Immediate Feedback for Students to Solve Notebook Reproducibility Problems in the Classroom

Christophe Casseau, Jean-Rémy Falleri, Xavier Blanc, Thomas Degueule

Immediate Feedback for Students to Solve Notebook Reproducibility Problems in the Classroom

Christophe Casseau*, Jean-Rémy Falleri*†, Xavier Blanc* and Thomas Degueule*
*Univ. Bordeaux, Bordeaux INP, CNRS, LaBRI, UMR5800
Talence, France
name.surname@labri.fr
†Institut Universitaire de France

***Abstract*—Jupyter notebooks have gained popularity in educational settings. In France, it is one of the tools used by teachers in post-secondary classes to teach programming.**

When students complete their assignments, they send their notebooks to the teacher for feedback or grading. However, the teacher may not be able to reproduce the results contained in the notebooks. Indeed, students rely on the non-linearity of notebooks to write and execute code cells in an arbitrary order. Conversely, teachers are not aware of this implicit execution order and expect to reproduce the results by running the cells linearly from top to bottom. These two modes of usage conflict, making it difficult for teachers to evaluate their students' work.

This article investigates the use of immediate visual feedback to alleviate the issue of non-reproducibility of students' notebooks. We implemented a Jupyter plug-in called Notebook Reproducibility Monitor (NoRM) that pinpoints the non-reproducible cells of a notebook under modifications. To evaluate the benefits of this approach, we perform a controlled study with 37 students on a programming assignment, followed by a focus group. Our results show that the plug-in significantly improves the reproducibility of notebooks without sacrificing the productivity of students.

***Index Terms*—notebooks, reproducibility, computer science education.**

I. Introduction

The Jupyter Notebook is an open-source web-based interactive environment that combines multi-language programming, rich text, mathematical equations, plots and more, into a single document. They are used by data scientists and teachers [1] to share "literate programming" documents [2].

In France, Jupyter notebooks are now proposed by the Ministry of National Education as part of its virtual learning environment for teachers and students. The context and challenges of this paper arise from our experience with the computer science courses of CPGE students. Scientific CPGEs are selective and intensive post-secondary curricula consisting of two years of study to prepare students before they enter engineering schools. Algorithms, data science, modeling, and simulation are at the heart of the computer science course for these students, and Jupyter notebooks are an ideal environment for teaching these subjects [3].

During programming assignments with notebook and Python, students write programs in code cells that yield an optional output in a dedicated area when executed. Students then use these outputs to check that their programs produce the expected result. While completing the assignment, students are free to add, modify, delete and (re-)execute code cells in any order. Once they are satisfied with the results, they send the notebook to their teacher.

Then, having no way of knowing the execution order followed by the students, teachers re-execute the cells linearly, from top to bottom, to make sure that they obtain the same results [4]. Ideally, the last outputs obtained by the students and those obtained by the teacher when re-executing the notebook should be the same. In such a case, the notebook is said to be *reproducible*, a key aspect of the scientific method [5]. Unfortunately, it is common for teachers to obtain different outputs when re-executing the notebooks due to a different execution order. Different outputs do not necessarily imply that the students failed. Therefore, the teacher must understand how the student arrived in this state to understand its reasoning and grade appropriately. This complex task could be avoided if the notebook was reproducible in the first place.

A naive solution would be to force the students to reset the notebook state and to execute the notebook linearly to ensure its reproducibility. However, this approach typically reveals several reproduciblity issues within the same notebook, making it difficult for the students to debug. In this paper, we propose to alleviate this problem with a simple Jupyter plug-in (NoRM) that gives immediate visual feedback [6, 7, 8, 9] to students regarding reproducibility while they are writing their notebooks. Our plug-in continuously attempts to reproduce the notebook in the background and highlights which cells are reproducible or not using colors. It allows the students to become aware of reproducibility issues as soon as possible to take action before they accumulate.

To assess the usefulness of our tool, we performed a controlled study involving 37 CPGE students on a two hours programming assignment to investigate the following questions:

RQ1 What are the most common reproducibility mistakes in non-reproducible notebooks?
RQ2 Does our plug-in improve the reproducibility of student notebooks?
RQ3 Since having to deal with reproducibility problems early might make the writing of a notebook more tedious, does our plug-in affect the productivity of students?

