Threats to Instrument Validity Within "in Silico" Research: Software Engineering to the Rescue

Serge Demeyer[1,2(✉)], Coen De Roover[3], Mutlu Beyazit[1], and Johannes Härtel[4]

[1] Universiteit Antwerpen, Antwerp, Belgium
serge.demeyer@uantwerpen.be
[2] Flanders Make vzw, Kortrijk, Belgium
[3] Vrije Universiteit Brussel, Brussels, Belgium
[4] Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Abstract. "In Silico" research drives the world around us, as illustrated by the way our society handles climate change, controls the COVID-19 pandemic and governs economic growth. Unfortunately, the code embedded in the underlying data processing is mostly written by scientists lacking formal training in software engineering. The resulting code is vulnerable, suffering from what is known as threats to instrument validity.

This position paper aims to understand and remedy threats to instrument validity in current "in silico" research. To achieve this goal, we specify a research agenda listing how recent software engineering achievements may improve "in silico" research (SE4Silico) and, conversely, how software engineering may strengthen its applicability (Silico4SE).

Keywords: In Silico · Threats to Validity · Instrument Validity

1 Introduction

"In Silico" research has driven several debates and policymaking on climate change [1], COVID-19 [2], and economic forecasts [3]. It has become a standard tool for scientists, complementing the traditional "in vitro" and "in vivo" research with carefully crafted simulation models for the phenomenon under investigation. Nowadays, even the director of the Max Planck Institute for Evolutionary Anthropology in Leipzig confesses the importance of code: "*How you fit the model is part of the model.*" ([4], page 39). Code is the new critical piece of research, and "*[when] something goes wrong, every piece of the machine may be suspect.*"

There is an inherent threat to validity (named *Instrument Validity*) stating that a data processing pipeline may not reliably produce what it is designed to produce, hence the results may not be truthful. Indeed, subtle changes in

T. Margaria and B. Steffen (Eds.): ISoLA 2024, LNCS 15222, pp. 82–96, 2025.
https://doi.org/10.1007/978-3-031-75387-9_6

the underlying program code may affect the results in unpredictable ways, completely invalidating the conclusions. The behavioural, the natural, and the life sciences have already experienced issues with code. We refer to three distinct examples.

- In biology, a research team investigating molecular structures had to withdraw five high-profile publications because of code accidentally flipping two columns of the data [5].
- In economics, a paper reportedly suffered from accidental data loss in the code examining the relation between a country's growth and depth. The resulting false conclusions have influenced the adoption of austerity politics in the EU [6].
- In anthropology, code written to handle missing data had a highly controversial influence on conclusions about moralising gods in cultures. A version of an article was published in nature [7], criticised for wrong treatment of missing data [8], subsequently withdrawn and later republished with updates [9].

As with any other threat to validity, threats to instrument validity need to be understood, detected and (if possible) resolved.

> $\Longrightarrow$ *In this position paper, we aim to understand and remedy threats to instrument validity in modern data processing pipelines. To achieve this goal, we pick achievements from software engineering research and elaborate on how these may be applied in today's data processing pipelines (SE4Silico). As a complementary perspective, we deduce a research agenda for the software engineering community, listing opportunities for strengthening applicability (Silico4SE).*

2 Instrument Validity Within "In Silico" Research

In this section, we elaborate on the nature of "in silico" research as an empirical discipline. We argue the need for treating the code embedded in the underlying data processing as a measurement device, similar to spectroscopes and oscilloscopes but also -as adopted in qualitative research- surveys and questionnaires. Such measurement devices should be subjected to established quality procedures to ensure they produce the intended results.

2.1 Defining "In Silico" Research

We refer to an overview paper from Life in Silico —a double peer-reviewed open access journal concerning all aspects of in silico studies— for a concise definition.

> In silico refers to tests or simulations carried out on a computer or by theoretical analysis, as opposed to "in vivo" (in a real organism) or "in vitro" (in a laboratory). [...] Many benefits of using in silico methods

> include the capacity to run tests more rapidly and at a lower cost, as well as testing ideas that would be difficult or impossible to test in vivo or in vitro. However, it should be noted that the outcomes of in silico trials should be confirmed by experimental or observational investigations.
>
> Mohd Yusuf, 2023 [10]

Note that "in silico" research is surprisingly similar yet quite different from data science. We quote a blog posted in the Communications of the ACM on the various favours of data science.

