A Prompt-Based Approach for Software Development

[*]Submission Type: Poster Paper, Research Track: CSCI-RTSE

Mohammed Hamdi
Department of Computer Science and Engineering
Oakland University
Rochester, MI, USA
mohammedhamdi@oakland.edu

Lewy D. Kim
Stoney Creek High School
Rochester Hills, MI, USA
djel2.kim@gmail.com

***Abstract*—Generative Language-Based AI (Gen-LBAI) models have drawn significant interest in a wide range of fields including software engineering. There has been increasing interest in leveraging Gen-LBAI models in various areas of software engineering for efficient software development. Successful adoption of AI models can save time by generating various types of software artifacts, which lets the developer focus on more critical and creative tasks. In this paper, we present a prompt-based approach for software development using an AI model. The approach presents a set of guidelines for designing prompts and discusses where and how AI models can be used in software development for improved efficiency. The approach is evaluated in a case study using ChatGPT-4 and compared with manual development. The evaluation shows that the approach increases 41% in efficiency compared to manual development.**

***Index Terms*—ChatGPT, generative artificial intelligence, prompt, software development.**

I. Introduction

The emergence of **G**enerative **L**anguage-**B**ased **AI** (**Gen-LBAI**) models such as GitHub Copilot [1] and ChatGPT [12] has the potential to revolutionize the field of software engineering. They can provide significant assistance throughout the software development cycle using their high-quality domain knowledge learned from vast amounts of data to generate automatically software artifacts such as designs and code. This can help reduce the time and cost in software development by generating various types of software artifacts (e.g., design, code, test cases), while allowing developers to concentrate on critical development and quality aspects.

Recently, there have been some works using **Gen-LBAI** models at the code level such as generating code [2], [3], [4], [5] and evaluating productivity in coding [6]. Some works [7], [8] studied on assessing the quality of generated code in various aspects such as correctness, understandability, and validity. However, it has not been explored how **Gen-LBAI** models can be adopted in software development.

In this paper, we present a prompt-based approach for software development as part of the development process covering the entire development lifecycle from requirements analysis to testing. The approach is evaluated through a case study using ChatGPT-4 where the approach was compared with manual development for efficiency. The evaluation shows that the approach increases efficiency of software development by 41%, but the approach suffers from significant inconsistency among the artifacts generated from the AI model.

The rest of the paper is structured as follows. Section II presents the proposed approach. Section III discusses the evaluation of the approach in a case study. Section IV concludes the paper.

II. A Prompt-Based Approach for Software Development

Figure 1 depicts the prompt-based approach for software development. The approach makes use of a **Gen-LBAI** model via prompt throughout the software development lifecycle which includes requirements analysis, design modeling, implementation, and testing. The AI model can be used to generate various types of artifacts in each phase, which improves development efficiency. In the use of a **Gen-LBAI** model, effective prompts are crucial to obtain useful and accurate responses from the model. The remainder of the section will discuss a set of guidelines for creating effective prompts and where and how a **Gen-LBAI** model can be used in each phase of software development.

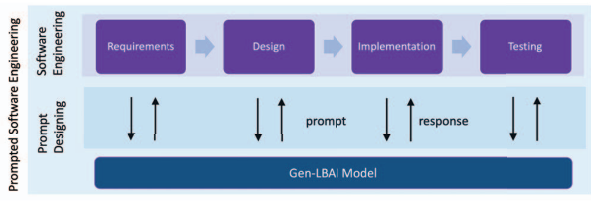


Fig. 1. Prompt-Based Approach for Software Development

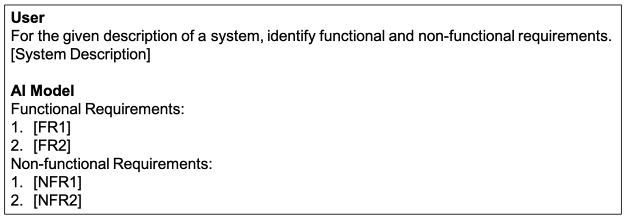
A. *Design Guidelines for Prompts*

Effective prompts play a key role in obtaining useful responses from AI models. The importance of well-crafted prompts lies in minimizing miscommunication between the user and the AI model. The outputs of prompts in software engineering are likely to be software artifacts (e.g., design, code, test cases) with various characteristics and forms. In order to produce such diverse artifacts, prompts should be devised effectively. We suggest the following guidelines for designing effective prompts.

