Exploring Code Comprehension in Scientific Programming: Preliminary Insights from Research Scientists

Alyssia Chen, Carol Wong
University of Hawai'i at Mānoa
Honolulu, Hawai'i, USA
abc8@hawaii.edu, carolw8@hawaii.edu

Bonita Sharif
University of Nebraska - Lincoln
Lincoln, Nebraska, USA
bsharif@unl.edu

Anthony Peruma
University of Hawai'i at Mānoa
Honolulu, Hawai'i, USA
peruma@hawaii.edu

***Abstract*—Scientific software, defined as computer programs, scripts, or code used in scientific research, data analysis, modeling, or simulation, has become central to modern research. However, there is limited research on the readability and understandability of scientific code, both of which are vital for effective collaboration and reproducibility in scientific research. This study surveys 57 research scientists from various disciplines to explore their programming backgrounds, practices, and the challenges they face regarding code readability. Our findings reveal that most participants learn programming through self-study or on-the-job training, with 57.9% lacking formal instruction in writing readable code. Scientists mainly use Python and R, relying on comments and documentation for readability. While most consider code readability essential for scientific reproducibility, they often face issues with inadequate documentation and poor naming conventions, with challenges including cryptic names and inconsistent conventions. Our findings also show low adoption of code quality tools and a trend towards utilizing large language models to improve code quality. These findings offer practical insights into enhancing coding practices and supporting sustainable development in scientific software.**

I. Introduction

Scientific software, which includes computer programs, scripts, algorithms, code, or models, plays a vital role in advancing scientific research [1]. Advancements in tools, Integrated Development Environments (IDEs), and software frameworks and libraries have made the development of scientific software more accessible to researchers across various fields [1], [2]. With scientists spending approximately 30% to 35% of their time developing software to support their research ([3], [4]), the quality of scientific code has become vital for research sustainability, reproducibility, and collaboration.

These challenges connect directly to program comprehension, a core practice in software development ([5]), where programmers can spend 58% of their time dedicated to analyzing code to understand its functionality for tasks such as bug fixing or feature changes [6]. This importance of code quality in enhancing comprehension has been extensively studied by researchers, focusing on various factors such as code structure, the semantics and structure of identifier names, and comments, among other characteristics [7]–[13]. However, these studies have primarily focused on general-purpose or consumer-oriented software systems [14].

Even though scientific programs are written in the same programming languages as other software systems ([4], [15]), they present distinct comprehension challenges. Scientific programs are written by scientists who prioritize domain knowledge over established software development practices [16]. Further, development environments like Jupyter Notebooks, while making programming more accessible through integrated live code, narrative text, and visualizations, introduce their own challenges: non-linear execution flows, hidden states, lack of modularity, contains redundant code, and organizational issues [15], [17]–[20]. Moreover, empirical studies show that Jupyter Notebooks tend to have more quality issues than traditional Python scripts, with higher rates of PEP8 violations, more stylistic problems, higher coupling, and often suffer from reproducibility challenges [18], [21]–[23].

A. Study Goal

While these prior studies demonstrate that scientific programs are not immune from code comprehension challenges, there is limited knowledge in our understanding of how scientific programmers themselves perceive and address these issues. *To bridge this gap, in this study, we survey research scientists across various disciplines, aiming to understand their perspectives on code comprehension, their current practices, and the challenges they face in maintaining readable and understandable scientific code.* We envision that this early-stage research will serve as a foundation for more comprehensive studies into the factors influencing code comprehension in scientific programming, potentially leading to the development of new tools and practices tailored for scientific programmers.

B. Contributions

The main contributions of this work are:

- Insights from scientific programmers conducting research across multiple disciplines regarding their approach to code comprehension in practice.
- Empirical evidence identifying the key challenges in scientific code comprehension, including insufficient documentation, poor identifier naming, and limited use of code quality tools, with a trend towards the adoption of AI-based code generation tools.
- Findings that serve as a foundation for future research in scientific code comprehension by identifying multiple research directions that aim to develop targeted solutions

tailored to the specific needs of scientific programmers and improve code quality in research software.

II. Study Design

This section describes our methodology.

A. Survey Design

We used the Qualtrics platform [24] to design a 20-question survey aimed at collecting data on participants' experiences and perceptions of code readability in scientific programming. The questions were developed based on the objectives of our study and a review of relevant literature. Additionally, following best practice [25], we conducted a pilot run with three research scientists, adjusting the questionnaire based on their feedback. The survey included a mix of single-choice, multiple-choice, and free-text questions. Due to space constraints, the complete survey instrument is not included in this paper. However, to ensure reproducibility and transparency, all relevant materials are available in our artifact package at [26].