> Data science also introduces a new scientific paradigm. [...] The third scientific paradigm was introduced only several decades ago and is the computational paradigm, in which scientists simulate complex phenomena using algorithms and computers. The fourth scientific paradigm, according to Gray, is data exploration, in which data is captured or simulated, and then analyzed by scientists to infer new scientific knowledge. [...] Data science is the current evolution of the fourth paradigm and described as "the conduct of data analysis as an empirical science, learning directly from data itself."
>
> Koby Mike and Orit Hazzan, 2023 [11]

Another area which also involves self-educated programmers is what is currently captured under "low-code" (or even "no-code") development platforms, enabling end-user development [12]. Spreadsheets are a notorious example of the spectacular results which enthusiastic laymen lacking formal training can produce. Nevertheless, lots of anecdotes concerning spreadsheet bugs circulate on the internet; quite recently the loss of 16 thousands COVID test results because of an import limit [13]. There is previous research which calls for supporting end-user programmers with solid software engineering principles such as code inspections [14].

For the remainder of this position paper, we will focus on "in silico" research as the third scientific paradigm. However, many of our conclusions would apply equally to data science or end-user development.

2.2 Illustrative Examples of "In Silico" Research

The University of Antwerp has a long tradition of governing what is known as *"citizen science"*, where citizens actively contribute to science either with their intellectual effort or surrounding knowledge, or with their tools and resources. For all practical purposes, many citizen science projects adopt an "in silico" approach. Three noteworthy examples are as follows.

- AIRBEZEN measures air pollution by using strawberry plants deposited on windowsills of citizens. After 2 months, the strawberry-hosts cut off a few leaves and bring them in to the distribution points. The researchers at the

University of Antwerp then measure the concentrations of particulate matter on the leaves, thus revealing to what the plants were exposed. https://www.uantwerpen.be/en/projects/airbezen/
- CURIEUZENEUZEN collects real-time data from 5,000 citizen scientists placing a measuring device in their gardens to assess the impact of heat and drought. https://curieuzeneuzen.be/home-en/
- ISALA aims to better understand the female microbiome with state-of-the-art DNA technology. The Isala[1] project found 6,000 women who volunteered to take some simple 'swabs' in the privacy of their own bathroom. https://isala.be/en/

Serge Demeyer (the first author of the position paper you are reading now) was personally involved in two "in silico" projects measuring indoor air quality, first in museums [15], and later in ships [16].

- AIRCHECK and ELGAS where we constructed special measurement devices collecting heterogeneous data concerning air quality (particle matter, concentration of pollutant gasses, light, temperature, etc.). Measurement campaigns spanning several months allowed us to perform routine monitoring, identifying periods where works of art (in case of museums) or human sailors (in case of ships) were exposed to risk. We assessed the impact of certain mitigation actions. https://www.belspo.be/belspo/brain-be/projects/AIRCHECQ_en.pdf

What all of these projects have in common is that they collect lots of data from various sources, often outside of explicit researcher control. As a consequence, the data processing pipelines need special care during intake and data cleansing. Once the data is deemed correct, researchers explore the data, coding various filters, correlations, manipulations, etc. to verify certain hypotheses. The results are presented usually in tabular form but often complemented with visualisations highlighting regions of interest. Flexibility is key, supporting the iterative and incremental nature of this exploration.

2.3 Threats to Instrument Validity

"In Silico" research is a special kind of empirical research, thus making observations about the phenomenon under investigation. These observations are enabled via special purpose measurement devices, such as oscilloscopes, microscopes and rRNA sequencers. The term "measurement device" should be interpreted in the broadest possible sense; in qualitative research, questionnaires and surveys are considered measurement devices as well.

As with all empirical research, there is an inherent risk that the observations made via such measurement devices are invalid. Such risks are typically covered

[1] The similarity between the name of this project (isala) and the name of the conference (isola) is pure coincidence. The project was named after the first female doctor in Belgium, Isala Van Diest (1842 – 1916). She had to study in Switzerland because women were not allowed to attend Belgian universities at that time.

in a section of the research protocol called "Threats to Validity" [17]. The one threat to validity we are concerned about here is "reliability": to what extent are the data and the analysis dependent on the specific researchers conducting the research? One of the sub-characteristics concerns the instruments used to collect measurements, and it is captured under "instrument validity": to what extent does the instrument measure what it is designed to measure? Typical mitigation actions there are properly calibrating the device, doing sufficient maintenance, providing training to the researchers, etc. But, here as well, mitigation actions should be interpreted in the broadest possible sense: a recommended practice for surveys is to conduct a pilot study to mitigate the risk that survey participants misunderstand the question.

