Reproducible HTML Notebooks

Computer Science Research Methods
INFO4990
Semester 1, 2023
Assignment 1

Tanaka Chitete

Supervisor:
Dr. Rahul Gopinath

The University of Sydney
School of Computer Science

April 16, 2023

Contents

Chapter 1

Top conferences & journals

In assessing top conferences and journals, I had proceeded with my identification using both qualitative and quantitative means. However, after further discussion with my research supervisor, I opted to employ a qualitative assessment, instead. This essentially came down to the fact that research and development in the computer sciences involves a substantial amount of communication amongst researchers and other concerned parties. As a result of this communication, researchers will come to identify the top venues and journals relevant to their respective fields through simple word of mouth. Naturally, they ultimately share their conclusions with their peers, mentors and other acquaintances. Considering that individuals are more likely to trust the conclusions of those within close proximity to themselves as opposed to conclusions drawn from quantitative information provided by distant entities, a qualitative assessment seemed far more appropriate for this task.

For all venues and journals, the following factors were considered:

- **Conference or journal theme** [1]
  Does the theme of the conference or journal match the theme of your project?
- **Organising and sponsoring entities** [1]
  Who organises the conference or releases issues of the journal? Who are the sponsoring entities, if any?
- **Review format** [1]
  What review format is utilised in the publication process?
- **Means of dissemination** [1]
  Where will the conference papers or journal articles primarily be published?

With consideration of these five factors, the top conferences I identified through discussion with my research supervisor and in conducting an ongoing literature review were:

- IEEE/ACM International Conference on Software Engineering (ICSE) [2]
- ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) [3]

- ACM Object-oriented Programming, Systems, Languages, and Applications (OOPSLA) [4]
- IEEE/ACM International Conference on Automated Software Engineering (ASE) [5]
- ACM International Conference on Human Factors in Computing Systems (CHI) [6]
- ACM Symposium on User Interface Software and Technology (UIST) [7]

Similarly, the top journals identified were:

- ACM Transactions on Software Engineering and Methodology (TOSEM) [8]
- IEEE Transactions on Software Engineering (TSE) [9]

1.1 Conferences

	Theme	**Organisers**	**Sponsors**	**Review**	**Dissemination**
ICSE	SWE	• ACM • IEEE	• SIGSOFT • SIGSAC	Double-blind	• ACM Digital Library • IEEE Xplore
ESEC/FSE	SWE	ACM	SIGSOFT	Double-blind	ACM Digital Library
OOPSLA	SWE	ACM	SIGPLAN	Double-blind	ACM Digital Library
ASE	SWE	• ACM • IEEE		Double-blind	• ACM Digital Library • IEEE Xplore
CHI	HCI	ACM	SIGCHI	Double-blind	ACM Digital Library
UIST	HCI	ACM	• SIGGRAPH • SIGCHI	Double-blind	ACM Digital Library

Table 1.1: Top conferences [2–7, 10]

1.2 Journals

	Theme	Organisers	Sponsors	Review	Dissemination
TOSEM	SWE	ACM	IEEE Computing Society	Single-blind	ACM Digital Library
TSE	SWE	IEEE		• Single-blind • Double-blind	ACM Digital Library

Table 1.2: Top journals [8, 9, 11, 12]

Chapter 2

Main research groups

To address the problem of identifying the main research groups in the fields relevant to my work, my research supervisor recommended CSRankings[13]–it proved to be an invaluable resource. It achieves the metric-based ranking of tertiary education institutions found all over the world[13]. Considering the number of works by faculty which have been published at the most prestigious conferences in each area of the computer sciences, CSRankings identifies top institutions and the faculty members actively engaged in research[13]. The metrics themselves are separated into **author-level** and **institution-level**.

Author-level metrics include *number of publications* and *normalised number of publications*. Ultimately, both of these metrics influence the overall rank of an institution.

Number of publications, p, is given by:

$$p = \text{total number of works published by this faculty member}$$

Normalised number of publications, n, is given by:

$$n = \frac{\text{total number of works published by this faculty member}}{\text{total number of co-authors on all works involving this faculty member}}$$

Institution-level metrics include *count* and *faculty*. Count, c, is given by:

$$c = \sum n = \text{the sum of normalised number of publications}$$

Faculty, f is given by:

$$f = \text{total number of faculty members who have published works in this area}$$

Taking into account research areas presented by CSRankings, the most applicable to my research are software engineering and human-computer interaction.