B. Survey Participants

Although this is an exploratory study, we aimed to obtain a diverse set of participants. Hence, we contacted researchers in multiple departments within the University of Hawai'i at Mānoa via email, some of whom forwarded our invitation or referred us to their colleagues or lab members. This combination of convenience and snowball sampling allowed us to identify qualified individuals [27]. Participation was voluntary and anonymous, and no compensation was provided. We contacted 600 individuals and received 112 responses. To ensure consistency in our analysis of results, we only considered participants who answered all mandatory questions, resulting in 57 valid responses for analysis.

C. Data Analysis

We analyzed the data using quantitative and qualitative techniques [28]. For the quantitative analysis, we used descriptive statistics, particularly for the single- and multi-choice questions. For the open-ended responses, we conducted a systematic thematic analysis. Three authors independently reviewed the data to identify and categorize key concepts, resolving any coding disagreements through consensus meetings.

III. Results

We received 57 completed responses (all mandatory questions answered). Due to space constraints, in some parts of the writeup, we only report on the frequent observations. Our artifact package ([26]) contains the complete dataset and the thematic coding of free-text responses.

***RQ1**: What are the backgrounds and extent of programming experiences of research scientists?*

Motivation. This RQ explores the educational qualifications and programming experience of research scientists. This knowledge provides insight into how scientists acquire programming skills and the extent of their programming expertise. The findings can help identify gaps in programming education and explain their software development practices and the challenges they face in code understandability and maintainability.

General Background: Most participants (51, or 89.47%) hold PhD degrees, followed by five with Master's degrees and one with a Bachelor's degree. We identify 35 unique majors among the 57 participants, reflecting a diverse range of educational specializations. Specifically, 11 participants have an economics background, six come from computer science, and three have a background in physics. The most common professions include professors, with 41 participants (71.93%), followed by six graduate students and four research scientists. When asked about their primary domains of work, the top three responses are 11 participants (12.50%) working in Economics and 10 each in Computer Science and Environmental Science.

Programming Experience: Python and R are the two most popular programming languages participants use for writing scientific programs, with 31 (27.93%) reporting Python and 29 (26.13%) reporting R. In contrast, languages such as C# and Java are reported by four and two participants, respectively. Since this is a multiple-choice question, the average number of languages reported by participants is 1.94, with Python and R being the common combination, occurring 14 times (18.42%). Further, participants report using statistical software such as Stata and SAS. Regarding preferred coding environments, most participants (46 or 43.81%) favor traditional IDEs (e.g., PyCharm, Visual Studio, etc.), followed by text editors (e.g., Sublime Text, Atom, Notepad++, etc.), which 23 participants utilize. Jupyter Notebooks and similar notebook interfaces/services are reported by 15 participants. On average, participants use 1.84 environments, with the most common pair being text editors and traditional IDEs, reported 14 times.

When asked where they initially learned to program, the top three answers are self-taught through online articles and blogs (31 responses or 22.30%), on-the-job training or learning (29 or 20.86%), and formal computer science courses at a university (23 or 16.55%). On average, participants select 2.44 options for this multiple-choice question, with the most common pair being on-the-job training or learning and being self-taught through online articles and blogs, which occurs 19 times. Moving on, 33 participants (57.9%) report that they have not received any education or training in writing readable and maintainable code. Among those who do receive such education/training, an analysis of free-text responses shows that most participants learn through formal education, while others acquire their knowledge from on-the-job experience, workshops, or peers. The topics covered in their education/-training include coding best practices, documentation, testing practices, and tools designed to enhance code quality.

***RQ2**: What are the practices, challenges, and perceptions regarding code understandability in scientific programming?*

Motivation. Code comprehension is an essential activity in software development and maintenance. This RQ identifies the specific practices scientists employ to write understandable code and the obstacles they encounter when comprehending scientific code. The findings provide insight into crucial areas

TABLE I
TOP FIVE FREQUENT CHALLENGES COMPREHENDING SCIENTIFIC CODE.

Challenge	Count	Percentage
Lack of or insufficient comments	44	16.18%
Lack of documentation (e.g., no README)	33	12.13%
Poor naming of methods/functions, variables, etc.	31	11.40%
Poor organization of project structure	31	11.40%
Hardcoded values without explanation	24	8.82%

TABLE II
TOP FIVE FREQUENT CHALLENGES WITH IDENTIFIER NAMES.

