Towards Developing a Comprehensive Methodical Framework for Evaluating Code Recommender Systems

Borst, Daniel

DOI:
10.57938/9f98cb58-fc8e-48c9-8343-42d666fdb24b

Published: 01/10/2024

Towards Developing a Comprehensive Methodical Framework for Evaluating Code Recommender Systems

Daniel Borst
daniel.borst@wu.ac.at
Institute for Complex Networks
Vienna University of Economics and Business
Vienna, Austria

01 October 2024

Abstract

The growing incorporation of code recommender systems is transforming software development, promising increased productivity, efficiency, and effectiveness through context-aware code suggestions. Despite these advancements, existing evaluation methods often fall short. They rely on computational metrics and synthetic benchmarks, failing to capture real-world software development environments' dynamic, human-centric nature. To enhance methods, techniques, and tools for evaluating code recommender systems, the aim is to create a new methodological framework. To achieve this goal, a multi-method approach will be employed, including a systematic literature review, along with user studies and online evaluations conducted in both a university setting and the automotive sector. By addressing practical challenges and involving human participants, the framework aims to thoroughly assess code recommender systems, ultimately providing empirical evidence of their impact on developers' productivity, efficiency, effectiveness, and perceived satisfaction. Furthermore, the research offers practical recommendations for designing more effective code recommender systems based on developers' needs and experiences.

1 Introduction

Recommender systems are information filtering systems designed to predict user preferences for a set of items. They have become omnipresent in various domains, such as e-commerce and social networks, providing personalized suggestions based on user preferences and historical data [1], [2]. A specialized subset of these systems, code recommender systems, aims to enhance software development by improving code quality and increasing programmers' productivity, efficiency, and effectiveness [3], [4]. These systems provide software developers with automated suggestions within the context of programming-language services, encompassing both syntactic aspects like static program analysis and semantic aspects such as automated code generation [5].

Code recommender systems have their roots in early programming tools of the 1960s, which were designed to assist developers with specific tasks throughout the software engineering life cycle [6]. Over time, these systems have evolved, with the incorporation of artificial intelligence (AI) techniques being one of the factors that have contributed to generating more accurate and context-aware recommendations [7], [8].

Recent advancements in large language models (LLMs) have further transformed code recommender systems [9], [10]. These advancements, along with other technological improvements, have enabled so-called *hybrid* code recommender systems that combine multiple recommendation strategies, such as collaborative filtering and content-based methods to enhance accuracy and robustness [11]. As a result, these systems can now perform more complex operations, including generating code from natural language descriptions, automating programming tasks, and fixing buggy code snippets [9], [10].

However, despite their advancements, it remains to be seen whether the evaluation of code recommender systems has kept pace, mainly due to a reliance on offline evaluation methods focusing on computational metrics and synthetic benchmarks [7], [10]. Software engineering, including programming, is considered a multidisciplinary and human-centric activity, involving complex social interactions [12]–[14]. Traditional evaluation methods often overlook these aspects by focusing solely on controlled and simulated settings, thereby failing to capture the behaviors of real-world actors (both people and systems) in natural environments. This discrepancy creates a gap between promised outcomes and actual results [7], [10]. Comprehensive evaluation methods involving human participants are critically needed to observe and characterize the quality attributes of these recommender systems [15], [16]. These methods may include user studies, where evaluations are conducted in live or laboratory settings with participants completing tasks defined by the researcher, as well as online evaluations, which involve field experiments where users perform self-selected tasks in real-world environments [15].

This PhD project aims to enhance methods, techniques, and tools for evaluating code recommender systems by reviewing existing approaches and developing a practical evaluation framework. It involves testing and refining this framework in studies to ensure applicability across diverse software languages, contexts and domains. Additionally, the research seeks to assess the impact of code recommender systems on developers' productivity, efficiency, effectiveness, and perceived satisfaction in software engineering.

