Review

Not peer-reviewed version

The Use of AI in Software Engineering: Synthetic Knowledge Synthesis of Recent Research Literature

Peter Kokol *

Posted Date: 11 March 2024

doi: 10.20944/preprints202403.0556.v1

Keywords: Software engineering; Artificial Intelligence; Machine learning; Synthetic Knowledge Synthesis doi:10.20944/preprints202403.0556.v1

Review

The Use of AI in Software Engineering: Synthetic Knowledge Synthesis of Recent Research Literature

Peter Kokol *

University of Maribor; Faculty of Electrical Engineering and Computer Science, Koroška ulica 46, 2000 Maribor, Slovenia
* Correspondence: peter.kokol@um.si

Abstract: Artificial intelligence (AI) has witnessed an exponential increase in its use in various applications. Recently, the academic community started to research and inject new AI-based approaches to provide solutions to traditional software engineering problems. However, a comprehensive and holistic understanding of the current status is missing. To close the above gap, synthetic knowledge synthesis was used to induce a research landscape of the contemporary research literature on the use of AI in software engineering. The synthesis resulted in 15 research categories and five themes, namely natural language processing in software engineering, use of artificial intelligence in the management of software development life cycle, use of machine learning in fault/defect prediction and effort estimation, employment of deep learning in intelligent software engineering and code management, and mining software repositories to improve software quality. The most productive country was China (n=2042), followed by the United States (n=1193), India (n=934), Germany (n=445), and Canada (n=381). A high percentage (n=47.4%) of papers were funded, showing a strong interest in this research topic. The convergence of AI and software engineering can significantly reduce needed resources, improve quality, increase user experience, and improve the well-being of software developers.

Keywords: software engineering; artificial intelligence; machine learning; synthetic knowledge synthesis

1. Introduction

In the last couple of years, artificial intelligence (AI) has witnessed exponential growth in development, the rise of AI use, and increased public interest on different levels, from individual to organizational [1–3]. Modern AI utilizes machine learning and other advanced techniques to generate new knowledge, content, hypotheses, and even innovative ideas by identifying patterns and information usually found in big data-size databases. This has catalyzed the use of AI in a broad spectrum of applications, including software design and development [4]. Software companies shifted their focus to deploying AI paradigms to their existing development processes. The academic community started to research and inject new AI-based approaches to provide solutions to traditional software engineering problems [5] and critical activities [6]. Examples include software testing [7], maintenance [8], requirements extraction [9], ambiguity resolution [10], software vulnerability detection [11], and software engineering education [12]. Despite the increasing prevalence of AI use in software engineering, a comprehensive and holistic understanding of the current status, possible target applications, practical software engineering usage scenarios, and unavoidable limitations, ethical concerns, and challenges remain unclear [6].

To close the above gap, this paper presents a comprehensive research landscape of the current research literature on the use of AI in software engineering. The landscapes aim to serve as a framework for informing and solving theoretical and practical challenges in software development and design related to AI SU. The research community and practicing software engineers can use it to improve their understanding of this fast-growing and highly innovative area. It can also inform novice researchers, grant administrators, software managers, and interested readers lacking specific

 doi:10.20944/preprints202403.0556.v1

domain knowledge to develop a perspective on essential research dimensions. Finally, the landscape can guide and inform further research and serve as a starting point for more formal knowledge and evidence synthesis approaches.

2. Materials and Methods

The research landscape representing AI use in software engineering was induced by Synthetic Knowledge Synthesis (SKS) [13]. SKS integrates quantitative and qualitative synthesis by triangulating descriptive bibliometrics, bibliometric mapping, and content analysis, thus reducing the weaknesses of traditional knowledge synthesis approaches [14]. A research landscape is a map/network of the relationships and associations between bibliometric units. In our present study, those units represent author keywords. Links on the map are current relations, proximity similarity, and node size popularity. Landscape areas (colored clusters) in our study represent strongly associated authors' keywords, either thematically or timewise. The third component of SKS is a content analysis [7], a resourceful approach which, in our case, was used for a qualitative analysis of phenomena contained in research publications to obtain their objective and holistic descriptions in the forms of categories and themes. Scopus (Elsevier, The Netherlands) was used as the source bibliographic database because it is deemed the largest abstract and citation database of the reviewed research literature. In addition to advanced analytics services, it enables 20,000 records to be exported simultaneously.

The search query shown below was constructed using the recommendation provided by Farooq et al. [15].

TITLE-ABS-KEY(("artificial intelligence" OR "machine learning" OR "deep learning" OR "intelligent system" OR "support vector machine" OR ("decision tree" AND (induction OR heuristic)) OR "random forest" OR "Markov decision process" OR "hidden Markov model" OR "fuzzy logic" OR "k-nearest neighbor" OR "naive Bayes" OR "Bayesian learning" OR "artificial neural network" OR "convolutional neural network" OR "recurrent neural network" OR "generative adversarial network" OR "deep belief network" OR "perceptron" OR {natural language processing} OR {natural language understanding} OR {general language model}) and ({software engineering} OR {software design} or {software development})) AND PUBYEAR > 2018 AND PUBYEAR < 2025

The search was performed on February 14th, 2024. The resulting corpus was analyzed using SKS, focusing on bibliometric mapping and content analysis. Finally, using the identified themes and categories as a basis, we performed a literature synthesis and review.

3. Results and Discussion

The search resulted in 9080 publications. Among them, there were 5187 conference papers, 3097 articles, 354 conference reviews, 185 book chapters,179 review papers, 15 editorials, ten retracted papers, nine errata, five short surveys, four notes, and 1 data paper. The recall value of the search was 0.95. The paper type distribution shows that most publications were conference-related papers, revealing that research is still in a maturation phase and that core knowledge is still forming. This finding is also confirmed by the fact that the three most prolific titles are conference proceedings, namely Advances In Intelligent Systems And Computing (n=456), ACM International Conference Proceeding Series (n=341), and Lecture Notes In Computer Science Including Subseries Lecture Notes In Artificial Intelligence And Lecture Notes In Bioinformatics (n=249). The first journal source title is Information Sciences (n=245), followed by another conference proceedings, Ceur Workshop Proceedings (n=190), indicating that the list of core journals has not yet been established. The H-index of the above source titles lies between 58 and 446, meaning that their quality is averagely high and that most publications are not yet published in top-tier journals. The H-index of the whole sub-field of AI use in software engineering has been 87 for the last five years.

The research productivity trend shown in Figure 1 is surprising since the productivity peak of the total number of publications was already reached in 2020. However, the number of publications stabilized in 2022. The decreasing number of publications is mainly due to the decreasing number of conference papers, while the number of articles increased in 2022. Both above facts might reveal the start of a positive trend toward reaching the research maturity.