> ⟹ *We argue that "instrument validity" also applies to a data processing pipeline in the sense that it may not produce what it is designed to produce, hence the results may not be truthful. Indeed, subtle changes in the underlying program code may affect the results in unpredictable ways, completely invalidating the conclusion. Therefore, data-processing pipelines should be subjected to a series of quality checks to ensure that they produce the intended results.*

3 Software Engineering Within "In Silico" Research

We first explore which software engineering tools and best practices are adopted in data processing pipelines and, if possible, the motivations for doing so. Next, we dive deeper into opportunities that scientists lacking formal training in software engineering are likely to miss and how this may affect the data processing pipelines they produce.

3.1 Software Engineering Best Practices

We had a series of conversations with the scientists involved in the aforementioned projects. These informal conversations were never recorded nor transcribed, so cannot be reproduced. The recurring theme is that all the data processing pipelines involved were created by a team of enthusiastic researchers without formal training in software engineering. We list the relevant observations regarding software engineering best practices which are (unconsciously) adopted.

R, *python*, *Jupyter notebooks* and *MatLab/Simulink* were the platforms of choice. There is seldom a deliberate decision on why a given platform is adopted; it is mainly based on previous experience, reputation and hearsay. Despite the claim that Jupyter notebooks are a de facto standard in the behavioural, the natural, and the life sciences (cfr. a publication in Nature [18]), R and python are most common in our limited sample.

Flexibility. The primary adoption criterion is the availability of libraries and packages that perform basic tasks and can be glued together easily. All these platforms embrace the flexibility needed for scientific exploration with a (sometimes implicit) "pipes and filters" architecture [19].

Literate Programming. Literate programming is an important coding guideline behind data science code that makes research transparent and understandable [20]. Here as well, glueing together a variety of libraries and packages makes the programs driving the data-processing pipeline self-documenting. In that sense, the code looks more like a domain-specific programming language.

Community. The community supporting the platform is certainly a contributing factor, both within the team and equally outside the confines of the lab. Q&A sites are consulted heavily, and code snippets found there get merged into the code base frequently. Only rudimentary quality checks on the imported code are performed.

Small, Interdisciplinary Teams. The teams involved are rather small (3 to 5 individuals), where a few take on the role of creating and maintaining the data-processing code. These scientists had minimal training in programming as part of their education and then improved their skills "on the job".

Modest Input Data Size. The sizes of the data sets pushed into the pipeline are surprisingly small. They usually fit on the local hard disk and are shared locally within the team. Data processing is done on local PCs and, sometimes, on a dedicated server. Speed of processing was seldom an issue: researchers just waited until the results were available.

Clone-and-Own. Experienced teams typically adopt a "clone-and-own" approach, i.e., making a full copy of the codebase of one project and adapting it to the needs of the new one [21]. Within software engineering circles, "clone-and-own" is a poor men's approach towards product-line engineering [22]. Nevertheless, it is quite appropriate for the projects creating and maintaining the code.

3.2 Software Engineering Missed Opportunities

Since the data-processing pipelines are created and maintained by scientists lacking formal training in software engineering, it is logical that they suffer from certain quality issues affecting the instrument validity. Below, we list observations from the aforementioned interviews, confirmed (if possible) by remarks made in the literature.

Lack of Reproducibility. Jupyter notebooks, as many other forms of data science code, suffer from quality issues, with lack of reproducibility being the most striking one [23]. Pimental *et al.* observed *"We were able to successfully run 24.11% of the unambiguous execution order Python notebooks. [...] However, the rate is way smaller (4.03%) when we count only notebooks that produce the same results."* [24]. A follow-up paper made similar observations: *"We were able to successfully run between 22.57% and 26.09% of the note-books that we attempted to run."* [25].

But reproducibility rates are alarming in other languages as well. Trisovic *et al.* found that 74% of the R code that accompanies academic papers crashed upon execution [26]. Boll *et al.* reported that from 65 recent papers employing MATLAB/Simulink models *"only 31% of the tools and 22% of the models used as experimental subjects are accessible. [...] We found none of the experimental results presented in these papers to be fully replicable, and 6% partially replicable."* [27].

