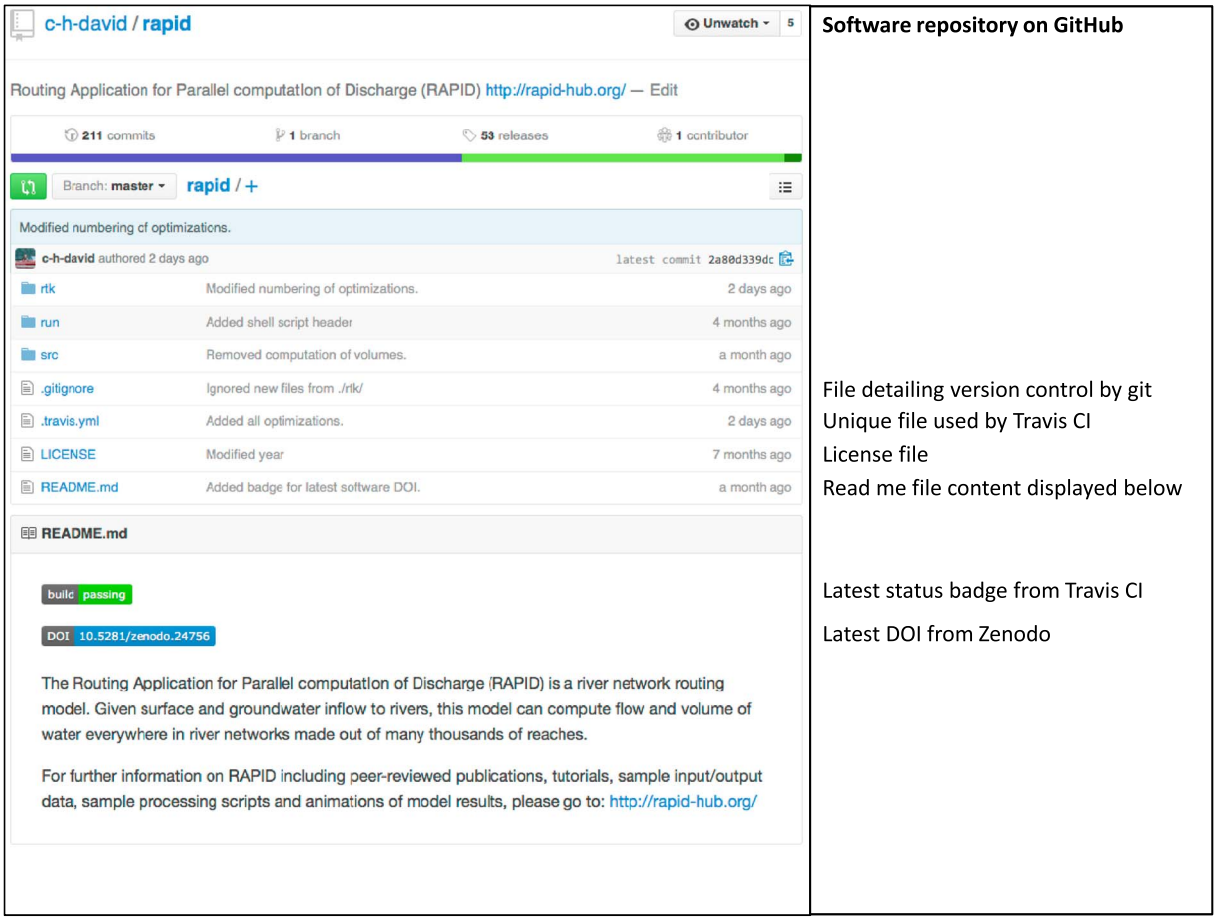

Figure 3. The GitHub repository for RAPID is available at http://github.com/c-h-david/rapid and offers direct access to all previous releases of the source code, including official versions. The successive modifications of all files are fully tracked and documented. The status badges in the README file automatically link to the pass/fail state of the latest automatic build, and to the latest citable version.

therefore serve as the basis for training material. Finally, such files can be used to check that potential new modifications to the software have not altered its overall functioning (i.e., software testing). There are therefore many aspects for which the sharing of files associated with previously published RAPID studies can be valuable, including data preservation, software training, and software testing.

As with *software* sharing, the choice of authorship, license, and repository are at the base of *data* sharing. The most widely used data licenses appear to be those of the Creative Commons (CC). CC licenses and their differences share many commonalities with the various software licenses (section 3.1) and specify details related to attribution, derivatives, distribution, and commercial use. Readers may find the online service for guiding the choice of a Creative Commons license (https://creativecommons.org/choose/) helpful when picking a license for their data. Note that the selection of a data license is a requirement before sharing data on online repositories. Three repositories currently seem to be the most popular for scientific data sharing: FigShare (http://figshare.com/), Dryad (http://datadryad.org/), and Zenodo (https://zenodo.org/). These repositories vary in their cost and storage capabilities, but all provide a unique Digital Object Identifier (DOI) making the data fully citable in peer-reviewed publications. Data repositories (like software repositories) also allow for the inclusion of a short description of the files.

The input and output files associated with RAPID simulations consist in a series of small Comma Separated Variables (csv) files and of larger netCDF files. The csv files contain information including river network connectivity, Muskingum model parameters, existing observations, and subbasin specifications. The netCDF files contain the inflow of water from land and aquifers into river reaches and the outflow of water from the river reaches.

The input and output files corresponding to peer-reviewed publications of RAPID are usually either generated from scratch or prepared based on off-the-shelf data sets or on files prepared by coauthors. Generally, the amount of work involved in the choice of data sources, data preparation, or data transformation is well aligned with the authorship of the associated papers. It was therefore decided that the authorship of the data sets used in the preparation of existing peer-reviewed RAPID publications would mirror the authorship of the corresponding papers.

The details associated with each of the few Creative Commons licenses vary from very permissive to very restrictive. Similarly to the choice of software license for RAPID, the selection of data license was made to maximize community usage. The Creative Commons Attribution (also known as CC BY) license was chosen for RAPID-related data sets as it allows for distribution, modifications, derived works, and commercial use. The CC BY license is the most accommodating of the Creative Commons licenses, and its only requirement is to credit the original author(s) of the work.

FigShare, a free data repository, is probably the most popular of the available data sharing services, but its current 250 MB limitation on any file in each data set makes it impossible to share RAPID files (often of larger than 1 GB). Dryad only accepts files corresponding to peer-reviewed publications, which would satisfy our needs, but their service is not free, and data publication charges increase steeply once the size of each data set goes beyond 10 GB (here again common in RAPID data sets). Zenodo was therefore chosen to host data associated with our peer-reviewed publications as it allows for many large files (each up to 2 GB) and remains free. Zenodo is supported by the European Center for Nuclear Research and appears to be functioning under stable funding. Furthermore, Zenodo guarantees survival of data sets and of their DOI. The files corresponding to the first two peer-reviewed articles using RAPID [i.e., *David et al*., 2011a, 2011d] were published in two separate Zenodo repositories (respectively, *David et al*. [2011b, 2011c]) for the purpose of this study. A short note (1–2 pages) included in each data publication summarizes the sources for all raw data used and explains the content of each file as well as how it was prepared.

