Releasing Scientific Software in GitHub: A Case Study on SWMM2PEST

Xuanyi Lin
Department of Electrical Engineering and Computer Science
University of Cincinnati
Cincinnati, OH, USA
linx7@mail.uc.edu

Michelle Simon
EPA Office of Research and Development
National Risk Management Research Laboratory
Water Systems Division
Cincinnati, OH, USA
simon.michelle@epa.gov

Nan Niu
Department of Electrical Engineering and Computer Science
University of Cincinnati
Cincinnati, OH, USA
nan.niu@uc.edu

***Abstract*—Release engineering involves code development, integration, testing, and software delivery. It has been widely applied to deliver high-quality software to users. While release engineering is a widespread practice in the software industry, there have been very few studies on the release engineering pipeline of scientific software. To shorten this gap, we present a case study in this paper to show a GitHub-driven release workflow on SWMM2PEST, a software system automating parameter calibration for the U.S. EPA's Storm Water Management Model (SWMM). Moreover, we analyze software version updates and requirements changes to develop strategies for improving the ongoing releases. The feasibility of improvement strategies is demonstrated by our consecutively released versions of SWMM2PEST. The results offered insights into the continuous release of scientific software.**

Keywords-Software engineering; release engineering; release scientific software; SWMM; SWMM2PEST; GitHub

I. INTRODUCTION

Release engineering is the discipline of code integration, build, test execution, deployment, and delivery of high-quality software releases to users [1]. It is of great importance to link the design and development of the software with the use and maintenance of the finished product. A growing number of cases shows that release engineering is becoming an increasingly common practice in the software development industry. For example, Schermann et al. [2] review continuous experimentation of industrial products by using the release engineering pipeline. One purpose of such experimentation is to collect practical usage feedback from new versions' users that give assistance to guide decisions about future releases. Continuous integration is a widely used development practice in the software development industry. Adopting this practice, the project team often performs integration and deployment in terms of development and release work, shortening the release cycle and improving software quality [3].

There is a large amount of literature indicating that release engineering can help accelerate software development and delivery, increase customer trust and satisfaction, provide faster feedback to developers, enhance the communication of integration systems, and improve software quality [3]. Nevertheless, little attention has been paid to the study of the release engineering pipeline in scientific software.

Drawing from our ongoing research of scientific software testing [4], we propose to release scientific software by using the release engineering pipeline. Specifically, we collect issues and relevant topics from user feedback to better understand user requirements. These issues and topics are then analyzed to improve our release process. The lessons provided by this paper are twofold: (1) We show how to release scientific software in GitHub through a case study; and (2) We illustrate the strategies of improving the quality of scientific software and its releases through detailed analyses.

The remainder of the paper is organized as follows. Section 2 reviews related work and describes the scientific software systems on which our release work is based. Section 3 details our release process. Section 4 presents the release improvements. Section 5 discusses the implications of our work, and finally, Section 6 concludes the paper.

II. BACKGROUND AND RELATED WORK

A. Releasing Scientific Software

Kelly [5] describes scientific software by three characteristics: (1) it is developed to answer a scientific question; (2) it relies on close involvement of expert(s) in its scientific domain; and (3) it provides results to be examined by the person who needs to answer the question. Examples of scientific software are weather prediction models, fluid mechanics models, water quality models, etc. When applying release engineering to scientific software, some processes may be distinct from non-scientific software. For example, at the phase of testing, developers of scientific software frequently face the oracle problem and hence metamorphic testing (MT) can be employed [6].

Scientists often develop their own software for research for a relatively small user set [7], which may not be well documented and debugged for general use of others [5]. Basic software engineering tools (e.g., version control systems) can help ensure that scientific software releases achieve expected levels of quality and reliability.

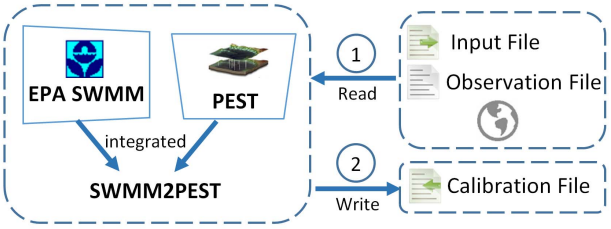


Figure 2. SWMM2PEST integrates SWMM and PEST

With more than 96 million active repositories, GitHub is a popular version control platform and internet hosting service for open-source software projects. Scientific programming projects are fast growing on the GitHub platform. Utilizing the GitHub platform serves as a straightforward and convenient method for publishing scientific software.

