EDITORS: Jeffrey Carver, carver@cs.ua.edu
Karla Morris, knmorri@sandia.gov

SPECIAL TRACK: SOFTWARE ENGINEERING

Discovering Metamorphic Relations for Scientific Software From User Forums

Xuanyi Lin, *University of Cincinnati, Cincinnati, OH, 45221, USA*
Michelle Simon, *Environmental Protection Agency, Cincinnati, OH, 45268, USA*
Zedong Peng, *University of Cincinnati, Cincinnati, OH, 45221, USA*
Nan Niu, *University of Cincinnati, Cincinnati, OH, 45221, USA*

Scientific software can be used for decades and is constantly evolving. Recently, metamorphic testing, a property-based testing technique, has shown to be effective in testing scientific software, and the necessary properties are expressed as metamorphic relations. However, the development of metamorphic relations is difficult: it requires considerable practical expertise for the software tester. In this article, we report our experience of uncovering metamorphic relations from a user forum's questions of the United States Environmental Protection Agency's Storm Water Management Model (SWMM). Our study not only illustrates a wealth of end users' expertise in interpreting software results, but also demonstrates the usefulness of classifying the user-oriented metamorphic relations into a nominal, ordinal, and functional hierarchy mainly from the software output perspective.

Scientific software development can last for decades. Compared to business/IT software, developers of scientific software often find it difficult to perform rigorous, systematic testing.[1] A root cause is the oracle problem when the mechanism for checking whether the program under testing produces the expected output. For example, the exact value of *sine*(12) can be hard to obtain, as determining this oracle would require knowledge about how the specific scientific software implementation handles round-off and truncation.

An emerging method to address the oracle problem is *metamorphic testing*.[2] The chief idea is to shift software testing from one input at a time to multiple inputs whose outputs shall meet cert`ain relations. An example of testing *sine*() is asserting *sine*(x) = *sine*(π-x) irrespective of the implementation specifics. Relations like *sine*(x) = *sine* (π-x) are called *metamorphic relations* (MRs), which are necessary properties of the program under test over a sequence of two or more inputs and their corresponding outputs.

Identifying MRs is critical because without them, metamorphic testing cannot be performed. Murphy *et al.*[3] made one of the first attempts to codify common MRs. They enumerated six MRs applicable to numerical or collection-like data types: additive, multiplicative, permutative, invertive, inclusive, and exclusive. These MRs focus only on change made to the input, and are used to verify the software correctness mainly at the method and function levels.

In this article, the focus is on both software correctness verification and user requirements validation. We are interested in testing scientific software as it is run by the end users as an individual, stand-alone application. As scientific software tends to have a large multiparameter input space, users would focus on a subset of inputs and outputs when executing the code. It is therefore through the actual usages of the scientific software that we expect diverse MRs to be discovered.

1521-9615 © 2020 IEEE
Digital Object Identifier 10.1109/MCSE.2020.3046973
Date of publication 23 December 2020; date of current version 25 March 2021.

MRS OF SCIENTIFIC SOFTWARE

In a survey conducted by Segura *et al.*[4] covering the papers published from 1998 (the year that metamorphic

TABLE 1. User-Oriented Studies Applying Metamorphic Testing to Scientific Software.

DOI link to the paper[1]	# of MRs	# of test cases	Code environment	Domain
10.1109/ICECCS.2009.28	4	700	Java; JMT	Queuing network
10.1145/1987993.1988003	5	N/A[2]	Java; JSim; MATLAB; GCS[3]	Health care simulation
10.1007/978-3-642-02138-1_19	4	N/A	N/A	Network simulation
10.1145/1982595.1982597	5	>40	FORTRAN; MPI[4]	Monte Carlo modeling
10.1007/s12243-014-0442-7	3	600	N/A	Cloud computing
10.1145/3194747.3194750	3	2336	C; Python	Storm water modeling
10.1145/2897676.2897678	5	12 300	Java; BBMap	Bioinformatics
10.1145/2896971.2896981	5	N/A	C99; C++; FORTRAN	Light scattering
10.1109/MET.2017.4	1	N/A	N/A	Bioinformatics
10.1109/MET.2017.1	1	N/A	N/A	Time series analysis
10.1145/3193977.3193981	3	N/A	Java; LingPipe	Bioinformatics
10.1145/3193977.3193979	2	N/A	JavaScript	Data analytics
10.1007/s11219-016-9337-3	5	>40	FORTRAN; MPI	Monte Carlo modeling

