Exploratory Metamorphic Testing for Scientific Software

Xuanyi Lin and Nan Niu
University of Cincinnati

Michelle Simon
Environmental Protection Agency

Abstract—Scientific model developers are able to verify and validate their software via metamorphic testing (MT), even when the expected output of a given test case is not readily available. The tenet is to check whether certain relations hold among the expected outputs of multiple-related inputs. Contemporary approaches require the relations to be defined before tests. Our experience shows that it is often straightforward to first define the multiple iterations of tests for performing continuous simulations, and then keep multiple and even competing metamorphic relations open for investigating the testing-result patterns. We call this new approach exploratory MT, and report our experience of applying it to detect bugs, mismatches, and constraints in automatically calibrating parameters for the United States Environmental Protection Agency’s Storm Water Management Model.

Testing scientific software is important because scientific simulations support critical decision making in many fields. For example, because nuclear weapons cannot be field tested, the code is used to determine the impact of the modifications in nuclear weapon simulations. Advances are made to help the scientific simulations team to run the tests with command-line input and control, to check code modifications before contributing changes, and to test the specific programming environments like MATLAB. For scientific model developers, the oracle problem remains a challenge for software testing. An oracle refers to the mechanism for
checking whether the program under testing produces the expected output when executed using a set of test cases. For reasons like the approximations involved in simulation models, oracles may not be available. The scientific software under our testing is such a model that lacks suitable oracles. This model presents the oracle problem, which makes it difficult to detect faults in scientific code.

An emerging method to overcome the oracle problem is metamorphic testing (MT). The idea is illustrated in Figure 1. When testing an individual input of a program that computes the sine function, the oracle may be difficult to obtain. For instance, the exact value of $\sin(12)$ could depend on how floating-point arithmetic and compiler optimization are implemented. If two test cases—12 and $(\pi-12)$—are run one after the other as shown in Figure 1, then their outputs shall be equivalent regardless of the implementation specifics. The relationship, $\sin(x) = \sin(\pi-x)$, is called a metamorphic relation. It is through checking whether the metamorphic relations hold or not that MT alleviates the oracle problem.

Researchers have applied MT to test scientific software. Contemporary approaches require the metamorphic relations to be defined a priori. In other words, source and follow-up tests are executed only after a relation is determined, because that relation is used both to guide the test case generation and to analyze the test execution results.

In our ongoing work of developing and testing software integration solutions, we realized that determining a relatively complete set of metamorphic relations without running some tests was difficult. We therefore proposed to develop a hierarchy of metamorphic relations with new relations formulated based on the testing results of a previous round of MT.

Building on the prior work, we present in this paper a new framework where the test cases are developed and executed to inform MT. We call this approach exploratory MT by highlighting the novelty of keeping the multiple, and even competing, metamorphic relations open while investigating the simulation software’s testing-result patterns. We demonstrate the effectiveness of exploratory MT with the code faults detected as well as the mismatches uncovered in scientific software integration.

The remainder of the paper is structured as follows. After briefly reviewing metamorphic testing applied to scientific software, we describe the integrated software system under test and our work flow. Then, we detail exploratory MT, analyze the results of applying exploratory MT to the system under test, and discuss the limitations of this research as well as our lessons learned. Finally, we present the conclusions and the future work.

**METAMORPHIC TESTING FOR SCIENTIFIC SOFTWARE**

Testing is a mainstream approach toward software quality. When the expected output of a test case’s execution is not available or is too expensive to be used, MT can be applied. MT is helpful to scientific model developers because they must address a tower of approximations in their software and the software may be written to find answers previously unknown.

Chen et al. performed one of the earliest studies of applying MT to scientific software, namely, a numerical program implementing the...
alternating direction implicit method to solve the Laplace equation with Dirichlet boundary conditions. Their work showed that once a metamorphic relation was determined, selecting a pair of source and follow-up test cases could reveal the subtle errors in the code.

In a survey covering the papers published from 1998 (the year that MT was introduced) to 2015, more than a quarter applications of MT were related to scientific software, ranging from numerical programs to simulation and modeling. This applicability continues. A case in point is the recent work by Ding et al. who defined five metamorphic relations to validate an open-source light scattering simulator. These relations were specified to test the effects of the software when the input biological-cell images were altered in size, shape, orientation, refractive index, etc. For other applications of MT and its further improvement, please refer to the survey conducted by Segura et al.