- **Context:** Provide the context of the prompt, so that the model can consider the context in producing a response. For instance, in requirements analysis, the context of the system under development can be provided, so that the model can use the domain knowledge of the system.
- **Intention:** Provide the intention so that it can be carried in the response. For instance, in domain modeling, the intention of understanding the problem domain can be provided, so that the AI model can incorporate the characteristics of domain models such as having no design decisions into the response.
- **Constraints:** Provide constraints so that the response can be framed to be relevant. For instance, in design modeling, constraints for excluding getters and setters can be provided, so that such default operations are omitted from the design reducing complexity.
- **Specificity:** Be specific with sufficient details to lead to a specific response. For instance, in test case development, the specific form of test cases or an example of a test case can be provided, so that the model can produce test cases in the desired form.
- **Structure:** Structure with logic to guide the model, which helps the model produce the desired response. For instance, in implementation, a specific sequence of code refactorings can be provided, so that the model can carry out refactoring in the specified sequence.

B. Applying Prompts to Software Development

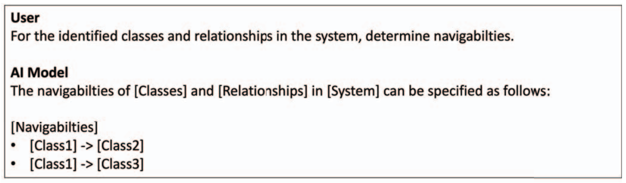
Requirements Analysis: This phase is concerned with gathering and analyzing requirements. It involves requirements identification, use case modeling, and domain modeling. In requirements identification, prompts can be used to (i) clarify the business context of the system under development and its intended users, (ii) sort out expected inputs and outputs, and (iii) identify functional and non-functional requirements. In use case modeling, prompts can be used to (i) identify use cases and actors, (ii) create use case specifications in the fully dressed form [9], (iii) identify system operations, and (iv) specify operation contracts defining the semantics of system operations. In domain modeling, prompts can be used to (i) identify domain concepts and their attributes, (ii) establish relationships, and (iii) specify multiplicities on relationships. Regarding prompt guidelines, the $Context$ and $Intention$ guidelines can be more effective as this phase focuses on clarifying requirements which deals with many ambiguities. Figure 2 shows an example prompt to identify functional and non-functional requirements from a given system description following the $Context$ guideline.


User
For the given description of a system, identify functional and non-functional requirements.
[System Description]

AI Model
Functional Requirements:
1. [FR1]
2. [FR2]
Non-functional Requirements:
1. [NFR1]
2. [NFR2]

Fig. 2. A Prompt in Requirements Analysis Using $Context$

Design Modeling: This phase focuses on creating solutions for the requirements gathered in the previous phase. It involves architecture modeling and component modeling. In architecture modeling, prompts can be used to (i) identify key architectural components and their relationships, (ii) recommend appropriate architectural patterns (e.g., layered, pipe-filter), and (iii) analyze trade-offs of patterns on quality (e.g., maintainability, performance). In component modeling, prompts can be used to (i) identify design classes with involved attributes and operations, (ii) establish the relationships between classes with navigability, (iii) apply design principles [9] (e.g., high cohesion, low coupling), and (iv) recommend and enforce design patterns [10] (e.g., Visitor, Singleton, Observer). In this phase, the $Constraints$ and $Specificity$ guidelines are more useful as the artifacts (e.g., design classes, sequential behaviors) involved in this phase are sophisticated. Figure 3 shows an example prompt to determine navigabilities of relationships per the $Specificity$ guideline.


User
For the identified classes and relationships in the system, determine navigabilties.

AI Model
The navigabilties of [Classes] and [Relationships] in [System] can be specified as follows:

[Navigabilties]
- [Class1] -> [Class2]
- [Class1] -> [Class3]

Fig. 3. A Prompt in Design Modeling Using $Specificity$

Implementation: This phase focuses on implementing the design. This phase involves design implementation and code refactoring. In design implementation, prompts can be used to (i) generate code for the design, (ii) fixing errors, (iii) conduct code review for conformance to the design, and (iv) generate in-line documentation for functions. In code refactoring, prompts can be used to (i) identify refactoring opportunities, (ii) recommend appropriate refactoring techniques [11], and (iii) apply refactoring techniques. In this phase, the $Constraints$ and $Specificity$ guidelines are more effective, given that coding itself is a complicated task. Figure 4 shows an example prompt to implement a specific operation per the $Constraints$ and $Specificity$ guidelines where the specific operation is designated with constraints on input and output.

Testing: The goal of this phase is to ensure that the software product meets the specified requirements with minimal

```
User
Implement [Operation] in [Class] to [Semantics] with [Parameter] of [Parameter Type] as input
parameter and returning a [Return Type] value.

AI Model
public class [Class] {
  private [Type] [Attribute];
  public [Return Type] [Method] ([Parameter Type] [Parameter]){
     [Implementation]}
}
```