[1]*The top five papers are also part of the "simulation and modeling" and "numerical programs" surveyed.*[4]
[2]*N/A means not applicable.*
[3]*GCS represents Glycemic Control Simulator.*
[4]*MPI represents Message Passing Interface.*

testing was introduced[2]) to 2015, 119 primary studies were identified. We deem this survey's categories of "simulation and modeling" and "numerical programs" as the most relevant to scientific software; these categories were in contrast to categories such as "web services and applications" and "variability and decision support." It is shown in Segura *et al.*[4] that these two scientific software categories are among the most popular domains where metamorphic testing is applied. To include more recent studies, we performed a citation analysis via Google Scholar on the nine "simulation and modeling" and "numerical programs" papers discussed in the survey of Segura *et al.*[4] In addition, we manually searched the proceedings of two relevant workshops since 2016: one on metamorphic testing (http://metwiki.net/MET19/series.html) and the other on software engineering for science (https://se4science.org/workshops/). Our citation analysis and workshop proceedings search was performed in March 2020.

Our search resulted in 14 papers. Combined with the nine papers discussed,[4] the total number of studies was 23. Among this group of work applying metamorphic testing to simulation, modeling, and numerical programs, we performed a final filtering with a user-oriented focus, i.e., we paid special attention to the scientific software that was developed for use by others and not only used by the original developers. To this end, we checked whether the software was publicly downloadable and whether there was evidence about end users' usage of the software. Applying our user-oriented filtering led to 13 papers. Table 1 summarizes these studies.

From Table 1, it can be seen that each study involved a handful MRs (from 1 to 5), which are identified manually by testers or researchers. Although a body of work uses machine learning to predict if a code unit (e.g., a method) exhibits certain MRs (e.g., "additive," "inclusive," and "permutative"),[5] the diverse domains listed in Table 1 show that focusing on particular input parameters and their influences on specific outputs can be a useful strategy for deriving scientific software's MRs. Our work expands the discovery of MRs by turning attention to the end users' point of view.

SWMM DEVELOPMENT AND EVOLUTION

The subject system of our study is the SWMM developed and maintained by the United States Environmental Protection Agency (EPA). SWMM is a dynamic rainfall-runoff simulation model that computes runoff quantity and quality from primarily urban areas. The development of SWMM began in 1971 and since then the software has undergone several major upgrades.

We studied SWMM version 5.1.015, released in July 2020. The top of Figure 1 shows a screenshot of the SWMM5 graphical user interface where inputs (e.g., properties of a weir) and output (e.g., link flow results) are illustrated. The computational engine of SWMM is written in C with about 46,300 lines of code, which can be compiled either as a DLL (dynamic link library) under Windows or as a stand-alone console application under both Windows and Linux. The bottom of Figure 1 shows that our metamorphic testing is file based and invokes the command-line environment to run SWMM. We share our testing code at https://doi.org/10.7945/ravz-y408.

The users of SWMM include hydrologists, engineers, and water resources management specialists