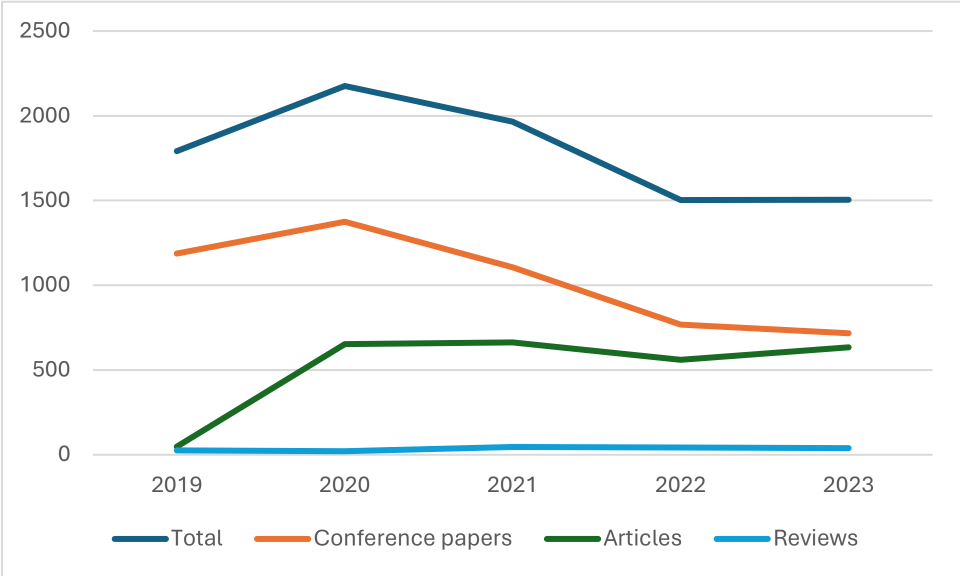

Figure 1. The dynamics of the research literature productivity.

The far most productive countries were China (n=2042), followed by the United States (n=1193), India (n=934), Germany (n=445), and Canada (n=381). That is in line with Scimago Country Rankings (Elsevier, Amsterdam, Netherlands), where the United States is first ranked in Software and second in Artificial Intelligence, China is first In Artificial Intelligence and second in Software, and other top countries are among the ten most productive in both categories. All top productive countries also belong to G20 [16]. China also prevails among the most productive institutions; among the first five, four are from China, namely the Ministry of Education of the People's Republic of China (n=90), Chinese Academy of Sciences (n=89), Nanjing University (n=72), and Peking University (n=71). The only non-Cina institution among the top five in third place is Monash University, Australia (n=73). The most productive USA institution is Chalmers University (n=46), which is in 16th place, and the most productive European one is the Chalmers University of Technology. Sweden (n=54) in 9th place.

Another important indicator of the research state of a scientific field/sub-field is research funding [17]. Our analysis showed that 41.1% of papers are funded, which is notably more than in many other disciplines [18], however, less than in a comparable subfield, namely the use of AI in pediatrics, where 47.4% of papers were funded [2]. The most prolific funding sponsor is the National Natural Science Foundation of China (n=987), the National Science Foundation, USA (n=284), the National Key Research and Development Program of China (n=226), the Horizon 2020 Framework Programme (n=154), and the European Regional Development Fund (n=105).

3.1. Identification of Main Research Themes

Content analysis of the research landscape consisted of 146 author keywords (Figure 2), revealing 15 categories and five themes, which are summarized in Table 1.

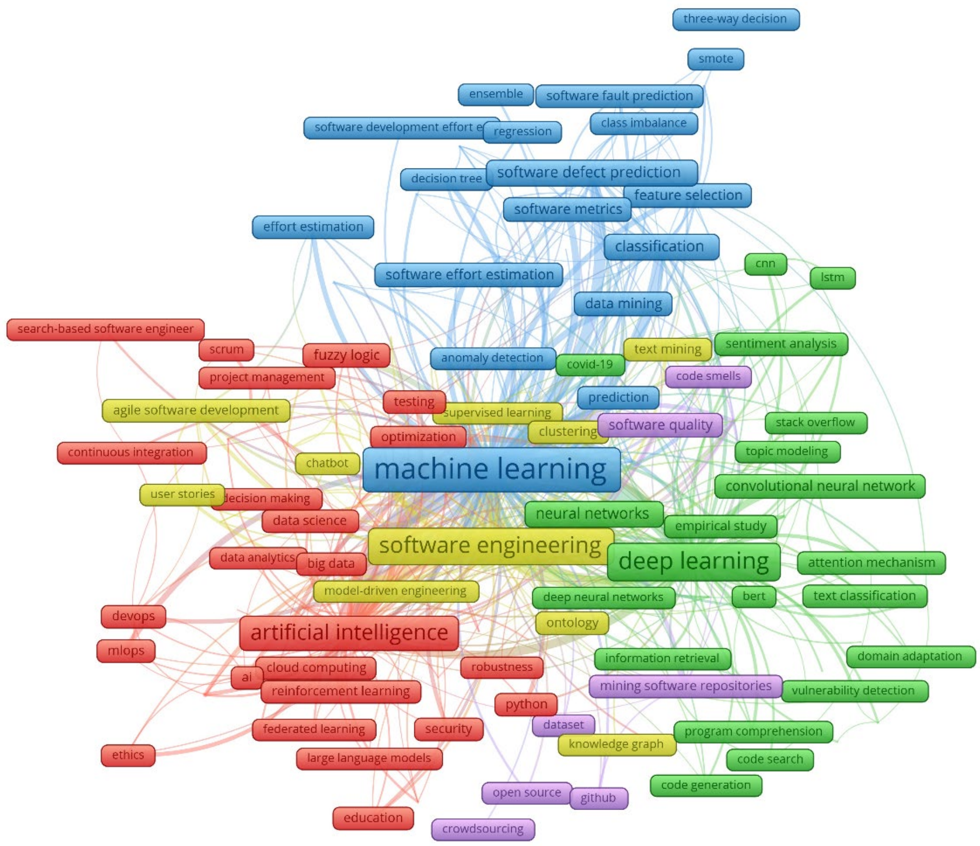

Figure 2. The research landscape of the AI use in software engineering. Keywords appearing in 20 or more publications are shown.

Table 1. This is a table. Tables should be placed in the main text near the first time they are cited.