Challenge	Count	Percentage
Names that are too short or cryptic (e.g., single letters abbreviations or acronyms)	40	21.28%
Inconsistent naming conventions and styles	30	15.96%
Names that don't accurately reflect their purpose	29	15.43%
Use of generic names (e.g., 'data', 'result', 'temp')	29	15.43%
Use of names that are too similar	25	13.30%

to support scientists in writing understandable code, leading to improved reproducibility of scientific results.

Practices: We asked participants, using a free-text question, what practices they frequently employ to improve code readability. An analysis of their responses shows that documentation practices, particularly code commenting, is a common approach (63.16% of responses), while 15.79% of participants report using multiple practices, including documentation, modular programming, and naming conventions. The remaining responses were distributed across specific focuses, such as using AI tools (e.g., ChatGPT) and refactoring. When surveyed about their use of automated code quality tools to improve code readability, 49.12% of participants indicate they 'Never' use them. Among those who do, many utilize AI and large language models (LLMs), particularly ChatGPT and Claude, to enhance code readability, accounting for 41.38% of responses. Approximately 24% mention using a combination of tools, including IDEs, LLMs, and online resources. The remaining responses focus on specific tools, with IDE features (e.g., refactoring and styling) used by 14% of participants.

Challenges: Through a Likert question, 54.55% of participants report that they 'Sometimes' encounter difficulties in understanding scientific software source code written by others, while 38.18%, 5.45%, and 1.82% select 'Often,' 'Rarely,' and 'Always,' respectively. Analyzing the challenges participants frequently encounter, we observe (as shown in Table I) that a lack of documentation in the form of code comments and project-level documentation (e.g., READMEs) are the top two challenges reported by 44 and 33 participants, respectively, closely followed by poor identifier naming and project structure, each with 31 instances. On average, participants select 4.77 challenges in this multi-choice question. Three challenges that frequently occur together (18 occurrences) are lack of project documentation, insufficient comments, and poor naming of methods/variables.

Perception on Reproducibility: All participants acknowledge the importance of code readability for ensuring reproducible scientific results, as indicated by a Likert-scale question. Specifically, 26 participants (45.16%) find it 'Very important,' while 22 participants (38.60%) rate it as 'Extremely important,' Five participants report it as 'Slightly important,' and four consider it 'Moderately important.' None of the participants select the option 'Not important at all.'

RQ3*: What is the importance of identifier names in scientific software readability, and what challenges arise in this area?*

Motivation. High-quality identifier names are crucial for understanding the correct behavior of code [29]. This RQ investigates scientists' views on identifier names and identifies the specific naming challenges they face with scientific code. The findings can guide the creation of automated tools to enhance naming practices in scientific coding.

Importance: Our 5-point Likert scale question regarding the importance of identifier names in making code readable shows that all participants recognize the importance of names. Specifically, 25 participants (43.86%) consider them 'Very important,' while 17 (29.82%) view them as 'Extremely important,' and 13 (22.81%) rate them as 'Moderately important.' Only two participants rate it as 'Slightly important,' and none select 'Not important at all.'

Challenges: When asked about the frequency at which they experience misunderstanding or errors in the code due to identifier names, 32 (or 56%) participants reported 'Sometimes,' while eighteen, five, and two participants reported 'Often,' 'Rarely,' and 'Never,' respectively. To gain more insight into their challenges, participants were presented with a multi-choice question regarding the common challenges they encounter with identifier names. As shown in Table II, the two most common issues were names that were too short or cryptic and names that followed inconsistent naming conventions, as reported by 40 (21.28%) and 30 (15.96%) participants, respectively. Participants chose, on average, 3.3 options for this question, with the most common two-pair combination being names that are too short or cryptic and the use of generic names. Finally, through a free-text question, participants were asked to provide examples of naming issues they have encountered. An analysis of their responses revealed issues, such as ambiguous and misleading names, using similar or same names for different purposes, overly short names (e.g., single characters), and inconsistencies in naming styles.

IV. DISCUSSION

This preliminary study introduces a novel perspective on code comprehension in the context of scientific program-

ming. While previous work has extensively focused on code comprehension in traditional software development, our study forms the foundation for examining the specific practices and challenges with scientific code comprehension.