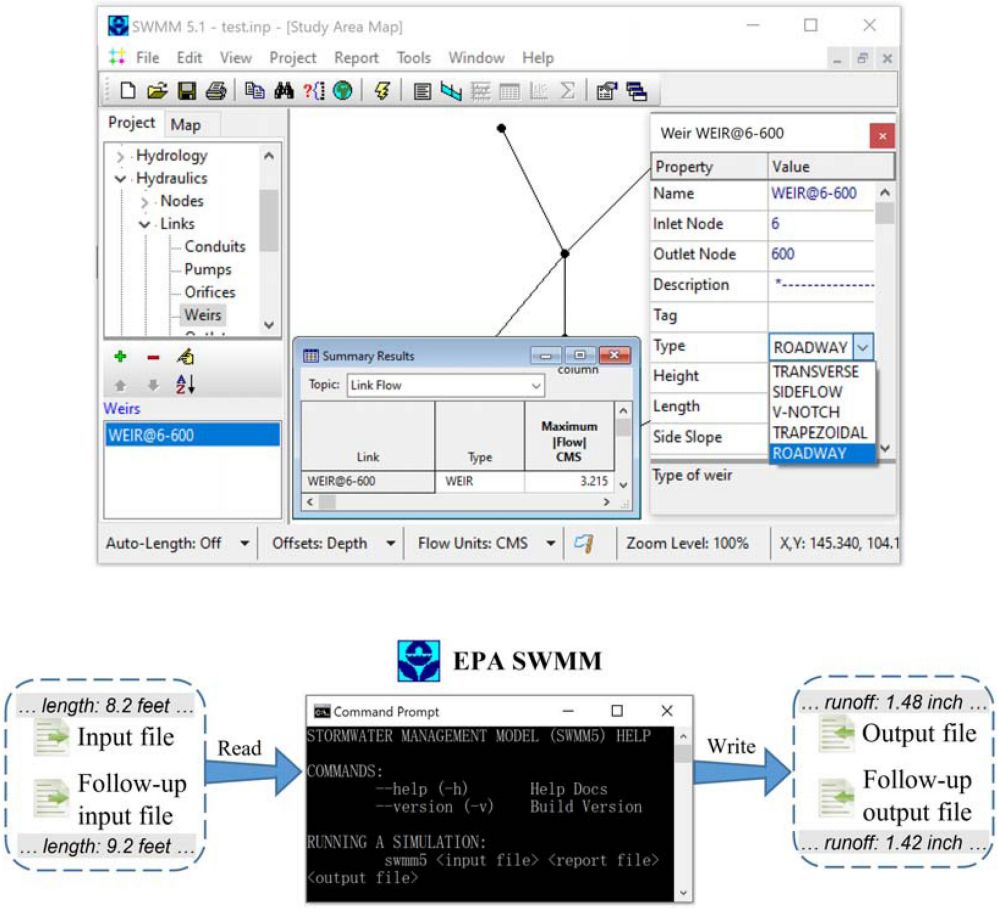


FIGURE 1. SWMM5 Graphical User Interface (top) and our file-based metamorphic testing (bottom).

who are interested in the planning, analysis, and design related to storm water runoff and quality, combined and sanitary sewers,[6] and other drainage systems in urban areas. Since end users generally face the oracle problem when validating each single execution of SWMM, metamorphic testing can help. In fact, our ongoing work on hierarchical and exploratory MRs[7,8] illustrates this by detecting defects when SWMM is integrated with an automated parameter calibration system.[9] Here, our interests are in SWMM itself.

Because the development of SWMM has lasted for five decades and continues to be active, we decide to extract potentially valuable information from user forums. A forum allows people who share similar interests to discuss issues related to certain topics. For SWMM end users, the topics typically include how to run the software and interpret its results. We rely mainly on Open SWMM (https://www.openswmm.org) to extract the software usage questions valuable toward formulating MRs.

DISCOVERING MRS FROM SWMM USAGE QUESTIONS

The first set of questions we noticed is that the software did not run at all under certain conditions. Figure 2 shows such an example where the formats of date in an external data file caused SWMM to "crash." Although how to format time series is specified in SWMM's user manual, which is available online (https://www.epa.gov/water-research/storm-water-management-model-swmm), it is clear that some users (e.g., User1 in Figure 2) do not follow the manual and feel unsatisfied after the continued error appearances.