978-1-6654-4592-4/21/$31.00 ©2021 European Union

Among the non-reproducible notebooks in the control group, we observed the same root causes as those identified in other studies [4, 10, 11, 12, 13, 14]. Our results show that the students using the plug-in were able to significantly improve the reproducibility of their notebooks (only 6% of non-reproducible notebooks compared to 47% in the control group). Moreover, the students using the plug-in were able to complete the assignment as well as the ones in the control group. In a focus group with students conducted after our study, one participant told us that: *in the absence of the plug-in I don't know exactly when the notebook is no longer in a reproducible state*.

II. Notebook Reproducibility

To study reproducibility, we consider that a notebook is a list of code cells and their output.

A. The Student's Point of View

When the student writes her notebook, she unknowingly uses a notebook kernel, *i.e.*, a computational engine that executes the code contained in the cells, updates the current memory state, and writes the output in the corresponding output cells.

Students mainly use the *Run Cell* command of Jupyter to execute individual cells in an arbitrary order. This execution mode is particularly suitable in the exploratory phase, where students write their notebooks incrementally following a trial-and-error approach. It enables them to edit and re-execute previous code cells as desired to experiment with their code.

In Figure 1a, the student wrote a first version of the Syracuse algorithm. The student then refactors her code with meaningful variable names (`s` becomes `syracuse`) but forgets to update one reference (Figure 1b, Cell 2, line 3). This issue goes unnoticed, as the variable `s` is still stored in memory and the algorithm still produces the expected result when running the entire notebook linearly. Then, in Figure 1c, the student expands the notebook with another usage of the algorithm in Cells 4 and 5. In her mind, the reader should first execute Cell 4 to initialize the variables, then Cell 2 to compute the Syracuse suite, and finally Cell 5 to display the results.

Once the assignment is finished, the student shares her notebook with her teacher via a file where the code cells and associated outputs are serialized, but not the kernel state.

B. The Teacher's Point of View

When the teacher opens the student's notebook, she is not aware of the execution order used by the student to obtain the outputs. Instead, she expects to reproduce the same results by executing the cells linearly. However, doing so on the notebook of Figure 1c yields a different output for Cell 5.

Indeed, Cell 2 must be re-executed every time the variable `N` is assigned a new value to compute the Syracuse suite. This example highlights the difference between non-linear and linear executions: the notebook executes successfully but does not produce the same outputs and thus *is not reproducible*.

If the teacher had received the second version of the notebook (Figure 1b), the situation would be worse. Running the cells linearly would yield a Python `NameError` on Cell 2, as depicted in Figure 1d. In this case, the notebook is not even *fully executable* by the teacher.

In the notebooks of Figure 1, Jupyter displays an *execution count* on the left of each code cell (*e.g.*, "[9]" for cell 3 in Figure 1c). It indicates the total number of code cell executions since the kernel was started, at the time of executing this cell. When the student re-executes a cell, it overwrites the previous execution count. As a consequence, the list of execution counts does not necessarily start from 1 (if the first executed cell was re-executed afterwards) and is not necessarily continuous.

One might think that executing the notebook in the order given by the *execution counts* of the notebook will help, but this is not the case [10, 4]. Executing the notebook of Figure 1c following the execution counts (*i.e.*, $[c_4, c_5, c_3, c_2, c_1]$) will also yield different outputs. In addition, execution counts are not guaranteed to appear if the student did no execute all cells before saving and sharing her assignment with the teacher or if the notebook was reset.

When grading, the teacher is left wondering if the code written by the student is simply wrong or if the issues arise from non-reproducibility, forcing her to conduct an in-depth analysis of the notebook written by the student.