2.1 Research field 1

Considering all institutions with a focus on software engineering, the main institutions are:

Rank	Institution	Count	Faculty
1	Nanjing University	21.5	31
2	Peking University	16.1	18
3	Carnegie Mellon University	13.6	12
4	Monash University	12.3	7
5	Concordia University	10.9	6

Table 2.1: Top institutions in the field of software engineering [13]

2.2 Research field 2

For the area of human computer interaction, the main institutions included the following:

Rank	Institution	Count	Faculty
1	Carnegie Mellon University	58.7	38
2	University of Washington	48.2	31
3	University of Toronto	31.6	18
4	Tsinghua University	29.5	16
5	University of Michigan	28.1	25

Table 2.2: Top institutions in the field of human-computer interaction [13]

Chapter 3

Exemplary works

3.1 Exemplary work 1

S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma and T. Barik, "What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities", *Conference on Human Factors in Computing Systems - Proceedings*, 2020, DOI: 10.1145/3313831.3376729

In this paper, the authors chronicle the most prohibitive pain points faced by users of computational notebooks. Employing the use of semi-structured interviews and surveys with industry professionals, they present highly insightful findings and lay the foundations for future work. Concerning their methods, the authors first conducted a thorough set of field interviews with data scientists and their teams to uncover when and why they face difficulties in their interactions with notebooks. In order to get a better understanding of the computational notebook landscape, the authors then went on to conduct a series of interviews to corroborate their initial findings. Opening up to a much wider population of data scientists, the authors conducted a final survey to corroborate the interviews they conducted. The manner in which the paper is structured easily guides the reader through the authors work extensive work through the use of a writing style that isn't mired with highly technical jargon. In regards to providing unfamiliar researchers with an introduction to the Jupyter notebook landscape, this work captures all of the core grievances facing users of such applications. By consequence, it provides an excellent road map for research and development in this field for the next few years.

3.2 Exemplary work 2

J. Wang, L. Li and A. Zeller, "Restoring Execution Environments of Jupyter Notebooks", *International Conference on Software Engineering - Proceedings*, 2021, DOI: 10.1145/3313831.3376729

This article presents a novel solution to addressing the issue of missing Jupyter notebook dependencies. As for the authors' methods, they begin by conducting a study to detail the extent of Jupyter notebook dependency issues. They sought to

identify and investigate the causes of non-executable Jupyter notebooks. The authors then proceed to present the design and accompanying implementation of their solution, *SnifferDog*, utilising the findings of their initial study to inform their solution. Ultimately, to validate SnifferDog's efficacy, the authors perform a large-scale notebook analysis and restoration, quantifying SnifferDog's overall performance. In yet another evaluation, the authors verified SnifferDog's acute ability to accurately determine dependencies of broken notebooks. By the authors opting to outline a high-level overview of their work's structure, they immediately provide readers with a digestible means to consume their paper. As a result, should a reader's interest be piqued, they are then more receptive to reading the paper in its entirety. In addition, the authors' style of presentation involving multiple diagrams, code snippets and figures only further promote reader engagement. Ultimately, the authors' solution proves to be a significant advancement to addressing the problem of restoring Jupyter notebook dependencies.

Chapter 4

Research problems & questions

Prior to my undertaking of this project, the main research problem had already been identified by my research supervisor. Therefore, the research questions I was to eventually propose needed to be fundamentally relevant to the research problem, and in turn, the project itself. Ultimately, the proposed research questions seek to address the main aspects of the computational workflow of Jupyter notebooks–(i) saving and restoration, (ii) dissemination, and (iii) collaboration. After numerous iterations spawning from multiple discussions, literature reviews, and feedback, the proposed research questions are as follows:

- **Research question 1**: *How could we achieve the saving and restoration of a partially-computed notebook?*
- **Research question 2**: *How could we achieve the lightweight sharing of a notebook–including all dependencies–with minimal expectations on the receiver?*
- **Research question 3**: *How could we achieve the collaboration of multiple users on a single notebook?*

4.1 Research question 1

How could we achieve the saving and restoration of a partially-computed notebook?