A. Key Findings and Their Implications

Extending programming education beyond syntax mastery: Our findings from RQ 1 support previous research showing that many scientists lack formal education in computer science or software engineering [30], [31]. Additionally, our results show that these scientists mainly develop their programming skills through self-study and practical experience, which aligns with earlier research [3]. However, our findings also offer additional insights showing that scientists typically lack an understanding of the best practices in writing maintenance-friendly code. These observations are important as they suggest potential risks to the long-term sustainability of research artifacts, which can include code misinterpretation, higher cognitive load, increased debugging time, and challenges in collaborative projects [32], [33]. These findings emphasize the importance of expanding programming education beyond just mastering syntax to include important software engineering practices. Hence, it is essential for academic and research institutions to implement training programs that focus on key software engineering concepts for scientists, establish coding standards for research software, and provide researchers with centralized resources for supporting code maintenance.

Need for focused program comprehension research on scientific programming: The study also highlights the scientific community's strong preference for Python and R over languages like Java and C#, aligning with prior research [4], [15]. Further, through RQ2, we identified code readability as a critical factor in scientific software, especially for ensuring reproducibility. RQ3 highlights the importance of high-quality identifier names in achieving this readability. These findings are of particular importance as prior research on code comprehension has predominantly concentrated on Java programs, examining aspects such as readability models, linguistic antipatterns, and identifier naming [7]–[9], showing that there is a clear need for targeted research that focuses on the unique aspects of scientific programming. Additionally, most research studies on code comprehension typically focus on code written in Java ([14]), further highlighting the need for research pertaining to the specific characteristics of scientific programming languages like Python and R. Finally, it is important for studies involving human subjects to broaden their recruitment beyond the traditional pool of computer science students and include researchers from various fields.

Need for better tool support and AI usage guidelines: Our RQ 2 findings reveal an interesting pattern in how scientific programmers approach code quality tools. While there is low adoption of traditional code quality tools (e.g., [34]), AI-powered tools, especially LLMs, are popular. This low adoption of traditional tools may indicate either a lack of knowledge among scientists or the inadequacy of these tools in addressing the specific needs of scientific programming and warrants further investigation. On the other hand, the shift toward LLMs mirrors trends in software engineering, where such AI-based tools are enhancing developer productivity [35], [36]. However, studies have shown that LLM-generated code can contain quality issues [37], [38]. This is particularly problematic in scientific programming, where many practitioners lack formal software engineering training. Hence, it is essential that they are equipped with the skills to critically evaluate AI-generated code [39], especially in a scientific context.

The documentation paradox: Our findings indicate that while documentation is the most commonly reported approach to improving code understandability, paradoxically, our findings also show that the lack of or insufficient comments and project documentation emerge as the top challenges in code comprehension. This contradiction presents multiple research opportunities, such as investigating what constitutes "sufficient" documentation across various scientific fields, understanding the gap between perceived and actual documentation practices, and examining its impact on reproducibility. Such studies can lead to improved documentation guidelines and document-generation tools for scientific code [19], [34].

V. Threats To Validity

We limited our survey participants to a single institute, which impacts the generalizability of our findings. Additionally, while we focused on individuals actively engaged in scientific research, our convenience and snowball sampling methods may have resulted in the exclusion of suitable candidates. Moreover, those who chose to respond to our survey might already prioritize code readability. To address potential subjectivity in our analysis of free-text responses, three authors independently categorized the responses and then reached a consensus.Finally, our research scope may not capture all aspects of code readability practices and challenges, posing a threat to the internal validity of our study. However, as this is an exploratory study, our goal is to identify preliminary trends and patterns that can guide future research in this area.

VI. Conclusion & Future Work

Readable and understandable code is vital for effective collaboration and reproducibility. Despite this importance, there is a lack of research on this topic in scientific programming.

This preliminary study begins to address this gap by surveying 57 research scientists across various disciplines. Our findings show that most scientists lack formal training in writing readable code, and while they recognize the importance of code readability for reproducibility, they face challenges with code comprehension. These challenges include insufficient documentation and poor identifier names, such as short and ambiguous names. Further, while a fair number never use automated code quality tools, those who do primarily rely on large language models, particularly ChatGPT. Our future work will investigate the relationship between identifier names and program behavior in scientific contexts, including the evolution of names over time and their impact on collaboration.