In addition to fulfilling recent requirements for increasing access to funded research [e.g., *Holdren*, 2013] data publication is therefore beneficial for data preservation and storage, for training, and for software testing. Perhaps more importantly for geoscience researchers, data publications can be officially cited, resulting in potential academic credit for authors [*Kratz and Strasser*, 2015], although the related benefits are complex (section 6.3).

4.3. Training Courses

To date, three training courses have been organized for RAPID (Figure 2). The first training course was held at the Institute of Atmospheric Physics of the Chinese Academy of Sciences in Beijing, China, on 24–26 May 2011. The second training course was part of the Community WRF-Hydro Modeling System Training workshop held at the National Center for Atmospheric Research in Boulder, Colorado, on 5–7 May 2015. The third training course was organized during the National Flood Interoperability Experiment Summer Institute held at the National Water Center in Tuscaloosa, Alabama, between 1 June 2015 and 17 July 2015. Each one of the training courses enabled further growth of the users community. Additionally, each training course allowed the evaluation of the ease of use of RAPID and of the quality of the tutorials. Note that while the first two training courses were taught by the lead software developer, the instructor of the third training was instead a member of the RAPID users community. The third training course hence marked a transition when RAPID users started teaching themselves, which can be seen as a milestone in user support. The organization of training courses therefore allows for the evaluation of software usability, but can also be seen as an integral part of community growth and engagement.

5. Facilitating Updates of a Geoscience Model—Consolidating RAPID

Inevitably, with years of development, the complexity of the source code for RAPID increased. Additionally, as RAPID reached a certain level of maturity, the users community gradually became more active, and the frequency of requests for updates, upgrades, and bug fixes grew. At this stage in model development, the need for regular testing of the code after each modification became crucial to avoid the risk of becoming inundated by bug reports from users. It followed that a more efficient approach for model testing was

needed to avoid being overwhelmed by user support activities, while sustaining community engagement and entertaining user requests.

5.1. The Transition From Manual to Automatic Testing

The many steps involved in testing software are often repetitive, tedious, and prone to human errors. The most basic testing steps consist of running the program with a given set of instructions and verifying that the outputs that are generated are as expected. If the source code and/or example data are not available locally, testing also involves downloading of a series of files. Additionally, when the software depends on other programs or libraries, their installation is required (see section 5.3) prior to the creation of the program executable from the source code. As the frequency of software updates increases, full testing can become a great consumer of developers' time and is therefore sometimes avoided, increasing the risk of releasing faulty source code.

The testing of RAPID—as that of many other geoscience models—involves all the aforementioned testing steps. When performing testing operations manually, many of these steps are achieved using Graphical User Interfaces (GUIs), i.e., the most natural tools for human-machine interactions. Downloading of the code and data is done using an Internet browser or a File Transfer Protocol client. The instructions for each simulation are created and modified manually using a text editor. Model outputs are checked visually by reading (for text outputs) or inspecting graphics (e.g., hydrographs created from binary outputs). However, despite the value of GUIs in easing human-machine interactions, the automation of tasks performed in GUIs is a challenge.

In most operating systems, text-only interfaces called shells allow users to request tasks from the computer by typing commands. Shells are the most fundamental way in which users can interact with the system. Actually, shells used to be the main tools for interactions with computers before the advent of GUIs. Each command entered in the shell results in an action performed and/or in text output. The success (or failure) of each action is then summarized by an integer number called exit code generated after each command execution, although it is kept hidden to the user unless specifically requested. A series of consecutive operations can therefore be easily automated by creating a text file containing the corresponding commands (i.e., a shell script), and their respective success can be checked using exit codes. Hence, the automation of model testing can be accomplished if all the steps involved can be summarized in a series of simple commands and included in a script.

Despite a few years of programming experience gained in the development of the RAPID source code, the author was not familiar with the many command line tools allowing such automation. However, after overcoming the initial learning curve associated with the discovery and use of a series of these tools, it was found that many of the steps involved in the testing of RAPID can be performed directly from the shell using existing programs (see the Appendix B for examples on Linux). The few tasks that could not be directly performed with off-the-shelf tools were specifically related to the outputs of RAPID and necessitated the creation of ad hoc programs.

As mentioned earlier (section 2), two main modes of usage currently exist for RAPID: the first consists in performing flow simulations, and the second is dedicated to parameter optimization. When simulating flows, RAPID generates a netCDF file containing the discharge for each river reach and at each time step. When checking simulations manually, hydrographs are created based on output files and verified visually. In order to allow for automated testing of simulations, a new approach was needed to compare two separate netCDF files and return an exit code for success if the files are similar. The issue here is that different computing environments can lead to slightly different results (generally on the order of 10^{-37} m^3/s in RAPID simulations) due to variations in ordering of floating-point arithmetic operations. A bit-to-bit comparison of files or of their corresponding floating points was therefore not possible. A FORTRAN program for testing the similarity of two RAPID output files was hence created to automatically compare among all simulated discharge values with an option for specifying acceptable absolute and/or relative tolerances. When run in parameter optimization mode, RAPID generates a text file including all parameter values tested and the corresponding cost functions obtained, along with the final optimized parameter values. However, because the optimization method used is unconstrained, the parameters found automatically are sometimes void of physical meaning in which case one has to handpick the best possible parameters from the list of physically valid values. Additionally, given

the dependence of the search space on the initial values used at the start of an optimization procedure, comparisons among a series of optimization experiments help selecting best possible parameters. Finally, these best possible parameters need be compared with previous results to perform software testing. Three automatic tasks were therefore needed to test the optimization procedure in RAPID: (1) finding the best valid parameters in a given optimization experiment, (2) picking the optimal parameters among a series of experiments, and (3) comparing these optimal parameters with previously computed values. Three shell scripts using off-the-shelf Linux programs were prepared to perform these three tasks. These programs all consist in a series of text editing tasks that were all made possible by combining existing Linux tools (see Appendix B).

Overall, the greatest challenge in creating programs for testing of RAPID was in the discovery of available command line tools. Once this learning curve was overcome, the automation process became straightforward. Despite their value for automatic testing (section 5.2), the custom programs that were created also turned out to be beneficial for day-to-day usage of the software. Indeed, the few manual tasks involved in basic postprocessing of RAPID outputs were time consuming and prone to human error. The development of automatic tools for testing, albeit motivated by user support, therefore turned out to be a valuable endeavor for the developer as well.

