2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC) | 979-8-3503-7696-8/24/$31.00 ©2024 IEEE | DOI: 10.1109/COMPSAC61105.2024.00096

Systematic Literature Review of Prompt Engineering Patterns in Software Engineering

Yuya Sasaki
Waseda University, Japan
sasaki.y@ruri.waseda.jp

Hironori Washizaki
Waseda University, Japan
washizaki@waseda.jp

Jialong Li
Waseda University, Japan
lijialong@fuji.waseda.jp

Dominik Sander
University of Hamburg, Germany
dominik.m.sander@t-online.de

Nobukazu Yoshioka
Waseda University, Japan
nobukazu@engineerable.ai

Yoshiaki Fukazawa
Waseda University, Japan
fukazawa@waseda.jp

***Abstract*—Advancements in large language models (LLMs) are transforming software engineering through innovative prompt engineering strategies. By analyzing prompt-driven enhancements across key software engineering tasks, we present a systematic literature review and a pioneering taxonomy elucidating the practical applications of prompt engineering in software engineering. Our taxonomy offers a foundational framework that clarifies the roles of prompt engineering and measures its impact, thereby guiding evolving AI-driven software engineering research and practices.**

***Index Terms*—prompt engineering, software engineering, systematic literature review, large language model, LLM application**

I. Introduction

Artificial intelligence (AI) has revolutionized numerous domains, creating new possibilities and efficiencies [1]. In software engineering, the AI field has burgeoned with approaches integrating large language models (LLMs) such as OpenAI's Generative Pre-trained Transformer (GPT) and Codex, which have demonstrated notable capabilities in code generation, natural language understanding and developer assistance tools [2]. However, despite the burgeoning application of these approaches, the domain of prompt engineering with LLMs for software engineering remains fragmented and lacks a comprehensive understanding [3]. Prompt engineering—the practice of tailoring inputs to direct the behavior of LLMs toward specific objectives—is essential for optimizing their practical utility in software engineering activities [4].

Although the potential of LLMs in software engineering is being explored [5], systematic assessments of how to employ prompt engineering effectively across diverse software engineering contexts are scant in the literature. Prior reviews have illuminated aspects of LLMs' abilities [6]; yet a compelling need remains to consolidate the knowledge around prompt strategies and patterns and their quantifiable effects on software engineering activities [7].

'Prompt Engineering Pattern' is a broader framework beyond prompt pattern and encompasses higher level adjustments from tuning model itself or its parameters for specific tasks or requirements to crafting prompts strategically. This holistic approach includes:

- **Prompt Design**: Developing reusable prompt patterns that help users effectively explain their tasks to the LLM.
- **Model Tuning**: Fine-tuning the model to better suit specific applications, enhancing its understanding and output relevancy.
- **Parameter Adjustment**: Modifying runtime settings such as temperature or top-p to tailor the model's generative behaviors.

This comprehensive approach enables a more nuanced interaction with LLMs, allowing for more precise and effective solutions tailored to complex problems. **Prompt Engineering Patterns** aim to systematize these interactions, providing a broader framework that can be applied across various domains, from software engineering to content creation. By introducing and defining these patterns within the context of software engineering, we aim to pave the way for more systematic, reliable, and ethically sound applications of LLMs in the industry.

We conducted a systematic literature review and answered the following research questions to elucidate the nature, roles, and implications of prompt engineering in software engineering, thus moving beyond superficial applications to a more strategic and informed utilization.

We thus answer the following research questions:

RQ1. What prompt engineering patterns are utilized to enhance the functionalities of large language models in software engineering tasks?

RQ2. What are attributes of prompt engineering patterns?

RQ3. What are the possible research directions?

We are examining these questions to attain a clear picture of how prompt engineering is used in creating and maintaining software. Our objective is to organize our findings into a taxonomy that helps both researchers and professionals using LLMs in their software work.

II. Related Work

In the present study, we build on the insights from several pivotal studies in the realm of LLMs and their application in software engineering.

