Contextual Understanding and Improvement of Metamorphic Testing in Scientific Software Development

Zedong Peng
University of Cincinnati
Cincinnati, OH, USA
pengzd@mail.uc.edu

Upulee Kanewala
University of North Florida
Jacksonville, FL, USA
upulee.kanewala@unf.edu

Nan Niu
University of Cincinnati
Cincinnati, OH, USA
nan.niu@uc.edu

ABSTRACT

Background: Metamorphic testing emerges as a simple and effective approach for testing scientific software; yet, its adoption in actual scientific software projects is less studied.
Aims: In order for the practitioners to better adopt metamorphic testing in their projects, we set out to first gain a deep understanding about the current qualify assurance workflow, testing practices, and tools.
Method: We propose to integrate various empirical sources, including artifact analysis, stakeholder interviews, and gap analysis from the literature.
Results: Applying our approach to the Open Water Analytics Stormwater Management Model project helped to identify four new needs requiring continued and more research: (1) systematic and explicit formulation of metamorphic relations, (2) metamorphic testing examples specific to the scientific software, (3) correlating metamorphic testing with regression testing, and (4) integrating metamorphic testing with build tools like CMake and continuous integration tools like GitHub Actions.
Conclusions: Integrating different empirical sources is promising for establishing a contextual understanding of software engineering practices, and for action research, such as workflow refinements and tool interventions, to be carried out in a principled manner.

CCS CONCEPTS

• **Software and its engineering** → *Software organization and properties*; **Software testing and debugging**; *Empirical software validation.*

KEYWORDS

metamorphic testing, scientific software, empirical software engineering

ACM Reference Format:
Zedong Peng, Upulee Kanewala, and Nan Niu. 2021. Contextual Understanding and Improvement of Metamorphic Testing in Scientific Software Development. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '21), October 11–15, 2021, Bari, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3475716.3484188

1 INTRODUCTION

Scientific software is an essential part in modern scientific discovery and many scientists today spend a considerable amount of their time writing and managing code. Kreyman *et al.* [15] defined scientific software as the software with a large computational component providing data for decision support. Kanewala and Bieman [10] adopted a broad view and referred to scientific software as the software used for scientific purposes.

High-quality scientific software can make the difference between valid, sustainable, reproducible research outputs and short-lived, potentially unreliable or erroneous outputs. There are cases that scientists retracted published work because of software faults [20]. There are also cases where scientists believed they needed to modify the physics model, but later found that the real problems were subtle faults in the code [7].

Software testing, a mainstream approach toward assuring software quality, is therefore important for developing scientific software. However, due to some characteristics of scientific software such as the complexity of the underlying scientific problem and the lack of knowledge among the scientists on software testing, applying testing in this domain is challenging. To understand scientific software testing, researchers have applied different empirical methods: reviewing the literature [10], surveying and interviewing practitioners [27, 33], and analyzing testing-related artifacts in software repositories [25, 26].

To integrate the various empirical sources, we present an approach by concentrating on a specific technique, namely *metamorphic testing* (MT) [12]. Our goal is to understand the contexts in which MT will be adopted, and to use the understanding to better support the adoption. To that end, we show our long-term plan in Figure 1 where five main research activities are depicted. In this paper, we report some initial results of the first four activities of Figure 1, in order to establish a contextual understanding of the workflow, practices, and tools relevant to MT adoption in the Open Water Analytics Stormwater Management Model (OWA SWMM) project [24].

The main contributions of this paper lie in our approach integrating the empirical sources of software repositories, stakeholders, and the research literature. In addition, the emerging results on the MT gaps can inform the researchers and tool builders to better address