B. *SWMM2PEST*

SWMM2PEST, the scientific software that we are releasing, is an integration of the SWMM and PEST scientific programs. 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 [8].

The latest version of SWMM (5.1.013) was released in 2018. Its calculation engine is written in C and contains approximately 45,500 lines of code. The other integrated part of SWMM2PEST is PEST, which is a software package for automated parameter estimation [9]. PEST is written in FORTRAN and its latest release (version 15, in 2018) has around 210,000 lines of code.

Because scientific software, like SWMM, usually has many input parameters and is complicated and time-consuming to manually calibrate their values, SWMM2PEST is built to achieve the automatic calibration for SWMM parameters. Figure 1 depicts the systematic integration of the three software systems SWMM, PEST, and SWMM2PEST. The observation file contains observed results which can be used to calibrate SWMM parameters via PEST. SWMM2PEST will invoke SWMM and PEST for calibrating the parameters, attempting to set their values as close as possible to the observed values.

C. *Metamorphic Testing*

MT is a software testing technique used to alleviate the oracle problem [4]. An oracle refers to the mechanism for checking whether the program under test produces the expected output in different test cases [6]. The oracle may be unavailable or too expensive to obtain in scientific software.

When testing a program that computes the sine function, even if the precise output of sin(1.2) is unknown, the relationship, sin(1.2)=sin(π-1.2), which is called a metamorphic relation (MR), can be verified. MT checks whether the MR holds or not among multiple test cases and indicates the failure if there is any violation detected. Our ongoing work [4, 6] applies MT to test SWMM2PEST.

TABLE I. THE CHANGES IN SWMM2PEST 2.0 COMPARING TO SWMM2PEST 1.0

Added & modified lines (source code)	Deleted lines (source code)	Added & modified UI files	Deleted UI files
1029	696	5	2

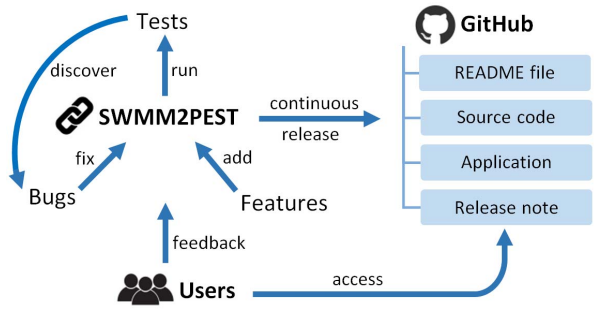


Figure 1. The release process. SWMM2PEST is developed locally and then uploaded to GitHub.

III. RELEASING SWMM2PEST

A. *SWMM2PEST 1.0 to SWMM2PEST 2.0*

Kamble developed SWMM2PEST 1.0 on the Windows platform with Python 3.5.4 and PyQt 5 to automate the calibration process of SWMM by integrating the essential modules of PEST [10]. This first version of SWMM2PEST was found to have many bugs, and the software underwent extensive testing, restructuring, and the addition of new features. As the difference shown in Table I, SWMM2PEST 2.0 has undergone major changes.

Kamble used GitHub mainly as a tool for version management. SWMM2PEST 1.0 [11] thus presents primarily source code in GitHub, with the README file specifying simple instructions and prerequisites. In contrast, SWMM2PEST 2.0 release [12] is a further development and informed by the literature [13], resulting the project repository having more complete content, including source code and a detailed readme file with a demo video to demonstrate the usability as well as various resource files and release notes.

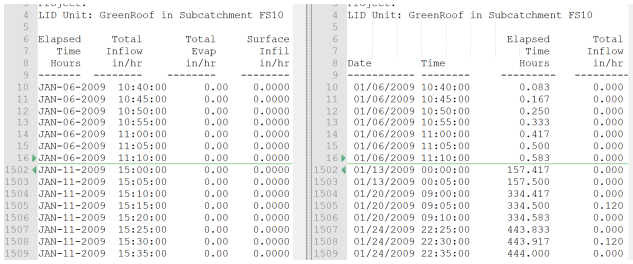
B. *Release Workflow*

Figure 2 depicts our GitHub-driven release process. SWMM2PEST 2.0 was developed on local computers, run through automated tests, bug fixed, restructured, and enhanced with new features. After accomplishing the new version of SWMM2PEST, we released it to GitHub to make it downloadable and available to users.