REFERENCES

[1] E.-M. Arvanitou, A. Ampatzoglou, A. Chatzigeorgiou, and J. C. Carver, "Software engineering practices for scientific software development: A systematic mapping study," *Journal of Systems and Software*, vol. 172, p. 110848, 2021. 1

[2] L. Vaughan, *Python Tools for Scientists: An Introduction to Using Anaconda, JupyterLab, and Python's Scientific Libraries*. No Starch Press, 2023. 1

[3] J. E. Hannay, C. MacLeod, J. Singer, H. P. Langtangen, D. Pfahl, and G. Wilson, "How do scientists develop and use scientific software?," in *2009 ICSE Workshop on Software Engineering for Computational Science and Engineering*, pp. 1–8, 2009. 1, 4

[4] P. Prabhu, T. B. Jablin, A. Raman, Y. Zhang, J. Huang, H. Kim, N. P. Johnson, F. Liu, S. Ghosh, S. Beard, T. Oh, M. Zoufaly, D. Walker, and D. I. August, "A survey of the practice of computational science," in *State of the Practice Reports*, SC '11, (New York, NY, USA), Association for Computing Machinery, 2011. 1, 4

[5] A. Von Mayrhauser and A. Vans, "Program comprehension during software maintenance and evolution," *Computer*, vol. 28, no. 8, 1995. 1

[6] X. Xia, L. Bao, D. Lo, Z. Xing, A. E. Hassan, and S. Li, "Measuring Program Comprehension: A Large-Scale Field Study with Professionals," *IEEE Transactions on Software Engineering*, vol. 44, no. 10, pp. 951–976, 2018. 1

[7] A. Sergeyuk, O. Lvova, S. Titov, A. Serova, F. Bagirov, E. Kirillova, and T. Bryksin, "Reassessing Java Code Readability Models with a Human-Centered Approach," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, ICPC '24, (New York, NY, USA), p. 225–235, Association for Computing Machinery, 2024. 1, 4

[8] V. Arnaoudova, M. Di Penta, and G. Antoniol, "Linguistic antipatterns: what they are and how developers perceive them," *Empirical Software Engineering*, vol. 21, pp. 104–158, Feb 2016. 1, 4

[9] G. Li, H. Liu, and A. S. Nyamawe, "A Survey on Renamings of Software Entities," *ACM Comput. Surv.*, vol. 53, Apr. 2020. 1, 4

[10] C. D. Newman, R. S. Alsuhaibani, M. J. Decker, A. Peruma, D. Kaushik, M. W. Mkaouer, and E. Hill, "On the Generation, Structure, and Semantics of Grammar Patterns in Source Code Identifiers," *CoRR*, vol. abs/2007.08033, 2020. 1

[11] D. Lawrie, C. Morrell, H. Feild, and D. Binkley, "What's in a Name? A Study of Identifiers," in *14th IEEE International Conference on Program Comprehension (ICPC'06)*, pp. 3–12, 2006. 1

[12] A. Kaur, "A Systematic Literature Review on Empirical Analysis of the Relationship Between Code Smells and Software Quality Attributes," *Archives of Computational Methods in Engineering*, vol. 27, pp. 1267–1296, Sep 2020. 1

[13] K. Park, J. Johnson, C. S. Peterson, N. Yedla, I. Baysinger, J. Aponte, and B. Sharif, "An eye tracking study assessing source code readability rules for program comprehension," *Empir. Softw. Eng.*, vol. 29, no. 6, p. 160, 2024. 1

[14] M. Wyrich, J. Bogner, and S. Wagner, "40 Years of Designing Code Comprehension Experiments: A Systematic Mapping Study," *ACM Comput. Surv.*, vol. 56, Nov. 2023. 1, 4

[15] 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*, CHI '18, (New York, NY, USA), p. 1–12, Association for Computing Machinery, 2018. 1, 4

[16] J. Segal, "Scientists and software engineers: A tale of two cultures," in *PPIG 2008: Proceedings of the 20th Annual Meeting of the Psychology of Programming Interest Group*, Lancaster University, 2008. 1

[17] A. Head, F. Hohman, T. Barik, S. M. Drucker, and R. DeLine, "Managing Messes in Computational Notebooks," in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, (New York, NY, USA), p. 1–12, Association for Computing Machinery, 2019. 1

[18] K. Adams, A. Vilkomir, and M. Hills, "A Comparison of Machine Learning Code Quality in Python Scripts and Jupyter Notebooks," *J. Comput. Sci. Coll.*, vol. 39, p. 96–108, Nov. 2023. 1

[19] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "Understanding and improving the quality and reproducibility of Jupyter notebooks," *Empirical Software Engineering*, vol. 26, p. 65, May 2021. 1, 4

[20] 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*, CHI '20, (New York, NY, USA), p. 1–12, Association for Computing Machinery, 2020. 1