INTEGRATION OF STORM WATER MANAGEMENT MODEL (SWMM) AND PEST

The United States Environmental Protection Agency’s Storm Water Management Model (SWMM) is a dynamic rainfall-runoff simulation model that computes runoff quantity and quality from primarily urban areas. Thousands of studies throughout the world have been carried out by using SWMM, assisting such stakeholders as hydrologists, engineers, and water resources management specialists in the planning, analysis, and design related to storm water runoff, combined and sanitary sewers, and other drainage systems in urban areas.

We work on the SWMM version 5.1.012 released in 2017. The computational engine is written in C with about 45 500 lines of code. The engine can be compiled either as a dynamic link library under Windows or as a stand-alone console application under both Windows and Linux. In addition to portability, SWMM embraces customizability for various stakeholders. Consequently, the software takes many input parameters, especially when the catchment is large and complex. As manually calibrating the parameters’ values is tedious and error-prone, we are developing solutions to integrate SWMM with PEST, a software package for automated parameter estimation. PEST is written in FORTRAN and its latest release in 2017 (version 14.2) has around 210 000 lines of code. Our connector software, called SWMM2PEST, is written in Python and has about 2500 lines of code.

Figure 2 shows how automated parameter calibration is accomplished via the integration of the three software systems: SWMM, PEST, and SWMM2PEST. The observation file, which is a static part of Figure 2, contains the real-world values of the SWMM output variables (e.g., runoff for a specific period of time in a given place) recorded by observers and hence defines the ideal simulation...
results. Correspondingly, the calibration file consists of the simulation results of the same variables in the observation file produced by the integrated software system. The dynamic part of Figure 2 is the iteration triggered by SWMM2PEST, which reads the SWMM input file and invokes PEST so as to match the simulation results with the observation values as closely as possible. In each iteration, an \( R^2 \) value, the coefficient of determination calculated based on the simulation results and the actual/observed values, can be measured to indicate statistically the goodness of fit of a model: the greater the \( R^2 \) value, the better the regression line approximates the real-data points. In this study, we calculated the \( R^2 \) value as follows:

\[
R^2 = 1 - \frac{\sum_{i=1}^{n} (O_i - S_i)^2}{\sum_{i=1}^{n} (O_i - \bar{O})^2}
\]

where \( O_i \) is the observed value and \( S_i \) is the corresponding SWMM simulated result at the same time. \( \bar{O} \) is the mean value of the observation data. Both the observation dataset and the simulation dataset have \( n \) values.

Specifically, the input file and the observation file in Figure 2 are the input datasets of this integrated software system. The calibration file, on the other hand, is the output data. At each iteration, the input file will be updated based on the calibration result from the previous iteration. The observation file is used after each iteration to compute measures like the \( R^2 \) coefficient. The \( R^2 \) values help check the effect of each iteration of simulation and calibration.

**EXPLORATORY MT**

An important aspect of Figure 2 is the iteration involving continuous simulations where the outputs from a previous round of calibration are fed to the subsequent round as inputs. We therefore structure MT around these output-as-input iterations. Starting from the initial round, any round of simulation and calibration serves as our MT’s source test case, and the next round is the follow-up test case. From the validation perspective, we focus on the \( R^2 \) value from each round to compare the source and follow-up tests.

Traditionally, a desired metamorphic relation would be defined as follows: \( R^2 \) (follow-up test) > \( R^2 \) (source test). This is because metamorphic relations represent necessary properties of the software under test in relation to multiple inputs and their expected outputs.\(^6\) For the automated and continuous simulations shown in Figure 2, it is the desired outcome for a follow-up simulation to be a better fit than the previous round. In other words, if the pattern of increased \( R^2 \) values during multiple testing rounds is not observed, the simulations performed by SWMM, PEST, and SWMM2PEST could have faults.

While specifying the desired metamorphic relation of increased \( R^2 \) values, we realize the importance to systematically analyze the undesired testing results as well. In fact, the novelty of our MT framework shown in Figure 3 is to explicitly classify the opposites of the desired property and leave all the paths open for exploration. In addition to \( R^2 \) (follow-up test) > \( R^2 \) (source test), Figure 3 shows three more patterns to investigate: simulation returns no result, \( R^2 \) (follow-up test) < \( R^2 \) (source test), and \( R^2 \) (follow-up test) = \( R^2 \) (source test). Compared to the “increased” relation, the other three relations (Resultless, Decreased, and Unchanged) can be seen as competing relations.

