ChatGPT in Action: Analyzing Its Use in Software Development

Arifa I. Champa, Md Fazle Rabbi, Costain Nachuma, Minhaz F. Zibran
Department of Computer Science, Idaho State University, Pocatello, ID, USA
{arifaislamchampa,mdfazlerabbi,costainnachuma,minhazzibran}@isu.edu

ABSTRACT

The emergence of AI tools such as ChatGPT is being used to assist with software development, but little is known of how developers utilize these tools as well as the capabilities of these tools in software engineering tasks. Using the DevGPT dataset, we conduct quantitative analyses of the tasks developers seek assistance from ChatGPT and how effectively ChatGPT addresses them. We also examine the impact of initial prompt quality on conversation length. The findings reveal where ChatGPT is most and least suited to assist in the identified 12 software development tasks. The insights from this research would guide the software developers, researchers, and AI tool providers in optimizing these tools for more effective programming aid.

KEYWORDS

ChatGPT conversation, software development tasks, task efficiency, prompt quality

ACM Reference Format:
Arifa I. Champa, Md Fazle Rabbi, Costain Nachuma, Minhaz F. Zibran. 2024. ChatGPT in Action: Analyzing Its Use in Software Development. In *21st International Conference on Mining Software Repositories (MSR '24), April 15–16, 2024, Lisbon, Portugal.*

1 INTRODUCTION

The integration of Artificial Intelligence (AI) into the field of software development represents a significant transformation, changing the traditional methods used in the industry. AI has been increasingly utilized in various technical and creative problem-solving aspects [21], as seen in tools like ChatGPT, developed by OpenAI [31]. ChatGPT showcases the power of generative pre-trained transformers [49] in comprehending and generating human-like text, making it a versatile asset in software development [3, 37, 47].

AI tools like ChatGPT have gained prominence in software development by assisting in tasks ranging from code generation [8, 24] to decision-making processes [7, 39]. These tools are reshaping the software development workflow, leading to increased efficiency and productivity [41]. However, despite the widespread adoption of AI tools in software development, there is a noticeable absence of comprehensive academic research that delves into the specific ways these tools are used in real-life scenarios. This research gap hinders the optimization and full utilization of AI tools in this domain.

MSR '24, April 15–16, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0587-8/24/04…$15.00
https://doi.org/10.1145/3643991.3645077

Our study aims to explore how software developers use ChatGPT in real-life situations. We want to understand what tasks they use it for, how well it works for different types of tasks, and whether the way they start a conversation affects its length. In our research, we address the following three research questions (RQs):

RQ1: What types of tasks do software developers seek assistance with from ChatGPT?
—Understanding these tasks can reveal the areas where ChatGPT can help one the most. This will also highlight the current limitations and guide future improvements of AI in software development.

RQ2: Are there any particular categories of tasks where the developer-ChatGPT collaborations/conversations can result in more or less effective assistance from ChatGPT?
— By analyzing how effectively the developer-ChatGPT collaboration/conversation yields solutions in various categories of software development tasks, we can identify areas where it performs excellently and areas that require further development. The findings will also inform developers about the most reliable uses of AI assistants, optimizing their workflow and reducing time spent on tasks that are less suited for AI intervention.

RQ3: Does the quality of the initial prompt affect the efficiency of developer-ChatGPT conversations/collaborations in completing a task?
— Examining how the quality of the initial prompt influences the length of ChatGPT conversations can provide valuable insights into optimizing human-AI communication. We measure the initial prompt quality in terms of sentence complexity, grammar errors, and readability score. This understanding is crucial for improving the user experience, enhancing the response accuracy of AI, and reducing time spent in clarifying or correcting misunderstandings.

To address the above three RQs, we analyze 2,865 conversations of software developers with ChatGPT in the DevGPT dataset, which contains a total of 17,622 ChatGPT prompts.

2 DATASET

We use the publicly available DevGPT [48] dataset, which contains a curated collection of JSON files derived from six different sources. The sources are "GitHub Code File", "GitHub Commit", "GitHub Issue", "GitHub Pull Request", "Hacker News", and "GitHub Discussion". We take the 'snapshot_20230914' that we get on September 15, 2023. This snapshot comprises of 17,622 pairs of prompts and answers in 2,865 conversations between developers and ChatGPT, which include 12,031 code snippets written in diverse programming languages.

