Learning I/O Variables from Scientific Software's User Manuals

Zedong Peng[†], Xuanyi Lin[‡], Sreelekhaa Nagamalli Santhoshkumar[†], Nan Niu[†], and Upulee Kanewala[⋆]

[†]University of Cincinnati, Cincinnati, OH, USA 45221
[‡]Oracle America, Inc., Redwood Shores, CA, USA 94065
[⋆]University of North Florida, Jacksonville, FL, USA 32224
{pengzd,linx7,nagamasa}@mail.uc.edu, nan.niu@uc.edu, upulee.kanewala@unf.edu

Abstract. Scientific software often involves many input and output variables. Identifying these variables is important for such software engineering tasks as metamorphic testing. To reduce the manual work, we report in this paper our investigation of machine learning algorithms in classifying variables from software's user manuals. We identify thirteen natural-language features, and use them to develop a multi-layer solution where the first layer distinguishes variables from non-variables and the second layer classifies the variables into input and output types. Our experimental results on three scientific software systems show that random forest and feedforward neural network can be used to best implement the first layer and second layer respectively.

Keywords: Scientific software, user manual, software documentation, classification, machine learning.

1 Introduction

The behavior of scientific software, such as a neutron transport simulation [9] and a seismic wave propagation [16], is typically a function of a large input space with hundreds of variables. Similarly, the output space is often large with many variables to be computed. Rather than requiring stimuli from the users in an interactive mode, scientific software executes once the input values are entered as a batch [46].

Recognizing the input/output (I/O) variables is a prerequisite for software engineering tasks like metamorphic testing [14]. In metamorphic testing, a change in some input variable(s) is anticipated to lead to a predictable effect on certain output variable(s). For instance, a metamorphic test case for the Storm Water Management Model (SWMM) [43] is: the surface water runoff is expected to decrease when bioretention cell is added [20], where "bioretention cell" is an input variable and "runoff" is an output variable.

In the previous study, we identified I/O variables manually from the user manual of SWMM [31]; however, this manual work was tedious and labor-intensive. The total cost of consolidating 807 input and 164 output variables was

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

approximately 40 human-hours; yet we found that the I/O classification was not exclusive, e.g., 53 variables introduced in the user manual of SWMM [34] were both input and output variables.

To automate the I/O variable identification from a scientific software system's user manual, we investigate the use of machine learning (ML) in this paper. Specifically, we build on the experience of our manual work to codify the natural-language features that are indicative of the variable types (I, O, both I and O). We then develop a two-layer ML approach by first distinguishing variables from non-variables, followed by the classification of the variable types. We report the experimental results of applying our ML solution to the user manuals of three different scientific software systems, and further reveal the most influential ML features for classifying I/O variables.

It is worth noting that user manuals represent only one source of I/O variable identification and yet a complementary source to source code. Not only are user manuals amenable to natural language processing techniques, but they tend to be relatively stable in the face of frequent code changes. For instance, since the user manual of SWMM v 5.1 was written in September 2015 [34], the code release has been updated five times from v 5.1.011 to v 5.1.015 [48]. While there exists an inherent tradeoff between code's transitory nature and user manual's stable status, our objective is to automatically process documentation in support of scientific software engineering tasks such as metamorphic testing.

The main contribution of this paper work is the two-layer ML approach along with the natural-language features employed in these layers. The ML solution could reduce the cost of identifying I/O variables from scientific software's user manuals and other types of documentation, e.g., user manuals [18] and release notes [19]. In what follows, we provide background information in Section 2. Section 3 presents our two-layer ML solution together with the exploited features, Section 4 describes the experimental results, and finally, Section 5 draws some concluding remarks and outlines future work.

2 Background

Documentation provides valuable sources for software developers. Aghajani *et al.* [2] showed that user manual was found helpful for most of the 15 surveyed software engineering tasks (especially the operations and maintenance tasks) by at least one fifth of the 68 industrial practitioners. In scientific software development, a user manual describes the scope, purpose, and how-to of the software. The survey by Nguyen-Hoan *et al.* [23] showed that 70% of scientific software developers commonly produced user manuals, and Pawlik *et al.* [27] confirmed that it is most likely that user manuals will be prepared when scientific software is expected to be used outside a particular, typically limited, group.