Consider the fact that the user should be able to (i) work on a notebook, (ii) save and close the notebook, and (iii) continue working on the notebook at a later stage. Thus, it is imperative to address the problem of saving and restoring complete notebook state–namely, the Jupyter/Python kernels and their internal state. Developing an efficient means to complete this task would help promote the user's continued workflow, and in turn, greatly improve user experience.

4.2 Research question 2

How could we achieve the lightweight sharing of a notebook–including all dependencies–with minimal expectations on the receiver?

Perhaps the biggest issue plaguing both prospective and existing users of Jupyter notebooks is setup and installation–but specifically, dependency management. Oftentimes, users will need to spend considerable time completing arduous tasks such as (i) downloading and installing dependencies on their local machine, (ii) fixing dependency conflicts resulting from incompatible versions, and (iii) manually upgrading deprecated APIs. The sheer difficulty involved in completing these tasks often disincentivises people from using Jupyter notebooks, altogether. Devising a means to embed these dependencies into a single file along with the notebook itself will serve to greatly lower the barrier of entry for using Jupyter notebooks.

4.3 Research question 3

How could we achieve the collaboration of multiple users on a single notebook?

Considering that computer science education and data science work are both highly-collaborative endeavours, the next logical step from being able to share a notebook would be to enable collaboration among its editors. Integration of versioning or synchronous (real time) editing are the two most likely avenues for collaboration. However, the complexity of designing and implementing a real-time solution may be beyond the scope of an Honours project. Further analysis will be conducted to assess its feasibility, but if it is ultimately deemed infeasible, it could be a rather interesting prospect for future work.

Chapter 5

Annotated bibliography

5.1 Research question 1

How could we achieve the saving and restoration of a partially-computed notebook?

5.1.1 Work 1

D. Wannipurage, S. Marru and M. Pierce, "A Framework to capture and reproduce the Absolute State of Jupyter Notebooks", *PEARC 2022 Conference Series - Practice and Experience in Advanced Research Computing 2022 - Revolutionary: Computing, Connections, You*, 2022, DOI: 10.1145/3491418.3530296.17

In this paper, the authors present novel solutions to facilitate the capturing of complete Jupyter notebook state, whilst navigating the challenges for making computational notebooks fully reproducible. The authors' remark that while significant advances have been made to making Jupyter notebooks highly accessible and configurable to a user's needs, very little attention has been placed on increasing the interoperability of Jupyter notebooks across various platforms. In light of these discoveries, the authors appeal to the relevance of their work by detailing the potential use cases of reproducible notebooks–from sharing educational materials to reproducing artifacts from scientific publications. To accomplish the objective of absolute notebook state capture, the authors utilised mechanisms contained within the Jupyter standard library and the IPython kernel to create a savable archive of a running notebook–capturing all of the information required to reproduce notebook state in its entirety. As the next logical step to their solution, the authors then proceeded to detail the means for the packaging and exportation of notebook state for reproduction across other systems. Ultimately, the authors performed a variety of stress tests across a range of different workloads to identify performance bottlenecks, highlighting that I/O operations were the least impacted by and CPU operations were most impacted. The paper expertly details not only the procedure for state exportation and restoration of Jupyter notebooks, but extends to deliver a well-developed implementation. The work conducted by the authors will prove to be highly valuable to researchers with a desire to improve upon reproducibility with Jupyter notebooks.

5.1.2 Work 2

M. Juric, S. Stetzler and C. T. Slater, "Checkpoint, Restore, and Live Migration for Science Platforms", 2021. ArXivID: 2101.05782v1

This article demonstrates a fully-functional implementation of check-pointing, restoration, and live migration for Jupyter notebook environments. *Check-pointing* refers to the capability of a program to "freeze" and save to storage the running state of its processes. When applied to the context of Jupyter notebooks, it can capture a snapshot of a user's session (variables, functions, and imports) to permanent storage. Calling to the fact that operating remote analysis paradigms can quickly become prohibitively expensive, they present their solution: a means to freeze an active Jupyter notebook server to persistent storage, and efficiently restore it to memory on-demand. The authors promise that their solution could serve to reduce operating costs for providers, whilst continuing to support existing user experience levels. With *Elsa*, the user is presented with a "pause" button situated in the notebook toolbar which, upon interaction, saves the notebook's state and frees computational resources. Thereafter, users can restore the session to resume computation on the original virtual machine or a new one with a different configuration. The authors provide sound rationale for their design decisions–all in the interest of improving reuse, interoperability, and ease of setup across various systems. Ostensibly, this work represents the first of its kind–the first working checkpoint-restore-migrate (CRM) solution available for Jupyter notebook environments. The authors have not only demonstrated that CRM is possible, but made a case for its cost-saving potential for vendors and practitioners. To those researchers motivated by advancing versioning with Jupyter notebooks, this work may prove to be exemplary.