Figure 1 summarizes the methodology of our study. Phase-0 includes the steps for extracting data from DevGPT. Each of the six sources of DevGPT contains a 'ChatGPTSharing' attribute that holds the ChatGPT conversations and the number of prompts in a conversation along with other attributes. Each conversation in the DevGPT dataset has three attributes: prompt, answer, and code-list (from the answer). In the context of DevGPT, a 'prompt' represents

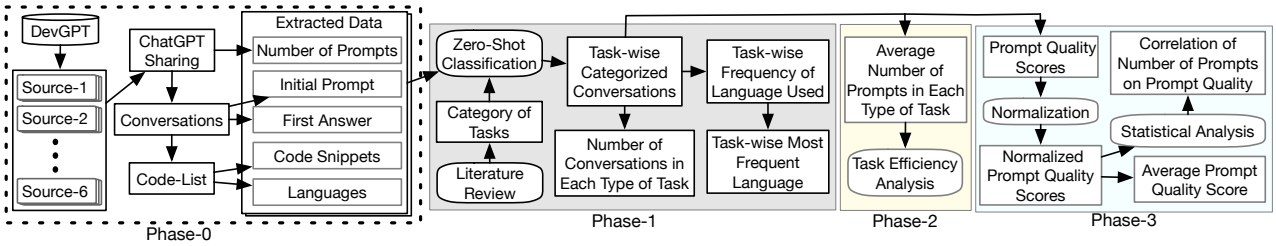

Figure 1: Procedural steps at different phases of our study

a user-input to ChatGPT, an 'answer' is the response from ChatGPT, and code-list includes the collection of code snippets present in an answer identified with the languages the snippets are written in.

For each of the 2,865 conversations from all of the six sources of DevGPT, we extract the total number of prompts, the initial prompt, the first answer, the list of any code snippets in the first answer, and the language in which each of the code snippets is written in.

3 ANALYSIS AND FINDINGS

The steps outlined in Phase-1, Phase-2, and Phase-3 in Figure 1 correspond to our approach for addressing RQ1, RQ2, and RQ3, respectively. We assume that each conversation between the developer and ChatGPT is meant to complete a task and we refer to such a conversation as simply a ChatGPT conversation.

3.1 Tasks Requiring ChatGPT's Assistance

To identify the types of tasks the developers mostly seek assistance from ChatGPT, we first identify 12 categories of software engineering tasks that are frequently discussed in the literature [3–5, 9–11, 19, 23, 26, 27, 29, 42, 43, 45]. These task categories are presented in the leftmost column of Table 1.

Table 1: Types of tasks developers present to ChatGPT

Types of Tasks	Sim.	C_T*	Lang.	Usage (%)
Code Quality Management (CQM)	0.60	890	Python	23.21
Commit Issue Resolution (CIR)	0.59	488	Python	18.92
Documentation Generation (DG)	0.60	438	Shell	27.06
New Feature Implementation (NFI)	0.81	257	Shell	73.68
Code Manipulation and Generation (CMG)	0.81	241	Shell	53.14
Energy-aware Development (ED)	0.66	172	Shell	28.41
Testing and Quality Assurance (TQA)	0.79	119	Shell	35.94
Development and Environment Setup (DES)	0.70	119	Shell	32.43
Debugging and Error Management (DEM)	0.74	91	Shell	34.72
Code Learning (CL)	0.81	30	Shell	72.00
Security Management (SM)	0.68	15	Bash	40.00
Software Development Management and Optimization (SMO)	0.47	5	None	-

*Here, C_T = number of conversations

3.1.1 Task-wise Categorization of Conversations. For each ChatGPT conversation, we determine what type of task is handled in that conversation. Thus, we task-wise categorize the conversations using the 'facebook/bart-large-mnli' model for zero-shot classification [12]. We choose zero-shot classification because it can categorize fairly without needing pre-labeled data. Moreover, the 'facebook/bart-large-mnli' model is particularly suitable for our task as it interprets and handles natural language tasks effectively [25]. For each conversation, the zero-shot classifier assigns a probability score to indicate how likely it is that the conversation belongs to a particular task category. Then, we associate each conversation with the task category that has the highest probability score among all the task categories.