5.2. Testing by Reproducing Entire Studies Programmatically

Once the programs that are necessary for automatic testing of a piece of software are prepared, the actual process of designing tests can begin. The choice of what tests to be performed therefore had to be made. In the case of RAPID, the creation of tests was motivated by the desire to check that any potential modification of the source code did not alter the overall functioning of the model. It was therefore decided to test software updates by automatically reproducing all model runs that had been performed in previously published studies. The first two RAPID papers [i.e., *David et al*., 2011a, 2011d] were selected for this endeavor. For each study, two main shell scripts were created: one for downloading the corresponding data and the other for reproducing and checking all model runs, hence fully exposing the provenance of past results. Creating programs that automatically download data is fairly straightforward if the data sets are published online (section 4.2) and can be done using off-the-shelf command line tools. The design of programs that automatically reproduce a series of model runs requires further effort. Because the generation of an executable from the source code (i.e., software build) can be time consuming, it is important that the code is designed so that all model options can be accessed at runtime (without rebuild). Such capability is usually achieved through the use of an input text file containing instructions. Several model runs can then be performed automatically solely by modifying the instruction file prior to execution. Finally, programs for resetting each instruction file to its respective default state (chosen arbitrarily) turned out to be useful when launching multiple model runs successively.

The programs that were created for the automatic reproduction of the first and second RAPID papers together combine 138 tests (23 and 115, respectively). The number of tests allowed by these programs is about an order of magnitude greater than what was used when performing manual testing and approximately an order of magnitude faster. Such a stricter testing procedure allowed for both increased robustness of the software and for temporal savings. Here again, the value of these tests goes beyond their use for development work, as they provide examples of model usage as well as a means for users to check their installation (or modifications) of RAPID. The tests corresponding to the first RAPID paper were actually run successfully by 25 attendees of the second RAPID training (section 4.3). Finally, the automatic reproduction of existing studies fully document the provenance of all simulations performed in the corresponding studies.

5.3. Continuous Integration

The principal strength of automatic tests is that they allow checking that the piece of software functions as expected *after creation of the executable* (build). However, geoscience models are often built upon existing software, all of which being already installed on the developer's computer. The only way for developers to ensure portability of their geoscience model—i.e., to verify that there are no any unexpected dependencies on their own system—is to install the model on a blank machine. This machine can be a dedicated computer requiring frequent reinstallation of the operating system or a less costly and less time consuming virtual machine (VM) that can be regularly reset using VM snapshots. Another approach to

guaranteeing portability, but does not require manual resetting of the machine (actual or virtual), is made possible by hosted Continuous Integration (CI) services.

The purpose of hosted CI services is to monitor changes to a source code repository and perform a set of given tasks upon publication of updates. The list of tasks to be performed is provided in a text file that is specific to the CI service and that is included in the source code. The most basic task performed by the CI server is to build the software using the instructions provided. Any time that an update is published on the code repository, the CI server automatically creates a clean operating system, downloads the source code, and builds the software.

Several services currently exist for hosted continuous integration, including Travis CI, Codeship, and Circle CI. These services share many commonalities including the capacity to interact with GitHub repositories. Travis CI was chosen here for RAPID because of its relative higher popularity, its tight coupling with GitHub (using the same identifier and password), and its free support of open source projects. Surprisingly, activating Travis CI for RAPID turned out to be a very straightforward two-step process. The first step consists in adding a text file called `.travis.yml` to the GitHub repository including the series of shell commands necessary to install RAPID and its dependencies. These commands were already summarized in one of the existing tutorials, which eased this procedure. The second step consists in logging in on Travis CI using GitHub credentials and activating the monitoring of the RAPID GitHub repository. The result of this two-step process is available at https://travis-ci.org/c-h-david/rapid (Figure 4) and includes information on all previous builds of RAPID since the continuous integration process was activated. Of particular interest here is the pass/fail status of the latest build.

Despite demonstrating the portability of the RAPID source code through automatic building of the software, one of the direct benefits of hosted continuous integration was to fully trace the software dependencies and environment variables that are needed to build RAPID. A successful build on a CI server guarantees that all necessary software were properly described and therefore that the installation procedure was fully documented. Setting up continuous integration for RAPID also proved to be immediately valuable as it allowed for illumination of some dangerous programming practices present in the source code (e.g., lack of variable initialization and implicit data type conversions). These weaknesses were exposed because the compiler collection used in the CI server provided different warnings than that of the development machine. Continuous integration therefore had an immediate impact on the quality of the RAPID code. Another benefit of continuous integration is the ability to advertise for the validity of the latest releases through a "status badge" confirming that the latest build was successful (Figure 3).

In addition to enhancing the portability and quality of the code, continuous integration helped with automatic testing of RAPID. Once automatic tests were included in the code repository, their use in the continuous integration process was easily activated through inclusion in the CI server instructions. The only difficulty associated with testing within the CI server was to ensure that the integration process—i.e., the combination of software building and testing—could be performed in a limited period of time (50 min for Travis CI), so tests had to be split into smaller groups running on 15 different Travis CI computers. Exit codes being the sole means of the CI server to determine success (or failure) of each command, their proper handing within testing scripts (section 5.2) was particularly key for continuous integration. The benefits of the temporal investments made in sharing code, data, and in preparing automatic tests therefore became even greater when using continuous integration, as they together allowed for machines to take over many of the time-consuming parts of code development.

6. Implications for Earth Science—Lessons Learned

6.1. On the Phases of Sharing and Their Associated Benefits

The first phase of sharing, i.e., "opening" a piece of software, consists in a series of consecutive steps. Perhaps the most important step in this phase is to determine the authorship and license hence clarifying potential ambiguities on permissions and restrictions intended by the authors prior to release. Software description —through cleaning, formatting, and commenting the code, and/or through preparation of a basic tutorial —allows potential peers to independently start using the software and evaluating the associated capabilities. Note that our recommendation for minimal initial software description is marginally more time consuming