5.1.3 Work 3

J. F. Pimentel, L. Murta, V. Braganholo and J. Freire, "noWorkflow: a tool for collecting, analyzing, and managing provenance from Python scripts", *Proceedings of the VLDB Endowment*, 2017, DOI: 10.14778/3137765.3137789

In this article, the authors showcase a novel and open-source solution to address the ever-present issue of collecting provenance from a variety of Python environments. The authors begin with an appeal to the relevance of provenance in scientific experiments: it captures both the steps and data from which a resultant output was derived, allowing scientists to analyse and draw conclusions upon the entire computational process. The authors then go on to detail some of the potential the use cases of provenance-based workflows in computation–from verifying inconclusive results to managing experiment evolution. With their solution, *noWorkflow*, the authors seek to demonstrate the differing levels of specificity at which provenance can be collected. There are three general challenges associated with collecting provenance: (i) selecting the appropriate level of granularity, (ii) processing scripts with a large set of control structures, and (iii) considering the implications of the fact that scripts aren't run in controlled environments. In light of this analysis, the authors detail how noWorkflow addresses these challenges by collecting provenance in

three forms: definition, deployment, and execution–capturing the script structure, execution environment, and execution log, respectively. In demonstrating their solution, the authors pose a hypothetical experiment. Start to finish, the authors thoroughly detail the inner workings of noWorkflow–from provenance collection to eventual provenance analysis and management. While the authors have not explored potential avenues for future work, they have presented an expert discussion on the implementation of provenance tools and their implications within scientific computing. The work presented here would appeal greatly to researchers with an interest in improving reproducibility of their own works through provenance-based solutions.

5.2 Research question 2

How could we achieve the lightweight sharing of a notebook–including all dependencies–with minimal expectations on the receiver?

5.2.1 Work 1

S. Chattopadhyay, I. Prasad, A. Z. Henley, A. Sarma and T. Barik, "What's Wrong with Computational Notebooks? Pain Points, Needs, and Design Opportunities", *Conference on Human Factors in Computing Systems - Proceedings*, 2020, DOI: 10.1145/3313831.3376729

Through the use of mixed interview modes–semi-structured interviews and surveys–this article reports on nine ever-present pain-points that data scientists face when attempting to interact with notebooks. While previous works have explored particular pain points, this work seeks to act as a supporting taxonomy. The authors begin with the remark that as data scientists seek to coordinate more of their activities by way of computational notebooks, they encounter more and more difficulties at every step along the way–from initial setup to deployment. Moreover, while numerous workarounds were reported, they were essentially "band-aid" fixes–ad-hoc and error-prone measures which often required manual-intervention. In their findings, they report that one of the key features that data scientists are calling for is dependency management. At present, data scientists have no functionality to update, remove, or even identify deprecated packages. To further complicate matters, oftentimes, it isn't even possible to obtain a record of installed packages from the notebook environment–forcing upon users the drudgery scouring console output for relevant commands and outputs. While the paper serves as an excellent taxonomy on the current state of using Jupyter notebooks, it ultimately fails to provide a much-needed *solution* to the issue of dependency management. However, the paper could still prove to be a highly valuable resource to those informing the design and implementation of future computational workbook solutions.

5.2.2 Work 2

J. Wang, L. Li and A. Zeller, "Restoring Execution Environments of Jupyter Notebooks", *International Conference on Software Engineering - Proceedings*, 2021, DOI: 10.1145/3313831.3376729