The PhD project operates within an industry-academia collaboration research project named Hybrid Approach to Intelligent Recommenders for cyber-physical systems of systems (HybridAIR)[1]. This project is in partnership with AVL, one of

[1] https://hybridair.wu.ac.at/ [Accessed: June 26, 2024]

the world's leading mobility technology companies specializing in development, simulation, and testing within the automotive industry. The project's goal is to develop, validate, and apply systematic empirical approaches for designing, implementing, and evaluating a multi-stakeholder, context-aware recommender system tailored to the automotive industry, specifically for the testing sector. The code recommender system is built for a domain-specific language (DSL), a software language designed for a particular application domain [17], used by domain experts to describe failure trees, which help in error identification in automotive testbeds. The system must provide accurate, trustworthy, and explainable recommendations to various stakeholders, such as testbed developers and operators, considering the severe consequences of misinterpretations in this critical sector. This involves understanding the stakeholders' roles and needs, defining effective methods for presenting explainable and responsible recommendations using tools like LLMs and eXplainable AI, and establishing suitable evaluation methods, including participant selection, evaluation metrics, and data collection strategies, while mitigating potential risks.

2 Related Work

Several frameworks and methodologies have been developed to evaluate various aspects of software engineering tools, DSLs, and IT products. The following examples provide valuable insights and approaches that can inform and guide the development of an evaluation framework for code recommender systems.

Kahraman and Bilgen's framework offers a qualitative approach to evaluating the success and quality of DSLs from multiple stakeholder perspectives. The framework identifies specific assessment goals based on each perspective, which in turn determine relevant quality attributes such as functional suitability, usability, or maintainability. These quality attributes are weighted according to the assessment goals, and a minimum attribute level is defined to ensure baseline standards. The evaluation outcomes can be categorized as incomplete, satisfactory, or effective, providing a clear measure of the DSL's performance [18].

The USE-ME framework, developed by Barisic et al., integrates usability evaluation into the iterative development cycle of DSLs. By involving end-users and emphasizing usability from the start, USE-ME aims to ensure DSLs are user-friendly and effective. It includes the steps context modeling, goal modeling, evaluation modelling, survey modelling, interaction modelling, and report modelling, providing a structured approach to continuous usability improvement [19].

Lundell and Lings developed the 2G method, a systematic approach for creating evaluation frameworks for complex IT products, combining organizational data with state-of-the-art technological capabilities. It generates two distinct frameworks: a strategic evaluation framework that captures ideal tool characteristics based on long-term needs, and a pragmatic evaluation framework adapted to technological capabilities. This tailored, iterative, and data-driven approach ensures continuous and targeted refinement [20].

The DESMET framework, developed by Barbara Kitchenham et al., offers a comprehensive methodology for evaluating software engineering methods and tools to enhance decision-making and improve software engineering practices. It addresses the need for systematic evaluations by providing nine distinct methods, such as formal experiments, quantitative case studies, and qualitative screenings. Designed for various stakeholders, DESMET includes guidelines for selecting appropriate evaluation methods and emphasizes a multi-component approach to ensure robust and context-sensitive evaluations [21].

These frameworks highlight the importance of comprehensive, iterative, and stakeholder-centered evaluations that can be tailored to specific technologies and settings. They serve as examples of how similar approaches can be applied to the context of code recommender systems, guiding the development of a robust and adaptable framework that is theoretically grounded and practically relevant.

3 Research Questions

- **RQ1:** What are the prevalent evaluation methods and metrics[2] used to assess the productivity, efficiency, effectiveness, and perceived satisfaction with code recommender systems, and how are these methods and metrics distributed across different studies?

- **RQ2:** How can a framework and guidelines be developed for evaluating code recommender systems in real-world settings, ensuring practical applicability and relevance?

[2]A metric is a higher-level, subjective attribute often based on quantifiable, observable, and objective data [22].

- **RQ3:** What is the impact of code recommender systems on developers' productivity, efficiency, effectiveness, and perceived satisfaction in real-world settings?

4 Methodology

To ensure scientifically rigorous and practically relevant research, the Design Science Research Methodology (DSRM) proposed by Peffers et al. [23] will be employed. DSRM is a structured approach consisting of six steps: *Identify Problem & Motivate*, *Define Objectives of a Solution*, *Design & Development*, *Demonstration*, *Evaluation*, and *Communication*. An integral part of DSRM is the ability to cycle back to previous steps based on feedback and evaluation, ensuring continuous refinement and improvement of the developed artefact.