Using software documentation to support metamorphic testing was first proposed by Chen *et al.* [8] in order to form I/O relations by systematically enumerating a pair of distinct complete test frames of the input domain. Zhou *et al.* [50] relied on the online specifications of the search engines to construct five

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

I/O relations for metamorphic testing, whereas Lin *et al.* [17, 18] exploited user forums to find such relations. No matter which documentation source is used, the identification of I/O variables themselves remains manual in these contemporary approaches.

Despite the lack of automated support for variable classification in the context of metamorphic testing, researchers have shown promise in building ML solutions to analyze software requirements written in natural languages: determining which statements actually represent requirements [1], separating functional and nonfunctional requirements [10], recognizing the temporal requirements that express the time-related system behaviors and properties [7], tracing requirements [47], just to name a few. Motivated by these ML solutions, we next present an automated approach to classifying variables from scientific software's user manuals.

3 Classifying Variables via Machine Learning

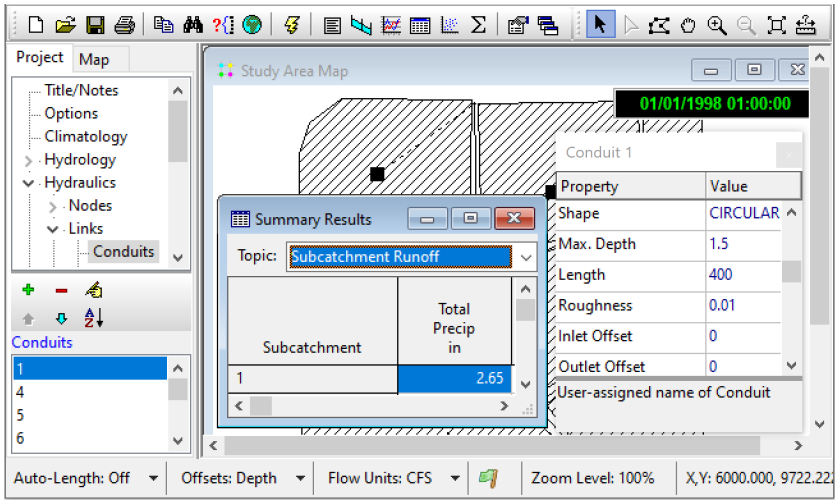
We gained experience in manually identifying variables from SWMM's user manual [31]. SWMM, created and maintained by the U.S. Environmental Protection Agency (EPA), is a dynamic rainfall-runoff simulation model that computes runoff quantity and quality from primarily urban areas. Figure 1-a shows SWMM's integrated environment for defining study area input data, running hydrologic, hydraulic, and water quality simulations, and viewing the results. Although graphical user interfaces like Figure 1-a help visualize some I/O information, user manual more completely introduces the I/O variables of the scientific software.

Figure 1-b shows an excerpt of SWMM user manual [34]. This excerpt is also annotated with the results from our variable classifier. We notice that variables are nouns or noun phrases (NPs); however, not every noun or NP is a variable. Indeed, non-variable greatly outnumbers variable, and this generally holds in scientific software's user manual. For this reason, we build a multi-layer variable classifier, the architecture of which is shown in Figure 2. Compared to a single, holistic classifier, the multi-layer architecture enables different ML algorithms to tackle problems at different granularity levels, thereby better handling imbalanced data and becoming more scalable for hierarchical training and classification [5].

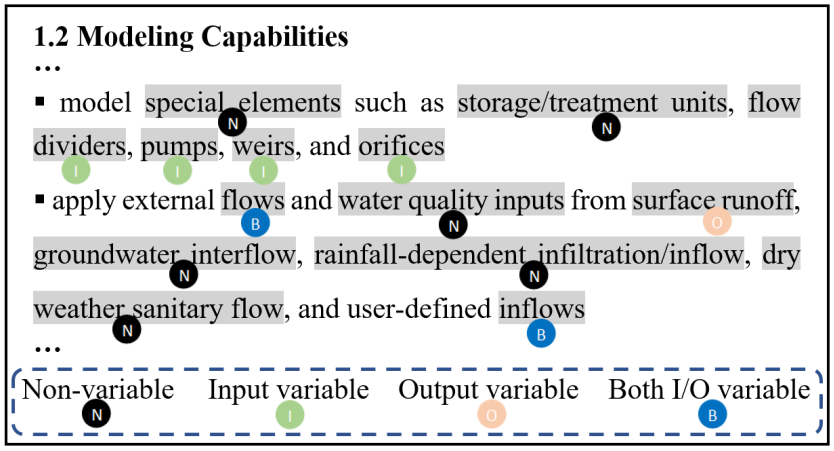
As shown in Figure 2, the preprocessing involves a few steps. If the user manual is a PDF file, we use the ExtractPDF tool [37] to convert it into a plain text file. We then apply NLTK in Python [26] to implement the tokenizer that breaks the text file's content into tokens (e.g., words, numbers, and symbols). NLTK is further used to split the tokens into sentences based on conventional delimiters (e.g., period and line break), and to assign the part-of-speech (PoS) tags (e.g., noun and verb) to each token. Our final preprocessing step uses the TextBlob Python library [39] to extract NPs.