This article reports on a novel solution of managing missing dependencies in Jupyter notebooks. The authors begin by highlighting the salient finding from recent studies that Jupyter notebooks oftentimes can't be executed by users, only read. The authors then discuss that the primary reason is that Jupyter notebooks are often highly-dependent on the environment in which they were originally authored. This is further compounded by the fact that the primary users of Jupyter notebooks–data scientists, *not* software engineers–are not familiar with the principles which constitute effective dependency management. Thus, the vast majority of notebooks don't state external dependencies, and of those which do, a sizable portion of them are unreliable. Resultantly, users with a desire to interact with these notebooks are often halted in their efforts by way of missing packages or incompatible software versions. The solution, *SnifferDog*, upon receipt of a Jupyter notebook as input, determines and installs the necessary packages, verifying if the notebook is executable and reproductive of the original results. In their evaluation, the authors demonstrate that, in addition to being a tool that data scientists have long since needed, *SnifferDog* is also highly performant. In this paper, the authors make significant improvements concerning the ever-present issue of restoring execution environments of Jupyter notebooks. With an extensive description of future work, not only do the authors demonstrate strong dedication and admirable commitment to advancing the scope of their work, they also better position interested researchers to succeed in their works.

5.2.3 Work 3

C. Zhu, R. K. Saha, M. R. Prasad and S. Khurshid, "Restoring the Executability of Jupyter Notebooks by Automatic Upgrade of Deprecated APIs", *36th IEEE/ACM International Conference on Automated Software Engineering - Proceedings*, 2021, DOI: 10.1109/ASE51524.2021.9678889

In this article, the authors address the challenge of managing deprecated dependencies in the context of Jupyter notebooks, utilising a machine-learning- and artificial-intelligence-driven approach. The primary motivation for the research and development detailed in this paper is the salient insight that the vast majority of publically-available Jupyter notebooks cannot be executed immediately after download. Specifically, the authors seek to address the abundant usage of deprecated APIs found within Jupyter notebooks. By upgrading deprecated APIs, *RELANCER* restores broken Jupyter notebooks. Employing an iterative approach driven by the occurrence of runtime errors, RELANCER identifies and fixes deprecation issues one package at a time. However, the rather less-than-satisfactory performance of RELANCER is made bare in subsequent analyses conducted by the authors. On a sample set of 255 unexecutable notebooks, their solution could only restore exe-

cutability of just 56% of the inputs. It should be noted, however, that RELANCER is still significantly more performant than similar, but not machine-learned models. This suggests that the rather poor performance *could* be suggestive of the difficulty of the task, rather than any incompleteness present in the solution. While more research and development evidently needs to be undertaken to make RELANCER a more viable solution, the article nevertheless provides valuable insight in regard to addressing Jupyter notebook dependency management. The article could have particular relevance to researchers seeking to revitalise older or legacy code in the form of Jupyter notebooks.

5.3 Research question 3

How could we achieve the lightweight sharing of a notebook–including all dependencies–with minimal expectations on the receiver?

5.3.1 Work 1

A. Y. Wang, A. Mittal, C. Brooks and S. Oney, "How data scientists use computational notebooks for real-time collaboration", *Proceedings of the ACM on Human-Computer Interaction*, 2019, DOI: 10.1145/3359141.17

In this article, the authors report on how real-time collaboration in the context of computational notebooks impacts the way in which data scientists work together and propose a multitude of implications to further support teamwork among data scientists. The authors remark that the ever-increasing complexity of data science work has prompted the need to facilitate collaborative environments for data scientists and end users to interact with one another. *Jupyter Notebook*, one such notebook, provides numerous functionalities which enable collaboration between varying end-users. However, despite such features, the authors then go on to recall that prior studies have uncovered numerous challenges associated with the usage of computational notebooks. They also point to further studies which have explored ways to lower the barriers for collaboration. Google's attempt to address the issue, *Colab*, introduced synchronous editing, a feature which promises to revolutionise collaboration in data science. However, the authors ground the reader in stating that synchronous editing inherently presents multiple issues. In the context of programming, the largest issue is the potential for programmers interfere with each other's code. Curious to observe the implications on data science, the authors proceeded to conduct various studies. In their findings, they conclude that synchronous editing establishes common ground between data scientists, however, the added complexity of having to be strategic in the means of collaboration may disincentivise its usage, and consequently, its initial implementation. In wake of these findings, the authors then go on to highlight that perhaps local version control could be a potential solution in assisting collaborators to track each others' edits. While the authors themselves don't present any implementations to support their solutions, their work nonetheless provides strong qualitative and quantitative analysis in regard to collaboration. Those seeking to implement or improve upon collaborative tools within

computational notebooks would derive great use from this work.

