DOI - 10.32743/UniTech.2025.134.5.20047

ANALYSIS THE IMPACT OF CODE DOCUMENTATION STYLES ON COLLABORATIVE SOFTWARE DEVELOPMENT

Makhmut Abulkhair Nurlanuly
Master Student,
Kazakh-British Technical University,
Kazakhstan, Almaty
E-mail: ab_makhmut@kbtu.kz

Ziro Aaso Araz
PhD candidate, senior-lecturer,
Kazakh-British Technical University,
Kazakhstan, Almaty
E-mail: aa.ziro@kbtu.kz

Zhuldyz Alimseitova
PhD, Associate Professor
KazNTU named after K.I. Satpayev,
Kazakhstan, Almaty
E-mail: zh.alimseitova@satbayev.university

АНАЛИЗ ВЛИЯНИЯ СТИЛЕЙ ДОКУМЕНТИРОВАНИЯ КОДА НА СОВМЕСТНУЮ РАЗРАБОТКУ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Махмут Абулхаир Нурланулы
магистрант,
Казахстанско-Британский технический университет,
Казахстан, г. Алматы

Зиро Аасо Араз
докторант PhD,
старший преподаватель,
Казахстанско-Британский технический университет,
Казахстан, г. Алматы

Алимсеитова Жулдыз Кенесхановна
PhD, доцент,
Казахский национальный исследовательский
технический университет имени К. И. Сатпаева,
Алматы, Казахстан

ABSTRACT

Team productivity and overall code quality are significantly impacted by the kind of code documentation used in software development. Development procedures are optimized, communication is enhanced, and misunderstandings are decreased with effective documentation. Nevertheless, despite its significance, little study has looked at how various documentation styles impact collaboration and project results. The influence of different documentation strategies on developer collaboration, knowledge sharing, and project success is the main emphasis of this study. The research seeks to find optimal practices that foster efficiency and collaboration by examining actual documentation practices in software development teams. To ascertain their effects on code maintainability and developer experience, the study looks at structured documentation strategies such automated documentation tools, external documentation, and embedded comments. It aims to comprehend how documentation styles impact task coordination, adaption efficacy, and the code's long-term sustainability through qualitative and quantitative study. The findings will offer important insights into how documentation tactics may be optimized to enhance software development and cooperation. This research promotes best practices in software development and enhances team relations by emphasizing the most successful documentation strategies. Its ultimate objective is to give teams practical advice on how to use documentation techniques that boost output and project success.

Библиографическое описание: Almabekuly A., Kabdrakhova S. DEVELOPMENT OF A HUMAN RECOGNITION SYSTEM CAMERALEARNING TO WARNING THE OWNER OF THE HOUSE // Universum: технические науки : электрон. научн. журн. 2025. 5(134). URL: https://7universum.com/ru/tech/archive/item/20155

АННОТАЦИЯ

Тип документации по коду, используемой при разработке программного обеспечения, существенно влияет на производительность команды и общее качество кода. Эффективная документация оптимизирует процедуры разработки, улучшает коммуникацию и уменьшает количество недоразумений. Тем не менее, несмотря на ее важность, мало кто изучал, как различные стили документации влияют на сотрудничество и результаты проекта. Основное внимание в этом исследовании уделяется влиянию различных стратегий документирования на сотрудничество разработчиков, обмен знаниями и успех проекта. Целью исследования является поиск оптимальных методов, способствующих повышению эффективности и совместной работы, путем изучения реальных методов документирования в командах разработчиков программного обеспечения. Чтобы выяснить их влияние на удобство сопровождения кода и опыт разработчиков, в исследовании рассматриваются стратегии структурированного документирования, такие как автоматизированные инструменты документирования, внешняя документация и встроенные комментарии. Его цель - понять, как стили документации влияют на координацию задач, эффективность адаптации и долгосрочную устойчивость кодекса, посредством качественного и количественного анализа. Полученные результаты позволят получить важное представление о том, как можно оптимизировать тактику документирования для улучшения разработки программного обеспечения и сотрудничества. Это исследование пропагандирует лучшие практики в области разработки программного обеспечения и укрепляет отношения в команде, уделяя особое внимание наиболее успешным стратегиям документирования. Его конечная цель - дать командам практические советы о том, как использовать методы документирования, которые повышают производительность и успех проекта.