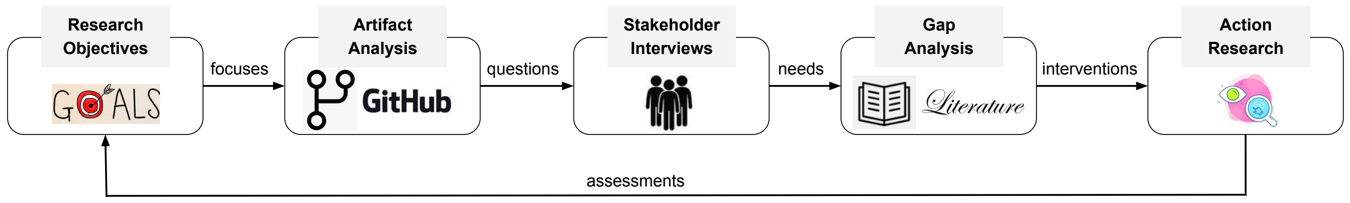

Figure 1: Integrating various empirical sources to build a contextual understanding via artifact analysis from software repositories, stakeholder interviews, and gap analysis from literature reviews. Such an understanding could assist in designing better interventions to improve the focused software engineering practices.

the adoption barriers. In what follows, we provide background information of SWMM and MT in Section 2. Section 3 discusses our research method, Section 4 presents the results, and finally, Section 5 concludes the paper.

2 BACKGROUND

2.1 Stormwater Management Model

Stormwater Management Model (SWMM) maintained by the Open Water Analytics (OWA) group [24] is a dynamic hydrology-hydraulic water quality simulation model, used for single event or long-term (continuous) simulation of runoff quantity and quality from primarily urban areas. The OWA SWMM repository is forked from the official SWMM source code repository maintained by the U.S. Environmental Protection Agency (EPA) Office of Research and Development [32].

SWMM source code is written in the C/C++ programming language and released in the public domain. In particular, all the OWA SWMM project materials created by the authors are released under the MIT License [24], allowing us as researchers to study the artifacts and publish our study results.

2.2 Metamorphic Testing (MT) for Scientific Software

Software testing is a systematic approach where the program under test is executed to verify that the produced output is correct according to the expected behavior of the program. However, the following quote from an R developer [33] reflects a major challenge in scientific software testing: "*It's frequently difficult to test scientific software, since you might not even know in advance what the answer should be.*" Given an input, not knowing the expected output of the software under test is called the *oracle problem* [2]. As in most scientific software systems, the oracle problem also exists in SWMM due to reasons like the unavailability of historical runoff measures and the modeling and computational approximations [9].

MT is an emerging technique for alleviating the oracle problem, and has shown to be effective for testing scientific software [12]. Rather than focusing on the correctness of output from a *single* execution of the software under test, MT checks whether a relation (called a "metamorphic relation") holds among *multiple* executions. For instance, "the simulated surface water runoff is expected to decrease when bioretention cells are added to an area of study" [18] is a sample metamorphic relation for SWMM, allowing MT to be performed by checking if the input change of adding bioretention cells indeed leads to the output change of decreased runoff. In this sense, the metamorphic relation effectively becomes a test oracle, eliminating the need for knowing the expected output of any individual SWMM simulation.

Despite the successful application of MT to a few simulation, modeling, and numerical programs [29], there exist plenty of opportunities to further advance MT. In particular, scientific software developers, similar to end-user programmers [3], normally have no formal training in software testing or software engineering [33]. This poses difficulty for them to fully understand the limitations and technical issues of MT; therefore, the availability of relevant automated testing tools is important [3]. Another opportunity is to apply MT to test software released in every iteration of agile development [3], which requires a deep understanding of the software testing practices between the iterations. Gaining such an understanding in the context of SWMM development is precisely the focus of our work.

3 METHOD

Our idea, depicted in Figure 1, is to integrate different empirical sources to achieve a contextual understanding for some software engineering practices in scientific software development. In this paper, our *research objectives* focus on SWMM developers' quality assurance (QA) workflow, software testing practices, and related tools, because our longer-term goal is to introduce MT in a way that will fit practitioners' workflow and enhance testing efficacy in scientific software development.

The focuses resulted from the first activity of Figure 1 guide our artifact analysis of SWMM's GitHub repositories. To model the current QA workflow, we paid special attention to the OWA SWMM development project [24] in our research. This project is aimed to take a community-driven approach to grow the capabilities of SWMM. We take advantage of this community-based project since its GitHub repository contains information about road-mapping, contributing, testing, etc, allowing us to construct a QA workflow, and to come up with several software testing and tooling questions related to the workflow.

