# SOFTWARE ENGINEERING FOR SCIENCE



Cite as:
Idaszak, R., D. G. Tarboton, H. Yi, L. Christopherson, M. J. Stealey, B. Miles, P. Dash, A. Couch, C. Spealman, D. P. Ames and J. S. Horsburgh, (2017), "HydroShare - A case study of the application of modern software engineering to a large distributed federally-funded scientific software development project," Chapter 10 in Software Engineering for Science, Edited by J. Carver, N. P. C. Hong and G. K. Thiruvathukal, Taylor&Francis CRC Press, p.217-233.

# Chapman & Hall/CRC
# Computational Science Series

## SERIES EDITOR

Horst Simon
Deputy Director
Lawrence Berkeley National Laboratory
Berkeley, California, U.S.A.

## PUBLISHED TITLES

COMBINATORIAL SCIENTIFIC COMPUTING
**Edited by Uwe Naumann and Olaf Schenk**

CONTEMPORARY HIGH PERFORMANCE COMPUTING: FROM PETASCALE TOWARD EXASCALE
**Edited by Jeffrey S. Vetter**

CONTEMPORARY HIGH PERFORMANCE COMPUTING: FROM PETASCALE TOWARD EXASCALE, VOLUME TWO
**Edited by Jeffrey S. Vetter**

DATA-INTENSIVE SCIENCE
**Edited by Terence Critchlow and Kerstin Kleese van Dam**

THE END OF ERROR: UNUM COMPUTING
**John L. Gustafson**

FROM ACTION SYSTEMS TO DISTRIBUTED SYSTEMS: THE REFINEMENT APPROACH
**Edited by Luigia Petre and Emil Sekerinski**

FUNDAMENTALS OF MULTICORE SOFTWARE DEVELOPMENT
**Edited by Victor Pankratius, Ali-Reza Adl-Tabatabai, and Walter Tichy**

FUNDAMENTALS OF PARALLEL MULTICORE ARCHITECTURE
**Yan Solihin**

THE GREEN COMPUTING BOOK: TACKLING ENERGY EFFICIENCY AT LARGE SCALE
**Edited by Wu-chun Feng**

GRID COMPUTING: TECHNIQUES AND APPLICATIONS
**Barry Wilkinson**

HIGH PERFORMANCE COMPUTING: PROGRAMMING AND APPLICATIONS
**John Levesque with Gene Wagenbreth**

HIGH PERFORMANCE PARALLEL I/O
**Prabhat and Quincey Koziol**

## PUBLISHED TITLES CONTINUED

HIGH PERFORMANCE VISUALIZATION:
ENABLING EXTREME-SCALE SCIENTIFIC INSIGHT
**Edited by E. Wes Bethel, Hank Childs, and Charles Hansen**

INDUSTRIAL APPLICATIONS OF HIGH-PERFORMANCE COMPUTING:
BEST GLOBAL PRACTICES
**Edited by Anwar Osseyran and Merle Giles**

INTRODUCTION TO COMPUTATIONAL MODELING USING C AND
OPEN-SOURCE TOOLS
**José M Garrido**

INTRODUCTION TO CONCURRENCY IN PROGRAMMING LANGUAGES
**Matthew J. Sottile, Timothy G. Mattson, and Craig E Rasmussen**

INTRODUCTION TO ELEMENTARY COMPUTATIONAL MODELING: ESSENTIAL
CONCEPTS, PRINCIPLES, AND PROBLEM SOLVING
**José M. Garrido**

INTRODUCTION TO HIGH PERFORMANCE COMPUTING FOR SCIENTISTS
AND ENGINEERS
**Georg Hager and Gerhard Wellein**

INTRODUCTION TO REVERSIBLE COMPUTING
**Kalyan S. Perumalla**

INTRODUCTION TO SCHEDULING
**Yves Robert and Frédéric Vivien**

INTRODUCTION TO THE SIMULATION OF DYNAMICS USING SIMULINK®
**Michael A. Gray**

PEER-TO-PEER COMPUTING: APPLICATIONS, ARCHITECTURE, PROTOCOLS,
AND CHALLENGES
**Yu-Kwong Ricky Kwok**

PERFORMANCE TUNING OF SCIENTIFIC APPLICATIONS
**Edited by David Bailey, Robert Lucas, and Samuel Williams**

PETASCALE COMPUTING: ALGORITHMS AND APPLICATIONS
**Edited by David A. Bader**

PROCESS ALGEBRA FOR PARALLEL AND DISTRIBUTED PROCESSING
**Edited by Michael Alexander and William Gardner**

SCIENTIFIC DATA MANAGEMENT: CHALLENGES, TECHNOLOGY, AND DEPLOYMENT
**Edited by Arie Shoshani and Doron Rotem**

SOFTWARE ENGINEERING FOR SCIENCE
**Edited by Jeffrey C. Carver, Neil P. Chue Hong, and George K. Thiruvathukal**

# SOFTWARE ENGINEERING FOR SCIENCE


Edited by

Jeffrey C. Carver
University of Alabama, USA

Neil P. Chue Hong
University of Edinburgh, UK

George K. Thiruvathukal
Loyola University Chicago, Chicago, Illinois



CRC Press
Taylor & Francis Group
Boca Raton London New York

CRC Press is an imprint of the
Taylor & Francis Group, an **informa** business
A CHAPMAN & HALL BOOK

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742



Printed on acid-free paper
Version Date: 20160817

International Standard Book Number-13: 978-1-4987-4385-3 (Hardback)



**Library of Congress Cataloging-in-Publication Data**

Names: Carver, Jeffrey, editor. | Hong, Neil P. Chue, editor. |
Thiruvathukal, George K. (George Kuriakose), editor.
Title: Software engineering for science / edited by Jeffrey Carver, Neil P.
Chue Hong, and George K. Thiruvathukal.
Description: Boca Raton : Taylor & Francis, CRC Press, 2017. | Series:
Computational science series ; 30 | Includes bibliographical references
and index.
Identifiers: LCCN 2016022277 | ISBN 9781498743853 (alk. paper)
Subjects: LCSH: Science--Data processing. | Software engineering.
Classification: LCC Q183.9 .S74 2017 | DDC 005.1--dc23
LC record available at https://lccn.loc.gov/2016022277

**Visit the Taylor & Francis Web site at**
**http://www.taylorandfrancis.com**

**and the CRC Press Web site at**
**http://www.crcpress.com**

# Chapter 10

## HydroShare – A Case Study of the Application of Modern Software Engineering to a Large Distributed Federally-Funded Scientific Software Development Project

**Ray Idaszak, David G. Tarboton (Principal Investigator), Hong Yi, Laura Christopherson, Michael J. Stealey, Brian Miles, Pabitra Dash, Alva Couch, Calvin Spealman, Jeffery S. Horsburgh, and Daniel P. Ames**

## Abstract

HydroShare is an online collaborative system under development to support the open sharing of hydrologic data, analytical tools, and computer models. With HydroShare, scientists can easily discover, access, and analyze hydrologic data and thereby enhance the production and reproducibility of hydrologic scientific results. HydroShare also takes advantage of emerging social media functionality to enable users to enhance information about and collaboration around hydrologic data and models.

HydroShare is being developed by an interdisciplinary collaborative team of domain scientists, university software developers, and professional software engineers from ten institutions located across the United States. While the combination of non–co-located, diverse stakeholders presents communication and management challenges, the interdisciplinary nature of the team is integral to the project's goal of improving scientific software development and capabilities in academia.

This chapter describes the challenges faced and lessons learned with the development of HydroShare, as well as the approach to software development that the HydroShare team adopted on the basis of the lessons learned. The chapter closes with recommendations for the application of modern software engineering techniques to large, collaborative, scientific software development projects, similar to the National Science Foundation (NSF)–funded HydroShare, in order to promote the successful application of the approach described herein by other teams for other projects.

---

## 10.1 Introduction to HydroShare

The HydroShare software development project is funded by the National Science Foundation (NSF) through its Software Infrastructure for Sustained Innovation program [333, 336]. Domain scientists, professional[1] software engineers, and academic software developers from ten academic institutions located across the United States[2] collaborate to develop HydroShare–an online,

[1] The term *professional*, as used here refers to an individual that has received formal education on software development and has applied this knowledge in a commercial or equivalent context.

[2] Brigham Young University, Caktus Group, Consortium of Universities for the Advancement of Hydrologic Science, Inc., Purdue University, Renaissance Computing Institute (RENCI) at the University of North Carolina at Chapel Hill, Tufts University, Institute for the Environment at the University of North Carolina at Chapel Hill, University of Texas at Austin, University of Virginia, and Utah State University.

collaborative system that extends the data-sharing capabilities of the Hydrologic Information System (HIS), which was developed by the Consortium of Universities for the Advancement of Hydrologic Sciences, Inc. (CUAHSI) [355]. HydroShare extends the data-sharing capabilities of HIS by broadening the classes of data that are accommodated, enabling the sharing of computer models and model components, and incorporating social media functionality in order to enhance communication and collaboration around hydrologic data and models [350, 351, 353].

In cooperation with CUAHSI, HydroShare is being used by the National Flood Interoperability Experiment (NFIE), which is a collaboration between the National Weather Service, government and commercial partners, and the academic community. NFIE is working to build a next-generation, high-resolution, near–real-time hydrologic simulation and forecasting model for the United States. With HydroShare, NFIE is able to better facilitate the flow of information between the federal, state, and local entities responsible for flood measurement, forecasting, and planning [338]. This near–real-time information also can be used by first responders during severe weather events to navigate to people in need of assistance [339].

The HydroShare project provides an example of the application of modern software engineering techniques to the development of scientific software. At the project's outset, most members of the HydroShare team did not fully understand the difference between software development and software engineering, nor were they familiar with iterative software methodology, code refactoring, continuous integration, or test-driven development (explained in Section 10.4.6). Much of the functionality of HydroShare–such as user interface, access control, social media incorporation, metadata handling, search and discovery, analytics, simulation, and storage capabilities–also was challenging for the team. While many members of the team had previous experience in the software development of hydrologic models, including models containing very complex algorithms and data structures, none of the models that had been developed by team members had the depth or complexity of the HydroShare software stack, and none required distributed code development and coordination across a large team. Thus, the team quickly realized the need to apply modern software engineering practices as part of the HydroShare experience. At the time of this writing, four years into the project, the team is now capable of applying advanced software engineering techniques to the development of HydroShare.

This chapter describes the approach, experience, and lessons learned when applying modern software engineering techniques to a large scientific software project, HydroShare. Recommendations are provided for how to integrate best practices in modern software engineering into large, collaborative research projects such as HydroShare. The overall intent is to support the advancement of science and expand the use of sustainable software engineering practices in academia. The goal is for other scientific software development teams to be able to adopt and adapt the techniques and practices described in this chapter.

## 10.2 Informing the Need for Software Engineering Best Practices for Science

Modern scientific research relies on software. Software enables scientists to collect data, perform analyses, run numerical and statistical models, and visualize data. With the aid of software, scientists are able to answer key research questions and test hypotheses that can revolutionize what is known about the world. Life-sustaining policies, products, and techniques–such as clinical therapies, pharmaceutical compounds, and solutions to environmental problems–derive from software-enabled scientific research.

Software such as HydroShare that supports data collection, analysis, and modeling is often used to accomplish research goals. Hannay, MacLeod, and Singer [349] have noted that scientists spend as much as 40% of their time using software. Often, existing software is ill-suited to a particular research project or, in the case of commercial software, prohibitively expensive. The result is that scientists often develop their own software–spending as much as 30% of their time doing so [349]–even though few incentives exist for software development in traditional tenure and promotion decision-making processes [352]. In other words, the time that an academic scientist spends developing software is not rewarded or recognized as a significant, independent accomplishment. Tenure and promotion, for example, are based on influential research, a successful publication record, the acquisition of grants, and teaching–not on whether one can author good software. Additionally, many funding agencies wish to see their funds going toward time spent on traditional research activities, not writing software.

While not incentivized, academic scientists continue to develop their own software. However, most academic scientists are not trained in software development or software engineering [342, 345, 359]. Software development courses, typically offered by computer science departments, are not required for most non-majors. Additionally, the training that scientists do receive from computer science departments often is perceived as overly general or abstract, and scientists may not see the relevance of such training [349]. As a result of the lack of training in software development and software engineering, the software that is developed by academic scientists often is not built to the development standards of the commercial sector. Software engineering best practices, such as documentation, versioning, and testing, may not be applied during the creation of academic scientific software. Furthermore, most academic software is developed to suit the needs of a specific research project and thus may not be applicable to other research projects or sustainable beyond the life of the initial project.

The lack of training in software development and software engineering can have dire consequences [346]. For instance, software that is developed without the use of proven software engineering techniques may lead to errors in the code. Even minor errors influence the validity of research findings; indeed, in some cases, papers have been retracted from scientific journals and careers have been ruined [357]. Paper retractions and irreproducible results due to poor-quality software impede the advancement of science and impart huge financial repercussions. Under the worst case scenario, programming errors can lead to the loss of lives if erroneous findings result in faulty medical technologies or misdirected policies on disaster response, to provide examples.

The detection of errors in academic software is extremely challenging, however. While manuscripts submitted for journal publication must undergo a peer review process, the software code that is used to generate the findings presented in manuscripts is rarely subjected to a peer review process or other measures of quality assurance. Yet, peer review and testing of software code are critical for the credibility of science and require software engineering best practices.

Of significance, the risk of introducing error into scientific research through the use of low-quality software provides a little recognized, but highly impactful, incentive for the adoption of software engineering best practices in academic scientific software development.

The HydroShare project addresses the challenges and highlights the benefits of the adoption of software engineering best practices through a collaborative scientific software project involving a large, geographically dispersed team of academic scientists, academic software developers, and professional software engineers.