C. Reproducibility

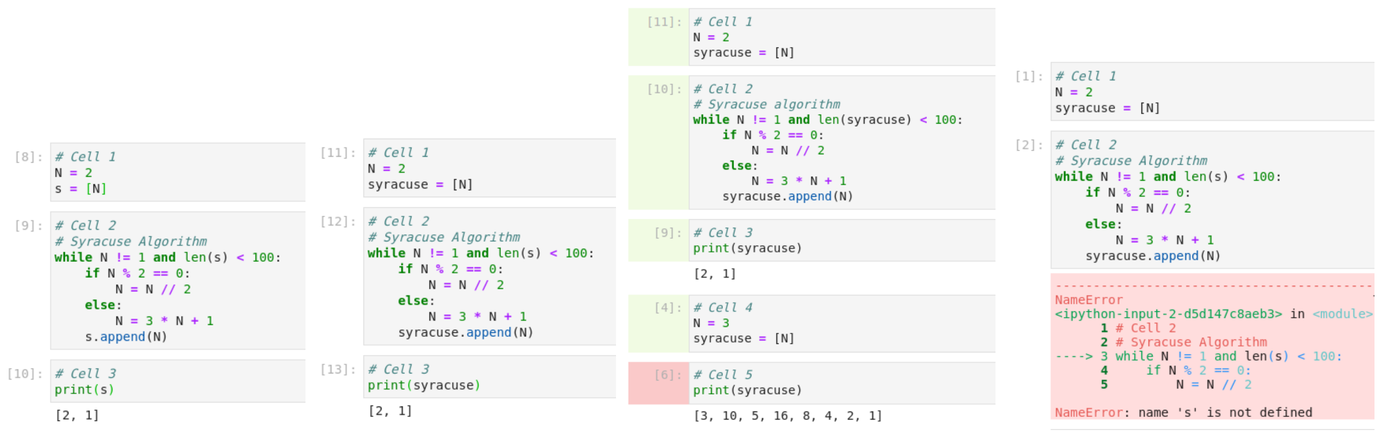
We consider two roles manipulating the notebooks: the writer (the student) and the reader (the teacher). As we have seen, it is natural for the writer to enjoy the freedom of non-linear execution when writing a notebook, and it is natural for the reader to execute and read the same notebook linearly. A notebook is *reproducible* if the outputs obtained by running cells from top to bottom are the same as those saved in the notebook. Our hypothesis is that students would benefit from knowing immediately when reproducibility problems arise in their notebooks to understand and solve them as early as possible. For our experiments, we implemented a simple plug-in (NoRM) highlighting reproducible cells in green and non-reproducible cells in red (Figure 1c). The current version of our plug-in compares, after each cell execution, the outputs of the notebook with the ones produced linearly by a second kernel using simple string comparison.

III. Study design

In this section, we describe the setup of our controlled study.

A. Procedure

Our participants are 37 CPGE students. In CPGE, the computer science course teaches scientific computing topics including the use of Python for data analysis and simulation. Jupyter notebooks are well-suited to support these requirements and prepare students for formatting their code and analyses in the same document [15]. Prior to the study, the professor (first author) introduced the notion of reproducibility and the NoRM plug-in. Then, the students had two hours to complete 13 questions with the instruction to use as many cells as desired. For every question, the subject contained the expected results that the programs should output for some example input values.


```
[8]:  # Cell 1
      N = 2
      s = [N]

[9]:  # Cell 2
      # Syracuse Algorithm
      while N != 1 and len(s) < 100:
          if N % 2 == 0:
              N = N // 2
          else:
              N = 3 * N + 1
          s.append(N)

[10]: # Cell 3
      print(s)

      [2, 1]
```

(a) First version

```
[11]: # Cell 1
      N = 2
      syracuse = [N]

[12]: # Cell 2
      # Syracuse Algorithm
      while N != 1 and len(s) < 100:
          if N % 2 == 0:
              N = N // 2
          else:
              N = 3 * N + 1
          syracuse.append(N)

[13]: # Cell 3
      print(syracuse)

      [2, 1]
```

(b) Second version

```
[11]: # Cell 1
      N = 2
      syracuse = [N]

[10]: # Cell 2
      # Syracuse algorithm
      while N != 1 and len(syracuse) < 100:
          if N % 2 == 0:
              N = N // 2
          else:
              N = 3 * N + 1
          syracuse.append(N)

[9]:  # Cell 3
      print(syracuse)

      [2, 1]

[4]:  # Cell 4
      N = 3
      syracuse = [N]

[6]:  # Cell 5
      print(syracuse)

      [3, 10, 5, 16, 8, 4, 2, 1]
```

(c) Final version

```
[1]:  # Cell 1
      N = 2
      syracuse = [N]

[2]:  # Cell 2
      # Syracuse Algorithm
      while N != 1 and len(s) < 100:
          if N % 2 == 0:
              N = N // 2
          else:
              N = 3 * N + 1
          syracuse.append(N)

      ---------------------------------------------------------------------------
      NameError
      <ipython-input-2-d5d147c8aeb3> in <module>
            1 # Cell 2
            2 # Syracuse Algorithm
      ----> 3 while N != 1 and len(s) < 100:
            4     if N % 2 == 0:
            5         N = N // 2

      NameError: name 's' is not defined
```

(d) Fresh execution of the second version

Figure 1: Successive versions of a notebook implementing the Syracuse algorithm.