Users involved into the release process as they may post feedback in GitHub repository including suggestions for the improvement of SWMM2PEST in terms of bugs and new features. We analyze such feedback to evaluate the software's limitations and the feasibility of improvements. New versions of SWMM2PEST are typically released after an extended period of bug fixes, MT, and improvements. For

TABLE II. THE COMPARISON OF VERSIONS OF SWMM AND PEST INTEGRATED IN DIFFERENT VERSIONS OF SWMM2PEST

	SWMM2PEST 1.0	SWMM2PEST 2.0	SWMM2PEST 2.1
SWMM	5.1.10	5.1.10	5.1.13
PEST	13.3	14.2	14.2



```
 4  LID Unit: GreenRoof in Subcatchment FS10
 5
 6  Elapsed     Total       Total     Surface
 7    Time     Inflow        Evap       Infil
 8   Hours      in/hr       in/hr       in/hr
 9  -------  --------    --------    --------
10  JAN-06-2009  10:40:00       0.00      0.0000
11  JAN-06-2009  10:45:00       0.00      0.0000
12  JAN-06-2009  10:50:00       0.00      0.0000
13  JAN-06-2009  10:55:00       0.00      0.0000
14  JAN-06-2009  11:00:00       0.00      0.0000
15  JAN-06-2009  11:05:00       0.00      0.0000
16  JAN-06-2009  11:10:00       0.00      0.0000
1502 JAN-11-2009  15:00:00       0.00      0.0000
1503 JAN-11-2009  15:05:00       0.00      0.0000
1504 JAN-11-2009  15:10:00       0.00      0.0000
1505 JAN-11-2009  15:15:00       0.00      0.0000
1506 JAN-11-2009  15:20:00       0.00      0.0000
1507 JAN-11-2009  15:25:00       0.00      0.0000
1508 JAN-11-2009  15:30:00       0.00      0.0000
1509 JAN-11-2009  15:35:00       0.00      0.0000
```

```
 4  LID Unit: GreenRoof in Subcatchment FS10
 5
 6                                   Elapsed       Total
 7                                      Time      Inflow
 8  Date         Time                   Hours       in/hr
 9  ---------- --------             ---------    --------
10  01/06/2009 10:40:00                 0.083       0.000
11  01/06/2009 10:45:00                 0.167       0.000
12  01/06/2009 10:50:00                 0.250       0.000
13  01/06/2009 10:55:00                 0.333       0.000
14  01/06/2009 11:00:00                 0.417       0.000
15  01/06/2009 11:05:00                 0.500       0.000
16  01/06/2009 11:10:00                 0.583       0.000
1502 01/13/2009 00:00:00               157.417       0.000
1503 01/13/2009 00:05:00               157.500       0.000
1504 01/20/2009 09:00:00               334.417       0.000
1505 01/20/2009 09:05:00               334.500       0.120
1506 01/20/2009 09:10:00               334.583       0.000
1507 01/24/2009 22:25:00               443.833       0.000
1508 01/24/2009 22:30:00               443.917       0.120
1509 01/24/2009 22:35:00               444.000       0.000
```

Figure 3. Comparison of the output of SWMM 5.1.10 (left side) and SWMM 5.1.13 (right side)

example, after users provided feedback on the requirements to integrate into the new version of SWMM, we released SWMM2PEST 2.1 by following this workflow, which fixed numerous bugs and added significant new features.

Our workflow will provide continuous benefits for releasing SWMM2PEST. One of those benefits is that releasing on the GitHub platform promotes communication between developers and users. Once issues are posted on GitHub, developers and users can view and respond directly.

IV. RELEASE IMPROVEMENTS

We propose a two-tiered strategy for future SWMM2PEST releases.

A. *Changes between Versions*

SWMM and PEST are continuously being updated, therefore SWMM2PEST will also need to work with the new versions of SWMM and PEST. Table II illustrates that as the release version of SWMM2PEST is updated, the newer version of SWMM and PEST are utilized. While we made many changes to the user interface from SWMM2PEST 1.0 to 2.0, we did only fine-tuning for PEST. The PEST version changed from 13.3 to 14.2 required minor changes to the code that calls PEST and the PEST configuration file.