For RQ1, DSRM guides the process of identifying prevalent evaluation methods and metrics by first defining the problem and then setting clear objectives for understanding how productivity, efficiency, effectiveness, and perceived satisfaction are assessed in various contexts. This structured approach ensures that the gathered insights are comprehensive and relevant.

For RQ2, the *Design & Development* step of DSRM is crucial. This step will involve creating an initial framework and a set of guidelines based on the insights from RQ1. The iterative nature of DSRM, with its cycles of demonstration and evaluation, will allow for continuous refinement of this framework to ensure it is practical and applicable in real-world settings.

For RQ3, the *Demonstration* and the *Evaluation* step of DSRM provide a systematic way to test the framework and assess the actual impact of code recommender systems. By applying the developed framework in real-world environments and collecting feedback, the impact of these systems on developers' productivity, efficiency, effectiveness, and perceived satisfaction can be empirically validated. The flexibility to cycle back to previous steps ensures that any issues or insights discovered during this step can be addressed.

By following the DSRM, I can ensure that the developed framework is theoretically sound and practically relevant.

5 Research Plan

To address the research questions and achieve the objectives of this dissertation, a multi-method approach will be employed, consisting of a systematic literature review (SLR), user studies, and online evaluations.

5.1 Systematic Literature Review

For the steps *Identify Problem & Motivate* and *Define Objectives of a Solution*, a SLR will be conducted to identify and synthesize the prevalent evaluation methods and metrics used to assess the impact of code recommender systems on developers' productivity, efficiency, effectiveness, and perceived satisfaction (RQ1). The SLR will follow the guidelines proposed by Kitchenham et al. [24] and the procedural guideline for a Quasi-Gold Standard Corpus by Zhang et al. [25], ensuring a rigorous and replicable process. Prominent databases, such as IEEE Xplore, ACM Digital Library, and Scopus, will be thoroughly searched using a predefined search string. Inclusion and exclusion criteria will be applied to select relevant studies, which will then be analyzed to extract data on evaluation methods and metrics. Additionally, quality criteria will be defined, prioritizing evaluation approaches that consider the role of human programmers. During data extraction, publications will be categorized by evaluation method (e.g., offline evaluation, online evaluation), supported software language (e.g., DSL, modeling language), software engineering task (e.g., Requirements, Testing), and code recommender system operation (e.g., complete, create, find). During data analysis, a particular focus will be set on the evaluation methods and metrics that consider human factors in software engineering (i.e. online evaluations and user studies).

5.2 User Studies and Online Evaluations

During the *Design & Development* step, an initial framework for evaluating code recommender systems in real-world settings (RQ2) will be drafted. In the subsequent *Demonstration* step, a series of user studies and online evaluations will be conducted to test, refine, and extend the framework. The evaluations will be carried out in controlled environments, such as university programming courses. Through these evaluations, valuable insights will be gained into the framework's practical applicability and utility. This approach will help determine whether the selected methods and metrics are suitable for their specific evaluation contexts and capable of addressing the dy-

namic, human-centric nature of software development environments.

In the *Evaluation* step, the framework will be used to assess the actual impact of code recommender systems on developers' productivity, efficiency, effectiveness, and perceived satisfaction in real-world settings (RQ3). This will be achieved by conducting user studies and online evaluations in collaboration with the industry partner within the HybridAIR project. These studies will evaluate the multi-stakeholder, context-aware code recommender system developed as part of the project, by monitoring its usage and impact over an extended period.

The data collected from the studies will be analyzed using appropriate statistical and qualitative methods, offering a thorough assessment of the systems' impact. Additionally, the studies will provide valuable insights into the success of the evaluation framework, contributing to its final version.

6 Expected Results

This PhD project aims to deliver several contributions to the field of code recommender systems, addressing both theoretical and practical aspects.

Firstly, the SLR is expected to provide a comprehensive overview of the current state-of-the-art evaluation methods and metrics used to assess code recommender systems. By synthesizing findings from various studies, this review is anticipated to highlight prevalent practices, identify gaps, and reveal trends in evaluation approaches. Through categorizing the publications (see section 5.1) and analyzing the data, the SLR aims to offer an overview of evaluation methods and metrics that are suited to specific types of code recommenders and contexts of use. Additionally, in the SLR defined quality criteria are expected to highlight promising evaluation methods and metrics that focus on the human aspect in programming. This foundational knowledge will inform the subsequent development of an initial evaluation framework.