5.3.2 Work 2

M. B. Kery and B. A. Myers, "Interactions for untangling messy history in a computational notebook", *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing*, 2018, DOI: 10.1109/VLHCC.2018.8506576

Here, the authors present, *Verdant*, a novel solution to address the problem of version control in the context of Jupyter notebooks–supporting automated and absolute notebook versioning. The authors begin by offering insight in regard to why versioning in data science doesn't exist in the same manner that it does for programming. They detail that the only artifact a programmer typically interacts with is code. However, data scientists have to interact with a *range* of different artefacts: code, data and parameters, visualisations, tables, text output from code cells, and lastly, any notes written during experimentation. Evidently, the matter of versioning with notebooks is inherently more complex. At present, the only "solutions" presented to data scientists are ad hoc and require manual implementation. In evaluating their solution, the authors conducted usability tests on data science practitioners involving simulated data analysis tasks and historical data. Ultimately, the results indicated that not only is Verdant impactful in its own right, it also extends upon functionalities offered by existing VCSs like Git–implementing novel features. While the performance of usability testing furthers the credibility of the authors' work, conducting it over such a short period of time (only a few hours) has limitations in depicting its ultimate usability as users utilise VCSs over the course of days, weeks, and even months. However, this would strongly appeal to researchers and even developers with an interest in implementing versioning in Jupyter notebooks.

5.3.3 Work 3

A. Head, F. Hohman, T. Barik, S. M. Drucker and R. DeLine, "Managing messes in computational notebooks", *Conference on Human Factors in Computing Systems - Proceedings*, 2019, DOI: 10.1145/3290605.3300500

In this work, the authors showcase numerous code-gathering tools to address the consequences of increasingly messy computational notebook workflows which arise from long-running analyses. Interestingly, the authors begin with a remark that the flexibility offered by computational notebooks can also manifest into the writing of messy code, stating that the design of these notebooks results in three types of messes: (i) *disorder*: the requirement to run cells non-linearly, (ii) *deletion*: the overwriting/deletion of a cell, while its effect is still recorded by the kernel, and (iii) *dispersal*: the generation of a result which permeates to multiple cells. To address these issues, the paper contributes the design and implementation of a set of code-gathering programs. Given a notebook and its resultant outputs, their solution works by slicing the set of code cells to obtain the minimal slices required to reproduce its outputs, ordering said slices in a linear-fashion. In their evaluation,

the authors conducted a 12-person in-house usability with professional analysts to examine the impact of their tools in the computational workflows of data science practitioners. Their findings report that their tools showed promising levels of engagement, despite the short duration of the study. The relatively small usability test sample size raises concerns over usability to more general audiences. For example, *What about its impact upon undergraduate computer science students?*–end users with relatively little technical background, if at all. Considering the set of use cases for their tools could reasonably encompass data science education, a usability test involving such individuals would detail its performance in a wider variety of contexts. With that in mind, their work would present great appeal to researchers and other individuals with an interest in improving the shareability of computational notebooks–cleaning messy notebooks prior to their dissemination.

Bibliography

(1) U. of Sydney, *Conferences and NTROs*, Website, Accessed on: 2023-03-19.

(2) ACM and IEEE, *ICSE Conference - Home*, Website, Accessed on: 2023-03-19.

(3) ACM, *ESEC/FSE Conference - Home*, Website, Accessed on: 2023-03-19.

(4) ACM, *SPLASH Conference - Home*, Website, Accessed on: 2023-03-19.

(5) ACM and IEEE, *ASE Conference - Home*, Website, Accessed on: 2023-03-19.

(6) ACM, *CHI Conference - Home*, Website, Accessed on: 2023-03-19.

(7) ACM, *CHI Conference - Home*, Website, Accessed on: 2023-03-19.

(8) ACM, *ACM Transactions on Software Engineering and Methodology - Home*, Website, Accessed on: 2023-03-19.

(9) IEEE, *IEEE Xplore: IEEE Transactions on Software Engineering*, Website, Accessed on: 2023-03-19.

(10) *A case for double-blind reviewing in software engineering*, Website, Accessed on: 2023-03-24.

(11) *ACM Peer Review Policy*, Website, Accessed on: 2023-03-19.

(12) *About the Peer Review Process - IEEE Author Center Journals*, Website, Accessed on: 2023-03-19.

(13) CSRankings, *CSRankings: Computer Science Rankings*, Website, Accessed on: 2023-03-19.