These questions drive our "stakeholder interviews" step of Figure 1. By interviewing six SWMM developers, we are able to clarify their different testing practices, learn about the tools employed by them, and elicit their perceptions toward MT. Based on the interview results, we identify four requirements for adopting MT in scientific software developers' daily practices. We then survey a sampled literature to investigate the extent to which the MT

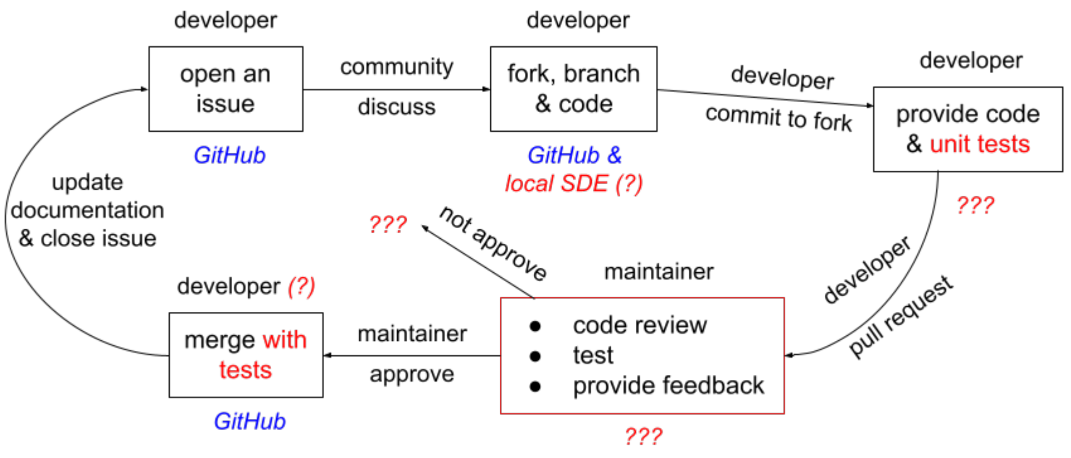

Figure 2: Quality assurance (QA) workflow extracted from the OWA SWMM's GitHub repository [24]. Boxes represent main activities, with the role annotated at the top and the tool annotated at the bottom. The red fonts and question marks highlight what our repository mining failed to identify. We therefore interviewed the SWMM stakeholders to find out the answers.

adoption requirements are satisfied, pointing out the gaps to be addressed by the research community.

4 RESULTS AND DISCUSSION

4.1 QA Workflow and Tools

We manually analyzed the community wiki in the OWA SWMM's GitHub repository [24], and extracted a QA workflow shown in Figure 2. The project welcomes issues (bugs or new features) raised by the community. The community has three defined roles: *users* are those who want to use SWMM as it is hosted on GitHub and get updates when they are released, *developers* are those who implement the changes to SWMM, and *maintainers* are gate keepers who are able to push to the upstream repo as well as pushing to their fork.

While the issues are discussed in GitHub, we could not identify a software development environment (SDE) recommended for SWMM developers to make code changes (and hence a red-fonted "local SDE" with question mark in Figure 2). Given that developers are asked to commit changed code and unit tests, learning about developers' local SDE helps to understand if and what MT support might be needed during coding and committing.

Developer's pull request (bottom right of Figure 2) informs others the completion of an issue, and our searching of the OWA SWMM repository identified only isolated practices of code review, testing, and providing feedback. However, we were not clear about their relationships and dependencies (if any). More specifically, it is not clear to us what kinds of testing were performed here by the maintainers and with which tools. Due to the lack of information surrounding the pull request approval, we were also not sure if and how tests were merged into a base branch.