Given that different evaluation methods are suited to different situations and systems, selecting the right approach is crucial to capturing both the system's and the user's behaviors in their natural environment, thereby ensuring realistic results. Therefore the selection process should take into account not only technical aspects but also social and organizational requirements, while considering the specific perspectives and needs of evaluators or stakeholders [18], [20], [21].

It is anticipated to develop a tailored framework that guides users in evaluating code recommender systems. This framework is expected to include decision criteria for selecting appropriate evaluation methods, taking into account both technical aspects (e.g., software language, code recommender system operation) and social and organizational factors (e.g., work context, stakeholders' needs). Additionally, it is intended to provide guidelines for integrating relevant quantitative and qualitative metrics, with the goal of capturing the behaviors of both the system and its users. For each metric, the framework is expected to offer meaningful ways to measure them effectively. The framework is anticipated to provide two key benefits. First, it is expected to facilitate the evaluation of code recommender systems by providing guidance with clear instructions. Second, by tailoring each evaluation and focusing on appropriate methods and metrics, the resulting evaluations are expected to actually reflect real-life usage.

An example of how this framework might structure decision making on designing evalutions is illustrated in Figure 1. The framework divides decision making into the following three phases:

Preparation Phase: This phase is centered on establishing the evaluation setting, considering both technical and organizational criteria that must be clearly defined. The technical criteria may involve considerations such as the supported software languages, the specific software engineering tasks, and the operations of the code recommender system. For instance, the supported languages could range from general-purpose programming languages (e.g., Python, Java) to domain-specific languages (e.g., YAML, SQL), as well as modeling languages (e.g., UML, UML extensions). The software engineering task might cover essential stages like requirements, design, and testing, while the code recommender system's operations could include functionalities such as code completion, generation, and search. On the organizational side, the criteria might be including the specific work contexts (e.g., education, research, corporate) and the nature of the work processes involved (e.g., agile methodologies, plan-driven approaches). Defining these criteria is crucial, as they form the basis upon which the framework will guide both the design and execution of the evaluation.

Design Phase: This phase focuses on the design of the evaluation process. Building on the previously defined criteria, the framework will guide the following steps, including participant selection, the definition of quality characteristics and metrics, the specification of the evaluation context, and the choice of evaluation method. If a comparative evaluation is required,

the next step would involve selecting an appropriate baseline for comparison. For instance, participants may include professional developers, students, or hobbyists, depending on the evaluation's goals. Quality characteristics and metrics could encompass factors such as accuracy, efficiency, and user satisfaction. The evaluation context might vary from a controlled laboratory setting to a real-world software development project or an online coding platform. The evaluation methods may cover user studies and online evaluations. This comprehensive design process ensures that the evaluation is tailored to the recommender system, its context of use, and the evaluators' needs, thereby enabling the collection of meaningful data.

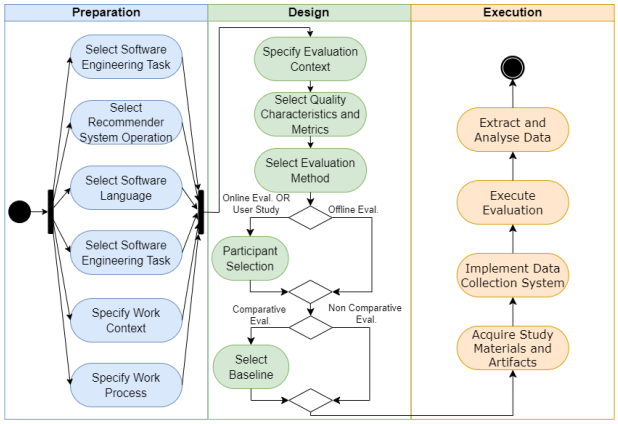

Figure 1: Exemplary Decision-Making Framework for Evaluating Code Recommender Systems inspired by Kahraman [18, p. 1512, Fig. 2] and Barisic [19, p. 135, Fig. 11].