Coding Driven by Copy-and-Paste (a.k.a. Clones). Programming practice appears to be driven by copying code from other sources (such as tutorials, Q&A websites and other projects). Mindlessly copying code from Q&A websites obviously induces risks. There is the infamous example of the most copied code on Stack Overflow (over 6,000 copies on GitHub alone) which happens to be buggy [28].

In software engineering, copy-and-paste programming is a well-studied phenomenon captured under the umbrella term of *clones*. Clones are considered a sign of technical debt and hence ought to be removed via refactoring [29]. However, it is recognized that there are several good reasons to copy-and-paste code (cfr. the award-winning paper "'Cloning Considered Harmful' Considered Harmful" [30].)

Several researchers examined clones within data-processing pipelines. Tang *et al.* analyzed 26 projects, consisting of 4.2 MLOC, along with 327 manually examined code patches. They found that code duplication is a major cross-cutting theme that particularly involves machine learning configuration and model code [31]. Koenzen *et al.* examined clones with Jupyter Notebook cells within a random sample of 1,000 GitHub repositories containing approximately 6,000 notebooks. Snippets of code that get duplicated the most within Jupyter notebooks are the ones whose main activity concerns visualization (21.35%), followed by machine learning (15.45%), definition of functions (12.85%) and data science (9.03%) [32]. According to Källén *et al.*, more than 70% of code snippets sampled from GitHub notebooks are exact copies of other snippets [33]. Around 50% of the notebooks do not have any unique piece of code.

Lack of Tests. During our interviews, researchers confirmed that there is rarely an explicit verification step on the outcome of the pipeline. The individual researchers conduct a cursory visual inspection of the resulting tables and visualisations, and, when the results make sense, they are deemed correct. This comes close to what is known as a *smoke test*, although the exploratory nature implies

that there is no objective oracle to verify the outcome. Regression tests to detect whether or not changes introduced unintended side effects are never used.

Bug Taxonomies. The example from the introduction referring to code accidentally flipping two columns of the data [5] is an example of the subtle bugs that may completely invalidate "in silico" results. Software engineering researchers investigated the kinds of mistakes that are often made in data-processing pipelines. De Santana *et al.* created a bug taxonomy for Jupyter Notebooks based on a mining study of 14,740 commits from 105 GitHub open-source projects and 30,416 Stack Overflow posts [34]. *"[. . .] the most frequent bugs in the Jupyter notebook are related to Environments and Settings, consisting of 43.2% of analyzed posts from StackOverflow and 35.5% of the issues analyzed in GitHub. [. . .] Incorrect algorithm implementations cause many bugs (44.2% of GitHub issues and 22% of StackOverflow posts). Most of them are related to coding and logical errors resulting in 'Incorrect Functionality'."* Noteworthy is that De Santana *et al.* also investigated the impact of bugs and found that *"The most frequent impacts from bugs in Jupyter notebooks are: Run Time Errors (Stack- Overflow - 57.5% | GitHub - 31.2%), [. . .]"*. This partially explains the alarming lack of reproducibility mentioned earlier.

Islam *et al.* performed a complementary study on the bugs made while using deep learning libraries [35]. They expanded upon an existing bug taxonomy and added a few categories quite specific for data-processing pipelines. Two categories are especially relevant from the perspective of Instrument validity: (i) Data Bug and (ii) API Change. *"2.4.3. Data Bug. This bug may arise if an input to the deep learning software is not properly formatted or cleaned well before processing in any deep learning model." "2.5.3 API Change. The reason for these bugs is the release of the new version of a deep learning library. In other words, the bug happens when the new API version is not backward compatible with its previous version."*

Divergent Opinions on Correctness. A central difficulty of data processing pipelines is that it is often unclear whether something is a mistake or a deliberate action. Take the example on moralizing gods from the introduction, where missing data was replaced with zeros, drastically altering the conclusions but not necessarily a mistake [9]. There exist various guidelines on how to treat data to avoid deriving incorrect conclusions; however, it is up to the researcher to consciously decide whether to follow the guideline. Take the example of *data leakage*, stating that training and testing data should not accidentally depend on one another. However, such dependency may also be employed for implicit control as, for instance, advocated in multilevel models [36].

This is particularly relevant for the research on bias and fairness in AI recommender systems, in which decisions are expected to be non-discriminatory with respect to an individual's protected traits such as gender, ethnicity, or religion [37]. Researchers tackle fairness through mathematical formalisms. However, such mathematical methods can only make guarantees about fairness based

on strong assumptions (reliable estimates of the ground truth, access to sensitive data and a lossy aggregation of discrimination effects) that are unrealistic in practice. Researchers therefore have to make task-specific assumptions when processing the data with the inherent risk of biased results.