If the categorization is correct, we expect that there will remain some similarities among the initial prompts of the conversation within a category. Hence, for each conversation pair within a task category, we compute the pair-wise cosine similarity. Then, for each type of task, we calculate the average of the computed pair-wise cosine similarities. The average similarities of the tasks of each type are presented in the second column from the left in Table 1. The high scores for almost all types of tasks imply that the classifier has correctly categorized the conversations. We also manually check 20 randomly picked prompts to assess task-wise conversation categorization and they seem to be assigned accurately. Thus the correctness of the categorization is verified in the aforementioned two ways.

3.1.2 Results. For each type of tasks, we compute the total number of conversations that deal with the tasks of that type, as presented in the third column in Table 1. We also identify the languages most frequently used in each type of task upon computing the frequency of all the languages used in that task. Task-wise most frequent languages and the percentages of code snippets written in the most frequent languages are shown in the rightmost two columns of Table 1. For every type of task, we find only one most frequent language except that no language is found for the software development management and optimization task.

As seen in Table 1, code quality management and commit issue resolution are the two types of tasks involving the highest number of conversations, and both of these task types deal with source code written in Python. The least number (05) of conversations deal with software development management and optimization tasks. This indicates that software developers seek the most assistance with code quality management but they need minimum assistance from ChatGPT in day-to-day software development management and optimization tasks.

It is also interesting to find that, in nine of the 12 task categories for which the developers seek assistance from ChatGPT, the most frequently involved code snippets are written as bash or other kinds of shell scripts. Based on the observations we derive the answer to RQ1 as follows.

Ans. to RQ1: *The software developers seek the most assistance from ChatGPT while dealing with Python code in quality management and commit issue resolution tasks. Across many different types of tasks, they also commonly seek aid in dealing with shell scripting.*

3.2 Task-wise Efficiency

Now that we have identified particular types of tasks in which the developers seek the most assistance, the next question that arises is how well ChatGPT assists in those tasks. If a task is resolved in a short conversation involving only a few prompts, we can consider the corresponding ChatGPT conversation to be efficient. Thus, for each type of task, we compute the average number of prompts in a conversation required to complete a task.

Table 2: Average number of prompts in each task category

Task Type	# of Prompts		Task Type	# of Prompts	
	Avg.	Std. Dev.		Avg.	Std. Dev.
SMO	2.80	1.30	ED	4.62	7.17
NFI	2.88	4.32	SM	5.13	6.12
CL	3.40	6.68	TQA	5.18	8.73
DEM	3.66	4.82	CQM	6.63	13.96
CIR	3.83	6.13	DG	11.03	20.84
CMG	4.02	7.72	DES	11.58	53.36

In Table 2, for each type of task, the average number of prompts and the standard deviations are presented in the ascending order of the average number of prompts. As seen in the table, the tasks in SMO (Software Development Management and Optimization) and NFI (New Feature Implementation) categories have the lowest average number of prompts (2.8 and 2.88 respectively) as well as the lowest standard deviation (1.3 and 4.32 respectively) compared to other types of tasks. This implies that the tasks of these two types are consistently accomplished efficiently with the help of ChatGPT.

On the contrary, the tasks in DES (Development and Environment Setup) and DG (Documentation Generation) categories on average require more than 11 prompts, with very high standard deviations of 53.36 and 20.84 respectively. These high numbers of prompts along with the high standard deviations suggest inconsistency and an extremely limited capacity of ChatGPT conversations/collaborations in completing the tasks. The task type CQM (Code Quality Management) is also worth mentioning here. The high standard deviation (13.96) for tasks of this type indicates that the ChatGPT conversation efficiency substantially fluctuates in completing tasks of this category. Based on the observations discussed above, we now formulate the answer to RQ2 as follows.