Execution Phase: The final phase involves executing the evaluation and collecting and analyzing the data. This phase is where the designed evaluation plan is implemented, and real-world data is gathered to assess the system's performance and impact. For example, data collection methods might include automated logging of user interactions, surveys, or interviews. Data analysis techniques could range from statistical analysis to qualitative coding of user feedback.

Additionally, this PhD project will assess the impact of code recommender systems on developers' productivity, efficiency, effectiveness, and perceived satisfaction. By piloting study designs in a university setting and in the field with the HybridAIR industry partner, empirical evidence on how these systems perform in practical settings will be gathered. The findings will provide insights into the benefits and limitations of code recommender systems, offering a nuanced understanding of their real-world applicability.

Furthermore, this PhD project is expected to contribute to practical recommendations for the design and implementation of more effective code recommender systems. By considering the actual needs and experiences of developers, the framework aims to inform the development of tools that enhance productivity and code quality in meaningful ways.

Overall, the expected results will advance the understanding of code recommender systems' evaluation and impact, providing both a theoretical foundation and practical tools for improving these systems in real-world software development environments.

7 Challenges

This PhD project anticipates several challenges that must be addressed. These challenges span defining evaluation metrics, measuring their impact, transferring findings across different programming environments, conducting real-world studies, addressing biases, and integrating diverse stakeholder needs.

Firstly, defining evaluation metrics like productivity, efficiency, effectiveness, or satisfaction in software engineering is challenging due to their multifaceted nature and variability across different contexts [26], [27]. The collaborative and creative aspects of software development further complicate these definitions [28]. Developing precise, operational definitions that are both comprehensive and adaptable is crucial for consistent and clear evaluations.

Secondly, measuring these metrics in the context of code recommender systems is inherently complex, as traditional measures like lines of code cannot fully capture the impact of these systems on development workflows [29], [30]. Effective measurement requires creating or adapting measures that better reflect real-world contexts. This includes using diverse data sources and considering subjective measures like developers' perception of productivity [30].

Transferring findings from evaluations in general-purpose programming environments to DSLs represents another significant challenge. While general-purpose programming environments provide a broad context for assessment, the unique characteristics and

requirements of DSLs may necessitate tailored evaluation methods and metrics. Ensuring that the developed framework is flexible and robust enough to be applied across diverse programming paradigms will be critical.

Moreover, conducting studies in real-world settings, particularly within industry collaboration, presents logistical and methodological challenges. Ensuring participant engagement, maintaining consistency in data collection, and mitigating external variables that may influence the outcomes are all factors that need careful management [31], [32]. Balancing academic rigor with practical feasibility in the online evaluations will be vital to obtaining valid and reliable results. Studies conducted in the field require significant design and execution efforts, often in corporate settings, which can be resource-intensive. Additionally, real-world settings are inherently complex, and controlling all variables to isolate the effects of the experimental manipulation can be difficult [31].

Addressing potential biases and ensuring the objectivity of the evaluations is a challenge that must be carefully managed. Biases can arise from various sources, including participant selection, experimental design, and data interpretation [31], [33]. Implementing strategies to minimize these biases and ensure the credibility of the research findings is paramount. Not properly addressing the validity of the research when industry is involved, such as issues with generalizability and control, can significantly affect the outcomes [33].

Finally, integrating multiple stakeholders' diverse needs and perspectives within the HybridAIR project adds another layer of complexity. Different stakeholders may have varying priorities and expectations regarding the recommender system's functionality and evaluation [33]. Achieving a consensus and ensuring that the evaluation framework addresses these diverse needs while maintaining scientific rigor will be a complex but necessary task.

8 Current Status & Next Steps

Currently, I am conducting the SLR to gather and synthesize prevalent evaluation methods and metrics. This review will provide the foundational insights necessary for defining the objectives of a framework design.

The next step involves building an initial framework, which will be tested through online studies conducted in a university programming course. This controlled environment will allow for iterative refinement of the framework based on practical feedback and demonstration results.