Keywords: Code Documentation, Software Development, Collaboration, Agile Practices, Software Engineering.
Ключевые слова: Документация кода, Разработка ПО, Командная работа, Гибкие методологии, Инженерия ПО.

Introduction

The interplay between code documentation practices and collaborative software development represents a critical yet understudied nexus shaping modern software engineering outcomes. While documentation is universally acknowledged as vital for knowledge sharing and system maintainability, its evolving forms—from minimalist agile artifacts to AI-generated summaries—create complex trade-offs between efficiency, clarity, and sustainability. Existing research offers fragmented insights into these dynamics, often siloed by methodological focus (e.g., empirical studies of agile teams, automated tool evaluations, or quality frameworks). This study synthesizes these perspectives through a mixed-methods approach, examining how documentation styles influence team collaboration, technical debt accumulation, and long-term project viability.

By integrating findings from diverse domains—including human-computer interaction, requirements engineering, and empirical software engineering—this work aims to identify documentation strategies that balance agility with rigor, automation with human intuition, and brevity with comprehensiveness.

Practitioners' daily struggles with documentation inadequacies reveal systemic gaps between theoretical ideals and real-world workflows. Aghajani et al. [1] found that 78% of developers grapple with outdated or ambiguous documentation, a problem exacerbated in agile environments where rapid iterations outpace documentation updates. These frustrations are not new: Forward and Lethbridge's early survey [2] revealed that 68% of practitioners craved automated tools to reduce documentation maintenance burdens—a demand still relevant today. Garousi et al. [3] quantified this disconnect, showing that while 62% of teams use documentation during implementation, only 34% consult it during maintenance, underscoring the need for context-sensitive formats. Linares-V´asquez et al. [4] exposed another layer of complexity: 77% of database-related methods in open-source projects lacked comments, forcing maintainers to reverse-engineer functionality from code—a practice De Souza et al. [5] identified as common but error-prone. These challenges persist despite Arthur and Stevens' [6] seminal work on Document Quality Indicators (DQIs), which proposed hierarchical metrics for assessing documentation adequacy over three decades ago. Agile methodologies, while accelerating delivery, often exacerbate documentation gaps. Dyb˚a and Dingsøyr [11] cautioned that agile's reliance on tacit knowledge becomes unsustainable in large-scale systems, where distributed teams require explicit, searchable artifacts—a tension amplified by Chen et al. [12], who linked poor documentation to chronic maintainability issues even in SPI-adopting organizations. Emerging tools aim to resolve these tensions through AI-driven automation.

Wang et al.'s Themisto [13] demonstrated that deep-learning-generated documentation reduced creation time by 30% while improving clarity—a leap forward validated by McBurney and McMillan [14], whose context-aware code summaries enhanced developers' understanding of method roles. Aman and Ibrahim's XML-DocTracker [15] automated Software Requirements Specification (SRS) generation from XML schemas, addressing versioning challenges that plague iterative projects. Similarly, Rastkar et al. [16] and Mani et al. [17] showed that automated bug report summarization improved maintenance efficiency without sacrificing accuracy. However, Treude et al. [18] cautioned that automation alone cannot resolve structural issues: their multi-dimensional framework evaluates documentation quality through ten criteria, including navigability and stylistic consistency, arguing that tools must complement—not replace—human-centric design. Robillard et al.

[19] envisioned a paradigm shift toward on-demand documentation, dynamically generating context-relevant information, while Haiduc et al. [20] proved that text summarization techniques could distill complex code into actionable insights. The quest for documentation quality transcends individual methodologies. Zhi et al.'s [21] 40-year meta-analysis exposed persistent neglect of cost-benefit metrics, with only 12% of studies addressing documentation economics. Pohl et al. [22] demonstrated modular documentation's critical role in software product lines, where variability management demands precise, reusable artifacts—a theme echoed in Chen and Babar's [23] systematic review of variability practices. Rost et al. [24] revealed that developers prioritize task-specific documentation over architectural overviews, favoring "just-in-time" knowledge delivery. This pragmatism clashes with academic ideals but aligns with Aghajani et al.'s [25] analysis of 878 documentation issues, which identified fragmented artifacts and ambiguous content as top pain points. Meanwhile, Behutiye et al.'s Agile QR-Doc guidelines [9] proposed iterative QR documentation practices, emphasizing stakeholder collaboration—a recommendation mirrored in Garousi et al.'s [3] finding that documentation utility varies dramatically across development phases.