4.2 Stakeholder Interviews and Responses

Analyzing the artifacts from the OWA SWMM's GitHub repository allowed us to build some initial understanding of the QA workflow: who does what to maintain and evolve the scientific software. As shown in Figure 2, our extracted QA workflow helps to raise several questions, which we group into two main areas to guide our stakeholder interviews: (1) local SDEs used and unit tests written by the individual SWMM developers, and (2) maintainers' testing practices to approve code contributions. In addition, we are open to receiving our interviewees' comments about what is modeled in Figure 2, correcting misunderstandings and filling in omissions.

We interviewed six active members of the SWMM community. Table 1 uses the pseudonyms to present some SWMM-related background information of the interviewees. While five interviewees are affiliated with the EPA, P6 represents a non-EPA contributor to SWMM who is a hydroinformatics engineer from industry. Half of the interviewees are developers who implement SWMM changes, and the other half are maintainers who also review, test, and provide feedback to other developers' code. All our interviewees have worked on SWMM for over two years, and are contributing to the scientific software project on a continuous and consistent basis.

Our interviews were carried out via web conferencing due to the COVID-19 pandemic. Each interview session lasted around 30 minutes. We began by introducing our research objectives (i.e., to understand SWMM's QA workflow, testing practices, and tools), and then projected Figure 2 to the shared screen with the interviewee in

Table 1: Interview Participants and Their SWMM-Related Status (the "Affiliation" and the SWMM "Role" columns show the current status, and the "Experience" column reports an estimated time period of total SWMM involvement)

Interviewee	Affiliation	Role	Experience
P1	EPA	developer	3.5 years
P2	EPA	developer	2.5 years
P3	EPA	developer	3 years
P4	EPA	maintainer	6 years
P5	EPA	maintainer	4 years
P6	industry	maintainer	3 years

Table 2: Participants' Responses to the Main Interview Questions

Questions	developer's code change		maintainer's change approval	
	local SDE	unit testing	regression testing	continuous integration (CI)
Responses	Visual Studio (P2, P4, P5, P6), MS Code (P4), XCode (P4), Eclipse (P4), Shell & text editors (P1, P3, P4, P6), CMake frees up SDEs & platforms (P4)	Boost library of C++ unit test (P1, P4, P5), CTest as part of CMake (P4), local regression testing (P1), *ad hoc* MT (P1, P6)	A set of regression tests viewed as the benchmark of the entire community (P3, P4, P5, P6)	GitHub Actions responsible for running the CI pipeline and notifying contributors of the workflow, failures, and degree of completion (P1, P3, P4, P5)

order to better contextualize our questions. We asked each SWMM stakeholder to respond to the main questions of software testing and QA during developer's code change and during maintainer's change approval. We were also flexible in eliciting comments beyond the main questions, and reserved a time slot toward the end of each interview to collect the feedback from the participant in an open-ended way.

The responses to our main interview questions are summarized in Table 2. The mostly mentioned SDE was Visual Studio, since SWMM was written in C/C++. However, using shell and text editors like notepad was another popular choice among our interviewees. An important aspect was emphasized by P4: SWMM's adoption of CMake frees up individual developers' SDEs (e.g., Visual Studio, XCode, or notepad) and platforms (e.g., Windows, Linux, and MacOS). The impact of CMake extends to unit testing in that the RUN_TESTS build target is aimed to run the tests registered with CTest (Req -DBUILD_TESTS=ON). The CMake build map from [24] illustrates the integral role that Boost plays in SWMM.

It is worth noting that a couple of the interviewees shared their experience of doing unit testing by using the core idea of MT, though the term "MT" was not mentioned by them. P1 stated that, "*In a physical system [and hence in SWMM], we should expect [that if] the infiltration rate increase, then the runoff will decrease.*" The relation of "infiltration rate increase leading to runoff decrease" is clearly a metamorphic relation that could help the developer to test the code and code changes. Although some SWMM developers already practiced MT, making the metamorphic relations more explicit and having good examples will help better convey the MT's core ideas and promote more MT practices in the context of SWMM.