From the symposium, I expect to gain valuable feedback on my research approach. Engaging with other researchers and domain experts will provide insights that can further refine the framework. Additionally, I want to establish collaborations that could support the planned studies and broader application of my research.

References

[1] G. Adomavicius and A. Tuzhilin, "Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions," *Trans. Knowl. Data Eng.*, vol. 17, no. 6, pp. 734–749, 2005. DOI: 10.1109/TKDE.2005.99.

[2] M. P. Robillard, R. J. Walker, and T. Zimmermann, "Recommendation Systems for Software Engineering," *Softw.*, vol. 27, no. 4, pp. 80–86, 2010. DOI: 10.1109/MS.2009.161.

[3] J. Lu, D. Wu, M. Mao, W. Wang, and G. Zhang, "Recommender system application developments: A survey," *Decis. Support Syst.*, vol. 74, pp. 12–32, 2015. DOI: 10.1016/J.DSS.2015.03.008.

[4] Y. Wan, Y. He, Z. Bi, *et al.*, "Deep Learning for Code Intelligence," *CoRR.*, 2024. DOI: 10.48550/ARXIV.2401.00288.

[5] S. Erdweg, T. Van der Storm, M. Völter, *et al.*, "The state of the art in language workbenches - conclusions from the language workbench challenge," in *Proc. 6th International Conference on Software Language Engineering (SLE 2013)*, Springer, 2013, pp. 197–217. DOI: 10.1007/978-3-319-02654-1_11.

[6] H. Ossher, W. H. Harrison, and P. L. Tarr, "Software Engineering Tools and Environments: A Roadmap," in *Proc. 2000 International Conference on Software Engineering, Future of Software Engineering Track (ICSE 2000)*, ACM, 2000, pp. 261–277. DOI: 10.1145/336512.336569.

[7] H. Tilen, L. Četina, T. Beranič, and L. Pavlič, "Evaluating the Usability and Functionality of Intelligent Source Code Completion Assistants: A Comprehensive Review," *Appl. Sci.*, vol. 13, no. 24, 2023. DOI: 10.3390/app132413061.

[8] E. Al-Hossami and S. Shaikh, "A Survey on Artificial Intelligence for Source Code: A Dialogue Systems Perspective," *CoRR.*, 2022. DOI: 10.48550/arXiv.2202.04847.

[9] S. Xinyu, L. Yue, Z. Yanjie, *et al.*, "Pitfalls in Language Models for Code Intelligence: A Taxonomy and Survey," *CoRR.*, 2023. DOI: 10.48550/ARXIV.2310.17903.

[10] J. Wang and Y. Chen, "A Review on Code Generation with LLMs: Application and Evaluation," in *Proc. 2023 International Conference on Medical Artificial Intelligence (MedAI) 2023*, IEEE, 2023, pp. 284–289. DOI: 10.1109/MedAI59581.2023.00044.

[11] P. K. Biswas and S. Liu, "A hybrid recommender system for recommending smartphones to prospective customers," *Expert Syst. Appl.*, vol. 208, p. 118 058, 2022. DOI: 10.1016/j.eswa.2022.118058.

[12] Y. Dittrich, "Beg, borrow, and steal – but what, and what for?," 2000.

[13] J. E. Hannay and M. Jørgensen, "The role of deliberate artificial design elements in software engineering experiments," *Trans. Software Eng.*, vol. 34, pp. 242–259, 2008. DOI: 10.1109/TSE.2008.13.

[14] J. Segal, "Some parallels between empirical software engineering and research in human computer interaction," in *Proc. 15th Annual Workshop of the Psychology of Programming Interest Group (PPIG 2003)*, Psychology of Programming Interest Group, 2003, p. 22.

[15] C. Bauer, E. Zangerle, and A. Said, "Exploring the Landscape of Recommender Systems Evaluation: Practices and Perspectives," *Trans. Recomm. Syst.*, vol. 2, no. 1, pp. 1–31, 2024. DOI: 10.1145/3629170.

[16] S. Proksch, S. Amann, S. Nadi, and M. Mezini, "Evaluating the evaluations of code recommender systems: A reality check," in *Proc. 2016 International Conference on Automated Software Engineering (ASE 2016)*, ACM, 2016, pp. 111–121. DOI: 10.1145/2970276.2970330.