## 10.3 Challenges Faced and Lessons Learned

This section describes the challenges faced and lessons learned when applying modern software engineering best practices to a software development project in hydrology. *Modern software engineering*, as used here refers to “the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software” [330].

### 10.3.1 Cultural and Technical Challenges

Early on, the HydroShare team identified several overarching culture challenges. First, the team found that it is not a tradition to use modern software engineering best practices in the development of academic software due to a lack of incentives and a failure to recognize the benefits, as discussed above. The perception was that good software engineering practices are not needed to obtain scientific results and to publish scientific papers. Second, graduate students often develop software for their faculty advisors, yet graduate students have very short-term goals (i.e., graduate in the next couple of years), so software sustainability is not a high priority. Third, graduate students and their faculty advisors typically have not received formal training in software development, let alone software engineering. Fourth and lastly, the rigorous metadata requirements necessary for reproducible science make scientific software systems more complex than other types of software and thus require significant time to create unit tests. This presents a paradox, as the more complex software is, the more benefit one gets from having comprehensive unit tests.

The team also encountered more specific technical challenges. For example, as implementation of our HydroShare project began, the team quickly realized that most members were not familiar with Git, GitHub, or continuous integration (i.e., a development practice that requires developers to integrate code into a shared repository on a very frequent basis). The decision was thus made to assign only members at the lead technical institution the task of implementing initial beta release functionalities in order to expedite creation of the code infrastructure for subsequent collaborative development and continuous integration by the broader team members. However, this limited HydroShare beta release functionality to only those functionalities that could be implemented by the lead technical institution. This approach did expedite the initial release of the system, but the approach also precluded the ability for other team members to contribute to the development. For HydroShare, this trade-off was acceptable as the other team members used the additional time to get versed on continuous integration, Git, GitHub, and other specific technologies and approaches used in HydroShare software development.

Early on in the project, the team held several in-person meetings, as well as weekly team teleconferences, that served to achieve the development objectives, including the development of a data model (i.e., a conceptual model of how data elements relate to each other) and access control policies (i.e., policies to restrict access to data) and thorough consideration of how to accommodate hydrologic models within HydroShare. As implementation progressed and software engineering principles, such as code versioning (i.e., management of revisions to source code) and continuous integration, were diffused from the professional software engineers to the hydrologists, additional challenges emerged. For example, the distributed development team experienced difficulty achieving short-release cycles of continuous integration of the Django-based system using Git and GitHub. Django is a large, complex, open source, python-based web development framework, in which its customization model and content data are stored in databases [328]. Django proved to be difficult to manage via version control by a team with members of various skill levels. Specifically, the challenge was how to manage multiple, distributed development teams that were simultaneously checking out their own branch[3] of HydroShare, while

[3]A *branch* in GitHub lives separately from the production codebase, thus allowing for experimentation without affecting the *master branch* (or production codebase).

maintaining consistency in the back-end Django database. This led to multiple, complex code feature branches being checked out and worked on at the same time–without a sufficient number of intermediate merges.

Described below are two specific challenges–waiting too long between code merges and establishing a development environment–examined in greater depth, including the downstream challenges and lessons learned.

### 10.3.2 Waiting Too Long between Code Merges

To highlight the complications that may arise from waiting too long between code merges, this section considers a release of HydroShare in which two key items were addressed on different branches: base class refactoring and a change in the approach to access control. This presented a non-trivial challenge because of the intertwining of these two items, along with the need to preserve existing resources.

The HydroShare base class refactoring branch endeavored to promote the HydroShare Generic Resource type functionality from being abstract to fully defined. Being able to extend upon a fully defined resource opened the door for easier searching, indexing, and database querying that wouldn't otherwise be possible if the team had kept extending from the previously defined abstract model. Once this was implemented and tested for the Generic Resource type, the team then needed to apply this to all of the other HydroShare resource types in close coordination with the developers that had originally created them in order to ensure against loss of context, extended metadata, or other resource-specific attributes.

The HydroShare access control branch endeavored to implement an access control model that the team designed to best suit the hydrology research community [347]. However, this uniquely designed HydroShare access control model meant that it was necessarily non-standard and non-compliant with basic use of Django; thus, the team took extra care in how this change was implemented. The first, albeit incomplete, implementation of the HydroShare access control was integrated on top of Django's permission system for the sake of simplicity and the need to get an initial working version of HydroShare. To implement the full HydroShare access control, the team needed to decouple from Django's permission system and enforce a HydroShare-specific model, thereby adding additional system complexity.

To further complicate things, the HydroShare access control and base class refactoring had to be implemented on top of existing HydroShare resources in production use. The integrated rule-oriented data system (iRODS) [334] is used as a data management back-end to HydroShare. The challenge however, was migrating all of the existing HydroShare resources that were in use by users when the new resource and access control schema didn't fit the existing iRODS storage policies. Multiple steps and operations on the resources and database were required to properly migrate resources into the new models and access control schema. This proved to be quite a challenging endeavor.

Each of these items on their own presented a significant task; however, the summation of all of these branches into a single release required numerous dry-runs and small intermediate tests based on the results of the dry-runs before the team was confident that it was right. The team put as much time into testing and validation as they did into coding the changes themselves. The main lesson learned from this experience is that it is best to perform smaller, but more frequent merges, rather than a large release with multiple complex merges. With the former approach, the merge complexity will be reduced and time will be saved.

### 10.3.3 Establishing a Development Environment

Another major challenge for the development team was setting up the integrated development environment for individual developers. This presented a challenge mainly due to the many Docker containers [329] that the HydroShare system uses, as well as the fact that most of the developers did not have knowledge of Docker configuration, which was a relatively new technology at the beginning of the HydroShare project. This challenge was resolved by scripting the development environment, such that with few commands, the team could quickly set up the HydroShare development environment–something that had previously taken hours. As the development team was distributed, weekly videoconferences were used to train new HydroShare developers on how to set up the development environment.

The HydroShare software developers faced major challenges in code contribution in the early stages of the HydroShare project due to the size of the project and their inexperience, particularly when working in a distributed team environment. In addition, the team didn't have procedures in place for how to effectively contribute code using GitHub (also discussed in Section 10.4.8), which was new to many team members. In order to solve these challenges, the team created very detailed documentation specific to the project on how to push/pull to/from GitHub. In addition, hands-on training was provided to all software developers on best practices for using GitHub. In order to improve code quality, the team adopted the GitHub pull request feature for code review, whereby all code had to be code reviewed by an independent team member prior to merging the pull request. We found these practices to be extremely beneficial in providing the momentum to establish our software development environment.

## 10.4 Adopted Approach to Software Development Based on the Lessons Learned

This section conveys the approach to software development that was adopted for HydroShare based on the lessons learned early on in the project.

The approach includes the adoption of an iterative approach to software development that incorporates best practices in modern software engineering. Highlighted are several best practices in software engineering, including the use of virtual machines, code versioning, code reviews, and test-driven development. This section concludes with a discussion of the role and importance of communication and DevOps in facilitating effective multi-institutional collaboration.

### 10.4.1 Adopting Best Practices in Modern Software Engineering

One of the goals of the HydroShare Project is to continually adopt modern software engineering techniques to all scientific software development efforts. Although a scientist can author high-value software code, s/he approaches software development as a means to an end, with the end being new research findings. A software engineer, in contrast, approaches software development with code quality and sustainability as primary goals–not application. To a scientist, the research process is emphasized, and the final product is a set of scientific findings, which should be accurate, reproducible, and generalizable. To a software engineer, the coding process is emphasized, and the software code is the product, which should be error-free and reusable for solving other problems. In the same way that a scientist carefully designs a study to answer a research question or test a hypothesis, a software engineer carefully designs the code s/he will write to create new functionality. For example, software engineers use *design patterns*, or reusable and generalizable units of code that solve common software problems. Most scientists are not familiar with the concept of design patterns. Instead of combining reusable, tested units of code into new software, scientists often choose to write code from scratch in order to address a specific research need; after that need has been met, the software code is often tossed aside. Scientists are not as concerned about ensuring that the code is free of bugs because the code is not the object of interest, so code testing is not common practice. Software engineers, however, are trained to ensure that the code is free of bugs because quality code is the object of interest, so testing the code for accuracy is common practice.

One could argue that if scientists had lavish funding, they could hire professional software engineers to develop higher quality code over a more expeditious timeline. However, while an abundance of funding is always desirable, this would prevent the realization of certain opportunities. For HydroShare the involvement of hydrologists, including both graduate students and their faculty advisors, in software coding was extremely important for several reasons:

- As subject matter experts, the scientists were able to illuminate salient uses cases.

- As co-creators and owners of the software, the scientific community will be more likely to adopt the software and shepherd it throughout its lifetime.
- As graduate students, the incorporation of modern software engineering practices into their training is imperative in order to better prepare the next generation of hydrologists.

The key is that HydroShare should not be viewed simply as a software artifact, but also as a project that captures human and scientific capital for the advancement of transformative science through mature software engineering methodology. This is an important point. A lot of new ideas and thought processes have been created in the minds of the HydroShare team (i.e. human capital) as a result of this project, and these need to be kept concomitant with the software. Modern software engineering, in part, helps us achieve this.

### 10.4.2 Iterative Software Development

The Waterfall approach to software development emphasizes a discrete planning phase that includes gathering all possible requirements before the coding phase commences [340]. After a Waterfall phase is concluded, the rule-of-thumb is that that phase should not be revisited. This type of approach does not recognize or make allowances for the unknown and unpredictable. In other words, the Waterfall approach does not provide flexibility regarding changes in requirements, new needs or system uses, or changes in project focus.

Brooks [343] claims that software engineers should be open and ready to throw out unworkable ideas. "The only question is whether to plan in advance to build a throwaway, or to promise to deliver the throwaway to customers" [344]. When a distributed software development team endeavors to integrate several new and existing software systems at the onset of a project, complications can arise that preclude the ability of the team to efficiently and/or practically overcome those challenges. This is especially true in academic software development projects that have limited time and funding and include team members with varying levels of skill. With HydroShare, the team was not exempt from the "throwaway principle" and indeed had to completely discard a well-developed early version of the software due to unforeseen complications with the integration of disparate software systems. This was the result of several factors:

1. The decision to go with a seemingly appropriate technology, with much community adoption in other circles, was flawed at the beginning and made by a subset of the team without full consideration by the broader team. A more inclusive decision process would have led to better articulation regarding the platform requirements and a better outcome.
2. The system that was chosen, while having widespread community adoption in other circles, was one in which the team had no expertise. The

learning curve proved to be too high for them, at least on a practical level and given the time constraints.

3. The team's lack of expertise was magnified when the team members that made the decision to adopt the system left the project and a new lead development team came onboard without any prior knowledge of the selected technology or understanding of the previous team's activities; this challenge was exacerbated by lack of transition documentation to guide the new team.

The team has since adopted a more flexible iterative approach with HydroShare, one that embraces change. The conclusion is that one should expect to throw out an early version of a software product and learn from the experience. Also, one should realize that it is so much more efficient (and easier to accept) if this is part of the team's plan from the start, for when planning to throw out an early version of developed software, a team can view the experience as an exceptional opportunity to learn what works and what doesn't from the perspectives of software and technology integration, team communication, meeting productivity, and process efficiency. The HydroShare team also found it beneficial to encapsulate functionality in small, loosely coupled systems. For example, the distributed data management system used by HydroShare can work separately from the content management system, which can work separately from the web applications system, and so forth. In the first iteration, the team found that the integration of systems too tightly presents limitations. Unforeseen challenges arise in every software development project; the key is to plan for this early on and in every facet of the project–and expect to throw away at least one early product.

### 10.4.3 Virtual Machines

The HydroShare team uses virtual machines (VM) in testing and production in order to facilitate the distributed team's concurrent prototyping and development of the many diverse features of HydroShare. VMs can be created and spun-up very quickly, with configurable memory, processor, disk storage, and operating system to meet the diverse and evolving project and feature requirements. For features that are complex and highly susceptible to error, the HydroShare team creates a VM to test the feature. The team also creates feature-testing VMs for contextually-related features. For example, the group working on the search and filtering functionality has their own VM; the federated identity management group has its own VM; the user interface group has their own VM, and so on. Git (i.e., a revision control system) and GitHub (i.e., a hosting service for Git repositories) are used to manage and eventually merge the work on these VMs into a production release. Generally, a given branch of code that requires testing and feedback from the team is given its own VM. The exception is that some Git branches–especially those for general fixes–don't require deployment to a VM since they don't intertwine with other parts of the system and can be tested locally. Production VMs share an allocation of fifty terabytes of project disk space and another fifty terabytes of replicated disk space located four miles away in order to ensure fault tolerance and disaster recovery.

### 10.4.4 Code Versioning

Code versioning is a must for any modern software development project, academic or otherwise. There are several popular code versioning systems. The HydroShare team chose Git due to its ability to support distributed development workflows. Unlike other version control systems, Git allows developers to clone the main code repository on their local machines and develop and experiment with new code safely, in an independent environment completely separated from the main codebase. New code can then be submitted for inclusion into the main codebase after being reviewed and tested by other members of the team. This enforces code review, allows for experimentation within a safety net, and enables concurrent streams of development for a speedier process.

### 10.4.5 Code Reviews

With HydroShare, code reviews have opened up the reading of code and stimulated discussion around the structure of the code–something that was not happening before the team implemented the code review process. However, the team took a while to acclimate to the code review process, whereby the person who reviews the code is always different from the person who authors the code. For HydroShare, a code review includes an evaluation of:

- How well the new units of code address the associated use case;
- Code quality, in terms of clarity, concision, lack of redundancy, and thorough inline documentation;
- How easy the code's functionality is to use; and
- How the code fared in unit testing (i.e., tests of individual modules written for a particular function that together comprise a larger set of code).