A majority of our interviewees pointed out the importance of regression testing as part of maintainer's QA workflow. In particular, about 50 regression tests were written by one of the original developers of SWMM. These tests cover such core computations that not only the maintainers but the entire SWMM community rely on them to assure the software preserves certain behaviors amid changes. Quoting P6 who is a non-EPA contributor, "*(The set of) regression tests is benchmark to me; I don't think anyone should be playing with these regression tests.*" This raises an interesting question: When certain behavior is desired to change, how to safely evolve the benchmark? We believe MT can play a role in this evolution.

Our question about the tooling for handling pull requests and merges was answered by more than half of the interviewees: GitHub Actions provide SWMM maintainers various continuous integration (CI) capabilities to run a series of commands after a specified event has occurred, e.g., commits, pull requests, etc. Thus, integrating MT into CI tools like GitHub Actions (e.g., via newly defined event-driven APIs) would fit the software QA workflow. Our stakeholder interviews uncovered the following needs to facilitate scientific software developers' adoption of MT in their daily work:

- $\mathbf{N}_1$**:** Systematic and explicit formulation of metamorphic relations;
- $\mathbf{N}_2$**:** MT examples specific to the scientific software;
- $\mathbf{N}_3$**:** Correlating MT with regression testing; and
- $\mathbf{N}_4$**:** Integrating MT with build tools like CMake and CI tools like GitHub Actions.

4.3 Literature Analysis

We conducted a literature analysis in July 2021 to determine how the current work addresses the four needs listed above. To identify the relevant articles, we conducted a Google Scholar search with the keywords: "scientific software", " metamorphic testing", and "metamorphic relations." We limited the search results to articles published since 2005 as we were interested in analyzing the latest work. Furthermore, we used the following criteria to filter the returned articles: (1) must describe a technique/tool developed for scientific software, and (2) should include an evaluation conducted on scientific software. From the returned search results, we identified the 15 papers listed in Table 3 as satisfying our criteria.

The majority of the identified papers (11/15) provided examples ($\mathbf{N}_2$) of applications of MT on scientific software from various domains such as bioinformatics and mathematics. However, reusing the MRs developed in these studies when testing a similar software is not straightforward, since these MRs are not available in a location such as a publicly accessible repository. Seven out of the 15 papers provided several techniques that could assist $\mathbf{N}_1$. These techniques include using machine learning to systematically identify MRs and manual yet systematic approaches for deriving MRs. However, these techniques are not integrated with scientific software development; thus, more work needs to be done to incorporate them with day-to-day scientific software development. None of the papers that we examined address the needs $\mathbf{N}_3$ and $\mathbf{N}_4$; thus, more work needs to be done on developing approaches of utilizing MT for regression testing and integrating MT with widely used tools for building and continuous integration.

4.4 Threats to Validity

Our study involves three key constructs: QA workflow, testing practices, and tools. We relied on artifact analysis from GitHub to devise a QA workflow model, and stakeholder interviews to

Table 3: Results of Analyzing Relevant Papers (ordered reverse-chronologically, and in the same year, alphabetically by the last name of the leading author): MR – metamorphic relation, ● – addresses the need, ○ – partially addresses the need.

Paper	**Goal of the paper**	$\mathbf{N}_1$	$\mathbf{N}_2$	$\mathbf{N}_3$	$\mathbf{N}_4$
Lin *et al.* - 2020 [17]	Iterative MR development for testing scientific simulations	○	●		
Olsen & Raunak - 2019 [23]	Application of MR for testing scientific software		●		
Shahri *et al.* - 2019 [30]	Application of MR for testing scientific software		●		
Akgün *et al.* - 2018 [1]	Application of MR for testing scientific software		●		
Hardin & Kanewala - 2018 [8]	Predicting MRs for testing scientific software	○			
Lin *et al.* - 2018 [16]	Incremental development of hierarchical MRs for testing scientific software	○			
Rahaman & Kanewala - 2018 [28]	Predicting MRs for testing scientific software	○			
Srinivasan *et al.* - 2018 [31]	Application of MR for testing scientific software		●		
Ding & Hu - 2017 [5]	Iterative MR development of scientific software	○	●		
Ding *et al.* - 2016 [6]	Iterative approach for developing MRs for testing a scientific program	○	●		
Kanewala *et al* - 2016 [11]	Predicting MRs for testing scientific software	○			
Kanewala *et al.* - 2016 [13]	Application of MR for testing scientific software		●		
Lundgren & Kanewala - 2016 [19]	Application of MR for testing scientific software		●		
Olsen & Raunak - 2016 [22]	Application of MR for testing scientific software		●		
Yan *et al.* - 2016 [34]	Application of MR for testing scientific software		●		

understand their testing practices and the tools that they used. We therefore believe the threats to *construct validity* are low.