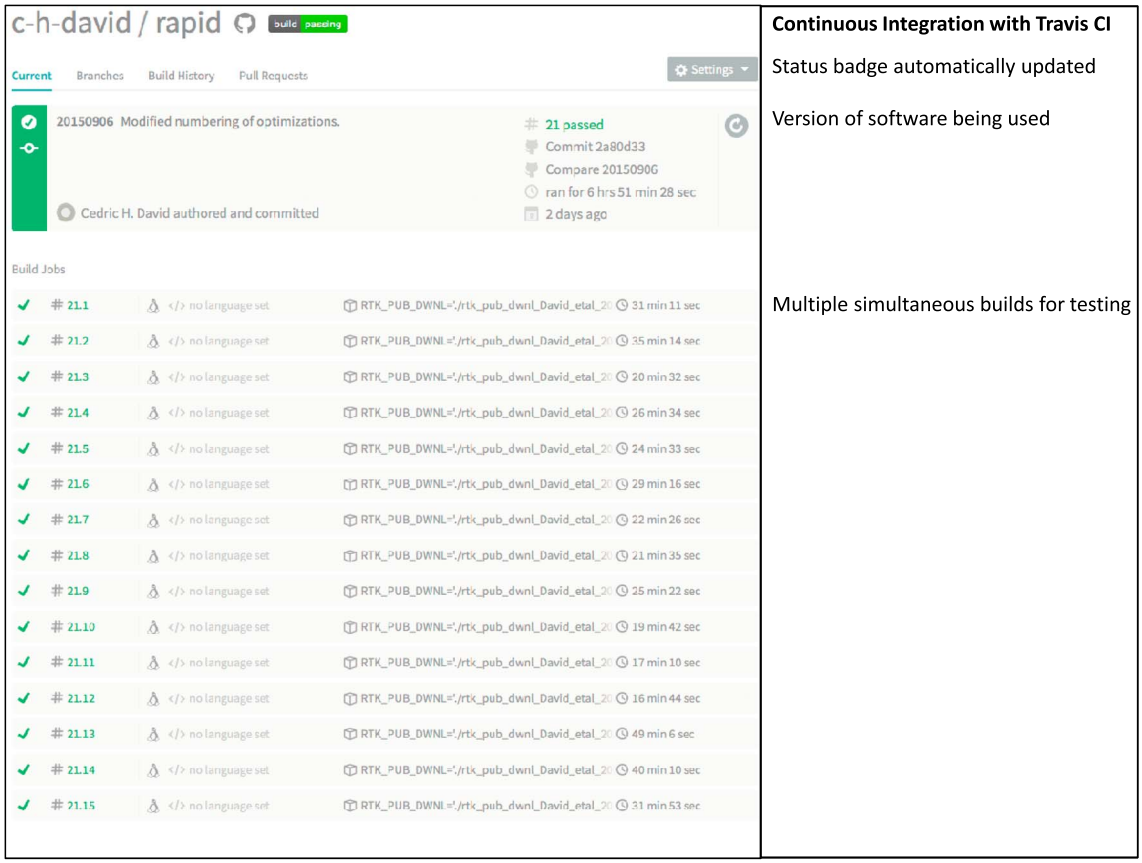

Figure 4. The continuous integration server for RAPID is available at http://travis-ci.org/c-h-david/rapid and provides details on all previous builds of the software. The continuous integration of RAPID is currently running on fifteen separate Travis CI workers. Note that the build time provided is the sum of the run times of all workers and that the wall clock time is shorter as several workers run concurrently.

than sharing code "as is" [e.g., *Barnes*, 2010]. The creation of a website is then the natural next step to provide continuous access to (and a means of discovery for) the software and associated documents; which together permit the fostering of a burgeoning community of users. One of the benefits of opening software, albeit not easily quantifiable, is the satisfaction one may get in having their research used by peers: hence contributing to one's community. In addition, opening software has direct benefits for the code itself. Initially, community feedback on potential bugs in the code and clarity of the code and tutorials are to be expected. Eventually, community contribution to the software knowledge base (tutorials and processing scripts) are even more rewarding.

The second phase of sharing, i.e., "exposing" the software and data, consists in using version control systems (VCSs) to track code changes, in publishing code and data through online repositories, and in organizing training courses. VCSs allow documenting code changes, labeling versions, comparing snapshots, and merging differences. Code repositories are companions to VCSs and facilitate browsing through code, increasing the transparency of changes, publishing official versions, and increasing accessibility to updates. The social media capabilities of code repositories also allow users to be kept appraised of software developments. Online availability is equally helpful for discussing specifics of the source code with remote users. Data repositories are valuable for storage and preservation, but also because sharing example data provide useful training material and later support automatic testing. Perhaps more importantly [e.g., *Kattge et al.*, 2014; *Kratz and Strasser*, 2015], data repositories enable potential furthering of academic credit through citable material (for both code and data) while fulfilling recent requirements in accessibility to direct products of funded research [*Holdren*, 2013]. Code and data sharing are particularly valuable once a small community of users exists. Training courses allow for the evaluation of software usability and are also an integral part of user engagement. Exposing software empowers users through eased access and usability and hence contributes to the growth of the users community while simultaneously saving developer time therefore making the

sharing process more sustainable. Finally, this phase demonstrates a certain level of maturity in the software and thus helps attract more users including potential industry partners.

The third phase of sharing, "consolidating" software, becomes necessary as the community of users continues to grow and the frequency of requests for code modifications increases. An active user base is very valuable because not only it generates motivation for software improvements but also it demands rigorous testing of updates at the risk of being inundated by bug reports. Manual testing—a repetitive, tedious, and time-consuming task that is prone to human error—is then no longer appropriate because the associated temporal investments become an impediment to user support and hence to sustainable sharing. At such stage of open source development, the creation of automatic testing tools and the activation of hosted continuous integration become necessary. The automatic tests facilitate tremendous savings in developer time and enable users to check their installation and/or potential code modifications. Continuous integration enhances software portability through ensuring a full and up-to-date description of software dependencies. Together, continuous integration and testing eventually allow saving time, strengthening the code, and benefit day-to-day operations. From a developer's perspective, continuous integration enables a more sustainable approach to software development through letting machines take over many of the time-consuming development tasks and therefore frees up availability to further community engagement.

The three phases of open source sharing that are presented here are incremental and align with the size of the users community. Therefore, while every phase enhances the sharing endeavor compared to its predecessor, each implementation can be spread out over the lifetime of the software (Figure 2). Note that the completion of the various steps in each phase (or of the phases themselves) can sometimes be made in a different order than that proposed here. For example, continuous integration can be implemented before automatic testing, or data can be published before using VCSs and code repositories. Likewise, VCSs can be used prior to sharing the software as can be needed if several developers are initially involved. However, we highly recommend the selection of a license—a step that is too often overseen in practice—before sharing.

6.2. Implications for Source Code Development

Our experience with the open source development that is presented in this study has highlighted three aspects of software design that are important to the sharing process: the necessity of a strong data model, the importance of instruction files, and the choice of the hosting location for material linking data and software.