The HydroShare team has found code reviews to be beneficial for encouraging discussion between scientists and software engineers around the structure of the code. These discussions have served as a vehicle for teaching software engineering best practices to the scientists involved in the project, particularly graduate students who are not required to take programming classes as part of their graduate work. In addition to these benefits, estimates suggest that rigorous code review can remove up to 90% of errors from a software product before any code testing is initiated [348] (code testing is discussed in next section).

The key point is that while it is necessary to establish regular code reviews early on, a period of acclimation should be expected.

### 10.4.6 Testing and Test-Driven Development

The testing of all code prior to release is extremely important for writing sustainable code – code that lasts over time because it has been tested for defects and performs consistently through new releases. There are a variety of tests that may be conducted during development, for example, unit testing, feature testing, regression testing, etc.

Unit testing is used to verify that the code does what it is expected to do without error. Ideally, using the software engineering concept of Test-Driven Development (TDD) [337], the test is written before the code is written. This forces the developer to think more carefully about the structure of the code, consider the best ways to satisfy the expectations of the unit of code, and plan for any error conditions *before the code is written*.

The HydroShare team has tied unit testing to Jenkins, which is an open source, continuous integration tool [335]. Jenkins is used to implement continuous integration by automating runs of unit tests for both new code submissions and nightly builds of the main codebase. Unit testing is beneficial because it allows developers to test new features within the context of the existing code prior to inclusion in the main codebase. This is done to verify that a new feature will not cause existing tests to fail after it is integrated into the main codebase. When many features are merged, they are tested together in order to ensure that their interactions do not cause failures. Roughly every two to three weeks, the development branch is merged into the production codebase (or master branch), which is the code that runs on the publicly visible HydroShare production site [332]. In this way, new functionality is both adequately reviewed and tested, as well as rapidly released.

While TDD provides an important model for software development, the HydroShare team implemented a hybrid approach by authoring some unit tests after functional HydroShare code was written. This approach was prompted by time constraints and the fact that TDD has a steep learning curve that may cause an initial decrease in developer productivity [356]. Indeed, even at this writing, the HydroShare team is still acclimating to the TDD process. Moreover, the HydroShare team does not yet use TDD for development of user interfaces, as the integration of emulated user interface actions, combined with all relevant user traversals of the HydroShare web page environment, is currently a prohibitively complex development endeavor for a project of the scale and complexity of HydroShare. Testing, combined with thorough initial design, has been shown to result in approximately 40% fewer defects compared to code developed with more ad-hoc testing [358]. The HydroShare team continues to strive toward more comprehensive use of TDD.

### 10.4.7 Team Communication

Invariably, a new project will commence with a series of meetings. Among the topics of those meetings should be the plan for both team communication and the software development infrastructure (i.e., the software and hardware used for development). With HydroShare, the establishment of communication protocols and development infrastructure early on in the project supported collaboration and productivity and likely will continue to serve the team well throughout the lifetime of the project.

For example, for weekly meetings of distributed team members, the team employs videoconferencing software with screen sharing capability. For communication outside of meetings, a team email list is used. HipChat [331], a synchronous chat tool, was adopted as a place solely for development-centric discussion, so as to avoid overloading subject matter experts (i.e., domain scientists who do not participate in development) with extraneous information or noise that only serves to distract from the research process. Furthermore, the team adopted a content management system to host all documents for the project, including meeting notes, presentations, use cases, architectural diagrams, API documentation, policies, etc. The team also uses email lists to disseminate community announcements (e.g., announce@hydroshare.org, support@hydroshare.org) and to allow people to obtain support for HydroShare. To describe the project to interested parties, the team has created public-facing web pages. Each of these activities has proven important to the success of HydroShare.

### 10.4.8 DevOps

In addition to effective communication among team members, close collaboration is essential. *Development Operations or DevOps* is an industry concept that can be defined as an approach to software development that emphasizes the importance of collaboration between all stakeholders [327]. DevOps recognizes that stakeholders (e.g., programmers, scientists) do not work in isolation. This principle was adopted for HydroShare; software developers and domain scientists work together, closely and continuously, in the development of the HydroShare code. For HydroShare, a software engineer was selected to fill the DevOps lead role because s/he must be a maestro of Git, GitHub, and coding, and few team scientist-developers were skilled with modern software engineering techniques at the start of the project. The appointment of an experienced software engineer as the DevOps lead allows the scientist-developers to learn tools such as Git as they develop and contribute code. The DevOps lead facilitates this learning process by writing task automation scripts in order to simplify and optimize code contributions in Git. With HydroShare, GitHub is used for issue tracking in order to drive new development or track defects (i.e. bugs). GitHub issues are also used to track the progress of code reviews, with developers giving a simple "+1" to indicate that the code has been reviewed and that the DevOps lead may proceed with a code merge. Task automation scripts help the DevOps lead groom the code repository and make Git's branching and merging processes more transparent. Together, these activities contribute to the DevOps lead's ability to successfully ensure continuous integration with automated testing. DevOps thus foster team collaboration on many levels over the course of a software development process.

## 10.5 Making Software Engineering More Feasible and Easier to Integrate into One's Research Activities

Many research projects do not have sufficient funding to support training in software development and the fundamentals of good software engineering [345]. Moreover, rigid or process-heavy software development approaches have been shown to be unappealing to scientists [345]. Thus, accepted software engineering approaches to the design, development, documentation, testing, and review of code, for example, may not be employed by scientists. The result is software that is not sustainable or usable by others.

In order to infuse the scientific community with good software engineering practices, it is important to make software engineering practices more appealing to scientists. One approach to encourage the adoption of modern software engineering practices is to emphasize the end result: software that is useful, high quality, and sustainable [341].

Through the HydroShare project, an approach has been identified to integrate software engineering best practices into a large, distributed scientific software development project in a manner that is feasible for scientists. Provided below are several specific recommendations for integrating software engineering practices into one's research activities.

First, an initial design specification should be completed at the very beginning of a project, followed by an iterative design review for continuous refinement throughout the project development cycle. This software engineering practice increases both software quality and productivity. The initial design should be just enough to get project development going. The design should be reviewed iteratively for continuous refinement as project development advances. The initial minimal set of specifications provides sufficient constraint and guidance for the first iteration of software development in order to ensure that no time is wasted in the present producing a specification that would be changed or abandoned later (especially if one plans to "throw one away" as covered in Section 10.4.2 herein). The design specification then evolves in conjunction with software development to best serve its purpose of guiding and planning software development in a most productive way. In practice, the project team needs to ensure that team members who are contributing to the system design communicate well with team members who are contributing to the system development throughout the project development cycle. This is in order to streamline the process in such a way as to produce an evolving design specification that is just enough to guide development of high-quality software.

Second, iterative software releases and the release of a prototype early in the development iteration are recommended in order to solicit feedback from end users. The software engineering practice of iterative software releases brings end users into the loop in such a way that their feedback can be integrated into the iterative software design and development process as early as possible, thereby ensuring the delivery of a software product with a large user base. It would be regrettable for any software project, especially large-scale, complex scientific projects that require several years of team development effort, to yield an end product with very few end users. The integration of end user feedback throughout the software development cycle via iterative software releases can prevent such a regrettable scenario from happening by addressing end user concerns in a timely manner. Early in the development process, the focus should be on simple designs that best fit the daily workflow of end users in order to ensure efficient delivery of an easy-to-use, high-quality end product.

Last, the adoption of software engineering practices is crucial to ensure software quality and sustainability, but these practices should be applied selectively to individual projects, so as not to hinder research productivity. Through the HydroShare experience three software engineering practices have been identified that warrant particular consideration for making software engineering more feasible and easier to integrate into one's research activities; namely, code refactoring, code review, and software testing. Code refactoring is needed on occasion in order to make changes to the underlying data structures and frameworks so that subsequent software development will be based on a better foundation, thereby resulting in improvements in software quality and development productivity. Because code refactoring can be very disruptive and may require a great deal of effort, careful consideration must be paid to the costs-benefits before adopting code refactoring. In certain circumstances, proof-of-concept prototyping will be needed in advance of any decision to adopt code refactoring in order to prove that the benefits outweigh the costs. While scientists often assume that output errors are the result of faulty theory rather than faulty software [354], the adoption of code review and software testing as precepts of sound software engineering in large-scale, scientific software development projects will help to minimize output errors and ensure that the final software product is high quality and sustainable.

## 10.6 Conclusion

The HydroShare project is a work in progress, and exploration, refinement, and implementation of the topics herein are by no means finished. Rather, the goal is to provide readers with insight into the HydroShare experience and lessons learned in order to minimize the learning curve and accelerate the development progress for other teams. The goal of this chapter is to provide readers with a basic understanding of why good software engineering for science is tantamount to the success and sustainability of a scientific research project and why poor software engineering will detract from research time, with more time spent managing poorly written code than actually conducting research. In the long run, good software engineering will foster research and one's research career by ensuring the validity of research findings, reducing the amount of time needed to maintain and extend code, and improving the ease at which new features can be adopted, thus supporting software reuse and sustainability.

---

## Acknowledgments

Technical editorial and writing support was provided by Karamarie Fecho, Ph.D. This material is based upon work supported by the NSF under awards 1148453 and 1148090; any opinions, findings, conclusions, or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF. The authors wish to thank many who have contributed to the HydroShare project, including but not limited to: Jennifer Arrigo, Larry Band, Christina Bandaragoda, Alex Bedig, Brian Blanton, Jeff Bradberry, Chris Calloway, Claris Castillo, Tony Castronova, Mike Conway, Jason Coposky, Shawn Crawley, Antoine deTorcey, Tian Gan, Jon Goodall, Ilan Gray, Jeff Heard, Rick Hooper, Harry Johnson, Drew (Zhiyu) Li, Rob Lineberger, Yan Liu, Shaun Livingston, David Maidment, Phyllis Mbewe, Venkatesh Merwade, Setphanie Mills, Mohamed Morsy, Jon Pollak, Mauriel Ramirez, Terrell Russell, Jeff Sadler, Martin Seul, Kevin Smith, Carol Song, Lisa Stillwell, Nathan Swain, Sid Thakur, David Valentine, Tim Whiteaker, Zhaokun Xue, Lan Zhao, and Shandian Zhe.

The authors wish to especially thank Stan Ahalt, Director of RENCI, and Ashok Krishnamurthy, Deputy Director of RENCI for their continued organizational and supplemental financial support of this project.

# *References*


[1] J. C. Carver. First International Workshop on Software Engineering for Computational Science and Engineering. *Computing in Science Engineering*, 11(2):7–11, March 2009.

[2] J. C. Carver. Report: The Second International Workshop on Software Engineering for CSE. *Computing in Science Engineering*, 11(6):14–19, Nov 2009.

[3] Jeffrey C. Carver. Software engineering for computational science and engineering. *Computing in Science Engineering*, 14(2):8–11, March 2012.

[4] Jeffrey C. Carver, Neil Chue Hong, and Selim Ciraci. Software engineering for CSE. *Scientific Programming*, (591562):2, 2015.

[5] Jeffrey C. Carver and Tom Epperly. Software engineering for computational science and engineering [guest editors' introduction]. *Computing in Science Engineering*, 16(3):6–9, May 2014.

[6] S. H. D. Haddock and C. W. Dunn. *Practical Computing for Biologists*. Sinauer Associates, 2011.

[7] Sushil K. Prasad, Anshul Gupta, Arnold L. Rosenberg, Alan Sussman, and Charles C. Weems. *Topics in Parallel and Distributed Computing: Introducing Concurrency in Undergraduate Courses*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2015.

[8] A. Scopatz and K. D. Huff. *Effective Computation in Physics*. O'Reilly Media, 2015.

[9] Agile methodology. http://agilemethodology.org/.

[10] IDEAS productivity: "How To" documents. https://ideas-productivity.org/resources/howtos/.

[11] Waterfall model. https://www.techopedia.com/definition/14025/waterfall-model.

[12] In L. I. Sedov, editor, *Similarity and Dimensional Methods in Mechanics*, pages 24–96. Academic Press, 1959.

[13] The FLASH code. http://flash.uchicago.edu/flashcode, 2000.

[14] A. Dubey, K. Weide, D. Lee, J. Bachan, C. Daley, S. Olofin, N. Taylor, P.M. Rich, and L.B. Reid. Ongoing verification of a multiphysics community code: FLASH. *Software: Practice and Experience*, 45(2), 2015.

[15] A. L. Atchley, S. L. Painter, D. R. Harp, E. T. Coon, C. J. Wilson, A. K. Liljedahl, and V. E. Romanovsky. Using field observations to inform thermal hydrology models of permafrost dynamics with ATS (v0.83). *Geosci. Model Dev. Discuss.*, 8:3235–3292, 2015.

[16] V. R. Basili, J. C. Carver, D. Cruzes, L. M. Hochstein, J. K. Hollingsworth, F. Shull, and M. V. Zelkowitz. Understanding the high-performance-computing community: A software engineer's perspective. *IEEE Software*, 25(4):29, 2008.

[17] M. Bauer, S. Treichler, E. Slaughter, and A. Aiken. Legion: Expressing locality and independence with logical regions. In *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis*, page 66. IEEE Computer Society Press, 2012.

[18] M. Berzins, J. Luitjens, Q. Meng, T. Harman, C. A. Wight, and J. R. Peterson. Uintah - a scalable framework for hazard analysis. In *TG '10: Proc. of 2010 TeraGrid Conference*, New York, NY, USA, 2010. ACM.

