ProvBuild: Improving Data Scientist Efficiency with Provenance (An Extended Abstract)

Jingmei Hu
Harvard University
Cambridge, MA, USA

Jiwon Joung
University of Michigan
Ann Arbor, MI, USA

Maia Jacobs
Harvard University
Cambridge, MA, USA

Krzysztof Z. Gajos
Harvard University
Cambridge, MA, USA

Margo I. Seltzer
University of British Columbia
Vancouver, BC, Canada

ACM Reference Format:
Jingmei Hu, Jiwon Joung, Maia Jacobs, Krzysztof Z. Gajos, and Margo I. Seltzer. 2020. ProvBuild: Improving Data Scientist Efficiency with Provenance (An Extended Abstract). In *42nd International Conference on Software Engineering Companion (ICSE '20 Companion), October 5–11, 2020, Seoul, Republic of Korea.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3377812.3390912

1 THE PROBLEM

Data scientists frequently analyze data by writing scripts. We conducted a contextual inquiry with interdisciplinary researchers, which revealed that parameter tuning is a highly iterative process and that debugging is time-consuming. As analysis scripts evolve and become more complex, analysts have difficulty conceptualizing their workflow. In particular, after editing a script, it becomes difficult to determine precisely which code blocks depend on the edit. Consequently, scientists frequently re-run entire scripts instead of re-running only the necessary parts. We present ProvBuild, a data analysis environment that uses change impact analysis [1] to improve the iterative debugging process in script-based workflow pipelines. ProvBuild is a tool that leverages language-level provenance [2] to streamline the debugging process by reducing programmer cognitive load and decreasing subsequent runtimes, leading to an overall reduction in elapsed debugging time. ProvBuild uses provenance to track dependencies in a script. When an analyst debugs a script, ProvBuild generates a simplified script that contains only the information necessary to debug a particular problem. We demonstrate that debugging the simplified script lowers a programmer's cognitive load and permits faster re-execution when testing changes. The combination of reduced cognitive load and shorter runtime reduces the time necessary to debug a script. We quantitatively and qualitatively show that even though ProvBuild introduces overhead during a script's first execution, it is a more efficient way for users to debug and tune complex workflows. ProvBuild demonstrates a novel use of language-level provenance, in which it is used to proactively improve programmer productively rather than merely providing a way to retroactively gain insight into a body of code. To the best of our knowledge, ProvBuild is a novel application of change impact analysis and it is the first debugging tool to leverage language-level provenance to reduce cognitive load and execution time.

ICSE '20 Companion, October 5–11, 2020, Seoul, Republic of Korea

ACM ISBN 978-1-4503-7122-3/20/05.
https://doi.org/10.1145/3377812.3390912

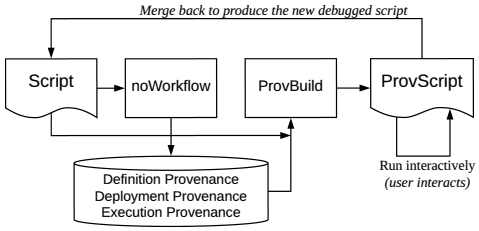

Figure 1: ProvBuild Architecture.

2 PROVBUILD: PIPELINE DEBUGGING USING PROVENANCE

ProvBuild utilizes noWorkflow [5] to collect *language-level provenance* [3] to record the actions a script takes and the dependencies between these actions, variables, values and functions. Provbuild uses these dependencies to identify precisely the parts of a script affected by a user's debugging. It then produces a shortened script, called the *ProvScript*, that contains only those parts of the script necessary to debug the original script. This shortened script provides two benefits: 1) it makes it easier for the user to reason about the script and the effect of a user's modification to it, and 2) it reduces the re-execution time.

ProvBuild consists of a backend engine (Figure 1) and a user interface (Figure 2). The interface allows users to debug functions or variables on the simplified *ProvScript* and seamlessly merge those modifications back into the original script. To facilitate evaluation, the ProvBuild prototype interface supports both conventional editing (i.e., editing on the entire script) and the ProvBuild provenance-driven editing of a *ProvScript*. In either case, the user begins by selecting the mode of interaction (conventional or ProvBuild) and identifying the script with which they are working. In ProvBuild mode, the interface activates the provenance tracking backend.

Users interact with their scripts through the three main modules shown in Figure 2.

- **Search**: The user inputs the name of the function or variable to edit (see (1) in Figure 2). ProvBuild extracts the object's dependencies based on the stored provenance information and generates a *ProvScript* containing only code pertaining to the chosen object.

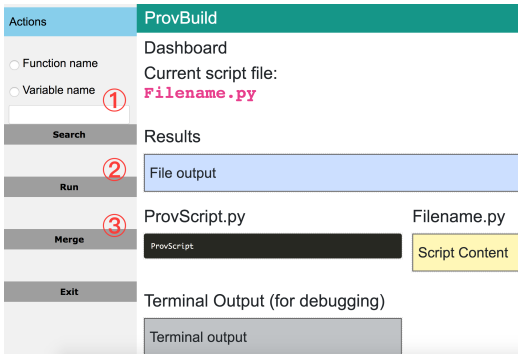

Figure 2: The ProvBuild interface features four boxes. We assume that all user scripts write to an output file. The blue box displays the contents of this output file. The yellow box displays the original script for user reference and the black box displays the *ProvScript*. The grey box displays terminal output, such as errors and print statements. The sidebar on the left provides easy access to the user commands. In ProvBuild mode, the user specifies the target, either a function or variable, to change/debug. After editing the *ProvScript*, the user runs it and examines the output. When the user is satisfied with the output, the user merges the changes into the original script. At this point, the yellow box updates to contain the revised script; the user can then select another object to edit.