The data model is the standard used to describe the content and format of the various files read or created by the Earth science model, along with how they relate to one another. This description includes the name and content of variables accompanied by their computer-based representation (i.e., character, logical, integer, floating point, and associated precision), the sorting details (e.g., arbitrary, ascending, and descending), and the file type used to store the data (e.g., csv and netCDF). The data model has a direct impact on the functioning and performance of an Earth science model and is therefore usually defined before or at the same time, but it is almost always refined with time. One should hence actively promote early data model stability for model development. Additionally, from a data sharing perspective, one should also wait for some stabilization of the data model before publishing data sets at the risk of making such data sets hastily obsolete. A stable data model is also advantageous because it facilitates the consistency of tutorials and automatic tests. In our experience, a good metric for determining the maturity of the data model has been its ability to accommodate different types of conceptually different inputs.

At the initial stage of development, many variables are often "hardwired" in the Earth science model source code, be it because of a strive for early results standing against recommended programming practices or perhaps because such can sometimes facilitate the detection of programming inconsistencies (e.g., array sizes) by compilers. This advantage comes at a price: the source code needs be rebuilt every time new instructions are used. As the software matures, it is therefore good practice to combine all possible instructions in a single text file that is read at runtime. The instruction file then enables multiple simulations to be run without rebuilding the source code, which greatly eases day-to-day operations and automatic testing.

Some files create a link between the source code and the data of an Earth system model. Examples include the data downloading script (which contains the names of all input and output files) and the instruction file (which can also contain their respective sizes). In the process of sharing source code and data, one may

therefore wonder which of the associated repositories is the most appropriate to store these specific files. Our experience has shown that it is helpful to include the data download scripts with the source code so that users can readily obtain example data to check their installation and/or modifications of the code and in order to ease the continuous integration process. Additionally, while the file names and sizes corresponding to a set of example input/output files tend to remain stable, the associated variables names within the source code might evolve (e.g., to improve readability) in which case old instruction files would no longer be applicable. It is therefore good practice to also keep example instruction files with the source code and not with the data.

6.3. Remaining Community Challenges and the Limits of Sharing

As geoscientists further embrace digital scholarship, a series of community challenges are to be expected. These challenges include matters related to the following: technical training, self-perceived inadequacy, community-based documentation, acknowledgment and assessment of digital scholarship, and sustainable sharing.

As mentioned earlier (section 3), geoscientists seem to often justify the lack of sharing by a lack of know-how, at least in conversation. This paper touches on many aspects related to code and data sharing including licenses, repositories, versioning, testing, and continuous integration. While these subject matters are commonly taught in computer science departments, they are typically absent from most geoscience departments. Such a lack of training has now been recognized by computer science colleagues [e.g., *Hey and Payne*, 2015]. If digital scholarship is expected of geoscientists, it must also become part of their curriculum in universities.

Another common justification for the lack of sharing seems to be that computer codes created are too simple (or "not good enough") to be made public. This apparent self-perceived inadequacy was humorously discussed by *Barnes* [2010] who argued that if the code does the task that it is designed for, then it is good enough to be shared. Our opinion is well in line with that of Barnes, as too many of we geoscientists spend much of our time writing code that others have written before, and more will write in the future. While much can be learned writing one's own code, having access to examples could lead to community-wide temporal savings.

The need to document software can also be seen as an impediment to sharing. Preparing documentation and keeping it up to date with latest code developments is indeed time consuming. While some argue for sharing as is [e.g., *Barnes*, 2010], our experience has shown that if there is in fact a community need for the tools, users are likely to manifest themselves with questions. We therefore recommend here a limited amount of editing and formatting of the source code—as was already advocated for by *Easterbrook* [2014]—and suggest that the preparation of a small tutorial suffices as initial supplementary documentation. Such an approach was taken for RAPID and was later rewarded by contributions to the documentation from enthusiastic members of the users community.

This case study suggests that the many steps involved in sharing together require substantial dedication; particularly as the users community grows. In the words of *Easterbrook* [2014], "making a code truly open source […] demands a commitment that few scientists are able to make." The geoscience community should therefore acknowledge sharing efforts in a way comparable to traditional scientific article publications [e.g., *Kattge et al*., 2014]. Fortunately, modern technology now allows for digital products to be fully citable (see sections 4.1 and 4.2), which is a significant step toward acknowledging contributions [*Kratz and Strasser*, 2015]. However, some scientific software eventually grow beyond the publishing scientific community, in which case an alternative measurement of their broader impact would be valuable. Unfortunately, there does not appear to be a way to track downloads in current data or software repositories. GitHub does provide a download count, but this capability is limited to the binary files associated with official releases which only represent a portion of the total number of downloads. The lack of this capability is perhaps why many developers wish to be contacted by users before granting access to their code. This might also explain the registration systems used by some modeling centers to track the usage of their codes [e.g., *Hurrell*, 2013]. Further, while we agree on the many benefits of citable digital research products [*Kattge et al*., 2014; *Scientific Data*, 2014; *Kratz and Strasser*, 2015] we further argue here that an additional step is needed to promote open research: a cultural shift in the assessment of research performance. Digital scholarship can be

seen as impactful for research, education, and outreach, and researchers might respond more enthusiastically to the added burden if their peers (i.e., colleagues and tenure and promotion committees) valued the efforts. This expectation in turn means digital scholars must become advocates of their own cause.

Finally, and contrary to common belief, open source software does not mean free user support [*Barnes*, 2010; *Easterbrook*, 2014]. This unfortunate misconception hinders sustainable sharing as it does sustainable research. An analogy between traditional publishing and digital scholarship can be made here. It is common that a given researcher *X* reads a scientific paper written by another researcher *Y*, applies the published methods to his/her case study, and writes their own paper citing the work of *Y*. However, it is less usual for *X* to ask *Y* for help with the data collection or application of the methods to the new case study without an implied understanding of coauthorship. Such is particularly true if the associated efforts require rigorous data collection, detailed data inspection, and/or enhancement of the methods. The same *modus operando* can reasonably be applied to digital scholarship. Citation of the digital research products is appropriate—and sufficient—when using these products as is. However, if user support requires "substantial" expertise or involvement from the developers, coauthorship seems appropriate. Similarly, if research proposals planning to use open source software are likely to necessitate assistance from the developers, a proportionate amount of funding can reasonably be requested. Such funding can then be used to answer new scientific questions and leveraged for support. Developers must therefore acknowledge that they too often drown—happily—in the time sink of user support. The benefits of community feedback cannot alone justify the associated efforts as developers' time could be very well spent instead on new publications or new research proposals. As we encourage geoscientists to enthusiastically embrace the open source approach, our community must therefore also strive for a proper balance between further sharing and sustainable research.

7. Conclusions

As geosciences gradually evolve to rely on increasing amounts of computer-aided methods and our society clamors for further transparency in the products of the research it supports, many geoscientists are faced with the challenges of digital scholarship. The importance of learning best sharing practices is particularly acute in the general field of Earth science modeling—i.e., the creation, update, and maintenance of numerical models used to study the dynamic elements of the Earth—which is a key component of current climate change studies. This paper focuses on the specific field of continental to global scale numerical modeling of flow wave propagation in rivers, one of potentially many scientific areas in which open development is uncommon, merely as an avenue to reflect on the open sharing process in Earth science.