The post shown in Figure 2 led us to formulate a metamorphic relation MR_{date} by checking how different date formats could result in errors. Discovering MR_{date} allowed us to run SWMM via the command line console automatically on 480 test cases by considering five factors (https://en.wikipedia.org/wiki/Date): basic component (year-month-day, month-day-year, or day-month-year), year (two-digit or four-digit), month (one to three

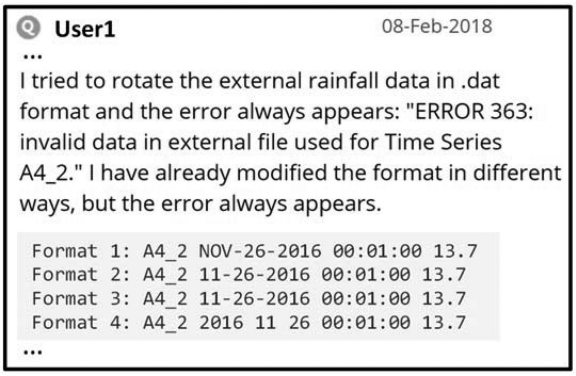


FIGURE 2. Question (https://www.openswmm.org/Topic/11247/question-about-error-363) provoking the formulation of MR_{date}.

characters, or fully spelled out), day (one to three characters, or fully spelled out), and separator ("/" [slash], "." [dot], "-" [dash], " " [space], or "" [no space]).

The automated testing results of MR_{date} are shown in Figure 3. User1's post corresponds to the middle bar of Figure 3. We used 480 different ways to format the date of time series in an external rainfall .dat file. SWMM encountered errors under 464 (97%) tests including the ones mentioned in User1's post. It is surprising to realize that when the same 480 cases were tested through SWMM's .inp input file, the error ratio dropped to 90%, as shown in the bottom bar of Figure 3. MR_{date} helped detect real defects of SWMM, confirmed by an original developer. The results further suggest that the code written to handle the "code input-in" data (.inp) is more buggy than the code written to handle the external input data files (.dat). The developers should test the "internal" code more rigorously, leaving less room for the "inside" errors to creep in. MR_{date} helps to achieve that.

The second class of questions that we analyzed concerned the SWMM runtime behavior in specific situations. In contrast to the first set of questions discussed above, SWMM was able to complete a run; however, certain software behaviors puzzled the users. For example, User2 raised a question: when a storm is at its peak the total inflow rate to a node (Qin) is much greater than the total outflow rate from a node (Qout) for an open storm ditch: however, Qout becomes much greater than Qin when the storm's intensity lessens. User2 expressed it as "a strange effect" of SWMM as shown in Figure 4 (left).

Commenter1 provided a useful hint to better understand Qin and Qout: In SWMM, each node assembly consists of the node itself and half of the length of each link connected to the node. As a result, SWMM exhibits a storage effect and the "strange effect" mentioned by User2 is actually a feature rather than a bug. Figure 4 (right) illustrates this storage effect with the junction node positioned in a sloped road. The node absorbs the storm to some extent when the storm becomes heavy, serving effectively as a surface water storage. In this case, Qin is much larger than Qout. When the storm's intensity lessens, the junction node outflows water, resulting in a greater Qout than Qin.

The post by User2 and the related discussions about the post led us to produce MR_{ditch}, which monitors how Qout changes via the alternation of Qin (storm volume). Automatically testing MR_{ditch} could better delineate the behavioral patterns of SWMM. Although no bugs would be detected with MR_{ditch}, the systematic testing associated with this MR could make better sense of the software behavior and further depict it in a more comprehensive form, e.g., via a piecewise linear function. Explaining what User2 referred to as the strange effect may require diagnostic methods like parameter

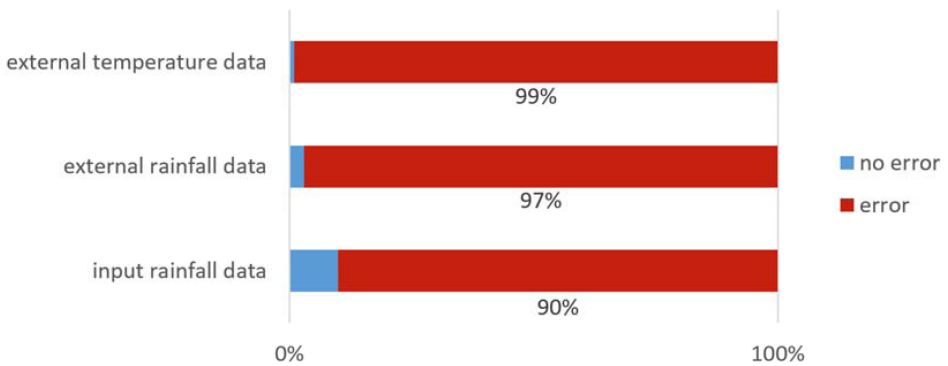


FIGURE 3. Testing results of MR_{date} where each bar is based on 480 test cases.