$\Longrightarrow$ *While computer science in general (and software engineering in particular) is still a relatively young discipline, the accessibility of the field has given rise to the adoption of many good practices, even for scientists without formal training in software engineering.*

State-of-the-art tools and languages are adopted, and the communities and supporting libraries in the corresponding eco-systems result in "in silico" being a very productive and effective research paradigm. However, the corresponding data-processing pipelines suffer from quality issues, lack of reproducibility being the most alarming one. Nevertheless, the risk of mistakes (accidental or not) within the data-processing pipelines has drastic repurcussions on the validity of the instrument.

4 Remediation of Threats to Validity in "In Silico" Research

In the previous section, we listed best practices which are (unconsciously) adopted "in silico" research. But we also illustrated that there are several missed opportunities which lead to data-processing pipelines producing irreproducible —sometimes even erroneous— results. To reduce the risk on instrument validity, we specify a research agenda listing how research in software engineering may improve "in silico" research (SE4Silico) and, complementarily, how the software engineering community may benefit from "in silico" research (Silico4SE).

4.1 Smoke Tests and Regression Tests

The International Software Testing Qualification Board (ISTQB) provides a glossary with terminology adopted within the software testing community [https://glossary.istqb.org/]. Two best practices are particularly relevant for "in silico" research.

- *smoke test* (synonyms: sanity test, intake test, confidence test)
  A test suite that covers the main functionality of a component or a system to determine whether it works properly before planned testing begins.
- *regression testing*
  A type of change-related testing to detect whether defects have been introduced or uncovered in unchanged areas of the software.

SE4Silico. As software engineers, we need to educate the “in silico” research community about lessons learned with respect to the reliability of software. The bare minimum would be that every reproduction package comes with one smoke test and a series of (preferably automatic) regression tests. The smoke test should explicitly document one valid suite of input files and the corresponding results irrespective of the tabular or visual output format. Once the smoke test confirms the minimum functionality is in place, the regression tests execute the basic scenarios to verify whether the results are equivalent.

Silico4SE. To accomplish (semi-automatic) smoke tests and (automatic) regression tests for “in silico” research, more work is needed to verify whether two subsequent results are to be considered equivalent. It is impossible to have full control over neither where the data processing is executed (certainly not when replication is a requirement) nor when it is executed (certainly not when the results are time-dependent). Therefore, deciding whether a test passes or fails is not a simple check for equality anymore; the verdict should have ways to allow for small but tolerated deviations. There exists decades worth of literature on ways to analyze time-series data (cfr. Kalman Filter, Autoregressive Integrated Moving Average, Dynamic Time Warping). Complementarily, there is lots of work that measures differences between manipulations on data streams (cfr. Mean Absolute Percentage Error, Weighted Average Percentage Error, Weighted Mean Absolute Percentage Error). Leveraging this research should allow the software engineering community to derive the appropriate verdicts. An interesting avenue for further research would be to mix these with the formulas for calculating the flakiness of a test suite [38].

Regression tests, on the other hand, will need more work on data-driven test cases, as it is the data which drives the pipeline, not the control flow. In a similar vein, the typical ways to measure the strength of a test suite (statement coverage, branch coverage, MC/DC coverage, etc.) must be expanded to incorporate boundary values within the data sets.

4.2 Clone Genealogies

Clones have a negative connotation in software engineering, but we must assume that, in data science code, clones are not always harmful [30]. Tang *et al.* found that copy-and-paste is often present in model construction, model training, and data preprocessing [31]. This assures understanding of such relevant steps; hence, code idioms are better kept explicit to facilitate a literate programming style. However, particular copies may include poor coding practices. And such problematic code constructs will also spread across tutorial sites, blog posts and Q&A sites.

Silico4SE. Software engineers study the families of clones by constructing so-called clone genealogies, tracing clones up to their origin in the version history. The idea of studying clone evolution originated in 2005, when Kim *et al.* coined the term “Code Clone Genealogies" for describing how a family of clones evolves

over time [39]. For example, they illustrated that clones are either short-lived and disappear due to natural code evolution or long-lived and get changed consistently over time since there is no proper way to refactor them into a single abstraction [39]. Other research also showed cloned code is generally more stable than non-cloned code [40], yet changes to clones are not always consistent [41]. We ourselves studied clone genealogies within test code and concluded that test code clones inherently differ from production clones, not only at their instantiation but also at every point of their evolution [42]. For the software engineering community, it would be interesting to study "in silico" code through the lens of clone genealogies. Given that copy-paste code is such a widespread practice, such an investigation could reveal interesting phenomena within the corresponding eco-systems.