The nouns and NPs are candidates for variable classification, and our classifier shown in Figure 2 works at the *sentence* level. Supervised learning is used

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42



(a) Storm Water Management Model (SWMM) [43] performs single event or long-term runoff simulations



(b) SWMM user manual [34] annotated with the I/O variables

Fig. 1. SWMM screenshot and its user manual excerpt.

at both layers: the first layer distinguishes which nouns or NPs are variables and which ones are not, and the second layer further predicts if a variable is the scientific software's input, output, or both input and output. The both category is of particular interest because the same variable can be one simulation's input and another's output. In SWMM, for example, "pollutant washoff" may be a key input variable for a wastewater treatment engineer, but at the same time can be an important output variable that a city manager researching low impact development (LID) pays attention to.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

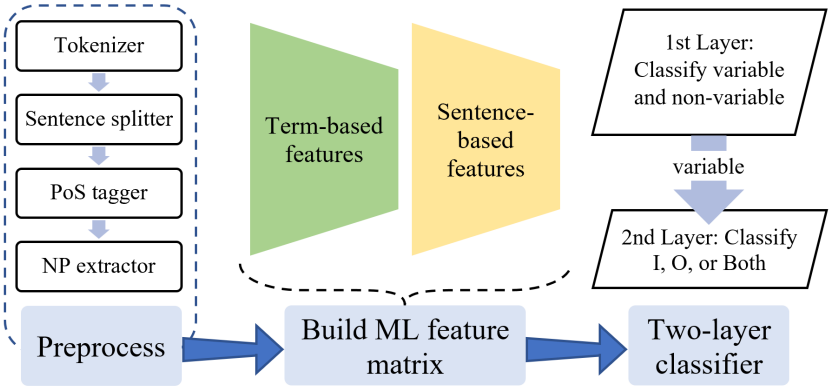


Fig. 2. Two-layer variable classifier: first layer distinguishes variables from non-variables, and second layer decides if a variable is input, output, or both.

As noun/NP and sentence represent the unit to classify and the context of classification respectively, our feature design is driven by the terms and the sentences. We group the features into term-based and sentence-based in Tables 1 and 2 respectively. These tables also provide our intuition behind each feature, leading up to the feature's inclusion and exclusion of the first layer (variable or non-variable) and the second layer (I, O, or both). We orient our features

Table 1. Term-Based Features for Variable Learning (• means the feature is used in a specific layer)

ID	Name	Description	First Layer	Second Layer
f1	hasNumerical-StatisticalTerm	A noun or NP containing a statistical term or symbol (e.g., "percent", "%", "ratio", "max", etc.) could imply an I/O variable.	•	
f2	hasInitTerm	A noun or NP containing "initial" or "init" could indicate an I variable.	•	•
f3	hasSummary-Term	A noun or NP containing statistical terms (e.g., "final", "average", and "total") could indicate an O variable.	•	•
f4	hasInputIn-Heading	If the heading of chapter, section, etc. to which the noun or NP belongs contains "input", the noun or NP is likely to be an I variable.	•	•
f5	hasOutputIn-Heading	If the heading of chapter, section, etc. to which the noun or NP belongs contains "output", "result", or "summary", the noun or NP is likely to be an O variable.	•	•
f6	hasHighFre-quency	If a noun or NP is part of the top 1% of most frequently occurred terms in the user manual, it is likely a domain concept of the software.	•	

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

Table 2. Sentence-Based Features for Variable Learning (• means the feature is used in a specific layer)