Cluster colour	Representative keywords	Categories	Themes
Red (42 author keywords)	Artificial intelligence (560), Software development (173), Software testing (123), Fuzzy logic (98), Software (73), Big data (65), Reinforcement learning (64)	Ethical use of AI-based software engineering, Use of fuzzy logic in software development and testing, Automation of software testing in an agile environment, Project management of software life cycle using fuzzy logic, Data science, and big data in software development	Use of artificial intelligence in management of software development life cycle
Yellow (25 author keywords)	Software engineering (673), Natural language processing (362), Requirement engineering (108), Agile software development (61)	Natural language processing in software development, Natural language processing in software requirements engineering, User stories understanding with natural language processing	Natural language processing (NLP) in software engineering
Blue cluster (31 author keywords)	Machine learning (1504), Software development effort estimation (156), Classification	Software development effort estimation, Data mining in software	Machine learning in fault/defect

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 11 March 2024 doi:10.20944/preprints202403.0556.v1

	(142), Software defect prediction (205), and Data mining (102). Artificial neural network (184), Software metrics (84), Feature selection (82)	fault/defect prediction. Machine learning and software metrics	prediction and effort estimation
Green (39 author keywords)	Deep learning (770), Neural networks (123), Empirical software engineering (62), Attention mechanism (68), Code generation (34), Code search (33), Covid 19 (30), Technical depth (26), program comprehension (31)	Deep learning in program comprehension and vulnerability detection; Technical depth and code smell detection, and classification, Covid 19 influence on software engineering	Deep learning in empirical software engineering focusing on code management
Viollet (9 author keywords)	Software quality (86), Software maintenance (62), Mining software repositories (43)	Mining software repositories to improve software quality and software maintenance, Crowdsourcing, Github, and Open source software as sources for mining software development data	Mining software repositories to improve software quality

3.1.1. Literature Review of Research Categories and Themes

Use of artificial intelligence in management of software development life cycle

Ethical use of AI-based software engineering

Vakkuri et al. [19] noted that ethical considerations are mostly ignored while developing AI-based software systems. Consequently, general and high-level guidelines for managing ethics issues have been proposed [20–22]-

Use of fuzzy logic in software development and testing.

Fuzzy logic techniques have been used in selecting software requirements from elicited software requirements or ordering them by preferences [23], cloud-based testing adaption [24], and software effort estimation [25]

Automation of software testing in an agile environment. Artificial intelligence has been used to generate test cases for automatic testing in agile environments [26–28] and to automate other phases of the software development lifecycle [29].

Project management of software life cycle using fuzzy logic.

Fuzzy logic techniques have been used to support project management activities like cost and effort estimation [30,31], imputation of missing values in empirical software project management [32], management of outsourcing [33], and risk assessment in the agile environment [34] and software product promotion *[35]*.

Data science and big data in software development.

Data science and the availability of extensive software development databases enabled the rise of computer, and AI-aided software engineering [36], estimation of story points in agile environments [37], and support of empirical software engineering in general *[38]*.

Natural language processing (NLP) in software engineering

Natural language processing in software development

NLP technology can drastically improve software development tasks [39]. It can support bug categorization [40], development of more secure software [41], program decomposition [42], classifying commitments [43], programming and coding [44], writing coherent and factually correct readmes [45], model-driven engineering [46], deployment of design patterns [47] and traceability management [48]-

Preprints.org (www.preprints.org) | NOT PEER-REVIEWED | Posted: 11 March 2024 doi:10.20944/preprints202403.0556.v1

Natural language processing in software requirements engineering

NLP can support human-performed linguistic analysis in requirements engineering [49,50], such as identifying domain concepts [51–54], establishing traceability links [55], requirement classification [56,57], handling ambiguity [58,59], preference extraction form scenarios [60], classification of non-functional requirements [61], standardization of requirements in agile approaches [62] and requirement elicitation [63].

User stories understanding with natural language processing

NLP has also been used to extract feature, goal, and domain models from user stories [51,54,64], improve the completeness of acceptance criteria [65], or build software structures from user stories [66].

Machine learning in fault/defect prediction and effort estimation

Software development effort estimation

Software development effort estimation is one of the most popular AI techniques used by Intelligent software engineering [67], and cost estimation is one of the most crucial software engineering tasks [68]. Different machine learning algorithms like random forests [69], differential evolution [70], or extreme learning [71]. AI-based effort and cost estimation are used in traditional [72] and agile environments [73].

Data mining in software fault/defect prediction

Data mining and machine learning are used to classify software faults [74] for their detection [75] and prediction [76,77].

Machine learning and software metrics

Machine learning is used to detect code smells [78,79], support software size metric estimation [80], asses software component reusability [81], or predict test flakiness [82].

Deep learning in empirical software engineering focusing on code management

Deep learning in program comprehension and vulnerability detection

In various ways, deep learning is used in software and program code comprehension [83]. For example, Hybrid-DeepCom and DeepComenter tools automatically generate code comments for Java functional units [84,85]. Similarly, deep learning is also used to generate pseudo-code from program code [86], classify code according to readability [87], or summarise code [88]. In the same context, deep learning is also used to detect code vulnerability [89,90]-

Technical depth and code smell detection

Machine learning has recently been often used to detect self-admitted technical depth [91–93] or technical debt in general [94,95]. Technical debt is closely linked to code smells and anti-patterns, which are spotted [96,97], classified [98,99], or identified [100]with artificial intelligence techniques.

COVID-19 influence on software engineering

During the COVID-19 pandemic, highly collaborative software development teams were bound to work online in a distributed manner, and many software companies transferred to hybrid models after the pandemic. In such environments, AI can be used to enhance the well-being of developers [101,102].

Mining software repositories to improve software quality

Mining software repositories to enhance the quality of software and software maintenance

Software fault triaging has become a significant activity in software maintenance. Guo et al. [103] used convolutional neural networks to learn about developers' fault reports and then automatically perform software fault triaging. Triaging has also been performed using the K Nearest Neighbour approach on stack traces and categorical features [104]. A long short-term memory algorithm has been used to estimate the fault-fixing times [105] and dependency graphs for semantic versioning of third-party library components [106]. A fine-tuned Transformer has been employed to predict both the objective behind opening an issue and its priority level in GitHub repositories [107].

GitHub and Open source software as sources for mining software development data

Data mining on GitHub or Open source repositories has been used to create evolving project data sets for intelligent/empirical software engineering [108,109], for example, for Eclipse Modelling

Framework metamodels formation [110,111] anomaly detection [112] or identification of migration reasons [113].

3.2. Timeline of the Recent Research and Hot Topics

Figure 3. reveals that according to the average age of author keywords, the period 2020-2024 started with the research on the use of data and text mining in combination with deep learning in search-based software engineering, focusing on program comprehension, software metrics, and effort estimation in agile environments. In the middle of the period, the research was mainly linked to machine learning in software testing, mining software repositories for model-driven engineering, code smells, and software development effort estimation. Hot topics seem to be the research on the use of large language models and explainable machine learning (i.e., decision trees) for fault prediction, code understanding, ethics, and vulnerability detection.