[19] M. Blazewicz, I. Hinder, D. M. Koppelman, S. R. Brandt, M. Ciznicki, M. Kierzynka, F. Löffler, E. Schnetter, and J. Tao. From physics model to results: An optimizing framework for cross-architecture code generation. *Scientific Programming*.

[20] A. C. Calder. Laboratory astrophysics experiments for simulation code validation: A case study. *Astrophysics and Space Science*, 298:25–32, July 2005.

[21] J. C. Carver. Software engineering for computational science and engineering. *Computing in Science & Engineering*, 14(2):8–11, 2012.

[22] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post. Software development environments for scientific and engineering software: A series of case studies. In *Software Engineering, 2007. ICSE 2007*, pages 550–559. IEEE, 2007.

[23] D. A. Case, V. Babin, J. Berryman, R. M. Betz, Q. Cai, D. S. Cerutti, T. E. Cheatham Iii, T. A. Darden, R. E. Duke, H. Gohlke, et al. Amber 2015. http://ambermd.org/, 2015.

[24] E. T. Coon, J. D. Moulton, and S. L. Painter. Managing complexity in simulations of land surface and near-surface processes. *Environmental Modelling & Software*, 78:134–49, 2016.

[25] G. Dimonte, D. L. Youngs, A. Dimits, S. Weber, M. Marinak, S. Wunsch, C. Garasi, A. Robinson, M. J. Andrews, P. Ramaprabhu, A. C. Calder, B. Fryxell, J. Biello, L. Dursi, P. MacNeice, K. Olson, P. Ricker, R. Rosner, F. Timmes, H. Tufo, Y.-N. Young, and M. Zingale. A comparative study of the turbulent Rayleigh–Taylor instability using high-resolution three-dimensional numerical simulations: The Alpha-Group collaboration. *Physics of Fluids*, 16:1668–1693, May 2004.

[26] A. Dubey, K. Antypas, M. K. Ganapathy, L. B. Reid, K. Riley, D. Sheeler, A. Siegel, and K. Weide. Extensible component-based architecture for FLASH, a massively parallel, multiphysics simulation code. *Parallel Computing*, 35(10-11):512–522, 2009.

[27] A. Dubey, A. C. Calder, C. Daley, R. T. Fisher, C. Graziani, G. C. Jordan, D. Q. Lamb, L. B. Reid, D. M. Townsley, and K. Weide. Pragmatic optimizations for better scientific utilization of large supercomputers. *International Journal of High Performance Computing Applications*, 27(3):360–373, 2013.

[28] A. Dubey and T. Clune. Optimization techniques for pseudospectral codes on MPPs. In *Proceedings of Frontiers 99*, 1999.

[29] A. Dubey, C. Daley, J. ZuHone, P. M. Ricker, K. Weide, and C. Graziani. Imposing a Lagrangian particle framework on an Eulerian hydrodynamics infrastructure in FLASH. *ApJ Supplement*, 201:27, Aug 2012.

[30] A. Dubey, D. Q. Lamb, and E. Balaras. Building community codes for effective scientific research on HPC platforms. http://flash.uchicago.edu/cc2012, 2012.

[31] A. Dubey, L. B. Reid, and R. Fisher. Introduction to FLASH 3.0, with application to supersonic turbulence. *Physica Scripta*, T132, 2008. Topical Issue on Turbulent Mixing and Beyond, results of a conference at ICTP, Trieste, Italy, August 2008.

[32] A. Dubey, A. Almgren, J. Bell, M. Berzins, S. Brandt, G. Bryan, P. Colella, D. Graves, M. Lijewski, F. Löffler, B. O'Shea, E. Schnetter, B. Van Straalen, and K. Weide. A survey of high level frameworks in block-structured adaptive mesh refinement packages. *Journal of Parallel and Distributed Computing*, 74(12):3217–3227, 2014.

[33] B. Fryxell, K. Olson, P. Ricker, F. X. Timmes, M. Zingale, D. Q. Lamb, P. MacNeice, R. Rosner, J. W. Truran, and H. Tufo. Flash: An adaptive mesh hydrodynamics code for modeling astrophysical thermonuclear flashes. *Astrophysical Journal, Supplement*, 131:273–334, 2000.

[34] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson. How do scientists develop and use scientific software? In

*Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8. IEEE Computer Society, 2009.

[35] M. A. Heroux and J. M. Willenbring. Barely sufficient software engineering: 10 practices to improve your cse software. In *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, SECSE '09, pages 15–21, Washington, DC, USA, 2009. IEEE Computer Society.

[36] L. Hochstein and V. R. Basili. The ASC-alliance projects: A case study of large-scale parallel scientific code development. *Computer*, (3):50–58, 2008.

[37] L. V. Kale, E. Bohm, C. L. Mendes, T. Wilmarth, and G. Zheng. Programming petascale applications with Charm++ and AMPI. *Petascale Computing: Algorithms and Applications*, 1:421–441, 2007.

[38] J. O. Kane, H. F. Robey, B. A. Remington, R. P. Drake, J. Knauer, D. D. Ryutov, H. Louis, R. Teyssier, O. Hurricane, D. Arnett, et al. Interface imprinting by a rippled shock using an intense laser. *Physical Review E, Statistical, Nonlinear, and Soft Matter Physics*, 63(5 Pt 2):055401, 2001.

[39] Q. Meng, J. Luitjens, and M. Berzins. Dynamic task scheduling for the uintah framework. In *Proceedings of the 3rd IEEE Workshop on Many-Task Computing on Grids and Supercomputers (MTAGS10)*, 2010.

[40] D. Monniaux. The pitfalls of verifying floating-point computations. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 30(3):12, 2008.

[41] D. Moulton, M. Berndt, M. Buskas, R. Garimella, L. Prichett-Sheats, G. Hammond, M. Day, and J. Meza. High-level design of Amanzi, the multi-process high performance computing simulator. Technical report, ASCEM-HPC-2011-03-1, US Department of Energy, Washington, DC, 2011.

[42] J. D. Moulton, J. C. Meza, and M. Day et al. High-level design of Amanzi, the multi-process high performance computing simulator. Technical report, DOE-EM, Washington, DC, 2012.

[43] L. Nguyen–Hoan, S. Flint, and R. Sankaranarayana. A survey of scientific software development. In *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, pages 12:1–12:10, New York, NY, 2010. ACM.

[44] P. K. Notz, R. P. Pawlowski, and J. C. Sutherland. Graph-based software design for managing complexity and enabling concurrency in multiphysics PDE software. *ACM Trans. Math. Softw.*, 39(1):1:1–1:21, November 2012.

[45] W. L. Oberkampf and C. J. Roy. *Verification and Validation in Scientific Computing*. Cambridge University Press, 2010.

[46] W. L. Oberkampf and T. G. Trucano. Verification and validation in computational fluid dynamics. *Progress in Aerospace Sciences*, 38(3):209–272, 2002.

[47] S. L. Painter, J. D. Moulton, and C. J. Wilson. Modeling challenges for predicting hydrologic response to degrading permafrost. *Hydrogeol. J.*, pages 1–4, 2013.

[48] S. G. Parker. A component-based architecture for parallel multi-physics PDE simulation. *Future Generation Comput. Sys.*, 22:204–216, 2006.

[49] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kale, and K. Schulten. Scalable molecular dynamics with namd. *Journal of Computational Chemistry*, 26(16):1781–1802, 2005.

[50] R. G. Sargent. Verification and validation of simulation models. In *Proceedings of the 30th Conference on Winter Simulation*, pages 121–130. IEEE Computer Society Press, 1998.

[51] J. Segal. When software engineers met research scientists: a case study. *Empirical Software Engineering*, 10(4):517–536, 2005.

[52] J. Segal and C. Morris. Developing scientific software. *Software, IEEE*, 25(4):18–20, 2008.

[53] The Enzo Collaboration, G. L. Bryan, M. L. Norman, B. W. O'Shea, T. Abel, J. H. Wise, M. J. Turk, D. R. Reynolds, D. C. Collins, P. Wang, S. W. Skillman, B. Smith, R. P. Harkness, J. Bordner, J.-H. Kim, M. Kuhlen, H. Xu, N. Goldbaum, C. Hummels, A. G. Kritsuk, E. Tasker, S. Skory, C. M. Simpson, O. Hahn, J. S. Oishi, G. C. So, F. Zhao, R. Cen, and Y. Li. Enzo: An Adaptive Mesh Refinement Code for Astrophysics. *ArXiv e-prints*, July 2013.

[54] D. Unat, J. Shalf, T. Hoefler, T. Schulthess, A. Dubey, et al. Programming abstractions for data locality. In *Workshop on Programming Abstractions for Data Locality (PADAL'14)*, 2014.

[55] Karen S. Ackroyd, Steve H. Kinder, Geoff R. Mant, Mike C. Miller, Christine A. Ramsdale, and Paul C. Stephenson. Scientific software development at a research facility. *IEEE Software*, 25(4):44–51, July/August 2008.

[56] Arne Beckhause, Dirk Neumann, and Lars Karg. The impact of communcation structure on issue tracking efficiency at a large business software vendor. *Issues in Information Systems*, X(2):316–323, 2009.

[57] Jacques Carette. Gaussian elimination: A case study in efficient genericity with MetaOCaml. *Science of Computer Programming*, 62(1):3–24, 2006.

[58] Jacques Carette, Mustafa ElSheikh, and W. Spencer Smith. A generative geometric kernel. In *ACM SIGPLAN 2011 Workshop on Partial Evaluation and Program Manipulation (PEPM'11)*, pages 53–62, January 2011.

[59] Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. Software development environments for scientific and engineering software: A series of case studies. In *ICSE '07: Proceedings of the 29th International Conference on Software Engineering*, pages 550–559, Washington, DC, USA, 2007. IEEE Computer Society.

[60] CIG. Mineos. `http://geodynamics.org/cig/software/mineos/`, March 2015.

[61] CRAN. The comprehensive R archive network. `https://cran.r-project.org/`, 2014.

[62] CSA. Quality assurance of analytical, scientific, and design computer programs for nuclear power plants. Technical Report N286.7-99, Canadian Standards Association, 178 Rexdale Blvd. Etobicoke, Ontario, Canada M9W 1R3, 1999.

[63] Andrew P. Davison. Automated capture of experiment context for easier reproducibility in computational research. *Computing in Science & Engineering*, 14(4):48–56, July-Aug 2012.

[64] Andrew P. Davison, M. Mattioni, D. Samarkanov, and B. Teleńczuk. Sumatra: A toolkit for reproducible research. In V. Stodden, F. Leisch, and R.D. Peng, editors, *Implementing Reproducible Research*, pages 57–79. Chapman & Hall/CRC, Boca Raton, FL, March 2014.

[65] Paul F. Dubois. Designing scientific components. *Computing in Science and Engineering*, 4(5):84–90, September 2002.

[66] Paul F. Dubois. Maintaining correctness in scientific programs. *Computing in Science & Engineering*, 7(3):80–85, May-June 2005.

[67] Steve M. Easterbrook and Timothy C. Johns. Engineering the software for understanding climate change. *IEEE Des. Test*, 11(6):65–74, 2009.

[68] Ahmed H. ElSheikh, W. Spencer Smith, and Samir E. Chidiac. Semi-formal design of reliable mesh generation systems. *Advances in Engineering Software*, 35(12):827–841, 2004.

[69] ESA. ESA software engineering standards, PSS-05-0 issue 2. Technical report, European Space Agency, February 1991.

[70] Sergey Fomel. Madagascar Project Main Page. `http://www.ahay.org/wiki/Main_Page`, 2014.

[71] Carlo Ghezzi, Mehdi Jazayeri, and Dino Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, Upper Saddle River, NJ, USA, 2nd edition, 2003.

[72] GRASS Development Team. GRASS GIS bringing advanced geospatial technologies to the world. `http://grass.osgeo.org/`, 2014.

[73] Michael Heath. *Scientific Computing: An Introductory Survey*. McGraw-Hill Publishing Company, New York, NY, 2nd edition, 2002.

[74] Timothy Hickey, Qun Ju, and Maarten H. Van Emden. Interval arithmetic: From principles to implementation. *J. ACM*, 48(5):1038–1068, September 2001.

[75] Daniel M. Hoffman and Paul A. Strooper. *Software Design, Automated Testing, and Maintenance: A Practical Approach*. International Thomson Computer Press, New York, NY, 1995.

[76] IEEE. *Recommended Practice for Software Requirements Specifications, IEEE Std. 830*. IEEE, 1998.

[77] ISTI. Earthworm software standards. `http://www.earthwormcentral.org/documentation2/PROGRAMMER/SoftwareStandards.html`, September 2013.

[78] Jeffrey N Johnson and Paul F Dubois. Issue tracking. *Computing in Science & Engineering*, 5(6):71–77, 2003.

[79] Diane Kelly. Industrial scientific software: A set of interviews on software development. In *Proceedings of the 2013 Conference of the Center for Advanced Studies on Collaborative Research*, CASCON '13, pages 299–310, Riverton, NJ, USA, 2013. IBM Corp.

[80] Diane Kelly. Scientific software development viewed as knowledge acquisition: Towards understanding the development of risk-averse scientific software. *Journal of Systems and Software*, 109:50–61, 2015.

[81] Diane F. Kelly, W. Spencer Smith, and Nicholas Meng. Software engineering for scientists. *Computing in Science & Engineering*, 13(5):7–11, October 2011.

[82] Brian W. Kernighan and Rob Pike. *The Practice of Programming*. Addison-Wesley Professional, Reading, MA, 1999.

[83] Oleg Kiselyov, Kedar N. Swadi, and Walid Taha. A methodology for generating verified combinatorial circuits. In *Proceedings of the 4th ACM International Conference on Embedded Software*, EMSOFT '04, pages 249–258, New York, NY, USA, 2004. ACM.