The students had to answer the questions while trying to keep the notebook reproducible top-to-bottom. For the study, we randomly assigned one group to the *control group* and the other group to the *experimental group* (with plug-in).

B. Measures

To answer RQ1, RQ2, and RQ3, we used the following metrics: reproducibility, executability (top-to-bottom), number of code cells (markdown and text cells excluded) and number of correct answers (evaluated by the professor).

C. Analysis

For RQ1, we use NoRM to assess non-reproducible cells and manually analyze the causes of non-reproducibility.

For RQ2, we test the two following null hypotheses: *our plug-in has no effect on the reproducibility of notebooks* and *our plug-in has no effect on the executability of notebooks*. As the reproducibility and executability of a notebook are Boolean variables and as our sample size is small, we use *Fisher's exact test* to assess the significance.

For RQ3, we test the following null hypotheses: *our plug-in has no effect on the number of cells* and *our plug-in has no effect on the number of correct answers*. Since we have no assumption about the distribution of the number of cells and correct answers, we use a non-parametric two-tailed Mann-Whitney test for both hypotheses to assess the significance.

Finally, to triangulate our quantitative results, we proceed to a focus group with the two groups of participants.

IV. Results

The examples, assignments, notebooks, data, and analyses are available on a companion webpage [16].

A. RQ1

We extracted three main categories, described below.

a) Out-of-order execution of dependent cells: In this category, non-reproducibility arises because the writer executed the cells in an other order than the top-down order.

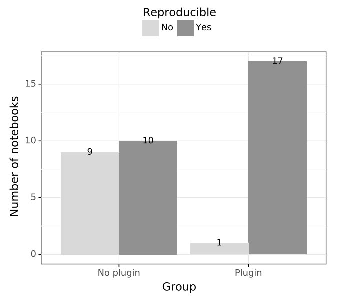
b) Missed cell execution after a code modification: In this category, non-reproducibility arises because the writer modified a cell, but forgot to execute the cell itself or a dependent cell.

c) Referencing a renamed or deleted variable: In this category, non-reproducibility arises because the writer modifies (or deletes) the name of a function or variable, and executes the corresponding cell. However, the previous name still exists in memory.

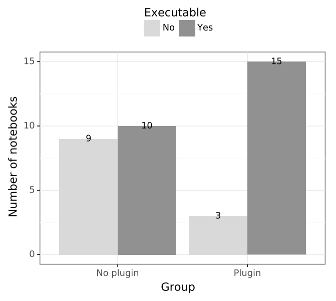
B. RQ2

In Figure 2a, 47% (9 out of 19) of the notebooks of the control group are not reproducible while 6% (1 out of 18) of the notebooks of the experimental group are in this situation. In the study of Pimentel et al., 96% of the notebooks with an unambiguous execution order were not reproducible [10]. In our study, we are not affected by some reproducibility problems such as incompatible environment or missing data. However, about half of the notebooks are still not reproducible. In the experimental group, the ratio of non-reproducible notebooks is greatly reduced, with only one non-reproducible notebook. To statistically assess the difference between the two groups, we perform a Fisher exact test on the contingency table of the amount of reproducible and non-reproducible notebooks in the control and experimental groups. The test yields a p-value $p = 0.008{*}{*}{*}$ which is significant under a strict 0.01 threshold. The risk ratio is 0.12, indicating that a notebook developed with the plug-in has 88% less risk to be non-reproducible.

We note in Figure 2b that 47% (9 out of 19) of the notebooks of the control group are not executable while 17% (3 out of 18) of the notebooks of the experimental group are in this situation. In the study of Pimentel et al., 75% of the notebooks with an unambiguous execution order were not executable. Similarly to the reproducibility, in our study, we are not affected by some execution problems such as missing dependencies. In the experimental group, the ratio of non-executable notebooks is greatly reduced, with only three non-reproducible notebook. To stastically assess the difference between the two groups,

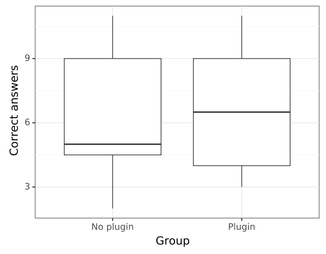


(a)

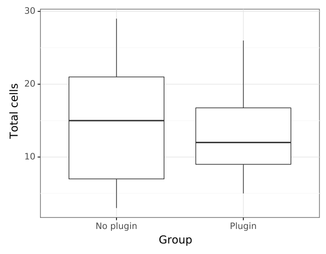


(b)

Figure 2: Number of a) reproducible and b) executable notebooks in the control and experimental groups



(a)



(b)

Figure 3: Number of a) correct answers and b) cells in the control and experimental groups.

we perform a Fisher exact test on the contingency table of the amount of reproducible and non-reproducible notebooks in the control and experimental groups. The test yields a p-value $p = 0.078*$ which is significant under a loose 0.1 threshold. The risk ratio is 0.35, indicating that a notebook developed with the plug-in has 65% less risk to be non-executable.