2836-3795/24/$31.00 ©2024 IEEE
DOI 10.1109/COMPSAC61105.2024.00096

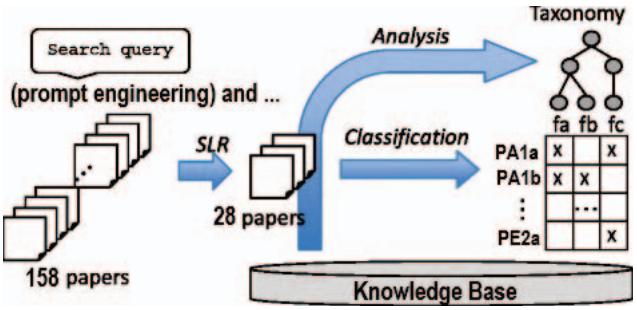

Fig. 1. Literature review process

In the context of LLMs and their integration into software engineering, several literature reviews provide foundational insights. Hou et al. (2023) [6] offers a comprehensive analysis of LLMs in software engineering, examining strategies for optimizing their performance across various software engineering tasks. Zhao et al. (2023) [5] delve into the evolution of LLMs, highlighting the challenges in their development and application. Fan et al. (2023) [8] present a bibliometric review of over 5,000 LLM publications, revealing trends and research paradigms across fields. Although these studies contribute to understanding LLMs' capabilities and applications, they do not specifically address the nuanced domain of prompt engineering in software engineering.

The present study fills this gap with a focused exploration of prompt engineering patterns in software engineering, building upon the broad perspectives provided by previous research. By offering a pioneering taxonomy of prompt engineering patterns, this paper makes a significant contribution to the field, systematically evaluating these patterns in software engineering and guiding future AI-driven research and practices in software development.

III. Systematic Literature Review

A. Queries and Paper Selection

We used systematic literature review (SLR) approach, focusing on unearthing, evaluating, and assimilating prompt engineering patterns relevant to software engineering tasks involving LLMs.

Eligibility Criteria: To align with the objectives of RQ1, we established criteria to include publications that provide insights into the use of prompt engineering to augment LLM functionality in software engineering tasks. Specifically, studies were required to discuss the application of prompts in LLMs within a software engineering context and to include evaluations of the prompts used.

Information Sources: For our literature review, we first used IEEE Xplore to source scholarly articles; however, no paper introducing prompt engineering patterns related to software engineering was available at the time. We, therefore moved on to Google Scholar. This choice was driven by its extensive coverage of repositories such as arXiv, ACM Digital Library, and IEEE Xplore. Most papers included in our review are gray papers (i.e., papers not yet subjected to peer review) because very few peer-reviewed papers were available at the time of our review.

Search Strategy: Our literature search was guided by a strategic combination of keywords reflecting the intention of RQ1:

```
("Prompt Engineering") AND ("Software
    Engineering" OR "Software Development")
```

This search yielded 149 studies from Google Scholar. In addition to the search results, we performed snowballing manually and added nine papers closely related to the topic, bringing the total to 158.

Study Selection: In the first step, titles and abstracts were reviewed to filter papers, narrowing the selection to 37 that met our eligibility criteria. In the second step, we closely examined these 37 papers, paying particular attention to the problems or challenges discussed and the prompt engineering patterns provided as the solution. This process resulted in further refinement, leaving 28 studies that explicitly employed prompt engineering to address software engineering challenges.

Data Extraction: We meticulously documented key findings from the 28 selected studies. This process included identifying 21 prompt engineering patterns within five categories, their applications in software engineering tasks, and their measured or observed efficacy. This data formed the backbone of our developed taxonomy.

Data Synthesis: Our synthesis efforts drew from the extracted data to construct a taxonomy that categorizes the prompt engineering patterns. This taxonomy, aiming to clarify the diverse strategies in the literature, addresses RQ1. In the process, we also identified the publication date, LLMs discussed, and target software engineering areas to answer RQ2.

Quality Assessment: Given the emergent nature of prompt engineering within software engineering, we focused on the relevancy, innovation, and methodological depth of studies rather than on formalized quality assessment tools. We ensured that each included study provided actionable insights related to our research questions, thereby enhancing the quality and utility of the developed taxonomy.

Risk of Bias in Individual Studies: While not formally structured around bias risk, our selection criteria preferred studies with transparent methodologies, ensuring replicable prompt engineering applications in software engineering.