[84] Donald E. Knuth. *Literate Programming*. CSLI Lecture Notes Number 27. Center for the Study of Language and Information, 1992.

[85] Adam Lazzarato, Spencer Smith, and Jacques Carette. State of the practice for remote sensing software. Technical Report CAS-15-03-SS, McMaster University, January 2015. 47 pp.

[86] Friedrich Leisch. Sweave: Dynamic generation of statistical reports using literate data analysis. In Wolfgang Härdle and Bernd Rönz, editors, *Compstat 2002 — Proceedings in Computational Statistics*, pages 575–580. Physica Verlag, Heidelberg, 2002. ISBN 3-7908-1517-9.

[87] Jon Loeliger and Matthew McCullough. *Version Control with Git: Powerful Tools and Techniques for Collaborative Software Development*. O'Reilly Media, Inc., 2012.

[88] Thomas Maibaum and Alan Wassyng. A product-focused approach to software certification. *IEEE Computer*, 41(2):91–93, 2008.

[89] NASA. Software requirements DID, SMAP-DID-P200-SW, release 4.3. Technical report, National Aeronautics and Space Agency, 1989.

[90] Nedialko S. Nedialkov. Implementing a Rigorous ODE Solver through Literate Programming. Technical Report CAS-10-02-NN, Department of Computing and Software, McMaster University, 2010.

[91] Suely Oliveira and David E. Stewart. *Writing Scientific Software: A Guide to Good Style*. Cambridge University Press, New York, NY, USA, 2006.

[92] Linda Parker Gates. Strategic planning with critical success factors and future scenarios: An integrated strategic planning framework. Technical Report CMU/SEI-2010-TR-037, Software Engineering Institute, Carnegie-Mellon University, November 2010.

[93] David L. Parnas. On the criteria to be used in decomposing systems into modules. *Comm. ACM*, 15(2):1053–1058, December 1972.

[94] David L. Parnas, P. C. Clement, and D. M. Weiss. The modular structure of complex systems. In *International Conference on Software Engineering*, pages 408–419, 1984.

[95] David L. Parnas and P.C. Clements. A rational design process: How and why to fake it. *IEEE Transactions on Software Engineering*, 12(2):251–257, February 1986.

[96] David Lorge Parnas. Precise documentation: The key to better software. In *The Future of Software Engineering*, pages 125–148, 2010.

[97] Matt Pharr and Greg Humphreys. *Physically Based Rendering: From Theory to Implementation*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2004.

[98] Michael Pilato. *Version Control With Subversion*. O'Reilly & Associates, Inc., Sebastopol, CA, USA, 2004.

[99] Patrick J. Roache. *Verification and Validation in Computational Science and Engineering*. Hermosa Publishers, Albuquerque, NM, 1998.

[100] Padideh Sarafraz. Thermal optimization of flat plate PCM capsules in natural convection solar water heating systems. Master's thesis, McMaster University, Hamilton, ON, Canada, 2014. http://hdl.handle.net/11375/14128.

[101] Judith Segal. When software engineers met research scientists: A case study. *Empirical Software Engineering*, 10(4):517–536, October 2005.

[102] Judith Segal and Chris Morris. Developing scientific software. *IEEE Software*, 25(4):18–20, July/August 2008.

[103] W. Spencer Smith and Nirmitha Koothoor. A document driven method for certifying scientific computing software used in nuclear safety analysis. *Nuclear Engineering and Technology*, Accepted, October 2015. 42 pp.

[104] W. Spencer Smith, Yue Sun, and Jacques Carette. Comparing psychometrics software development between CRAN and other communities. Technical Report CAS-15-01-SS, McMaster University, January 2015. 43 pp.

[105] W. Spencer Smith, Yue Sun, and Jacques Carette. Statistical software for psychology: Comparing development practices between CRAN and other communities. *Software Quality Journal*, Submitted December 2015. 33 pp.

[106] W. Spencer Smith. Systematic development of requirements documentation for general purpose scientific computing software. In *Proceedings of the 14th IEEE International Requirements Engineering Conference, RE 2006*, pages 209–218, Minneapolis / St. Paul, MN, 2006.

[107] W. Spencer Smith, Nirmitha Koothoor, and Nedialko Nedialkov. Document driven certification of computational science and engineering software. In *Proceedings of the First International Workshop on Software Engineering for High Performance Computing in Computational Science and Engineering (SE-HPCCE)*, November 2013. 8 pp.

[108] W. Spencer Smith and Lei Lai. A new requirements template for scientific computing. In J. Ralyté, P. Ȧgerfalk, and N. Kraiem, editors, *Proceedings of the First International Workshop on Situational Requirements Engineering Processes – Methods, Techniques and Tools to Support Situation-Specific Requirements Engineering Processes, SREP'05*, pages 107–121, Paris, France, 2005. In conjunction with 13th IEEE International Requirements Engineering Conference.

[109] W. Spencer Smith, Lei Lai, and Ridha Khedri. Requirements analysis for engineering computation: A systematic approach for improving software reliability. *Reliable Computing, Special Issue on Reliable Engineering Computation*, 13(1):83–107, February 2007.

[110] W. Spencer Smith, John McCutchan, and Fang Cao. Program families in scientific computing. In Jonathan Sprinkle, Jeff Gray, Matti Rossi, and Juha-Pekka Tolvanen, editors, *$7^{th}$ OOPSLA Workshop on Domain Specific Modelling (DSM'07)*, pages 39–47, Montréal, Québec, October 2007.

[111] W. Spencer Smith and Wen Yu. A document driven methodology for improving the quality of a parallel mesh generation toolbox. *Advances in Engineering Software*, 40(11):1155–1167, November 2009.

[112] Daniel Szymczak, W. Spencer Smith, and Jacques Carette. Position paper: A knowledge-based approach to scientific software development. In *Proceedings of SE4Science'16*, United States, May 16 2016. In conjunction with ICSE 2016. 4 pp.

[113] L. Andries van der Ark. *mokken: Mokken Scale Analysis in R*, 2013. R package version 2.7.5.

[114] Hans van Vliet. *Software Engineering (2nd ed.): Principles and Practice*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[115] Judith S. VanAlstyne. *Professional and Technical Writing Strategies*. Pearson Prentice Hall, Upper Saddle River, NJ, sixth edition, 2005.

[116] Gregory V. Wilson. Where's the real bottleneck in scientific computing? Scientists would do well to pick some tools widely used in the software industry. *American Scientist*, 94(1), 2006.

[117] Gregory V. Wilson, D.A. Aruliah, C. Titus Brown, Neil P. Chue Hong, Matt Davis, Richard T. Guy, Steven H. D. Haddock, Kathryn D. Huff,

Ian M. Mitchell, Mark D. Plumblet, Ben Waugh, Ethan P. White, and Paul Wilson. Best practices for scientific computing. *CoRR*, abs/1210.0530, 2013.

[118] A. Arcuri, M. Z. Iqbal, and L. Briand. Random Testing: Theoretical Results and Practical Implications. *IEEE Trans. Software Engineering*, 38(2):258–277, 2012.

[119] J. M. Bové. Huanglongbing: A Destructive, Newly-Emerging, Century-Old Disease of Citrus. *Journal of Plant Pathology*, 88:7–37, 2006.

[120] F. Brayton, A. Levin, R. Tryon, and J. C. Williams. The Evolution of Macro Models at the Federal Reserve Board. In *Carnegie Rochester Conference Series on Public Policy*, pages 43–81, 1997.

[121] I. Burnstein. *Practical Software Testing: A Process-Oriented Approach.* Springer, New York, NY, 2003.

[122] F. T. Chan, T. Y. Chen, S. C. Cheung, M. F. Lau, and S. M. Yiu. Application of metamorphic testing in numerical analysis. In *IASTED International Conference on Software Engineering*, pages 191–197, 1998.

[123] N. J. Cunniffe, R. O. J. H. Stutt, R. E. DeSimone, T. R. Gottwald, and C. A. Gilligan. Optimizing and Communicating Options for the Control of Invasive Plant Disease When There Is Epidemiological Uncertainty. *PLOS Computational Biology*, 2015.

[124] O. Diekmann, J. A. P. Heesterbeek, and J. A. J. Metz. The Oracle Problem in Software Testing: A Survey. *Journal of Mathematical Biology*, 28(4):365–382, 1990.

[125] J. W. Duran. An Evaluation of Random Testing. *IEEE Trans. Software Engineering*, 10(4):438–444, 1984.

[126] A. Geller and S. J. Alam. A Socio-Political and -Cultural Model of the War in Afghanistan. *International Studies Review*, 12(1):8–30.

[127] M. F. C. Gomes, A. Pastore y Piontti, L. Rossi, D Chao, I. Longini, M. E. Halloran, and A. Vespignani. Assessing the International Spreading Risk Associated with the 2014 West African Ebola Outbreak. *PLoS Curr*, 2014.

[128] T. R. Gottwald. Current Epidemiological Understanding of Citrus Huanglongbing. *Annual Review of Phytopathology*, 48:119–139, 2010.

[129] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson. How Do Scientists Develop and Use Scientific Software? In *Soft. Eng. for Computational Science and Eng., ICSE*, 2009.

[130] N. Hansen. The CMA Evolution Strategy: A Comparing Review. In J. A. Lozano, P. Larrañaga, I. Inza, and E. Bengoetxea, editors, *Towards a New Evolutionary Computation (Studies in Fuzziness and Soft Computing)*, pages 75–102. Berlin, Germany: Springer, 2006.

[131] N. Hansen. CMA-ES Source Code. `https://www.lri.fr/~hansen/cmaes\_inmatlab.html`, 2011. [Online]. Accessed 24 March 2016.

[132] N. Hansen, A. Auger, R. Ros, S. Finck, and P. Posik. Comparing Results of 31 Algorithms from the Black-Box Optimization Benchmarking BBOB-2009. In *Proc. 12th Genetic Evolutionary Computation Conf.*, pages 1689–1696, 2010.

[133] P. R. Harper and A.K. Shahani. Modelling for the Planning and Management of Bed Capacities in Hospitals. *Journal of the Operational Research Society*, 53(1):11–18, 2006.

[134] L. Hatton and A. Roberts. How Accurate is Scientific Software? *IEEE Trans. Software Engineering*, 20(10):785–797, 1994.

[135] S. Hettrick, M. Antonioletti, L. Carr, N. Chue Hong, S. Crouch, D. De Roure, I. Emsley, C. Goble, A. Hay, D. Inupakutika, M. Jackson, A. Nenadic, T. Parkinson, M. I. Parsons, A. Pawlik, G. Peru, A. Proeme, J. Robinson, and S. Sufi. UK Research Software Survey 2014. `https://zenodo.org/record/14809`, note = "[Online]. Accessed 24 March 2016".

[136] P.C. Jorgensen. *Software Testing: A Craftsman's Approach*. CRC Press, Boca Raton, FL, 4th edition, 2013.

[137] M. J. Keeling, M. E. J. Woolhouse, R. M. May, G. Davies, and B. T. Grenfell. Modelling Vaccination Strategies against Foot-and-Mouth Disease. *Nature*, 421:136–142, 2003.

[138] D. F. Kelly. A Software Chasm: Software Engineering and Scientific Computing. *IEEE Software*, 24(6):118–120, 2007.

[139] R. C. Martin. *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall, Upper Saddle River, NJ, 2008.

[140] F. Massey. The Kolmogorov–Smirnov Test for Goodness of Fit. *Journal of the American Statistical Association*, 46(253):68–78, 1951.

[141] R. K. Meentemeyer, N. J. Cunniffe, A. R. Cook, J. A. N. Filipe, R. D. Hunter, D. M. Rizzo, and C. A. Gilligan. Epidemiological Modeling of Invasion in Heterogeneous Landscapes: Spread of Sudden Oak Death in California (1990–2030). *Ecosphere*, 2(2), 2011.

[142] Z. Merali. Computational science: Error, why scientific programming does not compute. *Nature*, 467(7317), 2010.

[143] H. Motulsky. Comparing Dose-Response or Kinetic Curves with GraphPad Prism. *HMS Beagle: The BioMedNet Magazine*, 34, 1998.

[144] D. Orchard and A. Rice. A Computational Science Agenda for Programming Language Research. In *Proc. International Conference on Computational Science*, pages 713–727, 2014.

[145] D. L. Parnas. On the Criteria to Be Used in Decomposing Systems into Modules. *Communications of the ACM*, 15(12):1053–1058, 1972.

[146] M. Parry, G. J. Gibson, S. Parnell, T. R. Gottwald, M. S. Irey, T. C. Gast, and C. A. Gilligan. Bayesian Inference for an Emerging Arboreal Epidemic in the Presence of Control. *Proc. National Academy of Sciences*, 111(17):6258–6262, 2014.

[147] A. Piccolboni. Quickcheck. `https://github.com/RevolutionAnalytics/quickcheck`, 2015. [Online]. Accessed 24 March 2016.

[148] K. Salari and P. Knupp. Code Verification by the Method of Manufactured Solutions. Technical Report SAND2000-1444, Sandia National Laboratories, June 2000.

[149] S. Shamshiri, J. M. Rojas, G. Fraser, and P. McMinn. Random or Genetic Algorithm Search for Object-Oriented Test Suite Generation? In *Proc. GECCO*, pages 1367–1374, 2015.

[150] J. A. Sokolowski and C. M. Banks. *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*. Wiley, Hoboken, NJ, 4 edition, 2010.

[151] F. W. Thackeray and J. E. Findling. *Events That Formed the Modern World*. ABC CLIO, Santa Barbara, CA, 2012.

[152] J. Utts and R. Heckard. *Statistical Ideas and Methods*. Thomson, Belmont, CA, 2005.