ID	Name	Description	First Layer	Second Layer
f7	singleNoun	If a non-heading sentence contains only one noun or NP, it is likely a variable.	•	
f8	nounWith-NearUnit	If a sentence contains a noun or NP and a unit (e.g., inch or inches) within the 5-term neighborhood, the sentence likely introduces the noun or NP variable with unit.	•	
f9	beginsWith-Noun	If a non-heading sentence begins with a noun or NP, then the sentence likely provides details about the noun or NP details about the noun/NP variable.	•	
f10	hasNoun-Subject	If a non-heading sentence has a noun or NP subject, the sentence likely explains the noun or NP variable or describes the variable's operations.	•	
f11	hasBEVerb-WithNoun	If a sentence has the linking verb (e.g., "is") followed by a noun or NP, the sentence likely provides definitional information about the noun or NP variable.	•	
f12	nounWith-NearNum	If a sentence has a noun or NP and a value within the 5-term neighborhood, the sentence likely introduces the default value of the noun or NP input variable.	•	•
f13	nounSubject-WithNum	If a non-heading sentence has a noun or NP subject followed by a second sentence containing a numerical value, then it is likely the first sentence introduces the noun or NP input variable and the second sentence gives the default value.		•

around syntactic and semantic properties that are meaningful irrespectively of the exact characteristics of individual user manual. Of particular note are f8 and f12 that associate a noun/NP with a measuring unit or a numerical value within the 5-term neighborhood. It is worth bearing in mind that such a distance of five terms is not a parameter but a property of the English language [21].

Classification is carried out according to the feature matrix at either layer, and the output is the identification of scientific software's I/O variables. Examining which ML algorithms best implement our multi-layer variable classifier is addressed next in the evaluation section.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

4 Experimental Evaluation

4.1 Research Questions

We set out to answer three research questions.

RQ_1: Which ML algorithms yield the most accurate I/O variable classifications?

As ML classification algorithms can be broadly grouped into mathematical, hierarchical, and layered categories [38], we compare the classification accuracy measured by recall, precision, and F-measure of five ML algorithms: logistic regression and support vector machine (mathematical category), decision tree and random forest (hierarchical category), and feedforward neural network (layered category).

RQ_2: Is the two-layer approach better than the single-layer counterpart for classifying I/O variables?

We compare multi-layer with single-layer in I/O variable classifications. We implement the ML algorithms with scikit-learn in Python [36] and tune these algorithms to maximize classification accuracy.

RQ_3: What are the most influential ML features for classifying I/O variables?

We assess the importance of the features in Tables 1 and 2 via information gain [49], which measures how efficient a given feature is in our multi-layer variable classifier. The higher the information gain, the more discriminative power a feature has. Our analysis here also offers insights into the performance of a baseline classifier relying on the user manual headings (i.e., f4 and f5 of Table 1). These two features make use of terms like "input" and "output" directly, and hence provide straightforward classification cues.

4.2 Subject Systems

Our experiments are conducted on three subject systems. We choose these scientific software systems due to their sustained developments, and open accesses to their user manuals. Another reason was our familiarity to the water domain [6, 12, 22] and the EPA SWMM system [30–32]. Table 3 lists the characteristics of the three subject systems.

Table 3. Subject System Characteristics

Subject System	Basic Information			User Manual		Answer Set Size			
	years of existence	written in	# of LoC	source	# of pages	non-variable	input (I) variable	output (O) variable	both I and O
SWMM [43]	49	C	46,291	[34]	353	13,665	807	164	53
SWAT [40]	30	Fortran	84,296	[3]	650	24,894	1,006	454	78
MODFLOW [44]	36	Fortran	61,945	[41]	188	12,602	601	172	47

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

Besides SWMM simulating water runoff quantity and quality in primarily urban areas [43], the Soil & Water Assessment Tool (SWAT) is a small watershed to river basin-scale model widely used in regional management (e.g., simulating soil erosion prevention and non-point source pollution control) [40]. In contrast, the original scope of the Modular Hydrologic Model (MODFLOW) is solely limited to groundwater-flow simulation, though the simulation capabilities have since been expanded to include groundwater/surface-water coupling, solute transport, land subsidence, and other nonfunctional aspects [25, 44].

Although written in different programming languages and developed by different teams [4], the size of all three systems can be considered to be medium (between 1,000 and 100,000 LoC) according to Sanders and Kelly's study of scientific software [35]. Due to the thousands of users worldwide, each software maintains authoritative user manual where I/O variables are comprehensively documented [3, 34, 41].