SE4Silico. The automated tracing of the origin of copied code may help to understand, detect, and resolve threats to instrument validity. Koenzen *et al.* state that code is typically duplicated from tutorial sites (in 35% of the cases), API documentation (32%) and Stack Overflow (14%) [32]. Based on the frequency of cloned code snippets and whether they have changed inconsistently, we can identify "high-risk" code snippets. By repairing the corresponding code and rerunning the data pipeline, we can assess the impact on the actual results. Via such anecdotal evidence, we can demonstrate the necessity of good software engineering practices within "in silico" research.

4.3 Static and Dynamic Code Analysis

Software engineers have a long history of producing tools that flag potential programming errors, bugs, stylistic errors and suspicious constructs [43]. With the rising popularity of "in silico" research, it shouldn't come as a surprise that such attempts are made for data-processing pipelines as well.

SE4Silico. Urban and Müller propose a theoretical framework to automatically detect (unintended) filtering, which relies on established program analysis techniques, such as dependency analysis and strongly live variable analysis [44]. For "data leakage", Subotić *et al.* suggested a general static analysis framework, where user-provided annotations tag training and testing functions [45]. Tosch *et al.* developed a tool to check the validity of code for online experiments statically, identifying problems that arise in experimental design and causal inference [46]. Pimentel *et al.* created a proof-of-concept lint tool that is capable of identifying 21 potential issues within Jupyter notebooks [25]. Härtel *et al.* runs simulations with artificial data to verify whether the statistical tools adopted in a model produce acceptable results [47].

Silico4SE. As typical in today's software engineering research, the aforementioned proof-of-concept tools are validated on a selection of open-source projects by the creators of the tools. All of them do find quality issues with varying levels of precision. However, it remains unknown whether repairing these issues is

worth the effort. To bring these proof-of-concept tool prototypes from Technology Readiness Level 4 (technology validated in lab) to Technology Readiness Level 5 (validated in relevant environment) the software engineers need to reach out to the "In Silico" scientists. This is the normal way for research prototypes to achieve sufficient maturity. The good news is that it is likely that "In Silico" scientists will be open to such collaborations, because they share the same background.

5 Conclusion

In this position paper, we argue the eminent risk of so-called *instrument validity* within "in silico" research. As with any piece of software, the data processing pipelines used within this research paradigm incorporate mistakes, sometimes accidental and sometimes intentional. As a consequence, subtle changes in the underlying program code may affect the results in unpredictable ways, completely invalidating the conclusions. We illustrate that, despite the adoption of many good software engineering practices, there are still a lot of room for improvement. We derive a research agenda from two complementary perspectives.

(i) SE4Silico, where we pick achievements from software engineering research and elaborate on how these may be applied in today's data processing pipelines.
(ii) Silico4SE, where we sketch how software engineering research may benefit from the different perspective provided in today's data processing pipelines.

Since the two communities share the same background, setting up such an exchange should be possible. Indeed, "In Silico" researchers must publish papers which may serve as precise requirement documents so appraised by Software Engineers. Therefore it is ultimately a matter of investing the necessary time and effort for the greater benefit of our society.

References

1. Ahmstorf, S., Ganopolski, A.: Long-term global warming scenarios computed with an efficient coupled climate model. Clim. Change **43**(2), 353–367 (1999)
2. Sharma, M., et al.: Understanding the effectiveness of government interventions against the resurgence of Covid-19 in Europe. Nat. Commun. **12**(1), 1723–2041 (1999)
3. Kara, Y., Boyacioglu, M.A., Baykan, Ö.: Predicting direction of stock price index movement using artificial neural networks and support vector machines: The sample of the Istanbul stock exchange. Expert Systems with Appl. **38**(5), 5311–5319 (2011)
4. McElreath, R.: Statistical Rethinking: A Bayesian Course with Examples in R and STAN (2nd edition). Chapman and Hall/CRC (2020)