C. RQ3

We analyzed the students' notebooks to identify possible adverse effects of the plug-in on the number of correct answers and the total number of cells. In Figure 3a, the median number of correct answers appears slighlty greater in the experimental group, and the interquartile range is also greater in this group. To statistically assess the difference between the two groups, we perform a non-parametric two-tailed Mann-Withney test that yields a p-value $p = 0.78$. Thus, we cannot reject the null hypothesis that there is no difference between the groups.

In Figure 3b, the median number of cells is a little lower in the experimental group, and the interquartile range is also lower. To statistically assess the difference between the two groups, we perform a non-parametric two-tailed Mann-Withney test that yields a p-value $p = 0.21$. Thus, we cannot reject the null hypothesis that there is no difference between the groups.

D. Focus group

We first questioned the students about the notion of reproducibility and the importance they attached to it. As future engineers the students think that it is an important notion when it comes to sharing their work in physics, mathematics and chemistry. We then asked them if they could define the link between reproducibility and notebook. We received two spontaneous answers: "*the notebook can pose reproducibility problems when sharing it with someone because it will be necessary to execute the cells in the order they need to be executed for it to work*", "*when you save the file and open it later on, it doesn't work or it doesn't give the same thing as when you opened it*".

In a second step we asked the students if they made an effort to consider reproducibility when writing a notebook. Only 25% admit to thinking about it. Students say: "*when I look for the solution I don't think about reproducibility*", "*it is difficult to know when the notebook is no longer reproducible*".

Finally we asked student of the experimental group what were the advantages and drawbacks of the plug-in. Students say: "*when a cell is red I know there is a problem*", "*when it's green it's reassuring*", "*there is something missing to help us correct when it is red*".

E. Threats to Validity

a) Internal validity: All the students did not necessarily have the same level in programming and the best students may have ended up in the same group. A statistical study of the correct answers suggests that there is no significant difference.

b) External validity: The problem given to students has been specially designed to evaluate the plug-in in a controlled environment (no non-deterministic code or modules that could be problematic). Our study has a small sample size with a limited number of students and it lasted for only two hours.

V. Related work

Reproducibility is usually assessed after the notebook is finished [17, 12] to reconstruct a possible execution order for the cells. Wang et al. added the detection of packages needed to reproduce the results of a notebook [18]. Macke et al. developed a custom Jupyter kernel that uses run-time tracing and static analysis to automatically manage lineage associated with cell execution and global notebook state [14]. There are also two other exciting directions that we have not explored. The first allows data scientists to record the development history of the notebook [1, 19] and the second proposes a number of changes to the notebook core and interface that allow users to explicitly encode dependencies between cells by adding a unique and persistent identifier to each cell [4].

VI. Conclusion

In this paper, we study the reproducibility of notebooks in the context of programming courses for post-secondary students. We designed and implemented an approach giving immediate visual feedback on the reproducibility of a notebook's cells, as well as a prototype implementation, NoRM. To demonstrate the benefits of our approach, we conducted a controlled study showing that: (i) it significantly reduces the reproducibility problems of student notebooks, (ii) it has no significant adverse effects on the productivity of students and (iii) it is well perceived by the students.