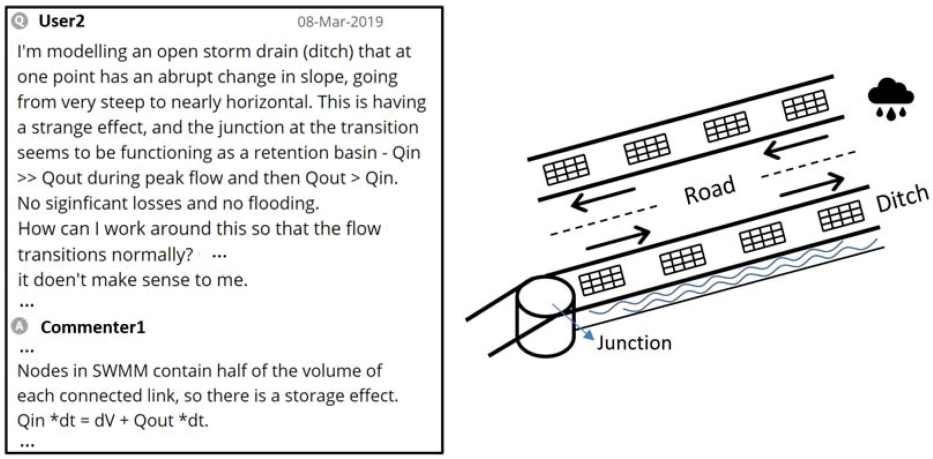


FIGURE 4. Question (https://www.openswmm.org/Topic/19102/junction-apparently-functioning-as-a-retention-basin) provoking the formulation of MR_{ditch}.

tracking[8] and domain knowledge like the temporary storage effect offered by Commenter1.

The third category of usage questions that emerged from our analysis embodies the end users' desire to understand the software output in a quantitative way. Figure 5 (top) presents a post where User3 did not choose the correct equation to calculate "Water Head." What User3 should be using was: Water Head = Storage Water Level – Storage Bottom Elevation, but what the user actually used was: Water Head = Storage Water Level – Downstream Node Invert.

Although User3 perceived SWMM's calculation to be wrong, he made sense of the situation by diagnosing which incorrect equation caused the problem, due to the visibility of "Water Head" values trackable from SWMM's output. However, in some situations, what the end users are interested in tracking may not be directly visible to them. The screenshot shown in Figure 1 is one such example. The user is provided with a drop-down list of five options to select the type of weir in the simulation. According to the specifications, SWMM implements the "Weir Type" as shown in Figure 5 (bottom).

Unfortunately, "Weir Type" values are not directly viewable to the end users. Had the user chosen an incorrect equation but was unaware of the mistake, it would be challenging for the user to figure out that SWMM's visible outputs were wrong due to the "Weir Type" selection. Fortunately, a class of MRs could be formed on the basis of the above "Weir Type" specifications. For example, the relation between V-NOTCH and ROADWAY can be inferred by how h varies, i.e., $(C_w S h^{5/2})/(C_w L h^{3/2}) = (S/L)h$, where S and L are constants. Figure 6 (left) plots MR_{weir} between V-NOTCH and ROADWAY. Using $(S/L)h$, the change from V-NOTCH to ROADWAY, e.g., triggered by the end user at SWMM runtime, can be quantitatively characterized.

Although the values of "Weir Type" are not directly accessible to the end users, the quantitative MR_{weir} makes it easier to locate surrogate variables to track the switch between "Weir Type" options and hence easier to correct the wrong choice at SWMM runtime. Figure 6 (right) shows two output variables directly visible to the end users: "maximum flow" and "average volume" tested when "Weir Type" was switched from V-NOTCH to ROADWAY. Although "average volume" had a nonlinear change, "maximum flow" reflected the linear function with the slope = (S/L). The output variable, "maximum flow," thus effectively serves as a surrogate for tracking V-NOTCH and ROADWAY.