5. Miller, G.: A scientist's nightmare: Software problem leads to five retractions. Science **314**(5807), 1856–1857 (2007)
6. Herndon, T., Ash, M., Pollin, R.: Does high public debt consistently stifle economic growth? A critique of Reinhart and Rogoff. Cambridge J. Econom. **38**, 257–279 (2013)
7. Whitehouse, H., et al.: RETRACTED ARTICLE: complex societies precede moralizing gods throughout world history. Nature **568**, 226–229 (2019)
8. Beheim, B., et al.: Treatment of missing data determined conclusions regarding moralizing gods. Nature **595**, E29–E34 (2021)
9. Whitehouse, H., et al.: Retraction note: complex societies precede moralizing gods throughout world history. Nature **595**, 320 (2021)
10. Yusuf, M.: Insights into the in-silico research: current scenario, advantages, limits, and future perspectives. Life in Silico **1**, 13–25 (2023)
11. Mike, K., Hazzan, O.: What is data science? Commun. ACM **66**, 12–13 (2023)
12. Lieberman, H., Paternò, F., Wulf, V. (eds.): End-User Development. Springer, Netherlands, Dordrecht (2006)
13. Hern, A.: Covid: how Excel may have caused loss of 16,000 test results in England. The Guardian (2020)
14. Roy, S., Deursen, A.V., Hermans, F.: Perceived relevance of automatic code inspection in end-user development: A study on VBA. In: Proceedings EASE 2019 (23rd International Conference on Evaluation and Assessment in Software Engineering), (New York, NY, USA), pp. 167—176, Association for Computing Machinery (2019)
15. Pernia, D.L., Demeyer, S., Schalm, O., Anaf, W.: A data mining approach for indoor air assessment, an alternative tool for cultural heritage conservation. In: Proceedings HERI-TECH 2018 (IOP Conference Series: Materials Science and Engineering), vol. 364 – 1, p. 012045 (2018)
16. Carro, G., et al.: A new approach to make indoor air quality in the accommodation of ships understandable and actionable for seafaring staff. In: Proceedings ICMT 2020 8th International Conference on Maritime Transport — Maritime Transport VIII Sept (2020)
17. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empir. Softw. Eng. **14**(2), 131–164 (2009)
18. Perkel, J.: Why jupyter is data scientists' computational notebook of choice. Nature **563**, 145–146 (2018)
19. Meunier, R., Rohnert, H., Sommerlad, P., Stal, M., Buschmann, F.: Pattern-Oriented Software Architecture, vol. 1. Wiley, A System of Patterns (1996)
20. Kery, M.B., Radensky, M., Arya, M., John, B.E., Myers, B.A.: The story in the notebook: Exploratory data science using a literate programming tool. In: CHI 2018 Proceedings 2018 CHI Conference on Human Factors in Computing Systems, pp. 1–11, Association for Computing Machinery, (2018)
21. Businge, J., Openja, M., Nadi, S., Berger, T.: Reuse and maintenance practices among divergent forks in three software ecosystems. J. Emp. Softw. Eng. **27**(2), 54 (2022)
22. Dubinsky, Y., Rubin, J., Berger, T., Duszynski, S., Becker, M., Czarnecki, K.: An exploratory study of cloning in industrial software product lines. In: Proceedings CSMR 2013 17th European Conference on Software Maintenance and Reengineering, pp. 25 – 34 (2013)
23. Wang, J., Li, L., Zeller, A.: Better code, better sharing: on the need of analyzing jupyter notebooks. In: ICSE-NIER 2020 Proceedings ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results, pp. 53—56, Association for Computing Machinery (2020)