[153] T. Weise. Global Optimization Algorithms - Theory and Application. `http://www.itweise.de/projects/book.pdf`, 2009. [Online]. Accessed 24 March 2016.

[154] E. J. Weyuker. On Testing Non-Testable Programs. *The Computer Journal*, 25(4):465–470, 1982.

[155] A. Dubey, K. Weide, D. Lee, J. Bachan, C. Daley, S. Olofin, N. Taylor, P.M. Rich, and L.B. Reid. Ongoing verification of a multiphysics community code: FLASH. *Software: Practice and Experience*, 45(2), 2015.

[156] Bamboo. https://www.atlassian.com/software/bamboo/.

[157] K. J. Bathe, editor. *Computational Fluid and Solid Mechanics*. Elsevier, 2001.

[158] K. Beck. *Test Driven Development*. Addison-Wesley, Boston, MA, 2003.

[159] K. Beck. *Extreme Programming (Second Edition)*. Addison-Wesley, 2005.

[160] Thirumalesh Bhat and Nachiappan Nagappan. Evaluating the efficacy of test-driven development: Industrial case studies. In *Proceedings of the 2006 ACM/IEEE International Symposium on Empirical Software Engineering*, ISESE '06, pages 356–363, New York, NY, USA, 2006. ACM.

[161] F. Brooks. *The Mythical Man-Month (second edition)*. Addison-Wesley, Boston, MA, 1995.

[162] CDash. www.cdash.org.

[163] E. Coon, J. D. Moulton, and S. Painter. Managing complexity in simulations of land surface and near-surface processes. Technical Report LA-UR 14-25386, Applied Mathematics and Plasma Physics Group, Los Alamos National Laboratory, 2014. To appear in *Environmental Modelling & Software*.

[164] CTest. https://cmake.org/Wiki/CMake/Testing_With_CTest.

[165] S. M. Easterbrook and T. C. Johns. Engineering the software for understanding climate change. *Computing in Science Engineering*, 11(6):65–74, Nov.-Dec. 2009.

[166] H. Erdogmus, M. Morisio, and M. Torchiano. On the effectiveness of the test-first approach to programming. *IEEE Transactions on Software Engineering*, 31(3):226–237, 2005.

[167] M. Feathers. *Working Effectively with Legacy Code*. Prentice-Hall, Upper Saddle River, NJ, 2004.

[168] M. Fowler. *Refactoring (Improving the Design of Existing Code)*. Addison Wesley, 1999.

[169] M. Gartner. *ATDD by Example: A Practical Guide to Acceptance Test-Driven Development*. Addison-Wesley, 2012.

[170] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, March 1991.

[171] gtest. http://wiki.ros.org/gtest.

[172] *hypre*: High Performance Preconditioners. . http://www.llnl.gov/CASC/hypre/.

[173] Parasoft Insure++. https://www.parasoft.com/product/insure/.

[174] Jenkins. https://jenkins-ci.org/.

[175] B. Koteska and A. Mishev. Scientific software testing: A practical example. In Z. Budimar and M. Hericko, editors, *Proceedings of the 4th Workshop of Software Quality, Analysis, Monitoring, Improvement, and Applications (SQAMIA 2015)*, Maribor, Slovenia, 8.-10.6.2015.

[176] X. S. Li, J. W. Demmel, J. R. Gilbert, L. Grigori, P. Sao, M. Shao, and I. Yamazaki. SuperLU Users' Guide. Technical Report LBNL-44289, Lawrence Berkeley National Laboratory, October 2014.

[177] S. McConnell. *Code Complete: Second Edition.* Microsoft Press, 2004.

[178] G. Miller. A scientist's nightmare: Software problem leads to five retractions. *Science*, 314(5807):1856–1857, 2006.

[179] J. Nocedal and S. Wright. *Numerical Optimization.* Springer, 1999.

[180] P. K. Notz, R. P. Pawlowski, and J. C. Sutherland. Graph-based software design for managing complexity and enabling concurrency in multiphysics PDE software. *Acm. T. Math. Software*, 39(1):1, 2012.

[181] M. Poppendieck and T. Poppendieck. *Implementing Lean Software Development.* Addison-Wesley, 2007.

[182] D. Post and L. Votta. Computational science demands and new paradigm. *Physics Today*, 58(1):35–41, 2005.

[183] J. Seward and N. Nethercote. Using valgrind to detect undefined value errors with bit-precision. In *Proceedings of the Annual Conference on USENIX Annual Technical Conference*, ATEC '05, pages 2–2, Berkeley, CA, USA, 2005. USENIX Association.

[184] A. K. Shuja and J. Krebs. *IBM Rational Unified Process Reference and Certification Guide: Solution Designer.* IBM Press, 2007.

[185] S. R. Slattery, P. P. H. Wilson, and R. P. Pawlowski. The Data Transfer Kit: A geometric rendezvous-based tool for multiphysics data transfer. In *International Conference on Mathematics & Computational Methods Applied to Nuclear Science & Engineering*, pages 5–9, 2013.

[186] SuperLU. http://crd-legacy.lbl.gov/~xiaoye/SuperLU/.

[187] D. Talby, A. Keren, O. Hazzan, and Y. Dubinsky. Agile software testing in a large-scale project. *Software, IEEE*, 23(4):30–37, 2006.

[188] The Trilinos Project. https://trilinos.org.

[189] H. Troger and A. Steindl. *Nonlinear stability and bifurcation theory: An introduction for engineers and applied scientists*. Springer, 1991.

[190] B. Turhan, L. Layman, M. Diep, H. Erdogmus, and F. Shull. How effective is test-driven development? In *Making Software: What Really Works and Why We Believe It*, pages 207–217. O'Reilly, 2010.

[191] xUnit. http://xunit.github.io/.

[192] Boris Beizer. *Software Testing Techniques*. Dreamtech Press, 2003.

[193] David L. Donoho, Arian Maleki, Inam Ur Rahman, Morteza Shahram, and Victoria Stodden. Reproducible research in computational harmonic analysis. *Computing in Science and Engineering*, 11(1):8–18, 2009.

[194] Steve M. Easterbrook and Timothy C. Johns. Engineering the software for understanding climate change. *Computing in Science and Engineering*, 11(6):64–74, 2009.

[195] Rob Gray and Diane Kelly. Investigating test selection techniques for scientific software using Hook's mutation sensitivity testing. *Procedia Computer Science*, 1(1):1487–1494, 2010.

[196] Daniel Hook. Using Code Mutation to Study Code Faults in Scientific Software. Master's thesis, Queen's University, Kingston, ON, Canada, April 2009. Available at `http://hdl.handle.net/1974/1765`.

[197] Daniel Hook and Diane Kelly. Mutation sensitivity testing. *Computing in Science and Engineering*, 11(6):40–47, 2009.

[198] Upulee Kanewala and James M. Bieman. Testing scientific software: A systematic literature review. *Information and Software Technology*, 56(10):1219–1232, 2014.

[199] Nicholas Jie Meng. A Model for Run-time Measurement of Input and Round-off Error. Master's thesis, Queen's University, Kingston, ON, Canada, September 2012. Available at `http://hdl.handle.net/1974/7508`.

[200] Glenford J Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. John Wiley & Sons, 2011.

[201] William L. Oberkampf, Timothy G. Trucano, and Charles Hirsch. Verification, validation, and predictive capability in computational engineering and physics. In *Proceedings of Foundations '02, a Workshop on Modeling and Simulation Verification and Validation for the 21st Century*, Laurel, MD, USA, October 2002. Johns Hopkins University.

[202] Rebecca Sanders and Diane Kelly. The challenge of testing scientific software. In *CAST '08: Proceedings of the 3rd Annual Conference of the Association for Software Testing*, pages 30–36, Toronto, ON, Canada, 2008. Association for Software Testing.

[203] Rebecca Sanders and Diane Kelly. Dealing with risk in scientific software development. *IEEE Software*, 25(4):21–28, 2008.

[204] A framework to write repeatable Java tests. available at http://junit.org, Accessed: 12-20-2015.

[205] A Unit Testing Framework for C. available at http://cunit. sourceforge.net, Accessed: 12-10-2015.

[206] A Unit Testing Framework for FORTRAN. available at https://rubygems.org/gems/funit/versions/0.11.1, Accessed: 20-12-2015.

[207] Cube 4.x series, 2015. Version 4.3.2, available at http://www.scalasca.org/software/cube-4.x/download.html, Accessed: 06-10-2015.

[208] D. Babic, L. Martignoni, S. McCamant, and D. Song. Statically-directed dynamic automated test generation. pages 12–22, 2011.

[209] G. B. Bonan. The land surface climatology of the NCAR land surface model coupled to the NCAR community. *Climate Model. J. Climate*, 11:1307–1326.

[210] C. Cadar, D. Dunbar, and D. Engler. Klee: Unassisted and automatic generation of high-coverage tests for complex systems programs. pages 209–224, 2008.

[211] D. Wang, W. Wu, T. Janjusic, Y. Xu, C. Iversen, P. Thornton, and M. Krassovisk. Scientific functional testing platform for environmental models: An application to community land model. *International Workshop on Software Engineering for High Performance Computing in Science*, 2015.

[212] R. E. Dickinson, K. W. Oleson, G. Bonan, F. Hoffman, P. Thornton, M. Vertenstein, Z. Yang, and X. Zeng. The community land model and its climate statistics as a component of the community climate system model. *J. Clim.*, 19:2302–2324, 2006.

[213] M. Feathers. *Working Effectively with Legacy Code*. Prentice-Hall, 2004.

[214] A. Knüpfer, C. Rössel, D. Mey, S. Biersdorf, K. Diethelm, D. Eschweiler, M. Gerndt, D. Lorenz, A. D. Malony, W. E. Nagel, Y. Oleynik, P. Saviankou, D. Schmidl, S. Shende, R. Tschüter, M. Wagner, B. Wesarg, and F. Wolf. Score-P - A Joint Performance Measurement Run-Time Infrastructure for Periscope, Scalasca, TAU, and Vampir. *5th Parallel Tools Workshop*, 2011.

[215] A. Knüpfer, H. Brunst, J. Doleschal, M. Jurenz, M. Lieber, H. Mickler, M. S. Müller, and W. E. Nagel. The Vampir Performance Analysis Tool-Set. In M. Resch, R. Keller, V. Himmler, B. Kramer, and A. Schulz, editors, *"Tools for High Performance Computing", Proceedings of the 2nd International Workshop on Parallel Tools for High Performance Computing*, Stuttgart, Germany, July 2008. Springer-Verlag.

[216] A. Kolawa and D. Huizinga. *Automated Defect Prevention: Best Practices in Software Management*. Wiley-IEEE Computer Society Press, 2007.

[217] K. Oleson, D. Lawrence, B. Gordon, M. Flanner, E. Kluzek, J. Peter, S. Levis, S. Swenson, P. Thornton, and J. Feddema. Technical description of version 4.0 of the community land model (clm). 2010.

[218] M. Pezze and M. Young. *Software Testing and Analysis: Process, Principles and Techniques*. Wiley, 2007.

[219] D. Wang, Y. Xu, P. Thornton, A. King, C. Steed, L. Gu, and J. Schuchart. A functional test platform for the community land model. *Environ. Model. Softw.*, 55(C):25–31, May 2014.

[220] Z. Yao, Y. Jia, D. Wang, C. Steed, and S. Atchley. In situ data infrastructure for scientific unit testing platform. *Proceeding Computer Science*, 80:587–598, Dec. 31, 2016.

[221] U. Kanewala and J. M. Bieman, "Testing scientific software: A systematic literature review," *Information and Software Technology*, vol. 56, no. 10, pp. 1219–1232, 2014. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0950584914001232`

[222] T. Clune and R. Rood, "Software testing and verification in climate model development," *Software, IEEE*, vol. 28, no. 6, pp. 49–55, Nov.–Dec. 2011.

[223] P. Dubois, "Testing scientific programs," *Computing in Science Engineering*, vol. 14, no. 4, pp. 69–73, Jul.–Aug. 2012.

[224] W. Wood and W. Kleb, "Exploring XP for scientific research," *Software, IEEE*, vol. 20, no. 3, pp. 30–36, May–June.

[225] G. Miller, "A scientist's nightmare: Software problem leads to five retractions," *Science*, vol. 314, no. 5807, pp. 1856–1857, 2006. [Online]. Available: `http://www.sciencemag.org/content/314/5807/1856.short`

[226] E. J. Weyuker, "On testing non-testable programs," *The Computer Journal*, vol. 25, no. 4, pp. 465–470, 1982. [Online]. Available: `http://comjnl.oxfordjournals.org/content/25/4/465.abstract`

[227] U. Kanewala and J. Bieman, "Using machine learning techniques to detect metamorphic relations for programs without test oracles," in *Proc. 24th IEEE International Symposium on Software Reliability Engineering (ISSRE)*, Pasadena, California, USA, Nov. 2013, pp. 1–10.

[228] A. Bertolino, "Software testing research and practice," in *Abstract State Machines 2003*, ser. Lecture Notes in Computer Science, E. Börger, A. Gargantini, and E. Riccobene, Eds. Springer Berlin Heidelberg, 2003, vol. 2589, pp. 1–21. [Online]. Available: http://dx.doi.org/10.1007/3-540-36498-6_1

[229] S. M. Easterbrook, "Climate change: a grand software challenge," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, ser. FoSER '10. New York, NY, USA: ACM, 2010, pp. 99–104. [Online]. Available: http://doi.acm.org/10.1145/1882362.1882383

[230] C. Murphy, M. S. Raunak, A. King, S. Chen, C. Imbriano, G. Kaiser, I. Lee, O. Sokolsky, L. Clarke, and L. Osterweil, "On effective testing of health care simulation software," in *Proc. 3rd Workshop on Software Engineering in Health Care*, ser. SEHC '11. New York, NY, USA: ACM, 2011, pp. 40–47.