Risk of Bias Across Studies: Our inclusive approach entailed sourcing material from both peer-reviewed and gray literature, the latter being especially important for capturing the latest yet-to-be-peer-reviewed findings in the rapidly evolving field of prompt engineering. This strategy enabled us to present a taxonomy that is both comprehensive and attuned to cutting-edge research, albeit with the understanding that such literature may vary in methodological rigor. To counteract potential biases, we critically evaluated the credibility of each study, ensuring a balanced representation of the state-of-the-

art in prompt engineering within software engineering.

B. Taxonomy of Prompt Engineering Patterns

To systematically categorize the diverse prompt engineering patterns, we developed a taxonomy as shown in Table I, addressing RQ1. The taxonomy encapsulates Learning-Driven Approaches, Interaction-Focused Methods, Task-Specific Patterns, and Systematization and Cataloging, each with distinct subcategories.

Overview of Taxonomy: Our inclusive approach entailed sourcing material from both peer-reviewed and gray literature, the latter being especially important for capturing the latest yet-to-be-peer-reviewed findings in the rapidly evolving field of prompt engineering (RQ3). This enabled us to present a taxonomy that is both comprehensive and attuned to cutting-edge research, albeit with the understanding that such literature may vary in methodological rigor. To counteract potential biases, we critically evaluated the credibility of each study, ensuring a balanced representation of the state-of-the-art in prompt engineering within software engineering.

The taxonomy is rooted in the emergent need for structure amidst the expanse of prompt engineering applications. Broadly categorized into four clusters, the taxonomy encompasses Learning-Driven Approaches, Interaction-Focused Methods, Task-Specific Patterns, and Systematization and Cataloging.

Each cluster delineates a distinct methodological approach toward LLMs in software engineering and further branches into finer subcategories, reflecting the spectrum of approaches unearthed during the systematic literature review. For instance, Learning-Driven Approaches include Few-Shot and Zero-Shot learning applications, with Differentiated Few-Shot modes tailored for specific software engineering tasks such as Code Generation (PA1a) and Code Interpretation (PA1b). This categorization demonstrates the multifaceted uses of prompt-based learning strategies in addressing the challenges stated in RQ2.

C. Main Categories and Patterns Description

The taxonomy is further elucidated through illustrative examples grounded in the literature. Each prompt pattern identified is anchored to specific software engineering activities. We present a comprehensive mapping of these patterns, referencing the fields they predominantly impact, thereby showcasing the concrete applications discussed in RQ2.

For example, in the Interaction-Focused Methods group, patterns such as User-Model Collaboration Refinement (PB1a), Mutual Prompt Refinement (PB1b), and Gradual Execution and Output (PB2a) emphasize the value of Interactive Prompting and Iterative Refinement within many contexts. The Task-Specific Patterns cluster elucidates techniques such as Leveraging Metadata and Docs (PC2a) and Prompt Wording Optimization (PC2b), which leverage existing documentation to enhance code generation tasks.

IV. Results

Through our systematic literature review, we analyzed 158 studies and selected 28 that applied and evaluated prompt engineering in the field of software engineering. These studies represent the growing academic interest and application of AI-augmented software development practices, with discussions aligning with the answers to our research questions (RQ1, RQ2, and RQ3).

A. *RQ1* What prompt engineering patterns are utilized to enhance the functionalities of large language models in software engineering tasks?

By carefully reading all 28 of the selected papers, we identified 21 prompt engineering patterns and classified them into the following five major categories (PA through PE), varying the general output format alignment to task-specific prompt engineering patterns.

- PA. Learning Approach: Seven papers address the pattern to provide the context necessary for their purpose to LLMs [9]- [15]. The few-shot approach was introduced in many papers, while some patterns were for the zero-shot approach.
- PB. Interaction-Focused Methods: Six papers discuss methods to enrich the output [20], [16]- [21]. These methods range from relatively simple requests to an LLM on the method to answer, to more extensive task decomposition through mutual communication.
- PC. Task Specific Patterns: Eight papers discuss the specific patterns to tasks around software engineering [22]- [29]. There were many patterns for code generation or specification or comment generation from code.
- PD. Model Optimization: Two papers introduced prompt engineering patterns around model tuning [31], [32]. In this category, they introduce prompts to tune the model parameters to specified tasks to bring out the potential of the model.
- PE. Systematization and Cataloging: Three papers involved cataloging the prompting patterns [34], [30], [33]. They provide their set of prompt engineering patterns for software engineering tasks.