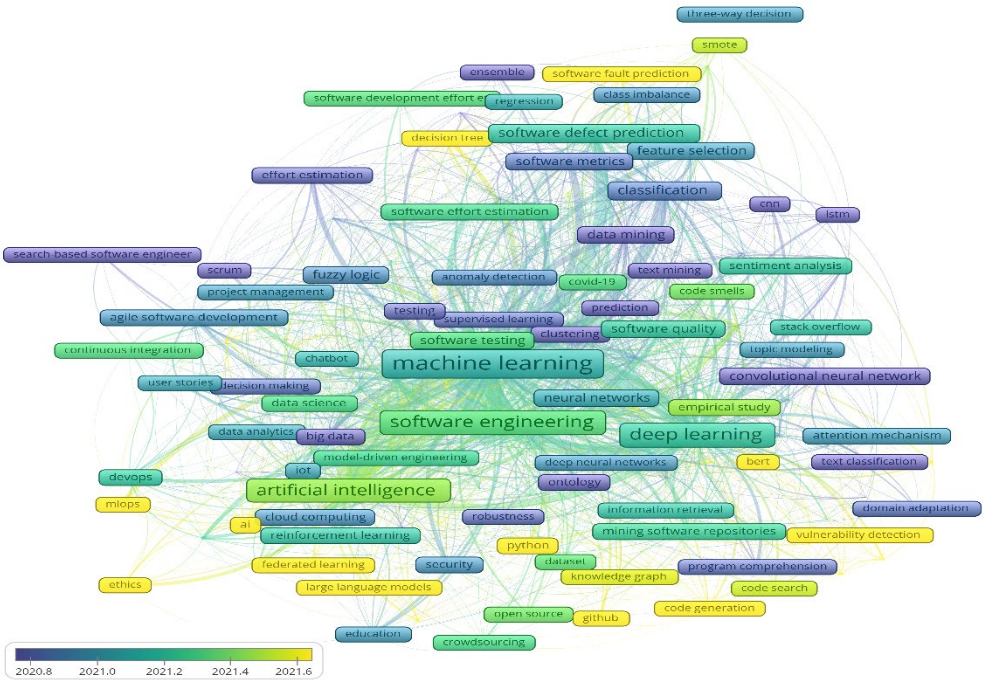

Figure 3. The research timeline landscape of the AI use in software engineering.

3.2. Research Gaps and Challenges

Our analysis also revealed the same research gaps and challenges that should be dealt with and require future research. At the time being, AI is not yet reliable enough, and software engineers and developers must thoroughly check its algorithms' output. Additionally, the increased use of AI in software engineering might result in more and more code, which will be less and less understood of how it works, establishing a positive feedback loop, which will lead to more and more checking, which might result in that, that more time will be spent on checking then developing.

AI needs vast amounts of data to create its models. That's not a problem if software engineers can train AI algorithms using the data from public and open repositories – but if they work in unique domains, appropriate training sets might not be available.

As with AI use in general, much more research is needed to resolve ethical concerns, even more so because software engineers are generally not trained and educated to deal with societal impact and ethical issues.

Extensive use of AI in software engineering will require new specialists' skills, which should be incorporated into software engineering curricula. Advanced tools, especially large language models, require sizeable computational power, storage space, and energy supply, which might increase costs. AI-based tools may require extra licensing fees.

3.4. Possible Future Research Trends

Based on the above analysis, several feature directions come to mind:

- Development of transparent, fair, ethical, responsible, and sustainable intelligent software development processes.
- Self-adapting software that adapts to evolving user requirements.
- Self-healing and self-reflecting software returns to a more functional condition after faults or performance and cybersecurity issues.
- Collaborative software development eco-systems where AI partners with human developers take team dynamics and self-organization into account.
- New software engineering curricula.
- Adaptive continuous learning platforms for software developers and engineers.

4. Conclusions

In conclusion, our analysis revealed that research on AI has significantly impacted software development in recent years. From natural language processing in software engineering, the use of artificial intelligence in the management of software development life cycle, the use of machine learning in fault/defect prediction and effort estimation, employment of deep learning in intelligent software engineering and code management to mining software repositories to improve software quality, AI has changed the way developers and engineers build software. The convergence of AI and software engineering has the potential to significantly reduce needed resources, improve quality, and increase user experience with more intelligent user-centric applications. On the other hand, it may enhance the well-being of software developers and engineers with automation of repetitive tasks, reduced workload, improved and more reliable/accurate predictive analysis, and speeding up the development cycle. Finally, it can also help software managers monitor overall team status, state, and performance, providing them with notifications if a team member has been over-utilized, over-extended, or is heading to burnout.

Funding: Please add: This research received no external funding

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: No new data were created or analyzed in this study. Data sharing is not applicable to this article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Ooi, K.-B.; Tan, G.W.-H.; Al-Emran, M.; Al-Sharafi, M.A.; Capatina, A.; Chakraborty, A.; Dwivedi, Y.K.; Huang, T.-L.; Kar, A.K.; Lee, V.-H.; et al. The Potential of Generative Artificial Intelligence Across Disciplines: Perspectives and Future Directions. *Journal of Computer Information Systems* **2023**, *0*, 1–32, doi:10.1080/08874417.2023.2261010.
2. Završnik, J.; Kokol, P.; Žlahtič, B.; Blažun Vošner, H. Artificial Intelligence and Pediatrics: Synthetic Knowledge Synthesis. *Electronics* **2024**, *13*, 512, doi:10.3390/electronics13030512.
3. Lo, D. Trustworthy and Synergistic Artificial Intelligence for Software Engineering: Vision and Roadmaps 2023.
4. Belzner, L.; Gabor, T.; Wirsing, M. Large Language Model Assisted Software Engineering: Prospects, Challenges, and a Case Study. In Proceedings of the Bridging the Gap Between AI and Reality; Steffen, B., Ed.; Springer Nature Switzerland: Cham, 2024; pp. 355–374.
5. Batarseh, F.A.; Mohod, R.; Kumar, A.; Bui, J. 10 - The Application of Artificial Intelligence in Software Engineering: A Review Challenging Conventional Wisdom. In *Data Democracy*; Batarseh, F.A., Yang, R., Eds.; Academic Press, 2020; pp. 179–232 ISBN 978-0-12-818366-3.
6. Sofian, H.; Yunus, N.A.M.; Ahmad, R. Systematic Mapping: Artificial Intelligence Techniques in Software Engineering. *IEEE Access* **2022**, *10*, 51021–51040, doi:10.1109/ACCESS.2022.3174115.
7. Amalfitano, D.; Faralli, S.; Hauck, J.C.R.; Matalonga, S.; Distante, D. Artificial Intelligence Applied to Software Testing: A Tertiary Study. *ACM Computing Surveys* **2023**, *56*, doi:10.1145/3616372.