[231] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post, "Software development environments for scientific and engineering software: A series of case studies," in *Proceedings of the 29th International Conference on Software Engineering*, ser. ICSE '07. Washington, DC, USA: IEEE Computer Society, 2007, pp. 550–559. [Online]. Available: http://dx.doi.org/10.1109/ICSE.2007.77

[232] D. Kelly, S. Smith, and N. Meng, "Software engineering for scientists," *Computing in Science Engineering*, vol. 13, no. 5, pp. 7–11, Sep.–Oct. 2011.

[233] M. T. Sletholt, J. Hannay, D. Pfahl, H. C. Benestad, and H. P. Langtangen, "A literature review of agile practices and their effects in scientific software development," in *Proceedings of the 4th International Workshop on Software Engineering for Computational Science and Engineering*, ser. SECSE '11. New York, NY, USA: ACM, 2011, pp. 1–9. [Online]. Available: http://doi.acm.org/10.1145/1985782.1985784

[234] R. Sanders and D. Kelly, "The challenge of testing scientific software," in *Proc. of the Conference for the Association for Software Testing (CAST)*, Toronto, July 2008, pp. 30–36.

[235] C. Murphy, G. Kaiser, and M. Arias, "An approach to software testing of machine learning applications," in *Proc of the 19th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Boston, MA, USA, Jul. 2007, pp. 167–172.

[236] T. Y. Chen, J. W. K. Ho, H. Liu, and X. Xie, "An innovative approach for testing bioinformatics programs using metamorphic testing." *BMC Bioinformatics*, vol. 10, pp. 24–36, 2009.

[237] D. Kelly and R. Sanders, "Assessing the quality of scientific software," in *Proc of the First International Workshop on Software Engineering for Computational Science and Engineering*, 2008.

[238] J. Pitt–Francis, M. O. Bernabeu, J. Cooper, A. Garny, L. Momtahan, J. Osborne, P. Pathmanathan, B. Rodriguez, J. P. Whiteley, and D. J. Gavaghan, "Chaste: Using agile programming techniques to develop computational biology software," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 366, no. 1878, pp. 3111–3136, 2008. [Online]. Available: http://rsta.royalsocietypublishing.org/content/366/1878/3111.abstract

[239] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?" in *Proceedings of the 2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, ser. SECSE '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 1–8. [Online]. Available: http://dx.doi.org/10.1109/SECSE.2009.5069155

[240] J. Segal, "Scientists and software engineers: A tale of two cultures," in *PPIG 2008: Proceedings of the 20th Annual Meeting of the Pschology of Programming Interest Group*, J. Buckley, J. Rooksby, and R. Bednarik, Eds. Lancaster, UK: Lancaster University, 2008, proceedings: 20th annual meeting of the Psychology of Programming Interest Group; Lancaster, United Kingdom; September 10–12 2008. [Online]. Available: http://oro.open.ac.uk/17671/

[241] A. J. Abackerli, P. H. Pereira, and N. Calônego Jr., "A case study on testing CMM uncertainty simulation software (VCMM)," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 32, pp. 8–14, Mar. 2010.

[242] L. Nguyen–Hoan, S. Flint, and R. Sankaranarayana, "A survey of scientific software development," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '10. New York, NY, USA: ACM, 2010, pp. 12:1–12:10. [Online]. Available: http://doi.acm.org/10.1145/1852786.1852802

[243] M. A. Heroux, J. M. Willenbring, and M. N. Phenow, "Improving the development process for CSE software," in *Parallel, Distributed and Network-Based Processing, 2007. PDP '07. 15th EUROMICRO International Conference on*, Feb. 2007, pp. 11–17.

[244] D. Kelly, R. Gray, and Y. Shao, "Examining random and designed tests to detect code mistakes in scientific software," *Journal of Computational Science*, vol. 2, no. 1, pp. 47–56, 2011. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S187775031000075X

[245] D. Kelly, S. Thorsteinson, and D. Hook, "Scientific software testing: Analysis with four dimensions," *Software, IEEE*, vol. 28, no. 3, pp. 84–90, May–Jun. 2011.

[246] P. E. Farrell, M. D. Piggott, G. J. Gorman, D. A. Ham, C. R. Wilson, and T. M. Bond, "Automated continuous verification for numerical simulation," *Geoscientific Model Development*, vol. 4, no. 2, pp. 435–449, 2011. [Online]. Available: http://www.geosci-model-dev.net/4/435/2011/

[247] S. M. Easterbrook and T. C. Johns, "Engineering the software for understanding climate change," *Computing in Science Engineering*, vol. 11, no. 6, pp. 65–74, Nov.–Dec. 2009.

[248] D. E. Post and R. P. Kendall, "Software project management and quality engineering practices for complex, coupled multiphysics, massively parallel computational simulations: Lessons learned from ASCI," vol. 18, no. 4, pp. 399–416, Winter 2004. [Online]. Available: http://hpc.sagepub.com/content/18/4/399.full.pdf+html

[249] M. D. Davis and E. J. Weyuker, "Pseudo-oracles for non-testable programs," in *Proceedings of the ACM '81 Conference*, ser. ACM '81. New York, NY, USA: ACM, 1981, pp. 254–257. [Online]. Available: http://doi.acm.org/10.1145/800175.809889

[250] L. Hatton, "The T experiments: Errors in scientific software," *IEEE Computational Science Engineering*, vol. 4, no. 2, pp. 27–38, Apr.– Jun. 1997.

[251] T. Chen, J. Feng, and T. H. Tse, "Metamorphic testing of programs on partial differential equations: A case study," in *Computer Software and Applications Conference, 2002. COMPSAC 2002. Proceedings. 26th Annual International*, pp. 327–333.

[252] S. Brilliant, J. Knight, and N. Leveson, "Analysis of faults in an n-version software experiment," *Software Engineering, IEEE Transactions on*, vol. 16, no. 2, pp. 238–247, 1990.

[253] P. C. Lane and F. Gobet, "A theory-driven testing methodology for developing scientific software," *Journal of Experimental & Theoretical Artificial Intelligence*, vol. 24, no. 4, pp. 421–456, 2012. [Online]. Available: http://www.tandfonline.com/doi/abs/10.1080/0952813X.2012.695443

[254] R. Sanders and D. Kelly, "Dealing with risk in scientific software development," *IEEE Software*, vol. 25, no. 4, pp. 21–28, Jul.–Aug. 2008.

[255] D. Hook and D. Kelly, "Testing for trustworthiness in scientific software," in *Software Engineering for Computational Science and Engineering, 2009. SECSE '09. ICSE Workshop on*, May 2009, pp. 59–64.

[256] L. Hochstein and V. Basili, "The ASC-Alliance projects: A case study of large-scale parallel scientific code development," *Computer*, vol. 41, no. 3, pp. 50–58, March.

[257] J. Mayer, A. A. Informationsverarbeitung, and U. Ulm, "On testing image processing applications with statistical methods," in *Software Engineering (SE 2005), Lecture Notes in Informatics*, 2005, pp. 69–78.

[258] M. Cox and P. Harris, "Design and use of reference data sets for testing scientific software," *Analytica Chimica Acta*, vol. 380, no. 2–3, pp. 339–351, 1999. [Online]. Available: `http://www.sciencedirect.com/science/article/pii/S0003267098004814`

[259] Z. Q. Zhou, D. H. Huang, T. H. Tse, Z. Yang, H. Huang, and T. Y. Chen, "Metamorphic testing and its applications," in *Proc. 8th International Symposium on Future Software Technology (ISFST 2004)*. Xian, China: Software Engineers Association, 2004, pp. 346–351.

[260] X. Xie, J. W. Ho, C. Murphy, G. Kaiser, B. Xu, and T. Y. Chen, "Testing and validating machine learning classifiers by metamorphic testing," *Journal of Systems and Software*, vol. 84, no. 4, pp. 544–558, 2011.

[261] S. Yoo, "Metamorphic testing of stochastic optimisation," in *Proc. Third International Conference on Software Testing, Verification, and Validation Workshops (ICSTW)*, Apr. 2010, pp. 192–201.

[262] T. Chen, J. Feng, and T. Tse, "Metamorphic testing of programs on partial differential equations: A case study," in *Proc. 26th International Computer Software and Applications Conference on Prolonging Software Life: Development and Redevelopment*, ser. COMPSAC '02. Washington, DC, USA: IEEE Computer Society, 2002, pp. 327–333.

[263] T. Y. Chen, T. H. Tse, and Z. Q. Zhou, "Fault-based testing without the need of oracles," *Information and Software Technology*, vol. 45, no. 1, pp. 1–9, 2003.

[264] T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," Department of Computer Science, Hong Kong University of Science and Technology, Hong Kong, Tech. Rep. HKUST-CS98-01, 1998.

[265] C. Murphy, G. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in *Proc of the 20th International Conference on Software Engineering and Knowledge Engineering (SEKE)*, Redwood City, CA, USA, Jul. 2008, pp. 867–872.

[266] J. Mayer and R. Guderlei, "On random testing of image processing applications," in *Quality Software, 2006. QSIC 2006. Sixth International Conference on*, Oct. 2006, pp. 85–92.

[267] C. Murphy, K. Shen, and G. Kaiser, "Using JML runtime assertion checking to automate metamorphic testing in applications without test oracles," in *Proc. 2009 International Conference on Software Testing Verification and Validation*, ser. ICST '09. Washington, DC, USA: IEEE Computer Society, 2009, pp. 436–445.

[268] R. Guderlei and J. Mayer, "Statistical metamorphic testing – testing programs with random output by means of statistical hypothesis tests and metamorphic testing," in *Proc. 7th International Conference on Quality Software (QSIC)*, Portland, Oregon, USA, Oct. 2007, pp. 404–409.

[269] U. Kanewala, J. M. Bieman, and A. Ben-Hur, "Predicting metamorphic relations for testing scientific software: A machine learning approach using graph kernels," *Software Testing, Verification and Reliability*, 2015, in press.

[270] F. E. Allen, "Control flow analysis," *SIGPLAN Not.*, vol. 5, no. 7, pp. 1–19, Jul. 1970. [Online]. Available: `http://doi.acm.org/10.1145/390013.808479`

[271] R. Vallee–Rai and L. J. Hendren, "Jimple: Simplifying Java bytecode for analyses and transformations," 1998.

[272] R. I. Kondor and J. Lafferty, "Diffusion kernels on graphs and other discrete structures," in *Proc. 19th International Conf. on Machine Learning*, 2002, pp. 315–322.

[273] T. Gärtner, P. Flach, and S. Wrobel, "On graph kernels: Hardness results and efficient alternatives," in *Learning Theory and Kernel Machines*, ser. Lecture Notes in Computer Science, B. Schölkopf and M. Warmuth, Eds. Springer Berlin Heidelberg, 2003, vol. 2777, pp. 129–143.

[274] J. Huang and C. Ling, "Using AUC and accuracy in evaluating learning algorithms," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 3, pp. 299–310, Mar. 2005.

[275] J. H. Andrews, L. C. Briand, and Y. Labiche, "Is mutation an appropriate tool for testing experiments?" in *Proceedings of the 27th International Conference on Software Engineering*, ser. ICSE '05. New York, NY, USA: ACM, 2005, pp. 402–411.

[276] Y. Jia and M. Harman, "An analysis and survey of the development of mutation testing," *IEEE Transactions on Software Engineering*, vol. 37, pp. 649–678, 2011.

[277] Y.-S. Ma and J. Offutt, "Description of method-level mutation operators for java," November 2005. [Online]. Available: `http://cs.gmu.edu/~offutt/mujava/mutopsMethod.pdf`

[278] David Abrahams and Aleksey Gurtovoy. *C++ Template Metaprogramming: Concepts, Tools, and Techniques from Boost and Beyond*. Addison Wesley, 2004.

[279] Kaitlin Alexander and Stephen M. Easterbrook. The software architecture of climate models: A graphical comparison of CMIP5 and EMICAR5 configurations. *Geoscientific Model Development Discussions*, 8(1):351–379, 2015.

[280] Ansible Incorporated. Ansible documentation. `http://docs.ansible.com`, 2015.

[281] Apple Incorporated. The Swift programming language – language reference. `https://developer.apple.com/library/ios/documentation/Swift/Conceptual/Swift_Programming_Language/AboutTheLanguageReference.html`, 2015.

[282] Simonetta Balsamo, Antinisca Di Marco, Paola Inverardi, and Marta Simeoni. Model-based performance prediction in software development: A survey. *Software Engineering*, 30(5):295–310, 2004.

[283] Victor R. Basili, Daniela Cruzes, Jeffrey C. Carver, Lorin M. Hochstein, Jeffrey K. Hollingsworth, Marvin V. Zelkowitz, and Forrest Shull. Understanding the high-performance-computing community: A software engineer's perspective. *IEEE Software*, 25(4):29–36, 2008.

[284] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-driven software engineering in practice*. Number 1 in Synthesis Lectures on Software Engineering. Morgan & Claypool, 2012.

[285] Susanne Brenner and L. Ridgway Scott. *The Mathematical Theory of Finite Element Methods*. Springer, 3 edition, 2008.

[286] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Peter Sommerlad, and Michael Stal. Pattern-oriented software architecture, volume 1: A system of patterns, 1996.

[287] Fabien Campagne. *The MPS Language Workbench: Volume I*. Fabien Campagne, 2014.

[288] Jeffrey C. Carver, Richard P. Kendall, Susan E. Squires, and Douglass E. Post. Software development environments for scientific and engineering software: A series of case studies. In *Proceedings of the 29th International Conference on Software Engineering (ICSE 2007)*, pages 550–559. IEEE, 2007.