Ans. to RQ2: *Developer-ChatGPT conversation/collaboration is the most efficient in achieving assistance with Software Development Management and Optimization as well as New Feature Implementation but the least efficient with Documentation Generation, Development and Environment Setup as well as Code Quality Management. For the tasks of the rest seven categories, fairly efficient assistance is achieved from Developer-ChatGPT conversation/collaboration.*

3.3 Impact of Prompt Quality on Efficiency

Once we have distinguished categories of tasks for which a developer-ChatGPT conversation/collaboration can be the most and the least efficient, it is now worth investigating if the quality of the initial prompt affects the efficiency of developer-ChatGPT conversations/collaborations in completing a task.

3.3.1 Metrics. To measure the quality of a ChatGPT prompt, we use three metrics: readability score (RS), sentence complexity (SC), and grammar errors (GE). Prompts having complex sentences and grammar error scores can impose difficulty on ChatGPT in correctly interpreting the developers' questions and needs. A high readability score indicates that a higher school grade level competency is required to understand the text [34]. Therefore, lower scores of these three metrics suggest better prompt quality. We compute the scores of these three metrics to measure the quality of the initial prompt in each developer-ChatGPT conversation.

To measure the readability score for the initial prompt in each conversation, we first calculate individual scores using four distinct readability tests: Flesch-Kincaid, SMOG Index, Coleman-Liau Index, and Automated Readability Index [40]. Then, for each conversation, we average these four scores to obtain a singular readability score.

Then we measure the complexity of each sentence in the initial prompt in each conversation by first processing the text through a neural network-based NLP pipeline, which includes tokenization, part-of-speech tagging, dependency parsing, and named entity recognition [44]. Then we calculate the average of the complexity scores for all the sentences in the prompt.

In measuring grammar errors, an entire prompt is considered as a single unit of text. Thus, for the initial prompt in each conversation, we employ a rule-based system [32] that uses pattern matching and linguistic rules to capture the total number of all detected grammar errors. We utilize Spacy [33], 'language_tool_python' [32], and Textstat Python package [34] to compute sentence complexity, grammar errors, and readability score respectively. After computing the aforementioned three metrics for each initial prompt, we apply the min-max normalization [46] on them to scale the scores between 0.0 and 1.0. This is done to make the scores uniform and comparable, regardless of their initial ranges.

3.3.2 Results. For each type of tasks, we calculate the average readability scores for the initial prompts in all conversations in tasks of the category. Similarly, we separately compute the task-wise average sentence complexity scores and also the task-wise average scores for grammar errors.

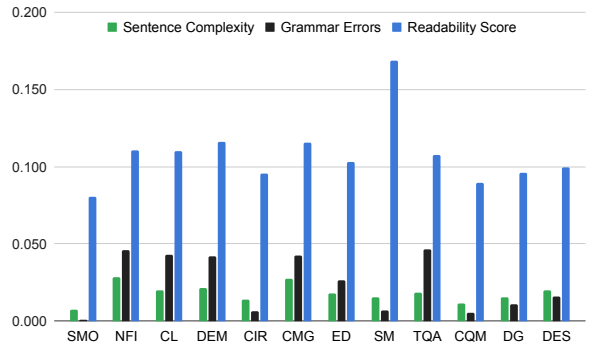

Figure 2: Quality of initial prompts for each type of tasks

Figure 2 displays the task-wise averages of sentence complexity, grammar errors, and readability scores. The consistently low scores for sentence complexity and grammar errors across all task types make it easier for ChatGPT to accurately interpret the questions and requirements of the developers. Although the readability scores across the types of tasks are consistently higher than those of sentence complexity and grammar errors, the prompts possibly

remain easy to comprehensible to ChatGPT. The interesting part is, no contrasting patterns of these three prompt quality metrics are observed across the efficient and inefficient ChatGPT conversations identified in Section 3.2.

Hence, we want to determine whether there is any correlation between the conversation length (measured in the number of prompts) in each category of tasks and any of these three metrics. Recall that the fewer the number of prompts (i.e., the shorter the conversation length) in a developer-ChatGPT conversation, the more efficient the collaboration is. So, we use mutual information [6], distance correlation [46], and Spearman correlation [46] to measure any possible correlation because our data do not exhibit normal distribution [46].