8. Majumdar, S.; Paul, S.; Paul, D.; Bandyopadhyay, A.; Chattopadhyay, S.; Das, P.P.; Clough, P.D.; Majumder, P. Generative AI for Software Metadata: Overview of the Information Retrieval in Software Engineering Track at FIRE 2023 2023.
9. Kulkarni, V.; Kolhe, A.; Kulkarni, J. Intelligent Software Engineering: The Significance of Artificial Intelligence Techniques in Enhancing Software Development Lifecycle Processes. In Proceedings of the Intelligent Systems Design and Applications; Abraham, A., Gandhi, N., Hanne, T., Hong, T.-P., Nogueira Rios, T., Ding, W., Eds.; Springer International Publishing: Cham, 2022; pp. 67–82.
10. Satpute, R.S.; Agrawal, A. A Critical Study of Pragmatic Ambiguity Detection in Natural Language Requirements. *International Journal of Intelligent Systems and Applications in Engineering* **2023**, *11*, 249–259.
11. S, P.; C. B., C.; Raju, L.K. Developer's Roadmap to Design Software Vulnerability Detection Model Using Different AI Approaches. *IEEE Access* **2022**, *10*, 75637–75656, doi:10.1109/ACCESS.2022.3191115.
12. Daun, M.; Brings, J. How ChatGPT Will Change Software Engineering Education. In Proceedings of the Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1; Association for Computing Machinery: New York, NY, USA, June 30 2023; pp. 110–116.
13. Kokol, P. Synthetic Knowledge Synthesis in Hospital Libraries. *Journal of Hospital Librarianship* **2023**, *0*, 1–8, doi:10.1080/15323269.2023.2291282.
14. Železnik, U.; Kokol, P.; Starc, J.; Železnik, D.; Završnik, J.; Vošner, H.B. Research Trends in Motivation and Weight Loss: A Bibliometric-Based Review. *Healthcare* **2023**, *11*, 3086, doi:10.3390/healthcare11233086.
15. Farooq, U.; Nasir, A.; Khan, K.I. An Assessment of the Quality of the Search Strategy: A Case of Bibliometric Studies Published in Business and Economics. *Scientometrics* **2023**, *128*, 4855–4874, doi:10.1007/s11192-023-04765-8.
16. G20 - Background Brief.
17. Kokol, P. Discrepancies among Scopus and Web of Science, Coverage of Funding Information in Medical Journal Articles: A Follow-up Study. *Journal of the Medical Library Association* **2023**, *111*, 703–709, doi:10.5195/jmla.2023.1513.
18. Kokol, P.; Železnik, D.; Završnik, J.; Blažun Vošner, H. Nursing Research Literature Production in Terms of the Scope of Country and Health Determinants: A Bibliometric Study. *Journal of Nursing Scholarship* **2019**, *51*, 590–598, doi:10.1111/jnu.12500.
19. Vakkuri, V.; Kemell, K.-K.; Jantunen, M.; Abrahamsson, P. "This Is Just a Prototype": How Ethics Are Ignored in Software Startup-Like Environments. *Lecture Notes in Business Information Processing* **2020**, *383 LNBIP*, 195–210, doi:10.1007/978-3-030-49392-9_13.
20. Vakkuri, V.; Kemell, K.-K.; Kultanen, J.; Abrahamsson, P. The Current State of Industrial Practice in Artificial Intelligence Ethics. *IEEE Software* **2020**, *37*, 50–57, doi:10.1109/MS.2020.2985621.
21. Pasricha, S.; Wolf, M. Ethical Design of Computers: From Semiconductors to IoT and Artificial Intelligence. *IEEE Design & Test* **2024**, *41*, 7–16, doi:10.1109/MDAT.2023.3277815.
22. Barletta, V.S.; Caivano, D.; Gigante, D.; Ragone, A. A Rapid Review of Responsible AI Frameworks: How to Guide the Development of Ethical AI.; 2023; pp. 358–367.
23. Nazim, M.; Wali Mohammad, C.; Sadiq, M. A Comparison between Fuzzy AHP and Fuzzy TOPSIS Methods to Software Requirements Selection. *Alexandria Engineering Journal* **2022**, *61*, 10851–10870, doi:10.1016/j.aej.2022.04.005.
24. Ali, S.; Ullah, N.; Abrar, M.F.; Yang, Z.; Huang, J.; Ali, R. Fuzzy Multicriteria Decision-Making Approach for Measuring the Possibility of Cloud Adoption for Software Testing. *Scientific Programming* **2020**, *2020*, doi:10.1155/2020/6597316.
25. Le, T.-A.; Huynh, Q.-T.; Nguyen, T.-T.-N.; Thi, M.-H.T. A New Method for Enhancing Software Effort Estimation by Using ANFIS-Based Approach. *Lecture Notes of the Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering, LNICST* **2021**, *379*, 195–210, doi:10.1007/978-3-030-77424-0_16.
26. Rathi, N.; Srivathsav, R.; Chitlangia, R.; Pachghare, V.K. Automatic Selenium Code Generation for Testing. *Advances in Intelligent Systems and Computing* **2020**, *1039*, 194–200, doi:10.1007/978-3-030-30465-2_22.
27. Hooda, S.; Sood, V.M.; Singh, Y.; Dalal, S.; Sood, M. *Agile Software Development: Trends, Challenges and Applications*; Agile Software Development: Trends, Challenges and Applications; 2023; p. 365; ISBN 978-1-119-89683-8.
28. Fehlmann, T.; Kranich, E. ART for Agile: Autonomous Real-Time Testing in the Product Development Cycle. *Communications in Computer and Information Science* **2021**, *1442*, 377–390, doi:10.1007/978-3-030-85521-5_25.
29. Shankar, S.P.; Chaudhari, S.S. Framework for the Automation of SDLC Phases Using Artificial Intelligence and Machine Learning Techniques. *International Journal on Recent and Innovation Trends in Computing and Communication* **2023**, *11*, 379–390, doi:10.17762/ijritcc.v11i6s.6944.
30. Jaiswal, A.; Raikwal, J.; Raikwal, P. A Hybrid Cost Estimation Method for Planning Software Projects Using Fuzzy Logic and Machine Learning. *International Journal of Intelligent Systems and Applications in Engineering* **2024**, *12*, 696–707.