Correspondingly, during the version change from SWMM2PEST 2.0 to SWMM2PEST 2.1, SWMM has been updated directly from version 5.1.10 to 5.1.13. The behaviors of SWMM are noticeably different between these two versions. As Figure 3 shows, when running SWMM 5.1.10 and SWMM 5.1.13 on exactly the same dataset, their outputs are not identical. First, the format of the date has changed (e.g., date JAN-06-2009 in SWMM 5.1.10 versus 01/06/2009 in SWMM 5.1.13). In addition, the position and the accuracy of parameters in the output file varied (e.g., the accuracy of parameter SurfaceInfil in SWMM 5.1.10 is 5 decimal places and in SWMM 5.1.13 is 3 decimal places). The name of some parameters differed between the two versions (e.g., one parameter name in SWMM 5.1.10 is

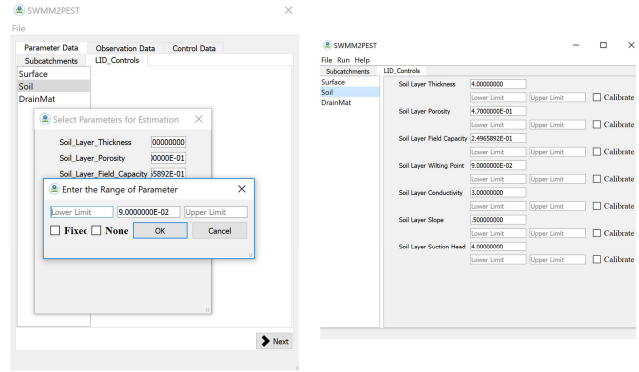


Figure 4. SWMM2PEST 1.0 (left) vs. SWMM2PEST 2.0 (right)

StorageDepth, which is called StorageLevel in SWMM 5.1.13). Further, the number of parameters in the output file is not the same (11 different output parameters in SWMM 5.1.10 versus 15 in SWMM 5.1.13). For these SWMM changes, the corresponding adaptations must be made in SWMM2PEST to allow the integrated software solution to run successfully.

The update of the integrated software will promote the new release of SWMM2PEST. In the process of software release, detailing the integration software information will not only help users better understand and deploy the software, but also can assist developers in discovering version errors if the release fails.

B. *Improvements as Requirements Change*

The software requirements specification (SRS) lays out non-functional and functional requirements. When the requirements of the stakeholders change, the software needs to be revised as well, contributing to the new version release.

There are many requirements changes during the update of SWMM2PEST from version 1.0 to 2.0. For example, as shown in Figure 4, restructuring the user interface (UI) of parameter selection is one of the non-functional requirements changes. In SWMM2PEST 1.0, the main interface and the parameter selection interface are separated. If the user expects to select a parameter for calibration, two more interfaces are required to be popped up to achieve this operation. If multiple parameters are to be calibrated, then the user has to repeatedly open and close interfaces. The parameter-input interface performs data input, however, once it is closed, the selected parameters and the inputted calibration range will become invisible. This UI structure not only makes the operation cumbersome but also prone to errors, such as lack of input parameters or incorrectly selected parameters to be calibrated. Therefore, a new non-functional requirement is warranted. To satisfy this change, we restructured and combined the main UI with the parameter selection UI in SWMM2PEST 2.0, so that one can more easily select parameters, directly enter the calibration parameter range, and clearly view all the data in the main interface. This improves the efficiency and usability of operating the software.

Updating software based on the changes in SRS can promote more systematically software development, bring

the software closer to users' need, and improve the quality of releases.

V. DISSCUSION

One of the limitations is that our main concern with a continuous release is to improve the software, yet we lack a sufficient number of released versions to assess the effectiveness of our pipeline. An example is our software upgrade from SWMM2PEST 2.0 to SWMM2PEST 2.1. While the upgrade matches the new version of SWMM, adds new features to satisfy SWMM changes, and hopefully improves the software's functional quality, there has not been enough time to determine how much it actually improved quality and user satisfaction.

We shared three most important lessons learned from releasing scientific software as follows.