As shown in Figure 3, we define the exploratory MT as the process from the exploration phase of constructing output-as-input iterative MT to fault detection phase based on the diagnoses of explored results. The first step is continuous simulation, which is the process of iteratively running MT on the integrated software system as illustrated in Figure 2. After obtaining and comparing the \( R^2 \) values at all the iterations in the initial step, we found four testing-result patterns including resultless, decreased, unchanged, and increased. Furthermore, we used diagnostic methods such as parameter monitoring to detect the faults based on the testing-result patterns. Finally, we resolved the related issues and used MT to check the resolutions.

The exploration phase of Figure 3 is diverging, i.e., four different paths are explored. However, the effectiveness of these exploratory relations, and more importantly, the practical value of MT lies in the capability of detecting faults. To that end, our framework converges via the phase of fault detection where we apply such diagnosis methods as monitoring the parameters embodied in different tests. These diagnoses can help uncover bugs in the code, constraints in running
scientific simulations, and mismatches between different software systems. A bug is defined as a segment of codes in error that would cause the failure of the system. A constraint means that certain operations are nonexecutable or some results are not achievable because of the specific software requirements or limitations. A mismatch exists when two software systems behave inconsistently, such as different definitions or different precision requirements for the same variable. This phase of fault detection is converging because our intention of diagnoses is to surface concrete elements of the software system rather than to continue exploring the different paths/patterns. Further actions, like debugging, can be taken to resolve the identified issues, and new MT patterns could be observed to better understand the issues’ characteristics and to assess the resolutions’ effects.

RESULTS AND ANALYSIS

We evaluated the exploratory MT framework depicted in Figure 3 with two datasets. FS10 green roof\(^1\) represents the fire station 10 located in Seattle, Washington, and the green roof is defined as a building’s roof that is partially or completely covered with vegetation together with the growing medium. The other research dataset is Villanova infiltration\(^2\) constructed in 2004, representing the infiltration trench retrofit at Villanova University in Pennsylvania. Some main differences between the FS10 and Villanova datasets lie in the low-impact development (LID) controls. The FS10 LID contains surface, soil, and drain mat layers, whereas Villanova LID consists of surface, storage, and drain layers. As a result, 33 parameters are used for FS10 and 31 parameters for Villanova.

Table 1 summarizes the results when our exploratory MT framework is applied to the two datasets. Because exhaustively testing the calibrations on either 33 or 31 parameters is impractical, we randomly developed a number of test cases to explore which metamorphic relation would hold. For example, from the 33 FS10 parameters, a randomly selected subset would be subject to calibration and the remaining parameters would not change their values. This particular

**Figure 3.** Exploratory metamorphic testing for scientific software. Continuous simulations serve as testing iterations, multiple testing patterns are explored, and diagnoses such as parameter monitoring are employed to detect faults.
test case would experience continuous simulations and the $R^2$ values from each simulation round could then be compared. As given in Table 1, 275 and 194 test cases were run against F10 green roof and Villanova infiltration, respectively. For each test, five iterations of simulations were executed.

In both the datasets, more than half of the tests followed the desired metamorphic relation (“Increased”) where a subsequent simulation resulted in a better fit than the one from the earlier iteration. This confirms the positive effect that automated parameter calibration had on scientific simulation. Meanwhile, the other three patterns all had tests falling into their categories, indicating the existence of faults from the perspective of validating SWMM, PEST, and SWMM2PEST. We illustrate next how diagnoses of our framework were carried out to detect faults.

When inspecting the FS10 exploratory MT results, we noticed that the $R^2$ values of 13 tests (4.7%) stayed unchanged in all the five simulation rounds. We performed the diagnoses by tracking which parameters were calibrated in those 13 tests. Such a monitoring is presented in Table 2. The pair of parameters, P14 and P15, had very consistent co-appearance and co-absence patterns in the 13 “Unchanged” tests. P13 and P18 formed another parameter pair. The actual names of P14 and P15 were “storage_depth_imperv” and “storage_depth_perv,” respectively, and those of P13 and P18 were “percent_impervious_area_treated” and “percent_impervious,” respectively.