Table 3: Correlation - conversation length and prompt quality

Measures	RS	SC	GE
Mutual Information	0.10	0.25	0.12
Distance Correlation	0.12	0.14	0.14
Spearman Correlation	-0.06	-0.13	-0.05

Table 3 shows the three measures of conversation length's correlation with the three prompt quality metrics. Very low values of all these correlation measures confirm that there exists no correlation between the initial prompt quality and conversation length/efficiency. Therefore, we derive the answer to RQ3 as follows:

Ans. to RQ3: *The quality of the initial prompt does not affect the efficiency of developer-ChatGPT conversations/collaborations in completing a task.*

4 THREATS TO VALIDITY

As we deal with unlabeled data in task-wise categorization of the ChatGPT conversations, we depend on zero-shot learning using the 'facebook/bart-large-mnli' model. One may question the accuracy of categorization achieved through this approach. However, this method is known to be effective in natural language processing tasks [25] such as ours. The cosine similarity-based verification and our manual validation yield high confidence that the categorization is accurate. In the computation of the usage of languages, the sizes of the code snippets are disregarded because we have not observed much variations in the sizes. The three prompt quality metrics might not be enough to sufficiently capture the quality of the prompts.

5 RELATED WORK

ChatGPT is used for bug fixing, programming, maintaining code quality, solving commit issues, documentation, and many other tasks in software development [3, 4, 9–11, 23, 26, 27, 29, 37, 42, 43, 45]. Although many aspects of software engineering were studied [1, 5, 13–18, 22, 28, 35, 38] in the past, studies of the role of generative AI tools in the field are relatively new.

Some recent work studied the role of ChatGPT in software engineering tasks such as bug fixing [10, 11, 42, 45], programming [4, 36], while others focused on multiple tasks [3, 9]. Besides, some studies assessed the efficiency of ChatGPT in providing assistance [20, 30], while some researchers conducted comparative studies of ChatGPT with other similar systems in assisting software development tasks [2, 23, 43].

Fraiwan et. al. [9] reported that ChatGPT could help developers in generating code, debugging, and software testing. However, it performed poorly in detecting code vulnerabilities. Sridhara et al. [43] explored how ChatGPT can be used to help with several software engineering tasks. They found that even though ChatGPT performed very well on tasks such as code summarization and code clone detection, it performed poorly in vulnerability detection. Their study is based on only 10 samples for each task. Our study is substantially larger. In our study, we used 2,865 ChatGPT conversations where each task category includes way more than 10 samples (except for software development management and optimization tasks).

Biswas et. al. [4] reported ChatGPT as a powerful tool for programming tasks such as code completion, code correction, code suggestion, automatic syntax error fixing, code optimization, refactoring, and chatbot development. The feasibility and efficiency of ChatGPT in debugging assistance, bug prediction, and bug explanation to assist in solving programming bugs were studied by Surameery et. al. [45]. Sobania et. al. [42] showed that ChatGPT can resolve software bugs to a similar extent as sophisticated deep learning systems.

Unlike the aforementioned studies, we have distinguished the tasks in which the developers seek assistance of ChatGPT the most and the least; in which tasks developer-ChatGPT conversations are efficient and where inefficient. Thus, we identify how developers can effectively leverage ChatGPT and where ChatGPT needs improvement to better assist in software engineering tasks.

6 CONCLUSION

This paper presents a quantitative study of the software development tasks where developers seek assistance from ChatGPT and how efficiently such tasks are completed through the developer-ChatGPT conversations/collaborations. Among the 12 categories of tasks studied in our work, code quality management and commit issue resolution are the two types of tasks in which the developers seek the most assistance from ChatGPT, especially in dealing with Python code. Across all types of tasks, assistance with shell scripting is sought the most from ChatGPT.