- *Release as required.* To meet stakeholders' operational needs in the software, we developed a new UI structure as shown in Figure 4, and released SWMM2PEST 2.0 to facilitate the use of software. Intuitively, a new release becomes necessary when requirements or requests of critical functional requirements changes emerged.
- *Connector versus connectee release.* Our released scientific software is integrated with two software systems. As shown in Table II, although the new release of PEST from version 13.3 to 14.4 has less impact on the SWMM2PEST, the update of SWMM from version 5.1.10 to 5.1.13 necessitated significant modifications on SWMM2PEST to make it compatible with SWMM 5.1.13 output. The release of SWMM2PEST 2.1 to accommodate the updates in SWMM motivated us to develop a new release strategy.
- *Release to help automated testing.* We integrated our ongoing work of MT in a way to support automated testing of SWMM2PEST [4, 6]. MT is based on the construction of MRs. The newly released version can help with building diversity relations, which will benefit more comprehensive software testing.

VI. CONCLUSIONS

In this paper, we provided the process of releasing scientific software in GitHub by demonstrating a release of SWMM2PEST to GitHub after testing and refinement as well as its continuous release of subsequent versions. In addition, through analysis of the differences between versions and changes in requirements, we developed our ideas for improving ongoing releases.

Our future work includes further investigating other repositories to improve the quality of ongoing releases, the software and accompanying release documents, and user satisfaction. Given the continuous release of SWMM2PEST, we expect to receive more comprehensive user feedback and other developers' opinions to improve our software.

ACKNOWLEDGMENT

This work was partially supported by the U.S. National Science Foundation (Award CCF 1350487) and the U.S. Environmental Protection Agency (EPA) via the Oak Ridge Institute Student Employment Agreement. The first version of SWMM2PEST was developed by Suraj Kamble, while a graduate student at the University of Cincinnati and an Oak Ridge Research Institute for Science and Education recipient at the U.S. EPA.

Disclaimer: The U.S. EPA through its Office of Research and Development partially funded and managed the research described herein. It has been subjected to the Agency's review and approved for release. The views of this article are those of the authors and not necessarily represent the views or policies of the U.S. EPA. Any mention of trade names or commercial products does not constitute endorsement or recommendation for use.

REFERENCES

[1] B. Adams, S. Bellomo, C. Bird, B. Debic, F. Khomh, K. Moir, and J. Oduinn. Release Engineering 3.0. IEEE Software, 35(2):22-25, March/April 2018.

[2] G. Schermann, J. Cito, and P. Leitner. Continuous Experimentation: Challenges, Implementation Techniques, and Current Research. IEEE Software, 35(2):26-31, March/April 2018.

[3] B. Fitzgerald, and K.-J. Stol. Continuous Software Engineering: A Roadmap and Agenda. Journal of Systems and Software, 123:176-189, January 2017.

[4] X. Lin, M. Simon, and N. Niu. Hierarchical Metamorphic Relations for Testing Scientific Software. In International Workshop on Software Engineering for Science (SE4Science), pages 1-8, Gothenburg, Sweden, June 2018.

[5] D. Kelly. An Analysis of Process Characteristics for Developing Scientific Software. Journal of Organizational and End User Computing. 23(4):64-79, October 2011.

[6] X. Lin, M. Simon, and N. Niu. Exploratory Metamorphic Testing for Scientific Software. Computing in Science & Engineering, DOI:10.1109/MCSE.2018.2880577, November 2018.

[7] C. Morris, and J. Segal. Some Challenges Facing Scientific Software Developers: The Case of Molecular Biology. In International Conference on e-Science (e-Science), pages 216-222, Oxford, UK, December 2009.

[8] US EPA Research. https://www.epa.gov/water-research/storm-water-management-model-swmm. Last accessed: March 2019.

[9] C. M. John Doherty, Jim Rumbaugh, Matt Tonkin. http://www.pesthomepage.org/. Last accessed: March 2019.

[10] S. Kamble, X. Jin, N. Niu, and M. Simon. A Novel Coupling Pattern in Computational Science and Engineering Software. In International Workshop on Software Engineering for Science (SE4Science), pages 9-12, Buenos Aires, Argentina, May 2017.

[11] S. Kamble. SWMM2PEST 1.0. https://github.com/SurajKamble/SWMM2PEST. Last access: March 2019.

[12] X. Lin. SWMM2PEST 2.0. https://github.com/XuanyiLin/SWMM2PEST2.0. Last accessed: March 2019.

[13] G. A. A. Prana, C. Treude, F. Thung, T. Atapattu, and D. Lo. Categorizing the Content of GitHub README Files. Empirical Software Engineering, DOI:10.1007/s10664-018-9660-3, October 2018.