Some factors can affect the *internal validity* of our study. First, our artifact analysis for modeling the QA workflow might be incomplete. For example, a couple of interviewees pointed out that "not approved" changes did exist in the GitHub repository and developers may work on them when they are free. Nevertheless, all the interviewees agreed that the workflow of Figure 2 captured the QA process well, which prevented us from asking open-ended questions, e.g., "Can you describe the QA workflow?" or "What testing tools do you use?" In line with Cohene and Easterbrook's experience [4], the workflow model of Figure 2 took longer to create yet the responses that we received tended to be shorter and more focused. Another threat to internal validity is our manual assessment of the literature identified in the gap analysis. To minimize the threat, two researchers independently assessed how well each identified paper addressed the needs, and their disagreements were resolved in a virtual meeting.

Our results may not generalize beyond the particular scientific software that we studied, presenting some threats to *external validity*. In particular, different QA processes and testing tools might apply to scientific software projects that are not open-sourced, or not producing a self-contained simulation application. For instance, there may be different needs to adopt MT in numerical and machine learning libraries, e.g., GNU Scientific Library (GSL), scikit-learn, OpenCV (Computer Vision), etc.

5 CONCLUSIONS

Testing software in the scientific domain faces some unique challenges due to inherent complexities in scientific software and the scientists' lack of knowledge in testing. MT can overcome many of these challenges and be an effective testing technique in this domain. However, not much work is done toward understanding scientific software QA workflow and how MT fits. To this end, we conducted an artifact analysis followed by stakeholder interviews and literature analysis.

Based on our artifact analysis and stakeholder interviews, we identified the following four needs that will facilitate the adoption of MT in scientific software development: $\mathbf{N}_1$: Systematic and explicit formulation of metamorphic relations, $\mathbf{N}_2$: MT examples specific to the scientific software, $\mathbf{N}_3$: Correlating MT with regression testing, and $\mathbf{N}_4$: Integrating MT with build tools and continuous integration tools. Subsequent literature analysis revealed that $\mathbf{N}_2$ is addressed by the majority of the studies that we examined. However, the developed MT examples and the related metamorphic relations are not easily accessible to the practitioners. Several studies could assist with $\mathbf{N}_1$, but they need to be applied and evaluated in the context of scientific software development. We could not find any studies that address the needs $\mathbf{N}_3$ and $\mathbf{N}_4$.

In the future, we plan to expand this study to several other scientific software development projects to confirm the identified needs and discover new requirements. To that end, theoretical replications [14, 21] may be worth conducting. We also plan to develop the tools and resources to facilitate the MT adoption in scientific software development based on the identified needs, continuing with the step of "Action Research" shown in Figure 1.

ACKNOWLEDGMENTS

We thank Colleen Barr, Faryn Dumont, Katherine Ratliff, Michelle Simon, Michael Tryby, and Jennifer Wu for partaking in our stakeholder interviews. We also thank the authors who contributed to the open-source repositories (in particular the OWA SWMM GitHub repository) and released the software artifacts under the MIT License.

REFERENCES

[1] Özgür Akgün, Ian P. Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. 2018. Metamorphic Testing of Constraint Solvers. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming (CP'20).* Lille, France, 727–736.

[2] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (May 2015), 507–525.

[3] Tsong Yueh Chen, Fei-Ching Kuo, Huai Liu, Pak-Lok Poon, Dave Towey, T. H. Tse, and Zhi Quan Zhou. 2018. Metamorphic Testing: A Review of Challenges and Opportunities. *Comput. Surveys* 51, 1 (April 2018), 4:1–4:27.