Further inspection revealed that P14 and P15 had the same name, “F0strg_dpth_,” in the intermediate file generated by SWMM2PEST. Similarly, P13 and P18 were both stored as “F0prcnt_mprv” in the intermediate file. It was not a coincidence that both intermediates transitioning from SWMM to PEST had 12 characters, because PEST only allowed at most 12 characters for each parameter name. SWMM2PEST, therefore, implemented some string truncate procedures shown

| P TC | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 30 | 31 | 32 | 33 |
|------|---|---|---|---|---|---|---|---|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 1    | ✗ |   |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 2    |   | ✗ |   |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 3    |   |   | ✗ |   |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 4    |   |   |   | ✗ |   |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 5    |   |   |   |   | ✗ |   |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 6    |   |   |   |   |   | ✗ |   |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 7    |   |   |   |   |   |   | ✗ |   |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 8    |   |   |   |   |   |   |   | ✗ |   |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 9    |   |   |   |   |   |   |   |   | ✗ |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 10   |   |   |   |   |   |   |   |   |   | ✗ |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 11   |   |   |   |   |   |   |   |   |   |   | ✗ |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 12   |   |   |   |   |   |   |   |   |   |   |   | ✗ |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |
| 13   |   |   |   |   |   |   |   |   |   |   |   |   | ✗ |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |    |

Table 2. Monitoring the appearance and absence of the 33 FS10 green roof parameters (columns) in the 13 (4.7%) “Unchanged” test cases (rows).
in Figure 4 to shorten the parameter name when it was longer than 12 characters. Clearly, this vowel-removal-then-cutting-off-the-tail implementation was buggy, as illustrated by P14 and P15, as well as by P13 and P18.

We classified this issue as a bug since the intent of SWMM2PEST was to transform SWMM parameters in a form compatible with PEST. This bug could be resolved by generating unique parameter identifiers, ensuring different parameter names via postchecking, or other methods. Our current debugging was performed by changing the names of P13, P14, P15, and P18 in the intermediate file. As shown in Figure 3, the debugging effect was observed by rerunning the 13 FS10 tests, which were previously “Unchanged.” This time, none of the 13 tests kept the $R^2$ values intact: 3 became “Resultless,” 2 “Decreased,” and 8 “Increased.”

In addition to locating bugs, our exploratory MT framework is capable of uncovering mismatches and constraints in scientific simulations performed by more than one software system. When diagnosing the “Resultless” tests, we realized that certain parameters should not be calibrated. In FS10, for example, the “area” (P1) of subcatchment is related to the “number_replicate_units” (P10) and “area_each_units” (P16). In fact, the relationship, $P1 \geq P10 \times P16$, shall hold throughout the simulation. If P10 is to be calibrated while P1 and P16 are fixed, then the constraint can be violated, causing the simulation to return no result. The constraints, such as $P1 \geq P10 \times P16$, encapsulate domain knowledge and must be made explicit for testers, users, and maintainers.

A mismatch between SWMM and PEST was found during the diagnoses of the “Decreased” tests. The mismatch was about how the two software systems handled round off. The parameter “top_width_overland_flow_surface” was tracked during testing to have the value of 15.618274 in SWMM. In the same testing session, however, its value in PEST turned out to be 15.6183. Similarly, the parameter “percent_impervious_area_treated” manifested the roundoff mismatch in a single testing session with the values of 49.395951 and 49.3960 in SWMM and PEST, respectively. The loss of precision led to a degrading performance of SWMM simulation. A root cause was the six-digit rounding operation implemented in PEST as shown in the following. Recognizing mismatches like roundoff errors could help discover new requirements for software integration. We are currently adding a new feature to SWMM2PEST to handle this numerical operation mismatch.

```
WRITE (ISNS, 299) APAR(IPP), PARGNME (IPARGP(IPP)), PVAL(IPP), SC (IES)/NOSBNZERO
FORMAT(3X, A12,T20,A12,T35,1PG13.6, T52,1PG13.6)
```

**DISCUSSION**

In this section, we present the code coverage of our testing, discuss a couple of limitations of our work, and share our lessons learned in applying exploratory MT to the integration of SWMM, SWMM2PEST, and PEST.

The code coverage of SWMM2PEST is 60% assessed by using Python’s library named coverage. However, SWMM and PEST are called in SWMM2PEST in the form of exe, which makes it
difficult to calculate the code coverage of these two applications.

One of the limitations is that the observation file in Figure 2 that is required in this research can be difficult to acquire. This is because our testing-result pattern is derived from the analysis of the change of $R^2$ values in multiple iterations of MT. The observation data from actual field measurements must be used together with the calibration data to obtain the $R^2$ value. Our exploratory MT, thus, relies on the observation file provided by the scientists or observers to conduct the entire testing process.