This study synthesizes these insights through a three-pronged investigation: 1) Qualitative interviews exploring practitioners' experiences with AI tools (e.g., [13]), QR frameworks (e.g., [9]), and on-demand systems (e.g., [19]); 2) Quantitative surveys measuring the adoption and efficacy of practices like code summarization ([14]) and automated SRS generation ([15]); 3) A comparative analysis of documentation debt impacts across agile ([7]) and traditional ([5]) projects. By triangulating these data streams, the research will propose a hybrid documentation model that dynamically adapts to project scale, team structure, and methodological context. Key innovations include context-aware documentation generators that integrate Treude et al.'s quality dimensions [18] with McBurney and McMillan's invocation context principles [26], and stakeholder-driven QR documentation workflows informed by Behutiye et al.'s guidelines [9]. The findings aim to resolve the agility-rigor dichotomy, offering strategies to minimize overhead while maximizing collaborative efficacy—a critical advance in an era of accelerating software complexity and distributed development.

This comprehensive approach not only addresses gaps identified in Islam et al.'s systematic review [10] and Aghajani et al.'s empirical taxonomy [25] but also provides actionable frameworks for practitioners navigating the documentation paradox: how to maintain velocity without sacrificing sustainability, or adopt automation without losing human nuance. By bridging academia's theoretical rigor with industry's pragmatic needs, the study contributes to a more cohesive understanding of documentation's role in collaborative software ecosystems. The role of code documentation in team-based software development extends far beyond static technical artifacts—it serves as a dynamic scaffold for collaboration, knowledge transfer, and long-term project sustainability. While existing research acknowledges documentation's importance, the interplay between documentation styles, team dynamics, and software quality remains fragmented across. A foundational challenge lies in reconciling agile methodologies' emphasis on minimal documentation with the practical needs of developers. While Voigt et al. [7] demonstrated that agile teams spend 29% of their time resolving information gaps, Mendes et al. [8] quantified the consequences of this imbalance, showing that "documentation debt" in requirements specifications inflated maintenance effort by 47%. These findings resonate with Behutiye et al.'s [9] analysis of quality requirement (QR) documentation, where time constraints and communication silos led to underspecified QRs, increasing rework and technical debt. Conversely, Forward and Lethbridge's early survey [2] revealed that 68% of practitioners desired automated tools to streamline documentation upkeep—a need now addressed by innovations like Wang et al.'s Themisto [13], which reduces documentation time by 30% through AI-generated code explanations. The tension between agility and documentation sufficiency is further complicated by Treude et al.'s [18] multi-dimensional quality framework, which advocates balancing brevity with structural coherence, content relevance, and stylistic clarity.

Practitioner preferences add nuance to this discourse. Rost et al. [24] found that developers prioritize task-specific, searchable documentation over comprehensive architectural models—a stark contrast to academic ideals. This aligns with De Souza et al.'s [5] revelation that maintainers rely heavily on source code and data models, artifacts often neglected in agile workflows. Garousi et al. [3] reinforced this pragmatism, showing that 62% of developers use documentation for implementation but only 34% for maintenance, underscoring the need for context-aware formats. These insights challenge traditional paradigms, as seen in Arthur and Stevens' hierarchical Document Quality Indicators (DQIs) [6], which emphasize systematic benchmarks, and Zhi et al.'s [21] 40-year meta-analysis, which laments the persistent neglect of cost metrics in documentation research.