This study presents reflections based on the 10 years of development of an open source river model called RAPID and highlights three consecutive but distinct phases of the sharing process: (1) opening, (2) exposing, and (3) consolidating. Each of these phases responds to a users community of increasing size. Phase 1 (opening) consists in selecting an open source license, cleaning and formatting of the source code associated with preparing a short tutorial, and publishing the source code on a dedicated website. This first phase is the minimal sharing phase and allows fostering a burgeoning community of users. Phase 2 (exposing) consists in tracking code changes using Version Control Systems (VCSs), publishing the source code and example files on citable software and data repositories, and organizing software trainings. This second phase becomes necessary as the user community grows, in order to facilitate transitions between existing versions of the source code and to provide example case studies. Phase 3 (consolidating) consists in creating a set of automatic tests and activating continuous integration of the software. This third phase becomes needed to ease the sharing process when the frequency of requests for software modifications increases along with the activity of the users community. Note that while the phases presented here mirror the RAPID development timeline, the order in which each phase (or its associated steps) is completed can vary among software projects.

The case study herein provides details on the several phases involved in open sourcing of a numerical model of a component of the Earth system and highlights the many benefits—mutual to developers and to users—of open source development. In addition to contributing to transparency in science, open source sharing allows for (1) improvement of software and documentation, (2) temporal savings through letting machines take over many of the repetitive aspects of software use and development, and (3) academic credit through citable digital research products. Note that all the services used in this study are available *at no cost* to open source developers.

However, the benefits of further sharing also come with a substantial time commitment as the sharing process becomes increasingly demanding with a growing user base. This added burden must be managed by the geoscience community as a whole; and several potential avenues are proposed here. First, the inclusion of digital scholarship methods in the geoscience curriculum of universities could greatly ease the associated learning curve. Second, geoscientists need to overcome the self-perceived inferiority of their computer code and instead embrace the many benefits of peer review for code development. Third, users of open source software should consider contributing to their own community through helping with the associated documentation. Fourth, traditional scholarship and digital scholarship should equally weigh in the acknowledgment, evaluation, and promotion of geoscientists, because digital scholarship equally impacts research, education, and outreach. Finally, the geoscience community might consider including open source developers in their peer-reviewed manuscripts and research proposals when making substantial use of the developers' expertise in their endeavors in order to foster sustainable sharing practices.

Appendix A: On the Apparent Level of Sharing in Selected River Models

The information below was retrieved at time of writing (14 August 2015) for selected river models with applicability from regional to global scale:

The code of *Lohmann et al.* [1996] is available for download from as part of the North American Land Data Assimilation System and does not appear to include a license: http://www.nco.ncep.noaa.gov/pmb/codes/nwprod/nldas.v2.0.3/sorc/nldas_rout.fd/

The website for LISFLOOD-FP [*Bates and De Roo*, 2000] states, "we are happy to provide a copy of the executable for noncommercial studies": http://www.bristol.ac.uk/geography/research/hydrology/models/lisflood/downloads/

RTM [*Branstetter*, 2001] can be downloaded from the Community Earth system model [*Hurrell et al*., 2013] website at http://www.cesm.ucar.edu/models/cesm1.2/, which states that "a short registration is required to access the repository."

The website for HRR [*Beighley et al*., 2009] states that "If you would like the source code, please email Dr. Beighley": http://www.northeastern.edu/beighley/hillslope-river-routing-hrr-model/

A similar statement is provided for CaMa-Flood [*Yamazaki et al*., 2011]: "Please contact to the developer (Dai Yamazaki) for the password to download the CaMa-Flood package": http://hydro.iis.u-tokyo.ac.jp/~yamadai/cama-flood/

Appendix B: Example Programs and Special Characters for Automatic Testing in Linux

The automation of RAPID testing in this study was made possible through the use of a series of programs and special characters including some that—despite a few years of experience with programming on Linux—were not already known to the author. A nonexhaustive summary is provided here in hope that readers might find these helpful for testing their own geoscience software; merely as a supporting information to comprehensive references [e.g., *Siever et al*., 2005; *Jones*, 2008].

The generation of an executable from the software source code (compilation and linking or build) is a convoluted and multistep process that can be transformed into a one-line command by using a program called `make` of which instructions are in a file called `makefile`. Word Count (`wc`) can be used to count the number of lines in a file. Globally search a Regular Expression and Print (`grep`) can locate a string of characters in a text file. The Stream EDitor (`sed`) can search for a string of characters in a file and replace it by another. The Basic Calculator (`bc`) can be used to compare two numbers and provide a Boolean value summarizing whether or not they are equal. The Worldwide web GET (`wget`) allows for downloading files from the Internet.

Special characters in a given Linux shell can also ease the automation of software testing. Here we focus on one of the most common Linux shells used in scientific computing called the Bourne Again Shell (`bash`). Helpful values can be obtained at the command line in `bash` or in a `bash` script: the number of arguments provided (`$#`), the list of all arguments (`$@`), the first argument (`$1`), and the exit code of the previous command (`$?`). The text resulting from the execution of one command can be redirected to a text file (`>`) or used as the input to another command (`|`).

Acknowledgments
This work was supported by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space Administration and by the University of California Office of the President Multicampus Research Programs and Initiatives; both institutions are gratefully acknowledged. This research was also partially supported by a Microsoft Azure for Research grant from Microsoft Research. The practical application of this study was enabled by the netCDF, MPICH, PETSc, and TAO scientific libraries that were all built with the GNU Compiler Collection installed on a Community Enterprise Operating System (CentOS). Version control was performed using git. This study was also made possible using the following free online services: GitHub (code repository), Zenodo, (data repository), and Travis CI (Continuous Integration). Comments from the Editor, Associate Editor, and two anonymous reviewers on earlier versions of this manuscript are gratefully acknowledged. The authors are thankful to Yolanda Gil for stressing the importance of software licenses and for continuous support throughout the research presented in this paper, to an anonymous reviewer of a National Science Foundation review panel for suggesting the use of GitHub, to Luke A. Winslow for demonstrating the power of Continuous Integration and for mentioning Travis CI, and to Lars Holm Nielsen of Zenodo for support on hosting the RAPID input and output files. Thank you to the EarthCube OntoSoft leadership team and the advisory committee members for enlightening discussions. The two data sets used in this paper [*David et al*., 2011b, 2011c] are openly available through their respective digital object identifiers (DOI). The piece of software used herein includes the scripts to reproduce all numerical experiments performed in this paper (hence describing provenance of results) and is openly available at https://github.com/c-h-david/rapid/tree/20150906; it will be assigned a DOI [e.g., *David*, 2010, 2011a, 2011b, 2013a, 2013b] pending publication of this study.