[21] J. Wang, L. Li, and A. Zeller, "Better code, better sharing: on the need of analyzing jupyter notebooks," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: New Ideas and Emerging Results*, ICSE-NIER '20, (New York, NY, USA), p. 53–56, Association for Computing Machinery, 2020. 1

[22] K. Grotov, S. Titov, V. Sotnikov, Y. Golubev, and T. Bryksin, "A large-scale comparison of Python code in Jupyter notebooks and scripts," in *Proceedings of the 19th International Conference on Mining Software Repositories*, MSR '22, (New York, NY, USA), p. 353–364, Association for Computing Machinery, 2022. 1

[23] J. F. Pimentel, L. Murta, V. Braganholo, and J. Freire, "A Large-Scale Study About Quality and Reproducibility of Jupyter Notebooks," in *2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)*, pp. 507–517, 2019. 1

[24] "Qualtrics XM: The Leading Experience Management Software." https://www.qualtrics.com/. 2

[25] J. Linåker, S. M. Sulaman, R. M. de Mello, and M. Höst, "Guidelines for conducting surveys in software engineering," 2015. 2

[26] "Artifact Package." https://doi.org/10.5281/zenodo.14676939. 2

[27] S. Baltes and P. Ralph, "Sampling in software engineering research: a critical review and guidelines," *Empirical Software Engineering*, vol. 27, p. 94, Apr 2022. 2

[28] S. Wagner, D. Mendez, M. Felderer, D. Graziotin, and M. Kalinowski, *Challenges in Survey Research*, pp. 93–125. Cham: Springer International Publishing, 2020. 2

[29] A. Schankin, A. Berger, D. V. Holt, J. C. Hofmeister, T. Riedel, and M. Beigl, "Descriptive compound identifier names improve source code comprehension," in *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, (New York, NY, USA), p. 31–40, Association for Computing Machinery, 2018. 3

[30] D. Leroy, J. Sallou, J. Bourcier, and B. Combemale, "On the Role of Computer Languages in Scientific Computing," *Computing in Science & Engineering*, vol. 24, no. 4, pp. 55–59, 2022. 4

[31] T. L. De Santana, P. A. D. M. S. Neto, E. S. De Almeida, and I. Ahmed, "Bug Analysis in Jupyter Notebook Projects: An Empirical Study," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, Apr. 2024. 4

[32] Z. Soh, A. Yamashita, F. Khomh, and Y.-G. Guéhéneuc, "Do Code Smells Impact the Effort of Different Maintenance Programming Activities?," in *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016. 4

[33] S. Fakhoury, Y. Ma, V. Arnaoudova, and O. Adesope, "The effect of poor source code lexicon and readability on developers' cognitive load," in *Proceedings of the 26th Conference on Program Comprehension*, ICPC '18, (New York, NY, USA), p. 286–296, Association for Computing Machinery, 2018. 4

[34] A. Rule, A. Birmingham, C. Zuniga, I. Altintas, S.-C. Huang, R. Knight, N. Moshiri, M. H. Nguyen, S. B. Rosenthal, F. Pérez, and P. W. Rose, "Ten simple rules for writing and sharing computational analyses in Jupyter Notebooks," *PLOS Computational Biology*, 07 2019. 4

[35] J. Sauvola, S. Tarkoma, M. Klemettinen, J. Riekki, and D. Doermann, "Future of software development with generative AI," *Automated Software Engineering*, vol. 31, Mar 2024. 4

[36] J. T. Liang, C. Yang, and B. A. Myers, "A Large-Scale Survey on the Usability of AI Programming Assistants: Successes and Challenges," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ICSE '24, (New York, NY, USA), Association for Computing Machinery, 2024. 4

[37] C. Dantas, A. Rocha, and M. Maia, "Assessing the Readability of ChatGPT Code Snippet Recommendations: A Comparative Study," in *Proceedings of the XXXVII Brazilian Symposium on Software Engineering*, SBES '23, (New York, NY, USA), p. 283–292, Association for Computing Machinery, 2023. 4

[38] M. L. Siddiq, L. Roney, J. Zhang, and J. C. D. S. Santos, "Quality Assessment of ChatGPT Generated Code and their Use by Developers," in *Proceedings of the 21st International Conference on Mining Software Repositories*, MSR '24, (New York, NY, USA), p. 152–156, Association for Computing Machinery, 2024. 4

[39] Y. Liu, T. Le-Cong, R. Widyasari, C. Tantithamthavorn, L. Li, X.-B. D. Le, and D. Lo, "Refining ChatGPT-Generated Code: Characterizing and Mitigating Code Quality Issues," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, June 2024. 4