As our multi-layer variable classifier (cf. Figure 2) uses supervised learning, labeled data are required for training. Three researchers therefore spent about 100 human-hours in total constructing the subject systems' answer sets manually. They first individually and independently labeled a randomly chosen chapter from each user manual, resulting in a 0.87 Fleiss' κ and hence achieving a strong inter-rater agreement [11]. The discrepancies were resolved in a joint meeting, and a protocol was agreed upon. The researchers then applied the protocol to label the remaining user manuals separately. From Table 3, we can see that each system contains hundreds of variables, confirming scientific software's large input and output spaces [46].

4.3 Results and Analysis

To answer $\mathbf{RQ}_1$, we use the manually labeled data to train the ML algorithms in a ten-fold cross validation procedure. The average recall, precision, and F-measure across the ten-fold validation are reported in Table 4. As F-measure is the harmonic mean of recall and precision, Table 4 highlights in **bold** the best accuracy. Among the five ML algorithms considered, random forest best implements the first layer of our variable classifier where a binary decision (variable or non-variable) is made. This result is in line with Ibarguren *et al.*'s experience that random forest almost always has lower classification error in handling uneven data sets [13]. Feedforward neural network, as shown in Table 4-b, is advantageous to implementing our second layer where variables are further classified into I, O, or both. Logistic regression can also be an option since it achieves the highest F-measure in SWAT. Table 4 demonstrates the modularity of our multi-layer variable classifier where different ML algorithms can be deployed for different classification tasks.

To answer $\mathbf{RQ}_2$, we compare our solution to a single-layer alternative. In particular, we train the five ML algorithms with all the 13 features of Tables 1 and 2 to classify four variable types at once. Table 5 shows that our multi-layer classifier outperforms the best single-layer solution where the average measure across ten-fold validation are presented. The multi-layer solution achieves better

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

Table 4. ML Algorithm Selection Results ($\mathbf{RQ}_1$)

(a) First layer distinguishing variables from non-variables

Accuracy of ML		SWMM	SWAT	MODFLOW
logistic regression	recall	0.755	0.767	0.693
	precision	0.778	0.769	0.717
	F-measure	0.765	0.768	0.704
support vector machine	recall	0.762	0.820	0.678
	precision	0.827	0.820	0.802
	F-measure	0.784	0.820	0.713
decision tree	recall	0.763	0.821	0.680
	precision	0.828	0.821	0.804
	F-measure	0.785	0.821	0.715
random forest	recall	0.812	0.847	0.750
	precision	0.828	0.852	0.804
	F-measure	**0.815**	**0.849**	**0.768**
feedforward neural network	recall	0.766	0.783	0.640
	precision	0.778	0.804	0.816
	F-measure	0.771	0.793	0.673

(b) Second layer classifying I, O, or both-I-and-O variables

Accuracy of ML		SWMM	SWAT	MODFLOW
logistic regression	recall	0.582	0.826	0.643
	precision	0.819	0.729	0.776
	F-measure	0.620	**0.763**	0.659
support vector machine	recall	0.573	0.815	0.643
	precision	0.814	0.727	0.777
	F-measure	0.614	0.757	0.660
decision tree	recall	0.582	0.810	0.643
	precision	0.819	0.724	0.776
	F-measure	0.620	0.753	0.659
random forest	recall	0.620	0.815	0.643
	precision	0.786	0.727	0.776
	F-measure	0.661	0.757	0.659
feedforward neural network	recall	0.705	0.815	0.696
	precision	0.819	0.723	0.780
	F-measure	**0.745**	0.756	**0.702**

performances especially in classifying I and O variables of the scientific software. We therefore suggest random forest and feedforward neural network to be the first layer and second layer ML algorithm respectively for classifying variables from scientific software's user manual.

Our answers to $\mathbf{RQ}_3$ are listed in Table 6. The singleNoun feature (f7) of Table 2 has the highest information gain and hence exhibits the most discriminative power in distinguishing variables from non-variables. Other important features in the first layer include keywords in headings (f4 and f5) and neighboring terms or verbs in a sentence (f8, f11, and f12). Surprisingly, f1—used only

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

Table 5. Comparing F-Measures ($\mathbf{RQ}_2$)

Accuracy of ML (F-measure)		non-variable	input (I) variable	output (O) variable	both I and O
SWMM	multi-layer	0.895	0.811	0.753	0.672
	single-layer (random forest)	0.900	0.396	0.499	0.652
SWAT	multi-layer	0.940	0.916	0.741	0.632
	single-layer (decision tree)	0.940	0.570	0.488	0.586
MOD-FLOW	multi-layer	0.917	0.713	0.731	0.661
	single-layer (random forest)	0.916	0.132	0.190	0.628