References

[1] M. B. Kery, B. E. John, P. O'Flaherty, A. Horvath, and B. A. Myers, "Towards Effective Foraging by Data Scientists to Find Past Analysis Choices," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, ser. CHI '19. New York, NY, USA: Association for Computing Machinery, 2019, event-place: Glasgow, Scotland Uk. [Online]. Available: https://doi.org/10.1145/3290605.3300322

[2] D. E. Knuth, "Literate Programming," *The Computer Journal*, vol. 27, no. 2, pp. 97–111, 1984. [Online]. Available: https://doi.org/10.1093/comjnl/27.2.97

[3] A. Rule, A. Tabard, and J. D. Hollan, "Exploration and Explanation in Computational Notebooks," in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, ser. CHI '18. New York, NY, USA: ACM, 2018, pp. 32:1–32:12, event-place: Montreal QC, Canada. [Online]. Available: http://doi.acm.org/10.1145/3173574.3173606

[4] D. Koop and J. Patel, "Dataflow Notebooks: Encoding and Tracking Dependencies of Cells," in *9th USENIX Workshop on the Theory and Practice of Provenance (TaPP 2017)*. Seattle, WA: USENIX Association, Jun. 2017. [Online]. Available: https://www.usenix.org/conference/tapp17/workshop-program/presentation/koop

[5] R. Peng, "The reproducibility crisis in science: A statistical counterattack," *Significance*, vol. 12, no. 3, pp. 30–32, 2015, _eprint: https://rss.onlinelibrary.wiley.com/doi/pdf/10.1111/j.1740-9713.2015.00827.x. [Online]. Available: https://rss.onlinelibrary.wiley.com/doi/abs/10.1111/j.1740-9713.2015.00827.x

[6] S. Gross, B. Mokbel, B. Hammer, and N. Pinkwart, "Learning Feedback in Intelligent Tutoring Systems," *KI - Künstliche Intelligenz*, vol. 29, May 2015.

[7] S. Narciss, K. Huth, and D. Narciss, "How to design informative tutoring feedback for multi-media learning," *Instructional Design for Multimedia Learning*, Dec. 2002.

[8] L. Benotti, F. Aloi, F. Bulgarelli, and M. J. Gomez, "The Effect of a Web-based Coding Tool with Automatic Feedback on Students' Performance and Perceptions," in *Proceedings of the 49th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '18. New York, NY, USA: Association for Computing Machinery, Feb. 2018, pp. 2–7. [Online]. Available: https://doi.org/10.1145/3159450.3159579

[9] F. Restrepo-Calle, J. Echeverry, and F. González, "Using an interactive software tool for the formative and summative evaluation in a computer programming course: an experience report," *Global Journal of Engineering Education*, vol. 22, pp. 174–185, Nov. 2020.

[10] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks," 2019, pp. 507–517.

[11] S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma, and T. Barik, "What's wrong with computational notebooks? pain points, needs, and design opportunities," in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, ser. CHI '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1–12. [Online]. Available: https://doi.org/10.1145/3313831.3376729

[12] J. Wang, T.-Y. KUO, L. Li, and A. Zeller, "Assessing and Restoring Reproducibility of Jupyter Notebooks," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Sep. 2020, pp. 138–149, iSSN: 2643-1572.

[13] J. Wang, L. Li, and A. Zeller, "Restoring execution environments of jupyter notebooks," *CoRR*, vol. abs/2103.02959, 2021. [Online]. Available: https://arxiv.org/abs/2103.02959

[14] S. Macke, H. Gong, D. J. L. Lee, A. Head, D. Xin, and A. G. Parameswaran, "Fine-grained lineage for safer notebook interactions," *CoRR*, vol. abs/2012.06981, 2020. [Online]. Available: https://arxiv.org/abs/2012.06981

[15] A. Willis, P. Charlton, and T. Hirst, "Developing Students' Written Communication Skills with Jupyter Notebooks," in *Proceedings of the 51st ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '20. New York, NY, USA: Association for Computing Machinery, Feb. 2020, pp. 1089–1095. [Online]. Available: https://doi.org/10.1145/3328778.3366927

[16] C. Casseau, J.-R. Falleri, X. Blanc, and T. Degueule, "NoRM companion webpage," https://github.com/christophe33/NoRM.git, 2021.

[17] H. Fangohr, V. Fauske, T. Kluyver, M. Albert, O. Laslett, D. Cortés-Ortuño, M. Beg, and M. Ragan-Kelly, "Testing with Jupyter notebooks: NoteBook VALidation (nbval) plug-in for pytest," *arXiv:2001.04808 [cs]*, Jan. 2020, arXiv: 2001.04808. [Online]. Available: http://arxiv.org/abs/2001.04808

[18] J. Wang, L. Li, and A. Zeller, "Restoring Execution Environments of Jupyter Notebooks," in *Proceedings of the 43rd International Conference on Software Engineering*, 2021, p. 12.

[19] M. B. Kery and B. A. Myers, "Interactions for Untangling Messy History in a Computational Notebook," in *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, Oct. 2018, pp. 147–155.