In practical terms, the identified prompt engineering patterns significantly contribute to refining software development methodologies by enabling more precise, efficient, and creative use of LLMs, thereby promising marked improvements in the pace and quality of software engineering tasks.

Answer RQ1: **Prompt engineering patterns are introduced in many papers to enhance the functionalities of LLMs across diverse software engineering activities.**

B. *RQ2: What are attributes of prompt engineering patterns?*

- **Number of Papers Published per Quarter:** The studies under consideration utilize various empirical research methods and have been conducted within the past 3 years. The publication trend shows increasing interest in this field over time. Starting from two papers in 2021-Q4, significant growth to eight papers in 2023-Q1 is observed. The trend suggests an evolving interest and approach toward more dynamic and collaborative AI-informed software development processes.
- **LLMs Used in Papers:** Various LLMs have been explored in the studies, with Codex being the most frequently discussed (18 papers), followed by GPT-3.5 (6 papers) and GPT-3 (3

TABLE I
EXTRACTED PROMPT ENGINEERING PATTERNS

ID	Category	Subcatgory	Pattern Name	Problem	Solution
PA1a	Learning Approaches	Few-Shot (In-context) Learning Applications	Iterative Example-Based Prompting for Code Generation [9] [10]	Improving LLMs' code generation efficiency for various tasks without extensive customization or fine-tuning.	Employing few-shot learning in LLMs with strategic examples or prompts for efficient, tailored code generation.
PA1b			Contextual Clue Integration for Code Interpretation [11]	Efficient retrieval of software engineering factual knowledge, specifically FQNs, from a code model.	Designing in-context learning prompts for accurate FQN inference and effective code analysis.
PA1c			Example-Guided Prompting for Conflict Resolution [12]	Time-consuming manual resolution of textual and semantic merge conflicts in software development.	Introduces merge tool using k-shot learning with GPT-3 for automated merge conflict resolution.
PA1d			Code Sample Retrieval [13]	Selecting effective prompts for code-related few-shot learning with LLMs.	Introduces the automated retrieval technique for selecting relevant code demonstrations for prompts.
PA2a		Zero-shot Applications	Zero Shot APR and vulnerability repair [14] [15]	Utilizing LLMs for programming error explanations or vulnerability patching in zero-shot settings.	Strategic zero-shot learning prompts for LLMs to clarify error messages and generate secure software patches.
PB1a	Interaction-Focused Methods	Interactive Prompting and Iterative Refinement	User-Model Collaboration Refinement [16] [17] [18]	Enhancing interactive dialogue and AI-assisted coding through improved user-model collaboration.	Implement interactive and iterative prompt designs for conversational assistants, object model synthesis, or educational coding.
PB1b			Mutual Prompt Refinement [19]	Aligning end-user programming queries with LLM capabilities for accurate code generation.	Using abstraction matching to translate user queries into clear, system-compatible prompts.
PB2a		Chain and Modular Output Design	Gradual Execution and Output [20]	Enhancing LLMs' performance on complex tasks by improving user control and fostering better collaboration.	Decomposing tasks using chained LLM steps for enhanced quality, transparency, and user satisfaction.
PB2b			Problem Decomposition and Integration [21]	Need for a comprehensive method covering large sets of software requirements.	Employing prompt templates for multi-class requirement classification.
PC1a	Task Specific Patterns	APR and Bug Fixing	Sequential Prompting for Bug Identification and Resolution [22] [23]	Addressing the limitations of traditional APR tools in generating diverse and complex bug fixes using LLM.	Employ strategic prompt design with LLM for diverse and effective software bug fixing, surpassing traditional APR methods.
PC1b			Security-Focused Prompt Refinement for Hardware Debugging [24]	Repairing hardware security bugs in Verilog (a HDL) designs using LLMs.	Framework using LLMs to generate functionally correct and secure HW code repairs.
PC2a		Code Synthesis and Quality Enhance	Leveraging Metadata and Docs [25]	Improving LLM code summarization by integrating file metadata and design documents for enhanced context and precision.	Utilizing repository context, tagged identifiers, and DFG in prompts to enhance LLM code summary precision and specificity.
PC2b			Prompt Wording Optimization [26]	Enhancing LLM output quality in code summarization through optimized prompt wording.	Comparing the output by different prompt wording, like adjusting adverbs, for better quality.
PC2c			Code Quality Enhancement through Detailed Prompts [27]	Exploring the impact of prompt detail level on the syntactic validity and functional correctness of AI-generated code.	Systematic evaluation and optimization of prompt details to enhance the quality and practicality of AI-generated code.
PC2d			Code Comment Integration [28]	Reducing the reproduction of known bugs in LLM-generated code completions.	Employ strategic comments in prompts to guide LLMs for bug-free code generation.
PC3a		Formal Specification Generation	Specification with Natural Language [29]	Automating natural language translation into formal specifications like OCL to enhance software modeling.	Using Codex with structured prompts for automatic OCL generation from natural language, improving model accuracy.
PC4a		Evaluation on Code Generation Security of LLMs	Vulnerability-Prone Code Descriptions [30]	Need for evaluation dataset for LLMs on how much of their generated code contains security flaws.	Provides the list of NL prompts based on CWEs to benchmark and improve the security of LLM-generated code.
PD1a	Model Optimization	Comparison to Fine-Tuning	Prompt Tuning for Code Intelligence Adaptation [31]	Adapting LLMs to specific code intelligence tasks with minimal data and avoiding extensive fine-tuning inconsistencies.	Using prompt tuning to adapt models efficiently to code intelligence tasks with limited data.
PD2a		Model Parameter Tuning	Input Parameter Impact Exploration [32]	Assessing how input parameters influence AI-generated code quality.	Systematic experiments to optimize AI code generation varying input parameters.
PE1a	Systematization and Cataloging	Prompt Engineering Frameworks	Domain-General Pattern Catalogs [33]	Need for structured prompt engineering for effective LLM interactions across various domains.	Creating domain-general catalogs to systematize prompt engineering for LLMs.
PE1b			SW Engineering Specific Pattern Catalogs [34]	Maximizing LLMs' utility in software engineering tasks.	Designing prompt engineering patterns tailored for software engineering applications.