References

Anthes, R. (1986), Summary of workshop on the NCAR Community Climate/Forecast Models 14–26 July 1985, Boulder, Colorado, *Bull. Am. Meteorol. Soc.*, *67*(2), 194–198.

Balay, S., W. D. Gropp, L. C. McInnes, and B. F. Smith (1997), Efficient management of parallelism in object oriented numerical software libraries, in *Modern Software Tools in Scientific Computing*, edited by E. Arge, A. M. Bruaset, and H. P. Langtangen, pp. 163–202, Birkhauser, Cambridge, Mass.

Balay, S., J. Brown, K. Buschelman, V. Eijkhout, W. D. Gropp, D. Kaushik, M. G. Knepley, L. Curfman McInnes, B. F. Smith, and H. Zhang (2013), *PETSc Users Manual (Revision 3.3)*, Argonne Natl. Lab, Argonne, Ill.

Barnes, N. (2010), Publish your computer code: It is good enough, *Nature*, *467*, 753, doi:10.1038/467753a.

Bates, P. D., and A. P. J. De Roo (2000), A simple raster-based model for flood inundation simulation, *J. Hydrol.*, *236*(1-2), 54–77.

Beighley, R. E., K. G. Eggert, T. Dunne, Y. He, V. Gummadi, and K. L. Verdin (2009), Simulating hydrologic and hydraulic processes throughout the Amazon River Basin, *Hydrol. Processes*, *23*(8), 1221–1235, doi:10.1002/hyp.7252.

Bell, C. G. (1987), The future of scientific computing, *Comput. Sci.*, *1*, 4–6.

Bell, G., T. Hey, and A. Szalay (2009), Beyond the data deluge, *Science*, *323*(5919), 1297–1298.

Branstetter, M. (2001), Development of a parallel river transport algorithm and applications to climate studies, PhD Thesis, Univ. of Texas, Austin.

Bryan, K., and M. D. Cox (1967), A numerical investigation of the oceanic general circulation, *Tellus*, *19*(1), 54–80, doi:10.1111/j.2153-3490.1967.tb01459.x.

David, C. H. (2010), RAPID v1.0.0, *Zenodo*, doi:10.5281/zenodo.27239.

David, C. H. (2011a), RAPID v1.1.0, *Zenodo*, doi:10.5281/zenodo.27241.

David, C. H. (2011b), RAPID v1.2.0, *Zenodo*, doi:10.5281/zenodo.27242.

David, C. H. (2013a), RAPID v1.3.0, *Zenodo*, doi:10.5281/zenodo.27243.

David, C. H. (2013b), RAPID v1.4.0, *Zenodo*, doi:10.5281/zenodo.24756.

David, C. H., F. Habets, D. R. Maidment, and Z.-L. Yang (2011a), RAPID applied to the SIM-France model, *Hydrol. Processes*, *25*(22), 3412–3425.

David, C. H., F. Habets, D. R. Maidment, and Z.-L. Yang (2011b), RAPID input and output files corresponding to "RAPID Applied to the SIM-France Model", *Zenodo*, doi:10.5281/zenodo.30228.

David, C. H., D. R. Maidment, G.-Y. Niu, Z.-L. Yang, F. Habets, and V. Eijkhout (2011c), RAPID input and output files corresponding to "River Network Routing on the NHDPlus Dataset", *Zenodo*, doi:10.5281/zenodo.16565.

David, C. H., D. R. Maidment, G.-Y. Niu, Z.-L. Yang, F. Habets, and V. Eijkhout (2011d), River network routing on the NHDPlus dataset, *J. Hydrometeorol.*, *12*(5), 913–934.

David, C. H., Z.-L. Yang, and J. S. Famiglietti (2013a), Quantification of the upstream-to-downstream influence in the Muskingum method and implications for speedup in parallel computations of river flow, *Water Resour. Res.*, *49*, 2783–2800, doi:10.1002/wrcr.20250.

David, C. H., Z.-L. Yang, and S. Hong (2013b), Regional-scale river flow modeling using off-the-shelf runoff products, thousands of mapped rivers and hundreds of stream flow gauges, *Environ. Modell. Software*, *42*, 116–132.

David, C. H., J. S. Famiglietti, Z.-L. Yang, and V. Eijkhout (2015), Enhanced fixed-size parallel speedup with the Muskingum method using a trans-boundary approach and a large sub-basins approximation, *Water Resour. Res.*, *51*, 7547–7571, doi:10.1002/2014WR016650.

Dongarra, J., et al. (1994), Special issue—MPI—A Message-Passing Interface standard, *Int. J. Supercomput. Appl. High Perform. Comput.*, *8*(3-4), 159–416.

Easterbrook, S. M. (2014), Open code for open science? *Nat. Geosci.*, *7*(11), 779–781.

Flipo, N., C. Monteil, M. Poulin, C. de Fouquet, and M. Krimissa (2012), Hybrid fitting of a hydrosystem model: Long-term insight into the Beauce aquifer functioning (France), *Water Resour. Res.*, *48*, W05509, doi:10.1029/2011WR011092.

Häfliger, V., et al. (2015), Evaluation of regional-scale river depth simulations using various routing schemes within a hydrometeorological modeling framework for the preparation of the SWOT mission, *J. Hydrometeorol.*, *16*(4), 1821–1842, doi:10.1175/JHM-D-14-0107.1.

Hey, T. (2010), Science has four legs, *Commun. ACM*, *53*(12), doi:10.1145/1859204.1859206.

Hey, T., and M. C. Payne (2015), Open science decoded, *Nat. Phys.*, *11*(5), 367–369.

Hey, T., S. Tansley, and K. Tolle (2009), Jim Gray on eScience: A transformed scientific method, in *The Fourth Paradigm. Data Intensive Scientific Discovery*, edited by T. Hey, S. Tansley, and K. Tolle, pp. xvii–xxxi, Microsoft Res, Redmond, Wash.

Holdren, J. P. (2013), *Memorandum for the Heads of Executive Departments and Agencies. Increasing Access to the Results of Federally Funded Scientific Research*, Exec. Off. of the Pres., Off. of Sci. and Technol. Policy, Washington, D. C.

Horsburgh, J. S., M. M. Morsy, A. M. Castronova, J. L. Goodall, T. Gan, H. Yi, M. J. Stealey, and D. G. Tarboton (2015), Hydroshare: Sharing diverse environmental data types and models as social objects with application to the hydrology domain, *J. Am. Water Resour. Assoc.*, doi:10.1111/1752-1688.12363.

Hurrell, J. W. (2013), NCAR in the 21st Century Building on a Distinguished Record of Achievement, Leadership and Service.