We find that developer-ChatGPT conversations/collaborations are the least efficient in dealing with tasks related to development environment setup, documentation generations, and code quality management. Thus, AI tools such as ChatGPT need to improve in these areas. The conversations/collaborations are found the most efficient in completing tasks associated with software development management and optimization as well as new feature implementation. Hence, developers can leverage AI tools such as ChatGPT in dealing with such tasks.

The findings are derived from thorough quantitative analyses of 2,865 distinct developer-ChatGPT conversations in the DevGPT dataset's 'snapshot_20230914' including 12,031 code snippets written in diverse languages. In the future, we will expand this work with qualitative analyses to capture deeper insights into the limitations and potential of AI tools in aiding software engineering tasks.

ACKNOWLEDGEMENT

This work is supported in part by the ISU-CAES Seed Grant at the Idaho State University, USA.

REFERENCES

[1] D. Alwad, M. Panta, and M. Zibran. 2018. An Empirical Study of the Relationships between Code Readability and Software Complexity. In *27th International Conference on Software Engineering and Data Engineering*. 122–127.

[2] Elissa Arias Sosa and Marco Godow. 2023. Comparing Google and ChatGPT as Assistive Tools for Students in Solving Programming Exercises.

[3] Adna Beganovic, Muna Abu Jaber, and Ali Abd Almisreb. 2023. Methods and Applications of ChatGPT in Software Development: A Literature Review. *Southeast Europe Journal of Soft Computing* 12, 1 (2023), 08–12.

[4] Som Biswas. 2023. Role of ChatGPT in Computer Programming.: ChatGPT in Computer Programming. *Mesopotamian Journal of Computer Science* 2023 (2023), 8–16.

[5] A. Champa, M. Rabbi, M. Zibran, and M. Islam. 2023. Insights into Female Contributions in Open-Source Projects. In *20th IEEE International Conference on Mining Software Repositories*. 357–361.

[6] Arifa Islam Champa, SM Mahedy Hasan, Md Atikur Rahman, and Md Fazle Rabbi. 2020. Hybrid technique for classification of hyperspectral image using quadratic mutual information. In *2020 IEEE Region 10 Symposium (TENSYMP)*. 933–936.

[7] Juan Dempere, Kennedy Prince Modugu, Allam Hesham, and Lakshmana Ramasamy. 2023. The impact of ChatGPT on higher education. *Dempere J, Modugu K, Hesham A and Ramasamy LK (2023) The impact of ChatGPT on higher education. Front. Educ* 8 (2023), 1206936.

[8] Yunhe Feng, Sreecharan Vanam, Manasa Cherukupally, Weijian Zheng, Meikang Qiu, and Haihua Chen. 2023. Investigating Code Generation Performance of ChatGPT with Crowdsourcing Social Data. In *Proceedings of the 47th IEEE Computer Software and Applications Conference*. 1–10.

[9] Mohammad Fraiwan and Natheer Khasawneh. 2023. A Review of ChatGPT Applications in Education, Marketing, Software Engineering, and Healthcare: Benefits, Drawbacks, and Research Directions. *arXiv preprint arXiv:2305.00237* (2023).

[10] Haotong Ge and Yuemeng Wu. 2023. An Empirical Study of Adoption of ChatGPT for Bug Fixing among Professional Developers. *Innovation & Technology Advances* 1, 1 (2023), 21–29.

[11] Md Asraful Haque and Shuai Li. 2023. The Potential Use of ChatGPT for Debugging and Bug Fixing. *EAI Endorsed Transactions on AI and Robotics* 2, 1 (2023), e4–e4.

[12] Hugging Face. 2023. facebook/bart-large-mnli. https://huggingface.co/facebook/bart-large-mnli. (Verified: Oct 2023).

[13] M. Islam and M. Zibran. 2016. A Comparative Study on Vulnerabilities in Categories of Clones and Non-Cloned Code. In *10th IEEE Intl. Workshop on Software Clones*. 8–14.

[14] M. Islam and M. Zibran. 2016. Exploration and Exploitation of Developers' Sentimental Variations in Software Engineering. *Internation Journal of Software Innovation* 4, 4 (2016), 35–55.