In summary, our analysis of SWMM user forums identified a broad range of questions raised by the end users themselves that enabled us to perform automated metamorphic testing. Compared to other testing techniques like fuzzing where invalid, unexpected, or random inputs are generated, the MRs in our work are formulated based on the actual uses of the scientific software. Moreover, fuzzing tends to crash or hang an application, whereas MRs attempt to check whether the expected output relations hold. For scientific software with a large input and a large output spaces, the "input change leading to output change" can be a more effective testing strategy than crashing a software system. During our collaboration meetings, we shared the MRs discovered from the user forums with the SWMM development team and received positive feedback. Although the MRs all seemed sensible, the team never thought testing the software by using these relations.

User3 12-Feb-2018

I have been encountering an issue where a previous SWMM model (v 5.1.006) is re-run in v5.1.011 and I am seeing different results for pond outlets using "tabular/depth" rating tables.
...
where the SWMM code seems to have changed for how the head on an outlet rating table is calculated:
SWMM v5.1.006 : Water Head = Storage Water Level – Storage Bottom Elevation.
SWMM v5.1.011 : Water Head = Storage Water Level – Downstream Node Invert.
...

Commenter2

... the SWMM 5.1 engine source code for Outlets in v5.1.011 is exactly the same as in v5.1.006. However, the equation you gave for v5.1.006 is the one used for Kinematic Wave routing while the one you gave for v5.1.011 is for Dynamic Wave routing.
...

Weir Type	Cross Section Shape	Flow Formula
TRANSVERSE	Rectangular	$C_w S h^{3/2}$
SIDEFLOW	Rectangular	$C_w L h^{5/3}$
V-NOTCH	Triangular	$C_w S h^{5/2}$
TRAPEZOIDAL	Trapezoidal	$C_w L h^{3/2} + C_{ws} S h^{5/2}$
ROADWAY	Rectangular	$C_w L h^{3/2}$

C_w = weir discharge coefficient, L = weir length, S = side slope of V-NOTCH or TRAPEZOIDAL weir, h = head difference across weir, C_{ws} = discharge coefficient through sides of TRAPEZOIDAL weir.

FIGURE 5. Question (https://www.openswmm.org/Topic/11274/epa-swmm-outlet-rating-discharge-differences-between-swmm-versions) provoking the formulation of MR_{weir}.

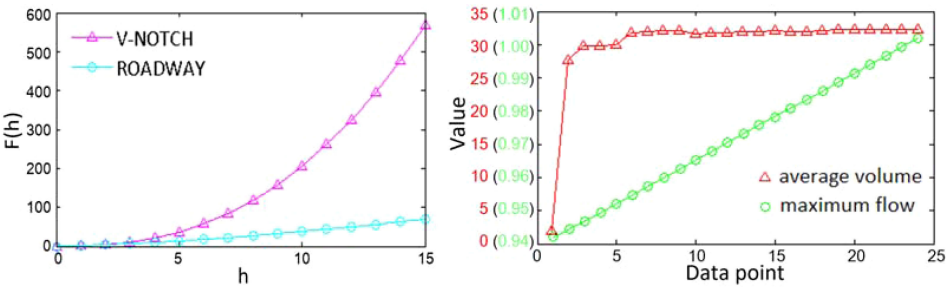


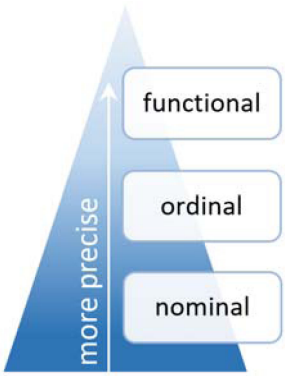
FIGURE 6. Plotting V-NOTCH and ROADWAY on h (left). Testing MR_{weir} of V-NOTCH and ROADWAY on two SWMM output variables (right).

In fact, the developers were surprised that bugs were uncovered by MR_{date} and user questions were raised regarding MR_{ditch}.