31. Singh, B.K.; Tiwari, S.; Mishra, K.K.; Punhani, A. Extended COCOMO: Robust and Interpretable Neuro-Fuzzy Modelling. *International Journal of Computational Vision and Robotics* **2021**, *11*, 41–65, doi:10.1504/IJCVR.2021.111873.
32. Abnane, I.; Idri, A.; Abran, A. Optimized Fuzzy Clustering-Based k-Nearest Neighbors Imputation for Mixed Missing Data in Software Development Effort Estimation. *Journal of Software: Evolution and Process* **2023**, doi:10.1002/smr.2529.
33. Sokolovska, Z.; Dudnyk, O. DEVISING A TECHNOLOGY FOR MANAGING OUTSOURCING IT-PROJECTS WITH THE APPLICATION OF FUZZY LOGIC. *Eastern-European Journal of Enterprise Technologies* **2021**, *2*, 52–65, doi:10.15587/1729-4061.2021.224529.
34. Anes, V.; Abreu, A.; Santos, R. A New Risk Assessment Approach for Agile Projects.; 2020; pp. 67–72.
35. Kataev, M.; Bulysheva, L.; Xu, L.; Ekhlakov, Y.; Permyakova, N.; Jovanovic, V. Fuzzy Model Estimation of the Risk Factors Impact on the Target of Promotion of the Software Product. *Enterprise Information Systems* **2020**, *14*, 797–811, doi:10.1080/17517575.2020.1713407.
36. Balasubramanian, P. Automation in Data Science, Software, and Information Services. *Springer Handbooks* **2023**, *Part F674*, 989–1014, doi:10.1007/978-3-030-96729-1_46.
37. Abadeer, M.; Sabetzadeh, M. Machine Learning-Based Estimation of Story Points in Agile Development: Industrial Experience and Lessons Learned.; 2021; Vol. 2021-September, pp. 106–115.
38. Klus, H.; Knieke, C.; Rausch, A.; Wittek, S. Software Engineering Meets Artificial Intelligence. *Electronic Communications of the EASST* **2023**, *82*, doi:10.14279/tuj.eceasst.82.1224.
39. Sawant, A.A.; Devanbu, P. Naturally!: How Breakthroughs in Natural Language Processing Can Dramatically Help Developers. *IEEE Software* **2021**, *38*, 118–123, doi:10.1109/MS.2021.3086338.
40. Ahmed, H.A.; Bawany, N.Z.; Shamsi, J.A. Capbug-a Framework for Automatic Bug Categorization and Prioritization Using Nlp and Machine Learning Algorithms. *IEEE Access* **2021**, *9*, 50496–50512, doi:10.1109/ACCESS.2021.3069248.
41. Althar, R.R.; Samanta, D.; Kaur, M.; Alnuaim, A.A.; Aljaffan, N.; Aman Ullah, M. Software Systems Security Vulnerabilities Management by Exploring the Capabilities of Language Models Using NLP. *Computational Intelligence and Neuroscience* **2021**, *2021*, doi:10.1155/2021/8522839.
42. Charitsis, C.; Piech, C.; Mitchell, J.C. Using NLP to Quantify Program Decomposition in CS1.; 2022; pp. 113–120.
43. dos Santos, G.E.; Figueiredo, E. Commit Classification Using Natural Language Processing: Experiments over Labeled Datasets.; 2020.
44. Wong, M.-F.; Guo, S.; Hang, C.-N.; Ho, S.-W.; Tan, C.-W. Natural Language Generation and Understanding of Big Code for AI-Assisted Programming: A Review. *Entropy* **2023**, *25*, doi:10.3390/e25060888.
45. Koreeda, Y.; Morishita, T.; Imaichi, O.; Sogawa, Y. LARCH: Large Language Model-Based Automatic Readme Creation with Heuristics.; 2023; pp. 5066–5070.
46. Di Sipio, C.; Di Rocco, J.; Di Ruscio, D.; Nguyen, P.T. MORGAN: A Modeling Recommender System Based on Graph Kernel. *Software and Systems Modeling* **2023**, *22*, 1427–1449, doi:10.1007/s10270-023-01102-8.
47. Jánki, Z.R.; Bilicki, V. The Impact of the Web Data Access Object (WebDAO) Design Pattern on Productivity. *Computers* **2023**, *12*, doi:10.3390/computers12080149.
48. Pauzi, Z.; Capiluppi, A. Applications of Natural Language Processing in Software Traceability: A Systematic Mapping Study. *Journal of Systems and Software* **2023**, *198*, doi:10.1016/j.jss.2023.111616.
49. Zhao, L.; Alhoshan, W.; Ferrari, A.; Letsholo, K.J.; Ajagbe, M.A.; Chioasca, E.-V.; Batista-Navarro, R.T. Natural Language Processing for Requirements Engineering. *ACM Computing Surveys* **2021**, *54*, doi:10.1145/3444689.
50. Umar, M.A.; Lano, K. Advances in Automated Support for Requirements Engineering: A Systematic Literature Review. *Requirements Engineering* **2024**, doi:10.1007/s00766-023-00411-0.
51. Arulmohan, S.; Meurs, M.-J.; Mosser, S. Extracting Domain Models from Textual Requirements in the Era of Large Language Models.; 2023; pp. 580–587.
52. Vahabi, S.; Hozhabri, A. Automatic Use Case Classification Based on Topic Grouping for Requirements Engineering. *Innovations in Systems and Software Engineering* **2024**, *20*, 85–96, doi:10.1007/s11334-023-00535-0.
53. Kochbati, T.; Li, S.; Gérard, S.; Mraidha, C. From User Stories to Models: A Machine Learning Empowered Automation.; 2021; pp. 28–40.
54. Gunes, T.; Aydemir, F.B. Automated Goal Model Extraction from User Stories Using NLP.; 2020; Vol. 2020-August, pp. 382–387.
55. Wang, Y.; Hu, K.; Jiang, B.; Xia, X.; Tang, X.-S. A Systematic Literature Review of Software Traceability Links Automation Techniques. *Jisuanji Xuebao/Chinese Journal of Computers* **2023**, *46*, 1919–1946, doi:10.11897/SP.J.1016.2023.01919.
56. Hey, T.; Keim, J.; Koziolek, A.; Tichy, W.F. NoRBERT: Transfer Learning for Requirements Classification.; 2020; Vol. 2020-August, pp. 169–179.