[15] M. Islam and M. Zibran. 2018. On the Characteristics of Buggy Code Clones: A Code Quality Perspective. In *12th IEEE Intl. Workshop on Software Clones*. 23 – 29.

[16] M. Islam and M. Zibran. 2020. How Bugs Are Fixed: Exposing Bug-fix Patterns with Edits and Nesting Levels. In *35th ACM/SIGAPP Symposium on Applied Computing*. 1523–1531.

[17] M. Islam and M. Zibran. 2021. What Changes in Where? An Empirical Study of Bug-Fixing Change Patterns. *ACM Applied Computing Review* 20, 4 (2021), 18–34.

[18] M. Islam, M. Zibran, and A. Nagpal. 2017. Security Vulnerabilities in Categories of Clones and Non-Cloned Code: An Empirical Study. In *11th ACM/IEEE Intl. Symposium on Empirical Software Engineering and Measurement*. 20–29.

[19] Md Rakibul Islam and Minhaz F Zibran. 2016. Exploration and exploitation of developers' sentimental variations in software engineering. *International Journal of Software Innovation (IJSI)* 4, 4 (2016), 35–55.

[20] Sajed Jalil, Suzzana Rafi, Thomas D LaToza, Kevin Moran, and Wing Lam. 2023. Chatgpt and software testing education: Promises & perils. In *2023 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. 4130–4137.

[21] Srecko Joksimovic, Dirk Ifenthaler, Rebecca Marrone, Maarten De Laat, and George Siemens. 2023. Opportunities of artificial intelligence for supporting complex problem-solving: Findings from a scoping review. *Computers and Education: Artificial Intelligence* (2023), 100138.

[22] R. Joseph, M. Zibran, and F. Eishita. 2021. Choosing the Weapon: A Comparative Study of Security Analyzers for Android Applications. In *Intl. Conference on Software Engineering, Management and Applications*. 51–57.

[23] Samia Kabir, David N Udo-Imeh, Bonan Kou, and Tianyi Zhang. 2023. Who Answers It Better? An In-Depth Analysis of ChatGPT and Stack Overflow Answers to Software Engineering Questions. *arXiv preprint arXiv:2308.02312* (2023).

[24] Muhammad Fawad Akbar Khan, Max Ramsdell, Erik Falor, and Hamid Karimi. 2023. Assessing the Promise and Pitfalls of ChatGPT for Automated Code Generation. *arXiv preprint arXiv:2311.02640* (2023).

[25] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461* (2019).

[26] Yiheng Liu, Tianle Han, Siyuan Ma, Jiayue Zhang, Yuanyuan Yang, Jiaming Tian, Hao He, Antong Li, Mengshen He, Zhengliang Liu, et al. 2023. Summary of chatgpt-related research and perspective towards the future of large language models. *Meta-Radiology* (2023), 100017.

[27] Wei Ma, Shangqing Liu, Wenhan Wang, Qiang Hu, Ye Liu, Cen Zhang, Liming Nie, and Yang Liu. 2023. The Scope of ChatGPT in Software Engineering: A Thorough Investigation. *arXiv preprint arXiv:2305.12138* (2023).

[28] D. Murphy, M. Zibran, and F. Eishita. 2021. Plugins to Detect Vulnerable Plugins: An Empirical Assessment of the Security Scanner Plugins for WordPress. In *Intl. Conference on Software Engineering, Management and Applications*. 39–44.

[29] Nathalia Nascimento, Paulo Alencar, and Donald Cowan. 2023. Comparing Software Developers with ChatGPT: An Empirical Investigation. *arXiv preprint arXiv:2305.11837* (2023).

[30] Nascimento Nathalia, Alencar Paulo, and Cowan Donald. 2023. Artificial Intelligence vs. Software Engineers: An Empirical Study on Performance and Efficiency using ChatGPT. In *Proceedings of the 33rd Annual International Conference on Computer Science and Software Engineering*. 24–33.