When it comes to assisting in the formulation of MRs, we classified the information extracted from the usage questions into three categories: those complaining the software is not able to run at all in certain conditions, those expressing the difficulty of making sense of a peculiar behavior of the software in some situations, and those trying to figure out what might go wrong

TABLE 2. MRs Discovered Manually From SWMM User Forums (Left) and a Hierarchy Focusing on Output Relations (Right).

MR	Definition
MR_{weir}	if weir type is switched at runtime, the resultant average volume and maximum flow will form a quantitative relation
MR_{ditch}	if inflow is increased when modelling the ditch, the resultant outflow will be changed
$MR_{iteration}$	if simulation iteration is increased, the resultant continuity error will decrease
MR_{date}	if the date format in the input data is changed, the error message will be prompted at runtime
MR_{file}	if the format type of input data file is changed, the error message will be prompted at runtime
MR_{flow}	if flow routing method is switched, the resultant flow will be changed



with the software output from the end user's control perspective and how to get it right. We conclude next by organizing these categories in a hierarchy of MRs and by sharing our lessons learned. Our proposed categories are: 1) errors that prevent successful initiation or completion of the program, 2) potential differences in output from expected behavior, and 3) potential differences in output related to operations, functions, or criteria outside the users' control.

CONCLUDING REMARKS

Metamorphic testing alleviates the oracle problem commonly faced by scientific software.[1] Selecting a proper set of MRs is useful for comprehensive software testing. While current approaches rely on researchers' domain knowledge or developers' implementation knowledge to identify MRs, we have shown in this article that forums where end users pose questions and exchange concerns are valuable sources to discover MRs. Compared to software documentation like SWMM user manual, forums contain more dynamic usage scenarios, allowing some MRs to be identified more easily. For example, the input variable of MR_{Weir} ("Weir Type") appears four times in the SWMM user manual (https://www.epa.gov/water-research/storm-water-management-model-swmm-version-51-users-manual); however, this input variable never coappears with the two output variables on the same page. The closest distance of "Weir Type" and "maximum flow" is 3 pages, and that of "Weir Type" and "average volume" is 68 pages, suggesting the difficulty of extracting MR_{Weir} only from software documentation. Our manual analysis of these SWMM forums helped extract half of a dozen MRs specific to SWMM, which we show in Table 2 (left).

Our results suggest a hierarchy of nominal, ordinal, and functional MRs as shown in Table 2 (right). This hierarchy is rooted in the theory of measurement scales,[10] where nominal groups data only by names (e.g., "run" or "not run") and ordinal allows data to be ordered (e.g., "less than" or "greater than"). The most precise MRs in our hierarchy are those of functional forms. When a change is made to the input (Δx), a functional MR allows the corresponding output change (Δy) to be predicted in an exact manner, e.g., the "Weir Type" switch from V-NOTCH to ROADWAY would follow a linear relationship. Compared with the MRs codified in Murphy *et al.*,[3] the hierarchy of MRs of Table 2 focus more on the behavior and the output of software.

Our ongoing work investigates automated ways of mining MRs from the usage questions, exploits serverless architectures and frameworks to efficiently execute metamorphic tests[11] and leverages numerical regression tests[12] to verify the correctness of the MRs.

ACKNOWLEDGMENTS

This work was supported in part by the U.S. National Science Foundation (Award CCF 1350487) and U.S. Environmental Protection Agency.

DISCLAIMER

The research described in this article has been funded in part by the U.S. Environmental Protection Agency

(EPA). It has been subjected to review by the Office of Research and Development and approved for publication. Approval does not signify that the contents reflect the views of the Agency, nor does mention of trade names or commercial products constitute endorsement or recommendation for use.