57. Alhoshan, W.; Ferrari, A.; Zhao, L. Zero-Shot Learning for Requirements Classification: An Exploratory Study. *Information and Software Technology* **2023**, *159*, doi:10.1016/j.infsof.2023.107202.
58. Ezzini, S.; Abualhaija, S.; Arora, C.; Sabetzadeh, M.; Briand, L.C. Using Domain-Specific Corpora for Improved Handling of Ambiguity in Requirements.; 2021; pp. 1485–1497.
59. Sarmiento-Calisaya, E.; do Prado Leite, J.C.S. Early Analysis of Requirements Using NLP and Petri-Nets. *Journal of Systems and Software* **2024**, *208*, doi:10.1016/j.jss.2023.111901.
60. Shen, Y.; Breaux, T. Stakeholder Preference Extraction from Scenarios. *IEEE Transactions on Software Engineering* **2024**, *50*, 69–84, doi:10.1109/TSE.2023.3333265.
61. García, S.E.M.; Fernández-y-Fernández, C.A.; Pérez, E.G.R. Classification of Non-Functional Requirements Using Convolutional Neural Networks. *Programming and Computer Software* **2023**, *49*, 705–711, doi:10.1134/S0361768823080133.
62. Tikayat Ray, A.; Cole, B.F.; Pinon Fischer, O.J.; Bhat, A.P.; White, R.T.; Mavris, D.N. Agile Methodology for the Standardization of Engineering Requirements Using Large Language Models. *Systems* **2023**, *11*, doi:10.3390/systems11070352.
63. Calle Gallego, J.M.; Zapata Jaramillo, C.M. QUARE: Towards a Question-Answering Model for Requirements Elicitation. *Automated Software Engineering* **2023**, *30*, doi:10.1007/s10515-023-00386-w.
64. Georges, T.; Rice, L.; Huchard, M.; König, M.; Nebut, C.; Tibermacine, C. Guiding Feature Models Synthesis from User-Stories: An Exploratory Approach.; 2023; pp. 65–70.
65. Rohmann, A. Improving the Completeness of Acceptance Criteria.; 2023; Vol. 3378.
66. Heng, S.; Snoeck, M.; Tsilionis, K. Building a Software Architecture out of User Stories and BDD Scenarios: Research Agenda.; 2022; Vol. 3134, pp. 40–46.
67. Perkusich, M.; Chaves e Silva, L.; Costa, A.; Ramos, F.; Saraiva, R.; Freire, A.; Dilorenzo, E.; Dantas, E.; Santos, D.; Gorgônio, K.; et al. Intelligent Software Engineering in the Context of Agile Software Development: A Systematic Literature Review. *Information and Software Technology* **2020**, *119*, doi:10.1016/j.infsof.2019.106241.
68. Jadhav, A.; Kaur, M.; Akter, F. Evolution of Software Development Effort and Cost Estimation Techniques: Five Decades Study Using Automated Text Mining Approach. *Mathematical Problems in Engineering* **2022**, *2022*, doi:10.1155/2022/5782587.
69. Priya Varshini, A.G.; Anitha Kumari, K.; Varadarajan, V. Estimating Software Development Efforts Using a Random Forest-Based Stacked Ensemble Approach. *Electronics (Switzerland)* **2021**, *10*, doi:10.3390/electronics10101195.
70. Singal, P.; Kumari, A.C.; Sharma, P. Estimation of Software Development Effort: A Differential Evolution Approach.; 2020; Vol. 167, pp. 2643–2652.
71. De Carvalho, H.D.P.; Fagundes, R.; Santos, W. Extreme Learning Machine Applied to Software Development Effort Estimation. *IEEE Access* **2021**, *9*, 92676–92687, doi:10.1109/ACCESS.2021.3091313.
72. Jadhav, A.; Shandilya, S.K. Reliable Machine Learning Models for Estimating Effective Software Development Efforts: A Comparative Analysis. *Journal of Engineering Research (Kuwait)* **2023**, *11*, 362–376, doi:10.1016/j.jer.2023.100150.
73. Rodríguez Sánchez, E.; Vázquez Santacruz, E.F.; Cervantes Maceda, H. Effort and Cost Estimation Using Decision Tree Techniques and Story Points in Agile Software Development. *Mathematics* **2023**, *11*, doi:10.3390/math11061477.
74. Gupta, N.; Sinha, R.R.; Goyal, A.; Sunda, N.; Sharma, D. Analyze the Performance of Software by Machine Learning Methods for Fault Prediction Techniques. *International Journal on Recent and Innovation Trends in Computing and Communication* **2023**, *11*, 178–187, doi:10.17762/ijritcc.v11i5s.6642.
75. Mcmurray, S.; Sodhro, A.H. A Study on ML-Based Software Defect Detection for Security Traceability in Smart Healthcare Applications. *Sensors* **2023**, *23*, doi:10.3390/s23073470.
76. Nasser, A.B.; Ghanem, W.; Abdul-Qawy, A.S.H.; Ali, M.A.H.; Saad, A.-M.; Ghaleb, S.A.A.; Alduais, N. A Robust Tuned K-Nearest Neighbours Classifier for Software Defect Prediction. *Lecture Notes in Networks and Systems* **2023**, *573 LNNS*, 181–193, doi:10.1007/978-3-031-20429-6_18.
77. Khan, M.A.; Elmitwally, N.S.; Abbas, S.; Aftab, S.; Ahmad, M.; Fayaz, M.; Khan, F. Software Defect Prediction Using Artificial Neural Networks: A Systematic Literature Review. *Scientific Programming* **2022**, *2022*, doi:10.1155/2022/2117339.
78. Hilmi, M.A.A.; Puspaningrum, A.; Darsih; Siahaan, D.O.; Samosir, H.S.; Rahma, A.S. Research Trends, Detection Methods, Practices, and Challenges in Code Smell: SLR. *IEEE Access* **2023**, *11*, 129536–129551, doi:10.1109/ACCESS.2023.3334258.
79. Soomlek, C.; van Rijn, J.N.; Bonsangue, M.M. Automatic Human-Like Detection of Code Smells. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **2021**, *12986 LNAI*, 19–28, doi:10.1007/978-3-030-88942-5_2.
80. Li, C.; Yuan, Y.; Yang, J. Causally Remove Negative Confound Effects of Size Metric for Software Defect Prediction. *Applied Sciences (Switzerland)* **2022**, *12*, doi:10.3390/app12031387.
81. Khan, F.; Lingala, G. Machine Learning Techniques For Software Component Reusability.; 2022.