[31] OpenAI. 2023. *ChatGPT*. Retrieved October 2023 from https://openai.com/chatgpt/

[32] PyPI. 2023. *language-tool-python 2.7.1 Project description*. Retrieved October 2023 from https://pypi.org/project/language-tool-python/

[33] PyPI. 2023. *spacy 3.7.2 Project description*. Retrieved October 2023 from https://pypi.org/project/spacy/

[34] PyPI. 2023. *textstat 0.7.3 Project description*. Retrieved October 2023 from https://pypi.org/project/textstat/

[35] Md Fazle Rabbi, Arifa I. Champa, Costain Nachuma, and Minhaz F. Zibran. 2024. SBOM Generation Tools Under Microscope: A Focus on the npm Ecosystem. In *Proceedings of ACM Symposium on Applied Computing (SAC 2024)*.

[36] Md Fazle Rabbi, Arifa I. Champa, Minhaz F. Zibran, and Md Rakibul Islam. 2024. AI Writes, We Analyze: The ChatGPT Python Code Saga. In *Proceedings of ACM International Conference on Mining Software Repositories (MSR 2024)*.

[37] Wahyu Rahmaniar. 2023. Chatgpt for software development: Opportunities and challenges. (2023).

[38] A. Rajbhandari, M. Zibran, and F. Eishita. 2022. Security Versus Performance Bugs: How Bugs are Handled in the Chromium Project. In *Intl. Conference on Software Engineering, Management and Applications*. 70–76.

[39] Thomas Ramge and Viktor Mayer-Schonberge. 2023. *Using ChatGPT to Make Better Decisions*. Retrieved November 27, 2023 from https://hbr.org/2023/08/using-chatgpt-to-make-better-decisions/

[40] Tanya Serry, Tonya Stebbins, Andrew Martchenko, Natalie Araujo, and Brigid McCarthy. 2023. Improving access to COVID-19 information by ensuring the readability of government websites. *Health Promotion Journal of Australia* 34, 2 (2023), 595–602.

[41] Inbal Shani and GitHub Staff. 2023. *Survey reveals AI's impact on the developer experience*. Retrieved October 29, 2023 from https://github.blog/2023-06-13-survey-reveals-ais-impact-on-the-developer-experience/

[42] Dominik Sobania, Martin Briesch, Carol Hanna, and Justyna Petke. 2023. An analysis of the automatic bug fixing performance of chatgpt. *arXiv preprint arXiv:2301.08653* (2023).

[43] Giriprasad Sridhara, Sourav Mazumdar, et al. 2023. ChatGPT: A Study on its Utility for Ubiquitous Software Engineering Tasks. *arXiv preprint arXiv:2305.16837* (2023).

[44] Bhargav Srinivasa-Desikan. 2018. *Natural Language Processing and Computational Linguistics: A practical guide to text analysis with Python, Gensim, spaCy, and Keras*. Packt Publishing Ltd.

[45] Nigar M Shafiq Surameery and Mohammed Y Shakor. 2023. Use chat gpt to solve programming bugs. *International Journal of Information Technology & Computer Engineering (IJITC) ISSN: 2455-5290* 3, 01 (2023), 17–22.

[46] Gábor J Székely and Maria L Rizzo. 2023. *The energy of data and distance correlation*. CRC Press.

[47] Muhammad Waseem, Teerath Das, Aakash Ahmad, Mahdi Fehmideh, Peng Liang, and Tommi Mikkonen. 2023. Using ChatGPT throughout the Software Development Life Cycle by Novice Developers. *arXiv preprint arXiv:2310.13648* (2023).

[48] Tao Xiao, Christoph Treude, Hideaki Hata, and Kenichi Matsumoto. 2024. DevGPT: Studying Developer-ChatGPT Conversations. In *Proceedings of the International Conference on Mining Software Repositories (MSR 2024)*.

[49] Gokul Yenduri, Gautam Srivastava, Praveen Kumar Reddy Maddikunta, Rutvij H Jhaveri, Weizheng Wang, Athanasios V Vasilakos, Thippa Reddy Gadekallu, et al. 2023. Generative Pre-trained Transformer: A Comprehensive Review on Enabling Technologies, Potential Applications, Emerging Challenges, and Future Directions. *arXiv preprint arXiv:2305.10435* (2023).