Table 6. Feature Importance ($\mathbf{RQ}_3$)

(a) Top-3 features ranked by the information gain scores

first layer			second layer		
SWMM	SWAT	MODFLOW	SWMM	SWAT	MODFLOW
f7	f7	f7	f4	f5	f5
f12	f4	f4	f13	f12	f4
f8	f11	f5	f12	f4	f12

(b) Information gain ranking of baseline features: f4 and f5

	first layer			second layer		
	SWMM	SWAT	MODFLOW	SWMM	SWAT	MODFLOW
f4	6th	2nd	2nd	1st	1st	2nd
f5	7th	4th	3rd	4th	3rd	1st

in the first layer—is not among the most discriminative features, and neither is f13 in the second layer for SWAT and MODFLOW. As shown in Table 6-b, keywords-in-the-heading as baseline features work well in the second layer, but other features play complementary, and often more dominant, roles in ML-based variable classification. From Table 6, we acknowledge the effectiveness of the simple baseline features, and also emphasize the important role played by sentence-based features such as f7 and f12.

4.4 Threats to Validity

We discuss some of the most important factors that must be considered when interpreting our experimental results. A threat to internal validity concerns the quality of the scientific software's user manual. Like other approaches based on software documentation [2], our ML-based I/O classification could be hindered if mistakes exist in the user manual. For this reason, we share our manual-labeling results in an institution's digital preservation repository [29] to facilitate reproducibility.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

A factor affecting our study's external validity is that our results may not generalize to other scientific software systems from SWMM, SWAT, and MODFLOW. In fact, the three systems that we studied are within the water domain and developed by government agencies. Producing user-oriented documentation has become a requirement for scientists-developers mandated by organizations like the U.S. EPA [42] and the U.S. Geological Survey (USGS) [45]. Therefore, it is interesting to extend the variable classification via ML to other scientific software systems.

5 Conclusions

In this paper, we have presented an automatic approach that classifies I/O variables from the user manual. Our evaluations on SWMM, SWAT, and MODFLOW show the accuracy of ML-based I/O classification, and support the hierarchy and modularity of a multi-layer ML solution. Specifically, we recommend random forest for distinguishing variables from non-variables, and feedforward neural network for further classifying the variables into input and output types.

Our future work includes expanding the experimentation to other scientific software systems, building more efficient ML solutions by using a subset of the most important features, releasing the solutions as cloud-based tools [33], integrating the classified variables into metamorphic testing [28], and performing theoretical replications [15, 24]. Our goal is to better support scientists in improving the effectiveness and efficiency of their software development and maintenance activities.

Acknowledgments. *We thank the EPA SWMM team, especially Michelle Simon, for the research collaborations. We also thank the anonymous reviewers for their constructive comments.*

References

1. S. Abualhaija, C. Arora, M. Sabetzadeh, L. C. Briand, and E. Vaz. A machine learning-based approach for demarcating requirements in textual specifications. In *International Requirements Engineering Conference*, pages 51–62, 2019.
2. E. Aghajani, C. Nagy, M. Linares-Vásquez, L. Moreno, G. Bavota, M. Lanza, and D. C. Shepherd. Software documentation: the practitioners' perspective. In *International Conference on Software Engineering*, pages 590–601, 2020.
3. J. G. Arnold, J. R. Kiniry, R. Srinivasan, J. R. Williams, E. B. Haney, and S. L. Neitsch. Soil & Water Assessment Tool (SWAT) Input/Output Documentation (Version 2012). https://swat.tamu.edu/media/69296/swat-io-documentation-2012.pdf Last accessed: April 9, 2022.
4. T. Bhowmik, N. Niu, W. Wang, J-R. C. Cheng, L. Li, and X. Cao. Optimal group size for software change tasks: a social information foraging perspective. *IEEE Transactions on Cybernetics*, 46(8): 1784–1795, 2016.
5. A. A. Burungale and D. A. Zende. Survey of large-scale hierarchical classification. *International Journal of Engineering Research and General Science*, 2(6): 917–921, 2014.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