papers). Notably, Codex is derived from the GPT-3 architecture and has been fine-tuned specifically for tasks related to coding [2], underscoring the preeminence of the GPT-3 lineage in the field. Other models such as CodeGen, CodeT5, and PolyCoder are also referenced, indicating a broad spectrum of LLMs have been deployed to tackle challenges in software engineering.

• **Software Development Knowledge Area in Papers:** Some papers focus on a specific knowledge area defined in SWEBOK [35]; however, most papers introduced a more general pattern applicable to multiple knowledge areas within software development. The most discussed topic is Software Construction, followed by Testing, Maintenance, and Design.

• **Between Development Knowledge Area and LLMs**: Codex is predominantly used in Construction and Testing,

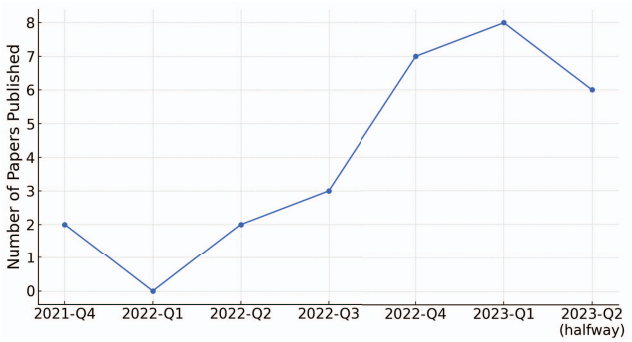

Fig. 2. Number of papers published per quarter

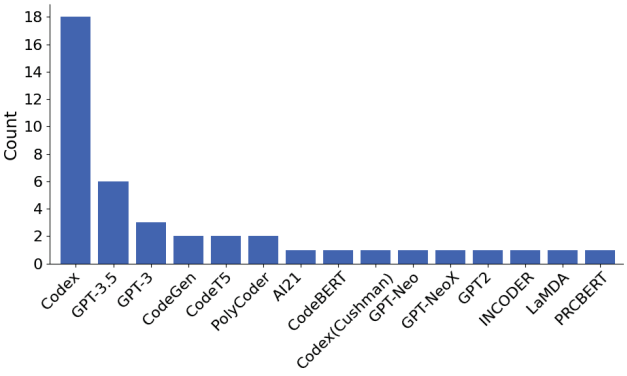

Fig. 3. LLMs discussed in papers

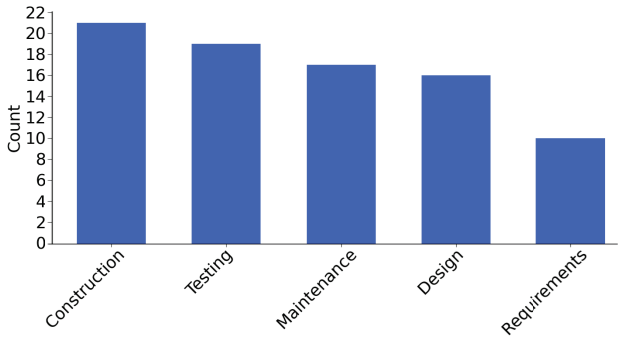

Fig. 4. Software development knowledge areas in papers

LLM	Software development knowledge area				
	Requirements	Design	Construction	Testing	Maintenance
Codex	4	8	12	14	10
GPT-3.5	3	5	7	3	5
GPT-3	1	3	2	3	3
CodeGen	1	1	2	2	1
PolyCoder	0	1	2	1	2
LaMDA	1	1	1	1	1
PRCBERT	1	1	1	1	1
Codex(Cushman)	0	1	2	1	1
GPT-2	0	1	1	0	1
AI21	0	1	1	0	1
GPT-J	0	0	0	1	1
GPT-NeoX	0	0	0	1	1
CodeT5	0	0	0	1	1
GPT-Neo	0	0	0	1	1
INCODER	0	0	0	1	1

Fig. 5. Number of papers per combination of software development knowledge area and LLM

underscoring its targeted efficacy in code-centric tasks. Conversely, GPT-3 variants (GPT-3.5 and GPT-3) are utilized less frequently and applied across a diverse array of software development phases. This diversity highlights their broad applicability in tackling a wide range of software engineering challenges beyond mere code generation.

Answer RQ2: **The attributes of prompt engineering patterns in software engineering are characterized by growing research interest, notable focus on Codex and GPT models, and diverse applications across major software development phases.**

C. *RQ3 What are the possible research directions?*

• **Exploration in Underexplored Domains:** Some knowledge areas, such as Software Requirements, had relatively limited prompt pattern investigation. New patterns are more likely to be found in these areas.

• **Formal Analysis on Prompt:** Some studies included a discussion of the effectiveness of their prompt pattern; however, how much room remains for improvement by changing the expressions or styles of prompts is unclear. It would be interesting if we could measure it.

• **Exploring Beyond Textual Prompts in Multimodal LLMs:** The advent of multimodal LLMs, such as GPT-4 and LLaMA, introduces the potential for leveraging nontextual inputs such as diagrams or photos in prompt engineering.

Answer RQ3: **Future research should focus on expanding prompt engineering into underexplored areas such as Software Requirements, examining the effectiveness of specialized versus general-purpose LLMs, and exploring innovative nontextual prompts in multimodal LLM environments. Research in this area could revolutionize how engineers interact with LLMs, making the process more intuitive and aligned with the multifaceted nature of software development tasks.**

V. Discussion

The publication trend highlights the rapid integration of AI in software engineering, underscoring the fast-paced evolution of this field. Our taxonomy offers a structured approach for leveraging LLMs, but it is crucial to note the temporal limitations inherent in such a dynamic domain. The field's swift advancements might outpace some of the findings presented here, necessitating ongoing research to ensure the continued relevance and applicability of these patterns.