REFERENCES

1. U. Kanewala and J. M. Bieman, "Testing scientific software: A systematic literature review," *Inf. Softw. Technol.*, vol. 56, no. 10, pp. 1219–1232, Oct. 2014.
2. T. Y. Chen, S. C. Cheung, and S. M. Yiu, "Metamorphic testing: A new approach for generating next test cases," The Hong Kong Univ. Sci. Technol., Hong Kong, China, Tech. Rep. HKUST-CS98-01, 1998.
3. C. Murphy, G. E. Kaiser, L. Hu, and L. Wu, "Properties of machine learning applications for use in metamorphic testing," in *Proc. 20th Int. Conf. Softw. Eng. Knowl. Eng.*, San Francisco, CA, USA, Jul. 2008, pp. 867–872.
4. S. Segura, G. Fraser, A. B. Sanchez, and A. R. Cortes, "A survey on metamorphic testing," *IEEE Trans. Softw. Eng.*, vol. 42, no. 9, pp. 805–824, Sep. 2016.
5. A. Nair, K. Meinke, and S. Eldh, "Leveraging mutants for automatic prediction of metamorphic relations using machine learning," in *Proc. 3rd Int. Workshop Mach. Learn. Techn. Softw. Qual. Eval.*, Tallinn, Estonia, Aug. 2019, pp. 1–6.
6. H. Gudaparthi, R. Johnson, H. Challa, and N. Niu, "Deep learning for smart sewer systems: Assessing nonfunctional requirements," in *Proc. 42nd Int. Conf. Softw. Eng. Softw. Eng. Soc. Track*, Seoul, South Korea, Jun./Jul. 2020, pp. 35–38.
7. X. Lin, M. Simon, and N. Niu, "Hierarchical metamorphic relations for testing scientific software," in *Proc. Int. Workshop Softw. Eng. Sci.*, Gothenburg, Sweden, Jun. 2018, pp. 1–8.
8. X. Lin, M. Simon, and N. Niu, "Exploratory metamorphic testing for scientific software," *Comput. Sci. Eng.*, vol. 22, no. 2, pp. 78–87, Mar./Apr. 2020.
9. S. Kamble, X. Jin, N. Niu, and M. Simon, "A novel coupling pattern in computational science and engineering software," in *Proc. Int. Workshop Softw. Eng. Sci.*, Buenos Aires, Argentina, May 2017, pp. 9–12.
10. S. S. Stevens, "On the theory of scales of measurements," *Science*, vol. 103, pp. 677–680, Jun. 1946. Art. no. 2684.
11. X. Lin, M. Simon, and N. Niu, "Scientific testing goes serverless: Creating and invoking metamorphic functions," *IEEE Softw.*, vol. 38, no. 1, pp. 61–67, Jan./Feb. 2021.
12. Z. Peng, X. Lin, and N. Niu, "Unit tests of scientific software: A study on SWMM," in *Proc. 20th Int. Conf. Comput. Sci.*, Amsterdam, The Netherlands, Jun. 2020, pp. 413–427.

XUANYI LIN is currently a Ph.D. candidate with the Department of Electrical Engineering and Computer Science, University of Cincinnati. His research interests include automated software testing, scientific software development, and requirements engineering. He received the M.Sc. degree in computer science from the University at Albany, State University of New York, NY, USA, in 2016. Contact him at linx7@mail.uc.edu.

MICHELLE SIMON is currently a Senior Chemical Engineer with the U.S. Environmental Protection Agency Office of Research and Development. She is also with the Center for Environmental Solutions and Emergency Response, Water Infrastructure Division, Cincinnati, OH, USA. He is a professional Engineer. Her research interests include stormwater modeling, green infrastructure, and wastewater infrastructure. She received the B.S. degree in chemical engineering from the University of Notre Dame, the M.S. degree in chemical engineering from the Colorado School of Mines, and the Ph.D. degree in environmental science from the University of Arizona . Contact her at simon.michelle@epa.gov.

ZEDONG PENG is currently a Ph.D. student with the Department of Electrical Engineering and Computer Science, University of Cincinnati. His research interests include requirements engineering, metamorphic testing, and natural language processing. He received the M.Sc. degree in computer science from the Ball State University, Muncie, IN, USA, in 2019. Contact him at pengzd@mail.uc.edu.

NAN NIU is currently an Associate Professor with the Department of Electrical Engineering and Computer Science, University of Cincinnati. His research interests include requirements engineering, scientific software development, and human-centric computing. He received the Ph.D. degree in computer science from the University of Toronto . Contact him at nan.niu@uc.edu.