6. H. Challa, N. Niu, and R. Johnson. Faulty requirements made valuable: on the role of data quality in deep learning. In *International Workshop on Artificial Intelligence and Requirements Engineering*, pages 61–69, 2020.
7. A. Chattopadhyay, N. Niu, Z. Peng, and J. Zhang. Semantic frames for classifying temporal requirements: an exploratory study. In *Workshop on Natural Language Processing for Requirements Engineering*, 2021.
8. T. Y. Chen. P.-L. Poon, and X. Xie. METRIC: METamorphic Relation Identification based on the Category-choice framework. *Journal of Systems and Software*, 116: 177–190, 2016.
9. K. Clarno, V. de Almeida, E. d'Azevedo, C. de Oliveira, and S. Hamilton. GNES-R: global nuclear energy simulator for research task 1: high-fidelity neutron transport. In *American Nuclear Society Topical Meeting on Reactor Physics: Advances in Nuclear Analysis and Simulation*, 2006.
10. F. Dalpiaz, D. Dell'Anna, F. B. Aydemir, and S. Çevikol. Requirements classification with interpretable machine learning and dependency parsing. In *International Requirements Engineering Conference*, pages 142–152, 2019.
11. J. L. Fleiss and J. Cohen. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. *Educational and Psychological Measurement*, 33(3): 613–619, 1973.
12. H. Gudaparthi, R. Johnson, H. Challa, and N. Niu. Deep learning for smart sewer systems: assessing nonfunctional requirements. In *International Conference on Software Engineering: Software Engineering in Society*, pages 35–38, 2020.
13. I. Ibarguren, J. M. Pérez, J. Muguerza, I. Gurrutxaga, and O. Arbelaitz. Coverage-based resampling: building robust consolidated decision trees. *Knowledge Based Systems*, 79: 51–67, 2015.
14. U. Kanewala and T. Y. Chen. Metamorphic testing: a simple yet effective approach for testing scientific software. *Computing in Science & Engineering*, 21(1): 66–72, 2019.
15. C. Khatwani, X. Jin, N. Niu, A. Koshoffer, L. Newman, and J. Savolainen. Advancing viewpoint merging in requirements engineering: a theoretical replication and explanatory study. *Requirements Engineering*, 22(3): 317–338, 2017.
16. Y. Li, E. Guzman, K. Tsiamoura, F. Schneider, and B. Bruegge. Automated requirements extraction for scientific software. In *International Conference on Computational Science*, pages 582–591, 2015.
17. X. Lin, Z. Peng, N. Niu, W. Wang, and H. Liu. Finding metamorphic relations for scientific software. In *International Conference on Software Engineering (Companion Volume)*, pages 254–255, 2021.
18. X. Lin, M. Simon, Z. Peng, and N. Niu. Discovering metamorphic relations for scientific software from user forums. *Computing in Science and Engineering*, 23(2): 65–72, 2021.
19. X. Lin, M. Simon, and N. Niu. Releasing scientific software in GitHub: a case study on SWMM2PEST. In *International Workshop on Software Engineering for Science*, pages 47–50, 2019.
20. X. Lin, M. Simon, and N. Niu. Scientific software testing goes serverless: creating and invoking metamorphic functions. *IEEE Software*, 38(1): 61–67, 2021.
21. Y. S. Maarek, D. M. Berry, and G. E. Kaiser. An information retrieval approach for automatically constructing software libraries. *IEEE Transactions on Software Engineering*, 17(8): 800–813, 1991.
22. N. Maltbie, N. Niu, M. Van Doren, and R. Johnson. XAI tools in the public sector: a case study on predicting combined sewer overflows. In *ACM Joint European*