One practical example of our taxonomy in action is evident in the use of "Gradual Execution and Output" (Pattern PB2a) for incrementally refining AI-generated code. This approach substantially reduces debugging time and improves code reliability, showcasing the immediate benefits of our findings in real-world scenarios.

However, the rapid evolution of AI technologies represents a limitation to our study. Future research should examine the applicability of these patterns in new AI models, and explore their effectiveness across the entire software development lifecycle, by digging deeper into individual tasks.

VI. Conclusion

In this study, we introduced a taxonomy of prompt engineering patterns in software engineering, derived from a systematic literature review. The key contributions are:

• **Identification of Patterns:** We identified and classified prompt engineering patterns, showcasing their application across different software engineering activities. This categorization provides a structured approach to harness LLMs for enhanced efficiency and innovation in software development.

• **Insights on Papers and Patterns:** The analysis of publication trends, LLMs utilized, and their target areas reveals a growing interest in prompt engineering and its potential to address key software engineering challenges. This enriches academic and practical discourse on the subject. • **Future Directions:** The study suggests potential research areas, including exploring underrepresented domains within software engineering and examining the effectiveness of prompt engineering across different LLMs.

Prompt engineering is a transformative approach in software engineering, facilitating more effective integration of AI technologies. Our taxonomy lays the groundwork for future advancements, advocating for the strategic application of LLMs to improve software development processes. As the discipline progresses, continuous research will be critical to refine and broaden the role of prompt engineering in AI-enabled software engineering.

VII. Acknowledgement

This work was supported by JSPS Bilateral Program JPJSBP120209936, JSPS KAKENHI 21KK0179, and JST-Mirai Program Grant Number JPMJMI20B8.