82. Pontillo, V.; Palomba, F.; Ferrucci, F. Toward Static Test Flakiness Prediction: A Feasibility Study.; 2021; pp. 19–24.
83. Yang, K.; Wang, J.; Song, Z. Learning a Holistic and Comprehensive Code Representation for Code Summarization. *Journal of Systems and Software* **2023**, *203*, doi:10.1016/j.jss.2023.111746.
84. Hu, X.; Li, G.; Xia, X.; Lo, D.; Jin, Z. Deep Code Comment Generation with Hybrid Lexical and Syntactical Information. *Empirical Software Engineering* **2020**, *25*, 2179–2217, doi:10.1007/s10664-019-09730-9.
85. Li, B.; Yan, M.; Xia, X.; Hu, X.; Li, G.; Lo, D. DeepCommenter: A Deep Code Comment Generation Tool with Hybrid Lexical and Syntactical Information.; 2020; pp. 1571–1575.
86. Yang, G.; Zhou, Y.; Chen, X.; Yu, C. Fine-Grained Pseudo-Code Generation Method via Code Feature Extraction and Transformer.; 2021; Vol. 2021-December, pp. 213–222.
87. Mi, Q.; Zhan, Y.; Weng, H.; Bao, Q.; Cui, L.; Ma, W. A Graph-Based Code Representation Method to Improve Code Readability Classification. *Empirical Software Engineering* **2023**, *28*, doi:10.1007/s10664-023-10319-6.
88. Li, M.; Yu, H.; Fan, G.; Zhou, Z.; Huang, J. ClassSum: A Deep Learning Model for Class-Level Code Summarization. *Neural Computing and Applications* **2023**, *35*, 3373–3393, doi:10.1007/s00521-022-07877-z.
89. Jain, R.; Gervasoni, N.; Ndhlovu, M.; Rawat, S. A Code Centric Evaluation of C/C++ Vulnerability Datasets for Deep Learning Based Vulnerability Detection Techniques.; 2023.
90. Iannone, E.; Guadagni, R.; Ferrucci, F.; De Lucia, A.; Palomba, F. The Secret Life of Software Vulnerabilities: A Large-Scale Empirical Study. *IEEE Transactions on Software Engineering* **2023**, *49*, 44–63, doi:10.1109/TSE.2022.3140868.
91. Zhu, K.; Yin, M.; Li, Y. Detecting and Classifying Self-Admitted of Technical Debt with CNN-BiLSTM.; 2021; Vol. 1955.
92. Mock, M. Utilization of Machine Learning for the Detection of Self-Admitted Vulnerabilities. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **2024**, *14484 LNCS*, 139–145, doi:10.1007/978-3-031-49269-3_15.
93. Wang, X.; Liu, J.; Li, L.; Chen, X.; Liu, X.; Wu, H. Detecting and Explaining Self-Admitted Technical Debts with Attention-Based Neural Networks.; 2020; pp. 871–882.
94. Tsoukalas, D.; Kehagias, D.; Siavvas, M.; Chatzigeorgiou, A. Technical Debt Forecasting: An Empirical Study on Open-Source Repositories. *Journal of Systems and Software* **2020**, *170*, doi:10.1016/j.jss.2020.110777.
95. Albuquerque, D.; Guimaraes, E.; Tonin, G.; Rodriguez, P.; Perkusich, M.; Almeida, H.; Perkusich, A.; Chagas, F. Managing Technical Debt Using Intelligent Techniques - A Systematic Mapping Study. *IEEE Transactions on Software Engineering* **2023**, *49*, 2202–2220, doi:10.1109/TSE.2022.3214764.
96. Zhang, Y.; Dong, C. MARS: Detecting Brain Class/Method Code Smell Based on Metric–Attention Mechanism and Residual Network. *Journal of Software: Evolution and Process* **2024**, *36*, doi:10.1002/smr.2403.
97. Malhotra, R.; Jain, B.; Kessentini, M. Examining Deep Learning's Capability to Spot Code Smells: A Systematic Literature Review. *Cluster Computing* **2023**, *26*, 3473–3501, doi:10.1007/s10586-023-04144-1.
98. Abdou, A.; Darwish, N. Severity Classification of Software Code Smells Using Machine Learning Techniques: A Comparative Study. *Journal of Software: Evolution and Process* **2024**, *36*, doi:10.1002/smr.2454.
99. Dewangan, S.; Rao, R.S.; Chowdhuri, S.R.; Gupta, M. Severity Classification of Code Smells Using Machine-Learning Methods. *SN Computer Science* **2023**, *4*, doi:10.1007/s42979-023-01979-8.
100. Shcherban, S.; Liang, P.; Tahir, A.; Li, X. Automatic Identification of Code Smell Discussions on Stack Overflow: A Preliminary Investigation.; 2020.
101. Awan, W.N.; Paasivaara, M.; Gloor, P.; Salman, I. Creating Happier and More Productive Software Engineering Teams through AI and Machine Learning.; 2024; Vol. 3621.
102. Georgiou, K.; Charmanas, K.; Papageorgiadis, K.; Mittas, N.; Christidis, G.; Angelis, L. A Data-Driven Framework for Knowledge Exchange Analysis of Development Issues in Medical Applications: A Case Study of COVID-19.; 2023; pp. 386–393.
103. Guo, S.; Zhang, X.; Yang, X.; Chen, R.; Guo, C.; Li, H.; Li, T. Developer Activity Motivated Bug Triaging: Via Convolutional Neural Network. *Neural Processing Letters* **2020**, *51*, 2589–2606, doi:10.1007/s11063-020-10213-y.
104. Sabor, K.K.; Hamdaqa, M.; Hamou-Lhadj, A. Automatic Prediction of the Severity of Bugs Using Stack Traces and Categorical Features. *Information and Software Technology* **2020**, *123*, doi:10.1016/j.infsof.2019.106205.
105. Ardimento, P. Enhancing Bug-Fixing Time Prediction with LSTM-Based Approach. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* **2024**, *14484 LNCS*, 68–79, doi:10.1007/978-3-031-49269-3_7.
106. Li, W.; Wu, F.; Fu, C.; Zhou, F. A Large-Scale Empirical Study on Semantic Versioning in Golang Ecosystem.; 2023; pp. 1604–1614.
107. Izadi, M.; Akbari, K.; Heydarnoori, A. Predicting the Objective and Priority of Issue Reports in Software Repositories. *Empirical Software Engineering* **2022**, *27*, doi:10.1007/s10664-021-10085-3.

108. Lewowski, T.; Madeyski, L. Creating Evolving Project Data Sets in Software Engineering. *Studies in Computational Intelligence* **2020**, *851*, 1–14, doi:10.1007/978-3-030-26574-8_1.
109. Kalliamvakou, E.; Singer, L.; Gousios, G.; German, D.M.; Blincoe, K.; Damian, D. The Promises and Perils of Mining GitHub.; 2014; pp. 92–101.
110. Babur, Ö.; Constantinou, E.; Serebrenik, A. Language Usage Analysis for EMF Metamodels on GitHub. *Empirical Software Engineering* **2024**, *29*, doi:10.1007/s10664-023-10368-x.
111. Robles, G.; Chaudron, M.R.V.; Jolak, R.; Hebig, R. A Reflection on the Impact of Model Mining from GitHub. *Information and Software Technology* **2023**, *164*, doi:10.1016/j.infsof.2023.107317.
112. Wijesinghe, N.; Hemmati, H. Log-Based Anomaly Detection of Enterprise Software: An Empirical Study.; 2023; pp. 12–23.
113. He, H.; He, R.; Gu, H.; Zhou, M. A Large-Scale Empirical Study on Java Library Migrations: Prevalence, Trends, and Rationales.; 2021; pp. 478–490.