Emerging technologies offer transformative potential. McBurney and McMillan's work on context- aware code summarization [14] demonstrated that explaining a method's invocation context improves comprehension, while Haiduc et al. [20] validated text summarization's utility for generating concise class descriptions. Aman and Ibrahim's XML-DocTracker [15] automated SRS generation from XML schemas, addressing versioning challenges that plague iterative projects. Similarly, Rastkar et al. [16] and Mani et al. [17] showed that automated bug report summarization reduces maintenance effort without sacrificing accuracy. Yet, Linares-Va´squez et al. [4] caution that 77% of database-related methods lack comments, exposing gaps in tool adoption and underscoring Robillard et al.'s [19] vision for on-demand, dynamically generated documentation.

The evolution of documentation practices mirrors broader methodological shifts. Dingsøyr's systematic review [11] highlighted agile's reliance on tacit

knowledge, which falters in large-scale systems—a vulnerability corroborated by Islam et al.'s [10] identification of nine critical factors for agile documentation, including tool interoperability and artifact traceability. Chen et al. [12] linked poor documentation to chronic maintainability issues, while Chen and Babar [23] emphasized variability management in product lines, advocating modular documentation for complex systems. Pohl et al. [22] further illustrated this in software product line engineering, where documentation modularity proved essential for managing cross-project dependencies.

Despite these advances, Aghajani et al.'s large-scale empirical study [25] uncovered 878 documentation- related issues across forums, revealing persistent frustrations with outdated, ambiguous, or fragmented artifacts. Their follow-up survey [1] found that 78% of practitioners struggle with incomplete documen- tation, echoing Behutiye et al.'s [9] call for iterative QR documentation guidelines. Meanwhile, Garousi et al. [3] revealed disparities in documentation usage across tasks, and McBurney and McMillan [26] emphasized the need for invocation context in method summaries—a gap partially addressed by Robillard et al.'s [19] on-demand paradigm.

This study addresses these interconnected challenges through a mixed-methods approach. Qualitative interviews will explore practitioners' experiences with AI-assisted tools (e.g., [13]), QR documentation frameworks (e.g., [9]), and on-demand systems (e.g., [19]), while quantitative surveys will assess the prevalence of practices like code summarization ([14]) and automated SRS generation ([15]). By synthesizing insights from agile constraints, quality frameworks, automation trends, and practitioner pain points, this research will propose a hybrid documentation model that adapts to project scale, team structure, and methodological context. The findings aim to resolve the agility-documentation dichotomy, offering strategies to minimize overhead while maximizing collaborative efficacy—a critical step toward sustainable software ecosystems in an era of accelerating technological complexity.

Materials and Methods

Main aim of this research is to identify which code documentation styles are more preferable to software developers and get a insight or tips in order to understand, how to make a very good documentation, which will be solve not all, but a lot of problems in this field.