[4] Tira Cohene and Steve Easterbrook. 2005. Contextual Risk Analysis for Interview Design. In *Proceedings of the International Requirements Engineering Conference (RE'05)*. Paris, France, 95–104.

[5] Junhua Ding and Xin-Hua Hu. 2017. Application of Metamorphic Testing Monitored by Test Adequacy in a Monte Carlo Simulation Program. *Software Quality Journal* 25, 3 (September 2017), 841–869.

[6] Junhua Ding, Dongmei Zhang, and Xin-Hua Hu. 2016. An Application of Metamorphic Testing for Testing Scientific Software. In *Proceedings of the International Workshop on Metamorphic Testing (MET'16)*. Austin, TX, USA, 37–43.

[7] Paul F. Dubois. 2012. Testing Scientific Programs. *Computing in Science and Engineering* 14, 4 (July/August 2012), 69–73.

[8] Bonnie Hardin and Upulee Kanewala. 2018. Using Semi-Supervised Learning for Predicting Metamorphic Relations. In *Proceedings of the International Workshop on Metamorphic Testing (MET'18)*. Gothenburg, Sweden, 14–17.

[9] Konrad Hinsen. 2015. The approximation tower in computational science: why testing scientific software is difficult. *Computing in Science and Engineering* 17, 4 (July/August 2015), 72–77.

[10] Upulee Kanewala and James M. Bieman. 2014. Testing Scientific Software: A Systematic Literature Review. *Information & Software Technology* 56, 10 (October 2014), 1219–1232.

[11] Upulee Kanewala, James M. Bieman, and Asa Ben-Hur. 2016. Predicting Metamorphic Relations for Testing Scientific Software: A Machine Learning Approach using Graph Kernels. *Software Testing, Verification & Reliability* 26, 3 (May 2016), 245–269.

[12] Upulee Kanewala and Tsong Yueh Chen. 2019. Metamorphic Testing: A Simple Yet Effective Approach for Testing Scientific Software. *Computing in Science and Engineering* 21, 1 (January/February 2019), 66–72.

[13] Upulee Kanewala, Anders Lundgren, and James M Bieman. 2016. Automated metamorphic testing of scientific software. In *Software Engineering for Science*. Chapman and Hall/CRC, 185–210.

[14] Charu Khatwani, Xiaoyu Jin, Nan Niu, Amy Koshoffer, Linda Newman, and Juha Savolainen. 2017. Advancing Viewpoint Merging in Requirements Engineering: A Theoretical Replication and Explanatory Study. *Requirements Engineering* 22, 3 (September 2017), 317–338.

[15] Konstantin Kreyman, David Lorge Parnas, and Sanzheng Qiao. 1999. *Inspection Procedures for Critical Programs that Model Physical Phenomena*. Technical Report. McMaster University, Hamilton, ON, Canada.

[16] Xuanyi Lin, Michelle Simon, and Nan Niu. 2018. Hierarchical Metamorphic Relations for Testing Scientific Software. In *Proceedings of the International Workshop on Software Engineering for Science (SE4Science'18)*. Gothenburg, Sweden, 1–8.

[17] Xuanyi Lin, Michelle Simon, and Nan Niu. 2020. Exploratory Metamorphic Testing for Scientific Software. *Computing in Science and Engineering* 22, 2 (March/April 2020), 78–87.

[18] Xuanyi Lin, Michelle Simon, and Nan Niu. 2021. Scientific Software Testing Goes Serverless: Creating and Invoking Metamorphic Functions. *IEEE Software* 38, 1 (January/February 2021), 61–67.

[19] Anders Lundgren and Upulee Kanewala. 2016. Experiences of Testing Bioinformatics Programs for Detecting Subtle Faults. In *Proceedings of the International Workshop on Software Engineering for Science (SE4Science'16)*. Austin, TX, USA, 16–22.

[20] Greg Miller. 2006. A Scientist's Nightmare: Software Problem Leads to Five Retractions. *Science* 314, 5807 (December 2006), 1856–1857.