Fig. 4. A Prompt in Implementation Using $Constraints$ and $Specificity$

defects. This phase includes test design, test execution, defect tracking, and test results analysis. In test design, prompts can be used to (i) evaluate testability of functions (e.g., executable) and (ii) create test cases for complete coverage of requirements. In test execution, prompts can be used to (i) prioritize test cases (e.g., based on complexity), (ii) generate test data, and (iii) execute test cases. In defect tracking, prompts can be used to document defects found during testing in terms of severity, steps to reproduce, and the test case in which they were found. In test results analysis, prompts can be used to (i) identify commonalities of defects such as where most defects are found or types of defects that occur most frequently and (ii) identify the underlying causes of defects. In this phase, the $Specificity$ guideline is more useful, given that test cases are sophisticated artifacts. Figure 5 shows an example prompt to test a specific method following the $Specificity$ guidelines.

```
User
For the given method, generate JUnit test cases.
[Method]

AI Model
import static org.junit.Assert.*;
import org.junit.Test;
public class [Class] {
  @Test public void test[Method](){
    [Test Code]}
}
```

Fig. 5. A Prompt in Testing Using $Specificity$

III. Evaluation: Case Study

We evaluated the approach by applying it to a case study using ChatGPT-4 to develop an online tour reservation system and comparing the application with manual development for efficiency. Both the application of the approach and the manual development were carried out by a graduate student with seven years of experience in software development and a high school research intern. Figure 6 shows the evaluation results. The table shows that the proposed approach produced a total of 125 artifacts in 626 minutes, while the manual development involved a total of 57 artifacts developed in 1066 minutes, which results in 41% increase in efficiency by the proposed approach. The majority of the time involved in the proposed approach was for reviewing the output of prompts from the AI model. A notable observation in the table is the 73.33% decrease in the development of design sequence diagrams. There were 84 design sequence diagrams produced one for each system operation in the use case specifications developed in requirements analysis. A major issue in the generated sequence diagrams was significant inconsistency among the diagrams, which took much time to be resolved. In fact, inconsistency were widespread among the artifacts generated by the AI model and much of the review time was spent to fix them.

Phase	Artifact	Prompt-Based Approach		Manual Development		Efficiency
		Count	Time (minutes)	Count	Time (minutes)	
Requirements	Use Cases	7	17	7	37	54.05%
Analysis	Use Case Diagrams	1	4	1	26	84.62%
	Use Case Specifications	7	52	7	210	75.24%
	Domain Classes	8	15	9	53	71.69%
	Domain Class Diagrams	1	16	1	35	54.29%
Design Modeling	Design Classes	8	13	9	43	69.77%
	Design Class Diagrams	1	9	1	32	71.88%
	Design Sequence Diagrams	84	260	13	150	-73.33%
Implementation	Classes	8	240	9	480	50%
Total		125	626	57	1066	41%

Fig. 6. Evaluation Results

IV. Conclusion

This work has presented a prompt-based approach for software development using **Gen-LBAI** models. The approach describes guidelines for designing prompts and where and how prompts can be used in software development. The evaluation of the approach shows that it increases efficiency of software development by 41% compared to manual development. However, the approach suffers from significant inconsistency among the artifacts generated from the AI model, which had to be resolved by human developers.

References

[1] GitHub, "Github copilot," https://copilot.github.com/, 2021.

[2] A. Mastropaolo, L. Pascarella, E. Guglielmi, M. Ciniselli, S. Scalabrino, R. Oliveto, and G. Bavota, "On the robustness of code generation techniques: An empirical study on github copilot," *arXiv preprint arXiv:2302.00438*, 2023.

[3] D. Sobania, M. Briesch, and F. Rothlauf, "Choose your programming copilot: a comparison of the program synthesis performance of github copilot and genetic programming," in *Proceedings of the Genetic and Evolutionary Computation Conference*, 2022, pp. 1019–1027.

[4] N. Nguyen and S. Nadi, "An empirical evaluation of GitHub copilot's code suggestions," in *Proceedings of the 19th International Conference on Mining Software Repositories*, 2022, pp. 1–5.

[5] 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*, 2022, pp. 62–71.

[6] S. Peng, E. Kalliamvakou, P. Cihon, and M. Demirer, "The impact of ai on developer productivity: Evidence from github copilot," *arXiv preprint arXiv:2302.06590*, 2023.

[7] A. M. Dakhel, V. Majdinasab, A. Nikanjam, F. Khomh, M. C. Desmarais, and Z. M. Jiang, "GitHub Copilot AI pair programmer: Asset or Liability?" *Journal of Systems and Software*, p. 111734, 2023.

[8] M. Wermelinger, "Using GitHub Copilot to Solve Simple Programming Problems," in *Proceedings of the 54th ACM Technical Symposium on Computing Science Education V. 1*, 2023, pp. 172–178.

[9] C. Larman, *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*. Prentice Hall, 2004.

[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design patterns: elements of reusable object-oriented software*. Pearson Deutschland GmbH, 1995.

[11] M. Fowler, *Refactoring*. Addison-Wesley Professional, 2018.

[12] O. Team, "Openai: Advances in large language models," https://chat.openai.com/, 2021.