Another limitation is that our MT is focused on analyzing the testing-result pattern, which is the trend of $R^2$ values after the test iterations. To obtain a more in-depth analysis, domain knowledge is required, such as whether the $R^2$ value is in line with expectations, and whether the change in $R^2$ value is due to one or several parameter changes.

We shared three most important lessons learned from exploratory MT, which are as follows.

- **Multiple datasets.** As given in Table 1, it is important to have more than one dataset to cross validate the detected issues. In Table 1, the string-truncate bug, the roundoff mismatch, and the subcatchment-area constraint were identified while testing FS10. These issues were confirmed under Villanova though the total number of tests and the testing-result distributions were different between the two datasets.

- **Less is more.** When diagnosing the FS10 testing results (cf. Table 1), we first investigated the “Unchanged” pattern. Our main rationale was tractability, i.e., the 13 (4.7%) test cases would allow us to thoroughly investigate the parameter-test relations as given in Table 2. Our experience here is in line with “smaller modules inspected first” strategy in defect detection.

- **Exploring root causes.** Although FS10 helped surface several issues, one constraint was uncovered by Villanova as given in Table 1. On one hand, this echoes the importance of having multiple datasets in MT. On the other hand, the diagnoses of Villanova’s “Decreased” category traced down to certain parameters like “percent_impervious” (P18) that had “0” as their values throughout the simulations. Initializing with non-zero values thus constrained how the simulations should be performed. Not only was this constraint confirmed with FS10, but it became another root cause of “Decreased” tests, complementing with the discovered roundoff mismatch.

### CONCLUDING REMARKS

Testing is one of the cornerstones of modern software engineering. Software development of scientific simulations faces the challenge of lacking proper oracles in testing the code. MT alleviates this challenge by shifting software testing from one input at a time to multiple ones whose outputs shall follow certain relationships. Previous work advocates the definition of some desired relations before writing tests.

In this paper, we propose a new MT approach by exploring multiple and competing relations. This metamorphic relation is based on the comparison of $R^2$ values resulted from multiple output-as-input iterations. When relating the $R^2$ value from the first iteration and that from the second iteration, for example, we keep all the options open (Resultless, Decreased, Unchanged, and Increased). The novelties of the proposed exploratory MT framework lie in our experience that, in scientific simulations, it is relatively easy and straightforward to define the tests, and more precisely, to define the multiple rounds of tests for continuous simulations. Rather than focusing only on the desired metamorphic relation, we show in our work that investigating a spectrum of testing-result patterns is valuable.

The value of our exploratory MT framework is demonstrated by the detection of faults such as bugs and mismatches. Our future work shall enhance the fault detection capabilities. We aim at overcoming one of the limitations of this research by introducing domain knowledge (e.g., merging different viewpoints) to acquire in-depth understating on the sources of faults based on the $R^2$ value changes. We are also interested in broadening exploratory MT to test non-functional aspects of scientific software, such as performance, security, and fault tolerance.
DISCLAIMER

The U.S. Environmental Protection Agency, through its Office of Research and Development, partially funded and collaborated in, the research described herein. It has been subjected to the Agency’s peer and administrative review and has been approved for external publication. Any opinions expressed in this paper are those of the author(s) and do not necessarily reflect the views of the Agency, therefore, no official endorsement should be inferred. Any mention of trade names or commercial products does not constitute endorsement or recommendation for use.

ACKNOWLEDGMENTS

This work is supported in part by the U.S. National Science Foundation (Award CCF 1350487) and the U.S. Environmental Protection Agency via the Oak Ridge Institute Student Employment Agreement.

REFERENCES

Xuanyi Lin is Graduate Student with the Department of Electrical Engineering and Computer Science, College of Engineering and Applied Science, University of Cincinnati, Cincinnati, OH, USA. His research interests include scientific software development, automated software testing, 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 the Associate Director of Science for the Water Systems Division for the EPA Office of Research and Development and National Risk Management Research Laboratory. She is the technical lead for the Storm Water Management Model. She received the Ph.D degree from the Department of Soil, Water, and Environmental Science, University of Arizona. Contact her at simon.michelle@epa.gov.

Nan Niu is an Associate Professor in Department of Electrical Engineering and Computer Science, College of Engineering and Applied Science at University of Cincinnati, OH, USA. His research interests include software requirements engineering, scientific software development, and human-centric computing. He received the Ph.D degree in computer science from the University of Toronto in 2009. He is a member of ACM and ASEE, and a Senior Member of IEEE. Contact him at nan.niu@uc.edu.