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

Software Engineering Conference and Symposium on the Foundations of Software Engineering, pages 1032–1044, 2021.
23. L. Nguyen-Hoan, S. Flint, and R. Sankaranarayana. A survey of scientific software development. In *International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2010.
24. N. Niu, A. Koshoffer, L. Newman, C. Khatwani, C. Samarasinghe, and J. Savolainen. Advancing repeated research in requirements engineering: a theoretical replication of viewpoint merging. In *International Requirements Engineering Conference*, pages 186–195, 2016.
25. N. Niu, Y. Yu, B. González-Baixauli, N. Ernst, J. Leite, J. Mylopoulos. Aspects across software life cycle: a goal-driven approach. *Transactions on Aspect-Oriented Software Development*, VI: 83–110, 2009.
26. NLTK. Natural Language Toolkit. https://www.nltk.org Last accessed: April 9, 2022.
27. A. Pawlik, J. Segal, and M. Petre. Documentation practices in scientific software development. In *International Workshop on Cooperative and Human Aspects of Software Engineering*, pages 113–119, 2012.
28. Z. Peng, U. Kanewala, and N. Niu. Contextual understanding and improvement of metamorphic testing in scientific software development. In *International Symposium on Empirical Software Engineering and Measurement*, pages 28:1–28:6, 2021.
29. Z. Peng, X. Lin, and N. Niu. Data of Classifying I/O Variables via Machine Learning. https://doi.org/10.7945/85j1-qf68 Last accessed: April 9, 2022.
30. Z. Peng, X. Lin, and N. Niu. Unit tests of scientific software: a study on SWMM. In *International Conference on Computational Science*, pages 413–427, 2020.
31. Z. Peng, X. Lin, N. Niu, and O. I. Abdul-Aziz. I/O associations in scientific software: a study of SWMM. In *International Conference on Computational Science*, pages 375–389, 2021.
32. Z. Peng, X. Lin, M. Simon, and N. Niu. Unit and regression tests of scientific software: a study on SWMM. *Journal of Computational Science*, 53: 101347:1–101347:13, 2021.
33. Z. Peng and N. Niu. Co-AI: a Colab-based tool for abstraction identification. In *International Requirements Engineering Conference*, pages 420–421, 2021.
34. L. A. Rossman. Storm Water Management Model User's Manual Version 5.1. https://www.epa.gov/water-research/storm-water-management-model-swmm-version-51-users-manual Last accessed: April 9, 2022.
35. R. Sanders and D. Kelly. Dealing with risk in scientific software development. *IEEE Software*, 25(4): 21–28, 2008.
36. scikit-learn. Machine Learning in Python. https://scikit-learn.org/stable/ Last accessed: April 9, 2022.
37. Spikerog SAS. ExtractPDF. https://www.extractpdf.com Last accessed: April 9, 2022.
38. S. Suthaharan. Machine Learning Models and Algorithms for Big Data Classification. *Springer*, 2016.
39. TextBlob. Simplified Text Processing. https://textblob.readthedocs.io Last accessed: April 9, 2022.
40. United States Department of Agriculture. Soil & Water Assessment Tool (SWAT). https://data.nal.usda.gov/dataset/swat-soil-and-water-assessment-tool Last accessed: April 9, 2022.
41. United States Department of the Interior & United States Geological Survey. Modular Hydrologic Model (MODFLOW) Description of Input and Output (Version 6.0.0). https://water.usgs.gov/ogw/modflow/mf6io.pdf Last accessed: April 9, 2022.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42

42. United States Environmental Protection Agency. Agency-wide Quality System Documents. https://www.epa.gov/quality/agency-wide-quality-system-documents Last accessed: April 9, 2022.
43. United States Environmental Protection Agency. Storm Water Management Model (SWMM). https://www.epa.gov/water-research/storm-water-management-model-swmm Last accessed: April 9, 2022.
44. United States Geological Survey. Modular Hydrologic Model (MODFLOW). https://www.usgs.gov/software/software-modflow Last accessed: April 9, 2022.
45. United States Geological Survey. Review and Approval of Scientific Software for Release (IM OSQI 2019-01). https://www.usgs.gov/about/organization/science-support/survey-manual/im-osqi-2019-01-review-and-approval-scientific Last accessed: April 9, 2022.
46. S. A. Vilkomir, W. T. Swain, J. H. Poore, and K. T. Clarno. Modeling input space for testing scientific computational software: a case study. In *International Conference on Computational Science*, pages 291–300, 2008.
47. W. Wang, N. Niu, H. Liu, and Z. Niu. Enhancing automated requirements traceability by resolving polysemy. In *International Requirements Engineering Conference*, pages 40–51, 2018.
48. Wikipedia. Storm Water Management Model. https://en.wikipedia.org/wiki/Storm_Water_Management_Model Last accessed: April 9, 2022.
49. I. H. Witten, E. Frank, and M. A. Hall. Data Mining: Practical Machine Learning Tools and Techniques. *Morgan Kaufmann*, 2016.
50. Z. Zhou, S. Xiang, and T. Y. Chen. Metamorphic testing for software quality assessment: a study of search engines. *IEEE Transactions on Software Engineering*, 42(3): 264–284, 2016.

ICCS Camera Ready Version 2022
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-08760-8_42