References

[1] T. B. e. a. Brown, "Language Models are Few-Shot Learners," July 2020.
[2] M. e. a. Chen, "Evaluating Large Language Models Trained on Code," July 2021.
[3] A. e. a. Srivastava, "Beyond the Imitation Game: Quantifying and extrapolating the capabilities of language models," June 2023.
[4] Y. e. a. Zhou, "Large Language Models Are Human-Level Prompt Engineers," March 2023.
[5] W. X. e. a. Zhao, "A Survey of Large Language Models," September 2023.
[6] X. e. a. Hou, "Large Language Models for Software Engineering: A Systematic Literature Review," September 2023.
[7] Z. e. a. Guo, "Evaluating Large Language Models: A Comprehensive Survey," October 2023.
[8] L. Fan, L. Li, Z. Ma, S. Lee, H. Yu, and L. Hemphill, "A Bibliometric Review of Large Language Models Research from 2017 to 2023."
[9] T. Ahmed and P. Devanbu, "Few-shot training LLMs for project-specific code-summarization," September 2022.
[10] P. Bareiß, B. Souza, M. d'Amorim, and M. Pradel, "Code Generation Tools (Almost) for Free? A Study of Few-Shot, Pre-Trained Language Models on Code," June 2022.
[11] Q. e. a. Huang, "SE Factual Knowledge in Frozen Giant Code Model: A Study on FQN and its Retrieval," December 2022.
[12] J. Zhang, T. Mytkowicz, M. Kaufman, R. Piskac, and S. K. Lahiri, "Using pre-trained language models to resolve textual and semantic merge conflicts (experience paper)," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*. Virtual South Korea: ACM, July 2022, pp. 77–88.
[13] N. Nashid, M. Sintaha, and A. Mesbah, "Retrieval-Based Prompt Selection for Code-Related Few-Shot Learning," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. Melbourne, Australia: IEEE, May 2023, pp. 2450–2462.
[14] J. e. a. Leinonen, "Using Large Language Models to Enhance Programming Error Messages," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. Toronto ON Canada: ACM, March 2023, pp. 563–569.
[15] H. Pearce, B. Tan, B. Ahmad, R. Karri, and B. Dolan-Gavitt, "Examining Zero-Shot Vulnerability Repair with Large Language Models," August 2022.
[16] S. I. Ross, M. Muller, F. Martinez, S. Houde, and J. D. Weisz, "A Case Study in Engineering a Conversational Programming Assistant's Persona," January 2023.
[17] A. Gu, T. Mitrovska, D. Velez, J. Andreas, and A. Solar-Lezama, "ObSynth: An Interactive Synthesis System for Generating Object Models from Natural Language Specifications," October 2022.
[18] P. Denny, V. Kumar, and N. Giacaman, "Conversing with Copilot: Exploring Prompt Engineering for Solving CS1 Problems Using Natural Language," in *Proceedings of the 54th ACM Technical Symposium on Computer Science Education V. 1*. Toronto ON Canada: ACM, March 2023, pp. 1136–1142.
[19] M. X. e. a. Liu, ""What It Wants Me To Say": Bridging the Abstraction Gap Between End-User Programmers and Code-Generating Large Language Models," in *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. Hamburg Germany: ACM, April 2023, pp. 1–31.