Hurrell, J. W., et al. (2013), The community Earth system model: A framework for collaborative research, *Bull. Am. Meteorol. Soc.*, *94*(9), 1339–1360, doi:10.1175/BAMS-D-12-00121.1.

Ince, D. C., L. Hatton, and J. Graham-Cumming (2012), The case for open computer programs, *Nature*, *482*(7386), 485–488, doi:10.1038/nature10836.

Intergovernmental Panel on Climate Change (2013), *Climate Change 2013: The Physical Science Basis. Contribution of Working Group I to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change*, edited by T. F. Stocker et al., pp. 1–1535, Cambridge Univ. Press, Cambridge, U. K.

Jones, M. T. (2008), *GNU/Linux Application Programming*, 2nd ed., Course Technol., Cengage Learn., Boston, Mass.

Kattge, J., S. Diaz, and C. Wirth (2014), Of carrots and sticks, *Nat. Geosci.*, *7*(11), 778–779.

Kratz, J. E., and C. Strasser (2015), Researcher perspectives on publication and peer review of data, *PLoS One*, *10*(2), e0117619, doi:10.1371/journal.pone.0117619.

Lin, P., Z.-L. Yang, X. Cai, and C. H. David (2015), Development and evaluation of a physically-based lake level model for water resource management: A case study for Lake Buchanan, Texas, *J. Hydrol.*, *4*(Part B), 661–674, doi:10.1016/j.ejrh.2015.08.005.

Lohmann, D., R. NolteHolube, and E. Raschke (1996), A large-scale horizontal routing model to be coupled to land surface parametrization schemes, *Tellus, Ser. A*, *48*(5), 708–721.

Lohmann, D., et al. (2004), Streamflow and water balance intercomparisons of four land surface models in the North American Land Data Assimilation System project, *J. Geophys. Res.*, *109*, D07S91, doi:10.1029/2003JD003517.

Maidment, D. R. (2015), *A Conceptual Framework for the National Flood Interoperability Experiment*, CUAHSI.

Manabe, S. (1969), Climate and the ocean circulation: 1. The atmospheric circulation and the hydrology of the Earth's surface, *Mon. Weather Rev.*, *97*(11), 739–774.
McCarthy, G. T. (1938), The unit hydrograph and flood routing.
Miller, J. R., G. L. Russell, and G. Caliri (1994), Continental-scale river flow in climate models, *J. Clim.*, *7*(6), 914–928, doi:10.1175/1520-0442(1994)007<0914:CSRFIC>2.0.CO;2.
Munson, T., J. Sarich, S. Wild, S. Benson, and L. Curfman McInnes (2012), *TAO User Manual (Revision 2.1)*, Math. and Comput. Sci. Div., Argonne Natl. Lab, Argonne, Ill. [Available at http://www.mcs.anl.gov/tao.]
Nature (2014), Code share, *Nature*, *514*, 536.
Nature Geoscience (2014), Towards transparency, *Nat. Geosci.*, *7*(11), 777.
Oleson, K., et al. (2013), *Technical Description of Version 4.5 of the Community Land Model (CLM)*, *Tech. Note NCAR/TN-503+STR*, NCAR, Boulder, Colo.
Peckham, S. D., E. W. H. Hutton, and B. Norris (2013), A component-based approach to integrated modeling in the geosciences: The design of CSDMS, *Comput. Geosci.*, *53*, 3–12.
Peng, R. D. (2011), Reproducible research in computational science, *Science*, *334*(6060), 1226–1227, doi:10.1126/science.1213847.
Phillips, N. A. (1956), The general circulation of the atmosphere: A numerical experiment, *Q. J. R. Meteorol. Soc.*, *82*(352), 123–164, doi:10.1002/qj.49708235202.
Rew, R., and G. Davis (1990), NetCDF—An interface for scientific-data access, *IEEE Comput. Graphics Appl.*, *10*(4), 76–82.
Rew, R., D. Heimbigner, and W. Fisher (2013), Announcing a transition to GitHub.
Rosen, L. E. (2005), *Open Source Licensing: Software Freedom and Intellectual Property Law*, 2nd ed., Prentice Hall, Upper Saddle River, N. J.
Saleh, F., N. Flipo, F. Habets, A. Ducharne, L. Oudin, P. Viennot, and E. Ledoux (2011), Modeling the impact of in-stream water level fluctuations on stream-aquifer interactions at the regional scale, *J. Hydrol.*, *400*(3–4), 490–500.
Scientific Data (2014), More bang for your byte, *Sci. Data*, *1*, 140010.
Siever, E., A. Weber, S. Figgins, R. Love, and A. Robbins (2005), *Linux in a Nutshell*, 5th ed., O'Reilly Media, Inc., Sebastopol, Calif.
Tavakoly, A. A., C. H. David, D. R. Maidment, Z.-L. Yang, and X. Cai (2012), An upscaling process for large-scale vector-based river networks using the NHDPlus dataset.
Tavakoly, A. A., D. R. Maidment, J. McClelland, T. Whiteaker, Z.-L. Yang, C. Griffin, C. H. David, and L. Meyer (2015), A GIS framework for regional modeling of Riverine nitrogen transport: Case study, San Antonio and Guadalupe basins, *J. Am. Water Resour. Assoc.*, *52*, 1–15, doi:10.1111/1752-1688.12355.
Thierion, C., et al. (2012), Assessing the water balance of the Upper Rhine Graben hydrosystem, *J. Hydrol.*, *424–425*, 68–83.
US Congress (1976), The copyright act of 1976.
Vardi, M. (2010a), Author's response, *Commun. ACM*, *53*(12), 6–7, doi:10.1145/1859204.1859206.
Vardi, M. (2010b), Science has only two legs, *Commun. ACM*, *53*(9), 5, doi:10.1145/1810891.1810892.
Williamson, D. L. (1983), *Description of NCAR Community Climate Model (CCM0B)*, *NCAR Tech. Note*, Natl. Cent. for Atmos. Res., Boulder, Colo.
Xia, Y., et al. (2012), Continental-scale water and energy flux analysis and validation for the North American Land Data Assimilation System project phase 2 (NLDAS-2): 2. Validation of model-simulated streamflow, *J. Geophys. Res.*, *117*, D03110, doi:10.1029/2011JD016051.
Yamazaki, D., S. Kanae, H. Kim, and T. Oki (2011), A physically based description of floodplain inundation dynamics in a global river routing model, *Water Resour. Res.*, *47*, W04501, doi:10.1029/2010WR009726.
Zhao, T., B. S. Minsker, J. S. Lee, F. R. Salas, D. R. Maidment, and C. H. David (2014), Real-time water decision support services for droughts, Pap. 77, pp. 1–10.