[289] Leonardo Dagum and Ramesh Menon. OpenMP: An industry standard API for shared-memory programming. *Computational Science & Engineering*, 5(1):46–55, 1998.

[290] Stephen M. Easterbrook and Timothy C. Johns. Engineering the software for understanding climate change. *Computing in Science & Engineering*, 11(6):65–74, 2009.

[291] Sven Efftinge, Moritz Eysholdt, Jan Köhnlein, Sebastian Zarnekow, Robert von Massow, Wilhelm Hasselbring, and Michael Hanus. Xbase: Implementing domain-specific languages for Java. In *Proceedings of the 11th International Conference on Generative Programming and Component Engineering*, pages 112–121. ACM, 2012.

[292] Moritz Eysholdt and Heiko Behrens. Xtext: Implement your language faster than the quick and dirty way. In *Proceedings of the ACM International Conference Companion on Object-Oriented Programming Systems Languages and Applications Companion*, pages 307–309. ACM, 2010.

[293] Stuart Faulk, Eugene Loh, Michael L. Van De Vanter, Susan Squires, and Lawrence G. Votta. Scientific computing's productivity gridlock: How software engineering can help. *Computing in Science & Engineering*, 11:30–39, 2009.

[294] Martin Fowler. *Domain-Specific Languages.* Addison-Wesley, 2010.

[295] Volker Grimm and Steven F. Railsback. *Individual-based Modeling and Ecology.* Princeton University Press, 2005.

[296] Jo ErsKine Hannay, Hans Petter Langtangen, Carolyn MacLeod, Dietmar Pfahl, Janice Singer, and Greg Wilson. How do scientists develop and use scientific software? In *Software Engineering for Computational Science and Engineering, 2009. SECSE'09. ICSE Workshop on*, pages 1–8. IEEE, 2009.

[297] Dustin Heaton and Jeffrey C. Carver. Claims about the use of software engineering practices in science: A systematic literature review. *Information and Software Technology*, 67:207–219, 2015.

[298] Siw Elisabeth Hove and Bente Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS 2005)*, pages 1–10. IEEE, 2005.

[299] Arne N. Johanson and Wilhelm Hasselbring. Hierarchical combination of internal and external domain-specific languages for scientific computing. In *Proceedings of the 2014 European Conference on Software Architecture Workshops*, ECSAW'14, pages 17:1–17:8. ACM, 2014.

[300] Arne N. Johanson and Wilhelm Hasselbring. Sprat: Hierarchies of domain-specific languages for marine ecosystem simulation engineering. In *Proceedings TMS SpringSim'14*, pages 187–192. SCS, 2014.

[301] Diane Kelly. A software chasm: Software engineering and scientific computing. *IEEE Software*, 24(6):118–120, 2007.

[302] Sarah Killcoyne and John Boyle. Managing chaos: Lessons learned developing software in the life sciences. *Computing in Science & Engineering*, 11(6):20–29, 2009.

[303] Anneke Kleppe. *Software Language Engineering: Creating Domain-Specific Languages Using Metamodels*. Addison-Wesley, 2008.

[304] Philipp Mayring. *Qualitative Inhaltsanalyse: Grundlagen und Techniken*. Beltz, 12 edition, 2015.

[305] Marjan Mernik, Jan Heering, and Anthony M. Sloane. When and how to develop domain-specific languages. *ACM Computing Surveys (CSUR)*, 37(4):316–344, 2005.

[306] Paul Messina. Gaining the broad expertise needed for high-end computational science and engineering research. *Computing in Science & Engineering*, 17(2):89–90, 2015.

[307] Christian Motika, Steven Smyth, and Reinhard von Hanxleden. Compiling SCCharts – a case-study on interactive model-based compilation. In *Leveraging Applications of Formal Methods, Verification and Validation. Technologies for Mastering Change*, pages 461–480. Springer, 2014.

[308] Prakash Prabhu, Thomas B. Jablin, Arun Raman, Yun Zhang, Jialu Huang, Hanjun Kim, Nick P. Johnson, Feng Liu, Soumyadeep Ghosh, Stephen Beard, Taewook Oh, Matthew Zoufaly, David Walker, and David I. August. A survey of the practice of computational science. In *State of the Practice Reports*, SC'11, pages 19:1–19:12. ACM, 2011.

[309] Doraiswami Ramkrishna. *Population Balances: Theory and Applications to Particulate Systems in Engineering*. Academic Press, 2000.

[310] Rebecca Sanders and Diane F. Kelly. Dealing with risk in scientific software development. *Software, IEEE*, 25(4):21–28, 2008.

[311] Ina Schieferdecker. Model-based testing. *IEEE Software*, 29(1):14–18, 2012.

[312] Erik Schnetter, Marek Blazewicz, Steven R. Brandt, David M. Koppelman, and Frank Löffler. Chemora: A PDE-solving framework for modern high-performance computing architectures. *Computing in Science & Engineering*, 17(2):53–64, 2015.

[313] Judith Segal. Models of scientific software development. In *Proceedings of the First International Workshop on Software Engineering for Computational Science and Engineering, SECSE'08*, pages 1–7, 2008.

[314] Judith Segal and Chris Morris. Developing scientific software. *Software, IEEE*, 25(4):18–20, 2008.

[315] Thomas Stahl and Markus Völter. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley, 2006.

[316] Mark Strembeck and Uwe Zdun. An approach for the systematic development of domain-specific languages. *Software: Practice and Experience*, 39(15):1253–1292, 2009.

[317] Gregory V. Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94(1):5–6, 2006.

[318] Gregory V. Wilson. Software carpentry: Lessons learned. *F1000Research*, 3:1–11, 2014.

[319] Satish Balay, Shrirang Abhyankar, Mark F. Adams, Jed Brown, Peter Brune, Kris Buschelman, Lisandro Dalcin, Victor Eijkhout, William D. Gropp, Dinesh Kaushik, Matthew G. Knepley, Lois Curfman McInnes, Karl Rupp, Barry F. Smith, Stefano Zampini, and Hong Zhang. PETSc Users Manual. Technical Report ANL-95/11 - Revision 3.6, Argonne National Laboratory, 2015.

[320] David M. Beazley. SWIG: An easy to use tool for integrating scripting languages with C and C++. In *Proceedings of the 4th USENIX Tcl/Tk Workshop*, pages 129–139, 1996.

[321] Thomas G. W. Epperly, Gary Kumfert, Tamara Dahlgren, Dietmar Ebner, Jim Leek, Adrian Prantl, and Scott Kohn. High-performance language interoperability for scientific computing through Babel. *International Journal of High Performance Computing Applications*, page 1094342011414036, 2011.

[322] William D. Gropp. Users manual for bfort: Producing Fortran interfaces to C source code. Technical Report ANL/MCS-TM-208, Argonne National Laboratory, IL (United States), 1995.

[323] William D. Gropp and Barry F. Smith. Simplified linear equation solvers users manual. Technical Report ANL–93/8, Argonne National Laboratory, IL (United States), 1993.

[324] William D. Gropp and Barry F. Smith. Scalable, extensible, and portable numerical libraries. In *Proceedings of the Scalable Parallel Libraries Conference*, pages 87–93, Mississippi State University, 1994. IEEE.

[325] Michael Metcalf. The seven ages of fortran. *Journal of Computer Science & Technology*, 11, 2011.

[326] Object Management Group. Common Object Request Broker Architecture (CORBA). http://www.corba.org, 2007.

[327] DevOps. https://en.wikipedia.org/wiki/DevOps.

[328] Django. https://www.djangoproject.com/.

[329] Docker. https://www.docker.com/.

[330] Guide to the Software Engineering Body of Knowledge (SWEBOK), V3. http://www.computer.org/web/swebok/v3-guide/.

[331] HipChat. https://www.hipchat.com/.

[332] HydroShare. http://www.hydroshare.org/.

[333] Implementation of NSF CIF21 Software Vision (SW-Vision). http://www.nsf.gov/si2/.

[334] iRODS. http://irods.org/.

[335] Jenkins. https://jenkins.io/.

[336] NSF collaborative HydroShare award numbers 1148453 and 1148090. http://www.nsf.gov/awardsearch/showAward?AWD_ID=1148453 and http://www.nsf.gov/awardsearch/showAward?AWD_ID=1148090.

[337] Test-Driven Development (TDD). https://www.agilealliance.org/glossary/tdd.

[338] National Flood Interoperability Experiment. http://www.caee.utexas.edu/prof/maidment/giswr2014/Synopsis/GISWRSynopsis12.pdf, 2014.

[339] Graduate students take part in National Flood Interoperability Experiment Summer Institute to develop new tools. http://www.tuscaloosanews.com/article/20150607/NEWS/150609776?p=2&tc=pg/, 2015.

[340] S. Ahalt, L. Band, L. Christopherson, R. Idaszak, C. Lenhardt, B. Minsker, M. Palmer, M. Shelley, M. Tiemann, and A. Zimmerman. Water Science Software Institute: Agile and open source scientific software development. *IEEE Computing in Science and Engineering (CiSE)*, 6(3):18–26, 2014.

[341] R. Bartlett, M. Heroux, and W. Willenbring. Overview of the TriBITS Lifecycle Model : A Lean/Agile Software Lifecycle Model for Research-based Computational Science and Engineering Software. In *Proceedings of the First Workshop on Maintainable Software Practices in e-Science*. Part of the IEEE International Conference on eScience, 2012.

[342] V. R. Basili, J. C. Carver, D. Cruzes, L. M. Hochstein, J. K. Hollingsworth, F. Shull, and M. V. Zelkowitz. Understanding the high-performance computing community: A software engineer's perspective. In *Software*, pages 29–36. IEEE, 2008.

[343] F. P. Brooks. *The Mythical Man-Month: Essays on Software Engineering, Anniversary Edition (2nd Edition).* Addison-Wesley Professional, 1995.

[344] Ibid Brooks. p. 116.

[345] J. C. Carver, R. P. Kendall, S. E. Squires, and D. E. Post. Software development environments for scientific and engineering software: A series of case studies. In *29th International Conference on Software Engineering (ICSE)*, pages 550–559. Minneapolis, MN: IEEE, 2007.

[346] Engineering Committee on Science, National Academy of Engineering Public Policy, National Academy of Science, and Institute of Medicine of the National Academies. *On Being a Scientist: A Guide to Responsible Conduct in Research*. Washington, DC: National Academies Press, 2009.

[347] A. Couch, D. G. Tarboton, R. Idaszak, J. S. Horsburgh, H. Yi, and M. J. Stealey. A Flexible File Sharing Mechanism for iRODS. In *iRODS User Group Meeting 2015 Proceedings*, pages 61–68, 2015.

[348] R. L. Glass. Frequently Forgotten Fundamental Facts about Software Engineering. *IEEE Software*, page 110, 2001.

[349] J. E. Hannay, C. MacLeod, and J. Singer. How do scientists develop and use scientific software? In *ICSE Workshop on Software Engineering for Computational Science and Engineering*, pages 1–8. Vancouver, Canada: IEEE, 2009.

[350] J. Heard, D. G. Tarboton, R. Idaszak, J.S. Horsburgh, D. Ames, A. Bedig, A. M. Castronova, A. Couch, P. Dash, C. Frisby, T. Gan, J. Goodall, S. Jackson, S. Livingston, D. Maidment, N. Martin, B. Miles, S. Mills, J. Sadler, D. Valentine, and L. Zhao. An Architectural Overview of Hydroshare, A Next-Generation Hydrologic Information System. In *11th International Conference on Hydroinformatics, HIC 2014*. CUNY Academic Works, 2014.

[351] J. S. Horsburgh, M. M. Morsy, A. M. Castronova, J. L. Goodall, T. Gan, H. Yi, M. J. Stealey, and D. G. Tarboton. Hydroshare: Sharing Diverse Environmental Data Types and Models as Social Objects with Application to the Hydrology Domain. *Journal of the American Water Resources Association (JAWRA)*, pages 1–17, 2015.

[352] J. Howison and J. D. Herbsleb. Scientific software production: Incentives and collaboration. In *Proceedings of the ACM 2011 Conference on Computer Supported Cooperative Work*, pages 513–522. Hangzhou, China: ACM, 2011.

[353] J. M. Sadler, D. P. Ames, and S. J. Livingston. Extending HydroShare to enable hydrologic time series data as social media. *Journal of Hydroinformatics*, 18(2):198–209, 2015.

[354] R. Sanders and D. Kelly. Dealing with risk in scientific software development. In *Software*, pages 21–28. IEEE, 2008.

[355] D. G. Tarboton, J. S. Horsburgh, D. Maidment, T. Whiteaker, I. Zaslavsky, M. Piasecki, J. L. Goodall, D. Valentine, and T. Whitenack. Development of a community hydrologic information system. In *18th World IMACS Congress and MODSIM09 International Congress on Modelling and Simulation*, pages 988–994. R. S. Anderssen, R. D. Braddock, and L. T. Newham (Ed.), 18th Modelling and Simulation Society of Australia and New Zealand and International Association for Mathematics and Computers in Simulation, 2009.

[356] B. Turhan, L. Layman, M. Diep, H. Erdogmus, and F. Shull. How Effective Is Test-Driven Development? In *Making Software: What Really Works and Why We Believe It*, pages 207–217. O'Reilly, 2010.

[357] R. Van Noorden. Science publishing: The trouble with retractions. *Nature*, 478:26–28, 2011.

[358] L. Williams, M. Maximilien, and M. Vouk. Test-Driven Development as a Defect-Reduction Practice. In *Proceedings of the 14th International Symposium on Software Reliability Engineering*, page 34, 2003.

[359] G. V. Wilson. Where's the real bottleneck in scientific computing? *American Scientist*, 94(1):5–6, 2006.