[20] T. Wu, M. Terry, and C. J. Cai, "AI Chains: Transparent and Controllable Human-AI Interaction by Chaining Large Language Model Prompts," March 2022.
[21] X. Luo, Y. Xue, Z. Xing, and J. Sun, "PRCBERT: Prompt Learning for Requirement Classification using BERT-based Pretrained Language Models," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*. Rochester MI USA: ACM, October 2022, pp. 1–13.
[22] J. A. Prenner, H. Babii, and R. Robbes, "Can OpenAI's codex fix bugs?: an evaluation on QuixBugs," in *Proceedings of the Third International Workshop on Automated Program Repair*. Pittsburgh Pennsylvania: ACM, May 2022, pp. 69–75.
[23] C. S. Xia, Y. Wei, and L. Zhang, "Automated Program Repair in the Era of Large Pre-trained Language Models," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. Melbourne, Australia: IEEE, May 2023, pp. 1482–1494.
[24] B. Ahmad, S. Thakur, B. Tan, R. Karri, and H. Pearce, "Fixing Hardware Security Bugs with Large Language Models," February 2023.
[25] T. Ahmed, K. S. Pai, P. Devanbu, and E. T. Barr, "Improving Few-Shot Prompts with Relevant Static Analysis Products," August 2023.
[26] W. e. a. Sun, "Automatic Code Summarization via ChatGPT: How Far Are We?" May 2023.
[27] B. Yetistiren, I. Ozsoy, and E. Tuzun, "Assessing the quality of GitHub copilot's code generation," in *Proceedings of the 18th International Conference on Predictive Models and Data Analytics in Software Engineering*. Singapore Singapore: ACM, November 2022.
[28] K. Jesse, T. Ahmed, P. T. Devanbu, and E. Morgan, "Large Language Models and Simple, Stupid Bugs," March 2023.
[29] S. Abukhalaf, M. Hamdaqa, and F. Khomh, "On Codex Prompt Engineering for OCL Generation: An Empirical Study," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*. Melbourne, Australia: IEEE, May 2023, pp. 148–157.
[30] C. Tony, M. Mutas, N. E. D. Ferreyra, and R. Scandariato, "LLMSecEval: A Dataset of Natural Language Prompts for Security Evaluations," March 2023.
[31] C. Wang, Y. Yang, C. Gao, Y. Peng, H. Zhang, and M. R. Lyu, "No more fine-tuning? an experimental evaluation of prompt tuning in code intelligence," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. Singapore Singapore: ACM, Nov. 2022, pp. 382–394.
[32] J.-B. Döderlein, M. Acher, D. E. Khelladi, and B. Combemale, "Piloting Copilot and Codex: Hot Temperature, Cold Prompts, or Black Magic?" February 2023.
[33] J. e. a. White, "A Prompt Pattern Catalog to Enhance Prompt Engineering with ChatGPT," February 2023.
[34] J. White, S. Hays, Q. Fu, J. Spencer-Smith, and D. C. Schmidt, "ChatGPT Prompt Patterns for Improving Code Quality, Refactoring, Requirements Elicitation, and Software Design," March 2023.
[35] P. Bourque and R. E. Fairley, Eds., *SWEBOK: Guide to the Software Engineering Body of Knowledge*, version 3.0 ed. Los Alamitos, CA: IEEE Computer Society, 2014.