[21] Nan Niu, Amy Koshoffer, Linda Newman, Charu Khatwani, Chatura Samarasinghe, and Juha Savolainen. 2016. Advancing Repeated Research in Requirements Engineering: A Theoretical Replication of Viewpoint Merging. In *Proceedings of the International Requirements Engineering Conference (RE'16)*. Beijing, China, 186–195.

[22] Megan M. Olsen and Mohammad S. Raunak. 2016. Metamorphic Validation for Agent-Based Simulation Models. In *Proceedings of the Summer Computer Simulation Conference (SummerSim'16)*. Montreal, Canada, 33.

[23] Megan M. Olsen and Mohammad S. Raunak. 2019. Increasing Validity of Simulation Models Through Metamorphic Testing. *IEEE Transactions on Reliability* 68, 1 (March 2019), 91–108.

[24] Open Water Analytics. 2021. Open Water Analytics Stormwater Management Model (OWA SWMM). https://github.com/OpenWaterAnalytics/Stormwater-Management-Model Last accessed: August 2021.

[25] Zedong Peng, Xuanyi Lin, and Nan Niu. 2020. Unit Tests of Scientific Software: A Study on SWMM. In *Proceedings of the International Conference on Computational Science (ICCS'20)*. Amsterdam, The Netherlands, 413–427.

[26] Zedong Peng, Xuanyi Lin, Michelle Simon, and Nan Niu. 2021. Unit and Regression Tests of Scientific Software: A Study on SWMM. *Journal of Computational Science* 53 (July 2021), 101347:1–101347:13.

[27] Christian R. Prause, Jürgen Werner, Kay Hornig, Sascha Bosecker, and Marco Kuhrmann. 2017. Is 100% Test Coverage a Reasonable Requirement? Lessons Learned from a Space Software Project. In *Proceedings of the International Conference on Product-Focused Software Process Improvement (PROFES'17)*. Innsbruck, Austria, 351–367.

[28] Karishma Rahman and Upulee Kanewala. 2018. Predicting Metamorphic Relations for Matrix Calculation Programs. In *Proceedings of the International Workshop on Metamorphic Testing (MET'18)*. Gothenburg, Sweden, 10–13.

[29] Sergio Segura, Gordon Fraser, Ana B. Sánchez, and Antonio Ruiz Cortés. 2016. A Survey on Metamorphic Testing. *IEEE Transactions on Software Engineering* 42, 9 (September 2016), 805–824.

[30] Morteza Pourreza Shahri, Madhusudan Srinivasan, Gillian Reynolds, Diane Bimczok, Indika Kahanda, and Upulee Kanewala. 2019. Metamorphic Testing for Quality Assurance of Protein Function Prediction Tools. In *Proceedings of the International Conference on Artificial Intelligence Testing (AITest'19)*. Newark, CA, USA, 140–148.

[31] Madhusudan Srinivasan, Morteza Pourreza Shahri, Indika Kahanda, and Upulee Kanewala. 2018. Quality Assurance of Bioinformatics Software: A Case Study of Testing a Biomedical Text Processing Tool Using Metamorphic Testing. In *Proceedings of the International Workshop on Metamorphic Testing (MET'18)*. Gothenburg, Sweden, 26–33.

[32] United States Environmental Protection Agency, Office of Research and Development, Center for Environmental Solutions & Emergency Response, Water Infrastructure Division. 2021. EPA ORD Stormwater Management Model. https://github.com/USEPA/Stormwater-Management-Model Last accessed: August 2021.

[33] Igor Wiese, Ivanilton Polato, and Gustavo Pinto. 2020. Naming the Pain in Developing Scientific Software. *IEEE Software* 37, 4 (July/August 2020), 75–82.

[34] Shiyu Yan, Xiaohua Yang, Meng Li, Hua Liu, and Zhaohui Liu. 2016. Research of Testing for Scientific Computing Software in the Area of Nuclear Power Based on Metamorphic Testing. In *Proceedings of the Pacific Basin Nuclear Conference (PBNC'16)*. Beijing, China, 501–512.