- **Execute**: Instead of running the original script, ProvBuild executes the shortened *ProvScript* for the user (see (2) in Figure 2). This reduces run time.
- **Merge**: ProvBuild allows users to easily merge edits from the *ProvScript* into the original file (see (3) in Figure 2).

When used in conventional mode, the interface is similar to common modern text editors. Users simply edit their scripts and re-execute them in their entirety.

3 RESULTS

We conducted three studies to evaluate ProvBuild's performance, effectiveness, and usability. First, we evaluated ProvBuild in a controlled laboratory experiment with 21 participants who performed a series of debugging tasks with and without ProvBuild, demonstrating that users prefer programming with ProvBuild to programming without it, that they complete programming tasks more quickly, and that ProvBuild reduces their cognitive load. Next, we ran benchmarks to quantify how much time ProvBuild saved during script re-execution, demonstrating that ProvBuild shortens re-execution time using stored provenance. Finally, we evaluated ProvBuild in a real-world deployment with 12 participants who accessed to ProvBuild for a week. We asked participants how and when they chose to use ProvBuild in their daily work and used surveys to assess ProvBuild's utility, where users explained that ProvBuild saved them time, helped them understand their workflow, and provided more immediate results.

Study 1: Controlled Laboratory Experiment: We conducted a controlled lab study to quantify ProvBuild's ability to reduce the time to complete a task and to obtain insight into real-world challenges. We asked 21 participants to perform a series of debugging tasks with and without ProvBuild. After each tasks, we examined their digital memorization behavior to evaluate cognitive load and gave them a questionnaire with NASA-TLX standard questions, which evaluate perceived workload [4], and questions about their perceived self-efficacy, subjective assessment of ease of use, and effectiveness. We found that after programming with ProvBuild, participants had significantly shorter average completion time ($F(1, 20) = 66.64$, raw $p < 0.0001$, adjusted $p < 0.0003$) and greater number recall accuracy ($F(1, 20) = 16.00$, raw $p = 0.0007$, adjusted $p = 0.0014$), and also reported being more satisfied overall ($F(1, 20) = 7.42$, raw $p = 0.0131$, adjusted $p = 0.0131$). Those main effect were statistically significant.

Study 2: Performance Evaluation: We collected Python scripts from published works, compared script length and running times with and without ProvBuild to evaluate performance. noWorkflow introduces overhead during dynamic provenance tracking, which caused execution time to increase dramatically, in the best case, by only 56%, but in the worst case by around a factor of 30. However, ProvBuild still saves time in later debugging phases. We measured ProvBuild's performance after making three types of changes: 1) directly altering script output, 2) altering an input file or input variable, and 3) modifying the parameter of a function in the script. The speedup inherently depends on the length of the code path following the edit and ranges from a factor of 1.78 to 39.31. The resulting *ProvScript*s retained, on average, 77% and 58% of the lines of the original script, respectively, while the speed-ups averaged 1.23X and 2.46X, respectively.

Study 3: Deployment in the wild: We conducted a real-world deployment to evaluate ProvBuild's usefulness and efficacy for data scientists from different domains. We gave participants access to ProvBuild for one week, which allowed them to explore and use the tool for Python debugging in their daily work. We used surveys to obtain feedback from 12 participants. All participants chose to use ProvBuild at least once, while four participants used it more than once in a one-week period. Participants mentioned different benefits and several concerns after their use with ProvBuild. They expressed a preference for using ProvBuild and mentioned that ProvBuild improves the debugging process mainly by reducing programming time, allowing users to find dependencies and understand their workflow more easily, reducing the need for memorization. They also addressed issues we knew about (e.g., initial run time) or that could be easily addressed (e.g., integration with Jupyter). Participants did not report any significant barriers to independent use of ProvBuild.

REFERENCES

[1] Robert S. Arnold. 1996. *Software Change Impact Analysis.* IEEE Computer Society Press, Los Alamitos, CA, USA.
[2] James Cheney, Amal Ahmed, and Umut a. Acar. 2011. Provenance As Dependency Analysis. *Mathematical. Structures in Comp. Sci.* 21, 6 (Dec. 2011), 1301–1337.
[3] Juliana Freire, David Koop, Emanuele Santos, and Cláudio T Silva. 2008. Provenance for computational tasks: A survey. *Computing in Science & Engineering* 10, 3 (2008), 11–21.
[4] Sandra G Hart. [n.d.]. NASA-task load index (NASA-TLX); 20 years later. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* ([n. d.]).
[5] Leonardo Murta, Vanessa Braganholo, Fernando Chirigati, David Koop, and Juliana Freire. 2015. noWorkflow: Capturing and Analyzing Provenance of Scripts. In *Provenance and Annotation of Data and Processes.* Springer International Publishing, Cham, 71–83.