[17] A. Iung, J. Carbonell, L. Marchezan, *et al.*, "Systematic mapping study on domain-specific language development tools," *Empir. Softw. Eng.*, vol. 25, no. 5, pp. 4205–4249, 2020. DOI: 10.1007/S10664-020-09872-1.

[18] G. Kahraman and S. Bilgen, "A framework for qualitative assessment of domain-specific languages," *Softw. Syst. Model.*, vol. 14, no. 4, pp. 1505–1526, 2015. DOI: 10.1007/s10270-013-0387-8.

[19] A. Barisic, V. Amaral, and M. Goulão, "Usability driven DSL development with USE-ME," *Comput. Lang. Syst. Struct.*, vol. 51, pp. 118–157, 2018. DOI: 10.1016/J.CL.2017.06.005.

[20] B. Lundell and B. Lings, "The *2G* method for doubly grounding evaluation frameworks," *Inf. Syst. J.*, vol. 13, no. 4, pp. 375–398, 2003. DOI: 10.1046/j.1365-2575.2003.00154.x.

[21] B. Kitchenham, S. Linkman, and D. Law, "Desmet: A method for evaluating software engineering methods and tools," *Comput. Control Eng.*, vol. 8, no. 3, pp. 120–126, 1997. DOI: 10.1049/cce:19970304.

[22] N. I. of Standards and Technology, *Metrics and measures*, https://www.nist.gov/itl/ssd/software-quality-group/metrics-and-measures, Retrieved June 26, 2024, 2021.

[23] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee, "A design science research methodology for information systems research," *J. Manag. Inf. Syst.*, vol. 24, pp. 45–77, 2008. DOI: 10.2753/MIS0742-1222240302.

[24] B. Kitchenham, "Procedures for Performing Systematic Reviews," 2004.

[25] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Inf. Softw. Technol.*, vol. 53, no. 6, pp. 625–637, 2011. DOI: 10.1016/j.infsof.2010.12.010.

[26] M. Solla, A. Patel, and C. Wills, "New metric for measuring programmer productivity," in *Proc. 2011 Symposium on Computers & Informatics*, IEEE, 2011, pp. 177–182. DOI: 10.1109/ISCI.2011.5958906.

[27] A. Seffah, M. Donyaee, R. B. Kline, and H. K. Padda, "Usability measurement and metrics: A consolidated model," *Softw. Qual. J.*, vol. 14, no. 2, pp. 159–178, 2006. DOI: 10.1007/S11219-006-7600-8.

[28] C. Jaspan and C. Green, "A Human-Centered Approach to Developer Productivity," *Softw.*, vol. 40, no. 1, pp. 23–28, 2023. DOI: 10.1109/MS.2022.3212165.

[29] R. Brasil-Silva and F. L. Siqueira, "Metrics to quantify software developer experience: A systematic mapping," in *Proc. 2022 Symposium on Applied Computing SAC 2022*, ACM, 2022, pp. 1562–1569. DOI: 10.1145/3477314.3507304.

[30] A. Ziegler, E. Kalliamvakou, A. X. Li, *et al.*, "Measuring GitHub Copilot's Impact on Productivity," *Commun.*, vol. 67, no. 3, pp. 54–63, 2024. DOI: 10.1145/3633453.

[31] K. Stol and B. Fitzgerald, "Guidelines for Conducting Software Engineering Research," in *Contemporary Empirical Methods in Software Engineering*, Springer, 2020, pp. 27–62. DOI: 10.1007/978-3-030-32489-6_2.

[32] D. I. Sjoberg, B. Anda, E. Arisholm, *et al.*, "Conducting realistic experiments in software engineering," in *Proc. 2002 International Symposium on Empirical Software Engineering (ISESE 2002)*, IEEE, 2002, pp. 17–26. DOI: 10.1109/ISESE.2002.1166921.

[33] V. Garousi, K. Petersen, and B. Özkan, "Challenges and best practices in industry-academia collaborations in software engineering: A systematic literature review," *Inf. Softw. Technol.*, vol. 79, pp. 106–127, 2016. DOI: 10.1016/J.INFSOF.2016.07.006.