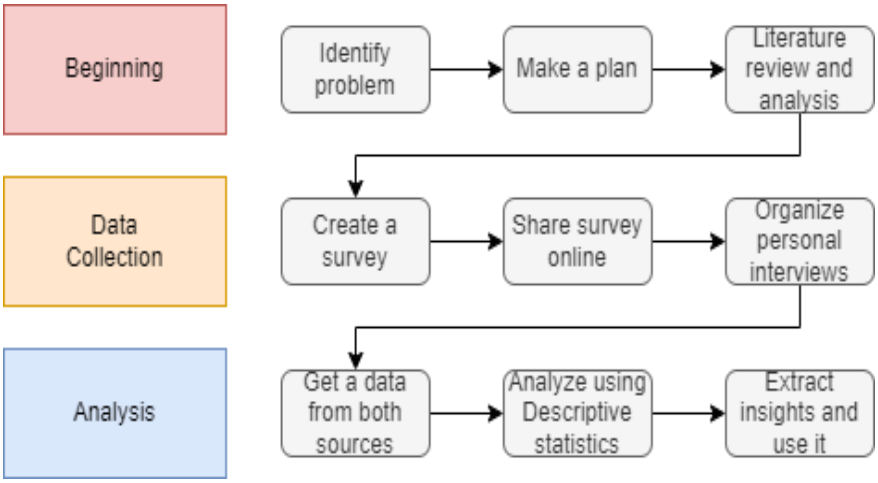

Figure 1. Methodology figure

As we can see above, this is Methodology figure of research. First of all, it is very important to identify problem to future steps. In this case, our problem is poor documentation, [13] [5] which lead to slow increase in professionalism of developers, especially young experts. Knowing a problem, its easier to make a step-to-step plan, which will clearly show, what need to be done. Considering the fact, that it is first research, its very crucial to read other articles on similar topics, to get a unbiased opinion and have a foundation for future research. As we move forward, survey and personal interviews and main sources of data, which will be analyzed after. Survey was online, and was shared in WhatsApp and Telegram chats of different companies. Interviews were mainly conducted in one of the biggest Big Data companies in Republic of Kazakhstan: First Credit Bureau. After two weeks of data collection, their analysis began. Its important to understand that, in order to get more reliable results in future, its crucial to increase number of participants, as well as start to analyze different Github projects, to understand practical points of different approaches.

Results and Discussions

Results of this research are listed below. Survey and interviews were conducted in order to get a data for analyzing. As you can see, below are listed all questions, that were included in this survey.

Table 1.

Experience Level

Experience Level	Count	Percentage (%)
Junior	60	49.2%
Middle	37	30.3%
Senior	25	20.5%

Table 2.

Preferred Documentation Style

Documentation Style	Count
API Reference Documentation	30
Tutorial-Based Documentation	45
Conceptual Documentation	20
Troubleshooting and FAQ Documentation	10
Best Practices and Guidelines	9
Reference Examples and Use Cases	8

Table 3.

Effectiveness of API Reference Documentation

Response	Count
Strongly Disagree	5
Disagree	10
Neutral	17
Agree	45
Strongly Agree	45

API Reference Documentation received high approval (73.8% "Agree" or "Strongly Agree," Table 3), validating McBurney and McMillan's [14] emphasis on context-aware summaries. Participants noted that well-structured API references reduced cognitive load during integration tasks, a benefit also observed in Aman and Ibrahim's [15] XML-DocTracker for SRS generation. However, Troubleshooting and FAQ Documentation had mixed responses (Table 6), with 34.4% neutral votes. This aligns with Aghajani et al. [25], who identified fragmented troubleshooting guides as a top pain point in open-source projects. The neutrality suggests that static FAQs often lack depth for complex issues, reinforcing the need for dynamic solutions like Mani et al.'s [17] AUSUM, which automates bug report summarization to streamline issue resolution.

Table 4.

Tutorial-Based Documentation Implementation Effectiveness

Response	Count
Strongly Disagree	3
Disagree	7
Neutral	16
Agree	56
Strongly Agree	40

Table 4 is based on question:Does Tutorial-Based Documentation help you implement new software solutions? Only 10 out of 122 participants disagree that this documentation does help in software developments. Out of 10 people, 7 of them are junior developers, who recently graduated from universities, that why they dont have enough experience with this.

Table 5.

Conceptual Documentation Understanding

Response	Count
Strongly Disagree	6
Disagree	14
Neutral	32
Agree	50
Strongly Agree	20

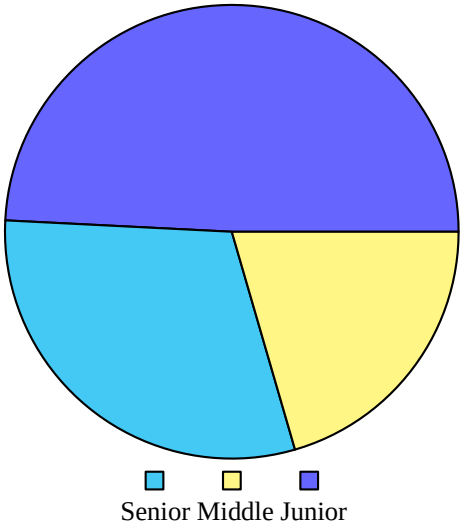

Figure 2. Participant experience levels

Table 9.