24. Pimentel, J.F., Murta, L., Braganholo, V., Freire, J.: A large-scale study about quality and reproducibility of jupyter notebooks. In: MSR 2019 Proceedings 2019 IEEE/ACM 16th International Conference on Mining Software Repositorie, pp. 507–517, IEEE (2019)
25. Pimentel, J.F., Murta, L., Braganholo, V., Freire, J.: Understanding and improving the quality and reproducibility of Jupyter notebooks. Emp. Soft. Eng. **26**(4), 1–55 (2021). https://doi.org/10.1007/s10664-021-09961-9
26. Trisovic, A., Lau, M.K., Pasquier, T., Crosas, M.: A large-scale study on research code quality and execution. Sci. Data **9**(60) (2022)
27. Boll, A., Vieregg, N., Kehrer, T.: Replicability of experimental tool evaluations in model-based software and systems engineering with matlab/simulink. Innov. Syst. Softw. Eng. (2022). https://doi.org/10.1007/s11334-022-00442-w
28. Lundblad, A.: The most copied stackoverflow snippet of all time is flawed!. programming.guide. https://programming.guide/worlds-most-copied-so-snippet.html
29. Demeyer, S., Ducasse, S., Nierstrasz, O.: Object-Oriented Reengineering Patterns. Morgan Kaufmann (2003)
30. Kapser, C., Godfrey, M.W.: Cloning considered harmful' considered harmful. In: Proceedings WCRE 2006 13th Working Conference on Reverse Engineering, pp. 19 — 28 (2006)
31. Tang, Y., Khatchadourian, R., Bagherzadeh, M., Singh, R., Stewart, A., Raja, A.: An empirical study of refactorings and technical debt in machine learning systems. In: ICSE 2021 Proceedings of 2021 IEEE/ACM 43rd International Conference on Software Engineering, pp. 238–250 (2021)
32. Koenzen, A.P., Ernst, N.A., Storey, M.A.D.: Code duplication and reuse in jupyter notebooks. In: Proceedings VL/HCC2020 2020 IEEE Symposium on Visual Languages and Human-Centric Computing, pp. 1–9 (2020)
33. Källén, M., Wrigstad, T.: Jupyter notebooks on github: characteristics and code clones. In: The Art, Science, and Engineering of Programming, vol.5, no. 3, (2021)
34. De Santana, T.L., Neto, P.A.D.M.S., De Almeida, E.S., Ahmed, I.: Bug analysis in jupyter notebook projects: an empirical study. ACM Trans. Softw. Eng. Methodol. **33** (2024)
35. Islam, M.J., Nguyen, G., Pan, R., Rajan, H.: A comprehensive study on deep learning bug characteristics. In: ESEC/FSE 2019 Proceedings 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, New York, NY, USA, p. 510-520, Association for Computing Machinery (2019)
36. Gelman, A., Hill, J.: Data Analysis Using Regression and Multilevel/Hierarchical Models. Cambridge University Press (2006)
37. Buyl, M., De Bie, T.: Inherent limitations of AI fairness. Commun. ACM **67**, 48–55 (2024)
38. Kowalczyk, E., Nair, K., Gao, Z., Silberstein, L., Long, T., Memon, A.: Modeling and ranking flaky tests at apple. In: Proceedings ICSE-SEIP 2020 42nd International Conference on Software Engineering: Software Engineering in Practice, pp. 110–119 (2020)
39. Kim, M., Sazawal, V., Notkin, D., Murphy, G.: An empirical study of code clone genealogies. In: Proceedings ESEC/FSE 2005 10th European Software Engineering Conference Held Jointly with 13th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 187–196 (2005)
40. Krinke, J.: Is cloned code more stable than non-cloned code?. In: Proceedings SCAM 2008 2008 Eighth IEEE International Working Conference on Source Code Analysis and Manipulation, pp. 57–66, IEEE (2008)

41. Krinke, J.: A study of consistent and inconsistent changes to code clones. In: Proceedings WCRE 2007 14th Working Conference on Reverse Engineering, pp. 170–178, IEEE, (2007)
42. van Bladel, B., Demeyer, S.: A comparative study of code clone genealogies in test code and production code. In: Proceedings VST 2023 IEEE Workshop on Validation, Analysis and Evolution of Software Tests, pp. 913 – 920, IEEE (2023)
43. Bessey, A., et al.: A few billion lines of code later: using static analysis to find bugs in the real world. Commun. ACM **53**, 66–75 (2010)
44. Urban, C., Müller, P.: An abstract interpretation framework for input data usage. In: Ahmed, A. (ed.) ESOP 2018. LNCS, vol. 10801, pp. 683–710. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-89884-1_24
45. Subotić, P., Milikić, L., Stojić, M.: A static analysis framework for data science notebooks. In: Proceedings ICSE-SEIP 2022 44th International Conference on Software Engineering: Software Engineering in Practice, (New York, NY, USA), pp. 13 – 22, Association for Computing Machinery (2022)
46. Tosch, E., Bakshy, E., Berger, E.D., Jensen, D.D., Moss, J.E.B.: PlanAlyzer: assessing threats to the validity of online experiments. Commun. ACM **64**, 108–116 (2021)
47. Härtel, J., Lämmel, R.: Operationalizing validity of empirical software engineering studies. Emp. Softw. Eng. **28**(6) (2023). https://doi.org/10.1007/s10664-023-10370-3