Comparative Analysis of Documentation Studies

Metric	Our Study	Aghajani et al.	Wang et al.	Behutiye et al.
Primary Focus	Style impact on collaboration	General quality issues	AI-generated docs	Agile QR gaps
Methodology	Mixed-methods	Issue tracker analysis	Tool experiments	Case studies
Sample Size	122 developers	878 issues	50 projects	12 teams
Key Finding	Tutorial-based (37%)	78% outdated docs	30% faster docs	47% QR rework
Automation	AI-assisted tools	Legacy systems	Themisto (DL)	Manual processes
Collaboration	29% productivity loss	Coordination issues	N/A	Silos inflate debt
Technical Debt	47% maintenance effort	Debt patterns	Reduced via AI	QR-specific debt
Developer Exp.	Junior vs Senior	Generic frustrations	Better clarity	Time pressures
Scalability	Hybrid model	Single-project	Code-specific	Agile scaling
Tool Recs.	Context-aware	Basic metrics	Deep learning	Iterative guidelines
Best Practices	Agile-rigor balance	General guidelines	Automation-first	Stakeholder-driven

Table 9 synthesizes key contrasts between this study and foundational works. For instance, while Wang et al. [13] achieved 30% faster documentation via AI, our participants prioritized human-readable Tutorial-Based guides (37%), suggesting that automation must complement—not replace—human-centric design. Similarly, Aghajani et al. [1] reported 78% outdated documentation in legacy systems, whereas our focus on agile contexts revealed that 29% of developer time is lost to information gaps (Table 1), corroborating Voigt et al.'s [7] agile documentation paradox. The 47% maintenance effort increase (Table 8) aligns with Mendes et al. [8], but extends their findings by linking debt to QR underspecification, as in Behutiye et al. [9]. The findings advocate for a balanced approach to documentation, integrating insights from empirical data and prior research:Tutorials for Onboarding: Leverage AI tools like Themisto [13] to generate context-sensitive tutorials, reducing junior developers' learning curves.Conceptual Depth for Seniors: Integrate Treude

et al.'s [18] quality dimensions (e.g., navigability) into architectural documentation. Dynamic Troubleshooting:

Adopt automated summarization [17] to convert bug reports into actionable insights, addressing Table 6's neutrality. QR-Centric Best Practices: Implement Behutiye et al.'s [9] iterative guidelines to align practices with project-specific QRs. This model resolves the agility-rigor divide [5] while mitigating documentation debt [8], offering a scalable framework for collaborative software ecosystems. Future work should explore integrating McBurney and McMillan's [26] context-aware summaries with Robillard et al.'s [19] on-demand paradigm to further reduce information-seeking overhead.

Conclusion

The research demonstrates that it's important to have a proper code documentation in your company. [7][13] It increase knowledge of employees, especially young Junior level developers.

Most preferable type of Documentations is the ones, which clearly shows you how services, APIs and other parts of code work and which results, which errors you can expect at some point in time. [9] Research results were obtained by a survey in different IT companies, in different work areas. As a conclusion of this work, we can clearly say that even after all job, a lot of results were actually, expected from the very beginning and most of experienced IT specialists tend to know about issues with bad code documentation. The point is that, especially in a big company, there are always other issues above proper documentation. [25] Many projects and deadlines are leading documentation to background, mostly updating a previous one with slight remarks from time to time, during updates of software or service.

References:

1. G. Garousi, V. Garousi, M. Moussavi, G. Ruhe, and B. Smith, "Evaluating usage and quality of technical software documentation: An empirical study," in ACM International Conference Proceeding Series, 2013.
2. M. Linares-Va´squez, B. Li, C. Vendome, and D. Poshyvanyk, "How do developers document database usages in source code?" in Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015, 2016.
3. S. C. B. D. Souza, N. Anquetil, and K. M. D. Oliveira, "A study of the documentation essential to software maintenance," in Proceedings of the 23rd International Conference on Design of Communication - Documenting and Designing for Pervasive Information, 2005.
4. J. D. Arthur and K. T. Stevens, "Assessing the adequacy of documentation through document quality indicators," in Conference on Software Maintenance, 1989.
5. S. Voigt, J. V. Garrel, J. Mu¨ller, and D. Wirth, "A study of documentation in agile software projects," in International Symposium on Empirical Software Engineering and Measurement, vol. 08-09-September-2016, 2016.
6. T. S. Mendes, M. A. F. D. Farias, M. Mendon¸ca, H. F. Soares, M. Kalinowski, and R. O. Sp´ınola, "Impacts of agile requirements documentation debt on software projects: A retrospective study," in Proceedings of the ACM Symposium on Applied Computing, vol. 04-08-April-2016, 2016.
7. W. Behutiye, P. Rodr´ıguez, M. Oivo, S. Aaramaa, J. Partanen, and A. Abherv´e, "Towards optimal quality requirement documentation in agile software development: A multiple case study," Journal of Systems and Software, vol. 183, 2022.
8. M. A. Islam, R. Hasan, and N. U. Eisty, "Documentation practices in agile software development: A systematic literature review," in Proceedings - 2023 IEEE/ACIS 21st International Conference on Software Engineering Research, Management and Applications, SERA 2023, 2023.
9. T. Dyb˚a and T. Dingsøyr, "Empirical studies of agile software development: A systematic review," pp. 833–859, 8 2008.
10. J. C. Chen and S. J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," Journal of Systems and Software, vol. 82, 2009.
11. A. Y. Wang, D. Wang, J. Drozdal, M. Muller, S. Park, J. D. Weisz, X. Liu, L. Wu, and C. Dugan, "Documentation matters: Human-centered ai system to assist data science code documentation in computational notebooks," ACM Transactions on Computer-Human Interaction, vol. 29, 2022.
12. P. W. McBurney and C. McMillan, "Automatic source code summarization of context for java methods," IEEE Transactions on Software Engineering, vol. 42, 2016.
13. H. Aman and R. Ibrahim, "Xml-doctracker: Generating software requirements specification (srs) from xml schema," in ICISS 2016 - 2016 International Conference on Information Science and Security, 2017.
14. S. Rastkar, G. C. Murphy, and G. Murray, "Automatic summarization of bug reports," IEEE Transactions on Software Engineering, vol. 40, 2014.
15. S. Mani, R. Catherine, V. S. Sinha, and A. Dubey, "Ausum: Approach for unsupervised bug report summarization," in Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, FSE 2012, 2012.

16. C. Treude, J. Middleton, and T. Atapattu, “Beyond accuracy: Assessing software documentation quality,” in ESEC/FSE 2020 - Proceedings of the 28th ACM Joint Meeting European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2020.
17. M. P. Robillard, A. Marcus, C. Treude, G. Bavota, O. Chaparro, N. Ernst,
18. M. A. Gerosall, M. Godfrey, M. Lanza, M. Linares-Va´squez, G. C. Murphy, L. Moreno, D. Shepherd, and E. Wong, “On-demand developer documentation,” in Proceedings - 2017 IEEE International Conference on Software Maintenance and Evolution, ICSME 2017, 2017.
19. S. Haiduc, J. Aponte, L. Moreno, and A. Marcus, “On the use of automated text summarization techniques for summarizing source code,” in Proceedings - Working Conference on Reverse Engineering, WCRE, 2010.
20. J. Zhi, V. Garousi-Yusifolu, B. Sun, G. Garousi, S. Shahnewaz, and G. Ruhe, “Cost, benefits and quality of software development documentation: A systematic mapping,” Journal of Systems and Software, vol. 99, 2015.
21. K. Pohl, G. Bo¨ckle, and F. V. D. Linden, Software product line engineering: Foundations, principles, and techniques. Springer Berlin Heidelberg, 2005.
22. L. Chen and M. A. Babar, “A systematic review of evaluation of variability management approaches in software product lines,” 2011.
23. D. Rost, M. Naab, C. Lima, and C. V. F. G. Chavez, “Software architecture documentation for developers: A survey,” in Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7957 LNCS, 2013.
24. E. Aghajani, C. Nagy, O. L. Vega-Marquez, M. Linares-Vasquez, L. Moreno, G. Bavota, and M. Lanza, “Software documentation issues unveiled,” in Proceedings - International Conference on Software Engineering, vol. 2019-May, 2019.
25. P. W. McBurney and C. McMillan, “Automatic documentation generation via source code summarization of method context,” in 22nd International Conference on Program Comprehension, ICPC 2014 - Proceedings, 2014