УДК 004.89

ИССЛЕДОВАНИЕ И ФОРМАЛИЗАЦИЯ ПАТТЕРНОВ ВЗАИМОДЕЙСТВИЯ "РАЗРАБОТЧИК – ИИ-АССИСТЕНТ" ДЛЯ ОПТИМИЗАЦИИ ПРОЦЕССА НАПИСАНИЯ, РЕФАКТОРИНГА И ОТЛАДКИ КОДА

Бочаров А.Н, студент

Белорусский государственный университет информатики и радиоэлектроники
г. Минск, Республика Беларусь

Скиба И.Г. – ассистент

Аннотация. В статье рассмотрена актуальная проблема взаимодействия разработчиков программного обеспечения с интеллектуальными ассистентами кодирования (ИИ-ассистентами). Проанализированы практики и проблемы, возникающие при использовании GitHub Copilot, AWS CodeWhisperer, Tabnine. Проведен анализ и предложены паттерны взаимодействия для оптимизации процессов написания, рефакторинга и отладки кода. Оценено потенциальное влияние внедрения данных паттернов на продуктивность разработчиков и качество программного продукта.

Ключевые слова. Искусственный интеллект, ИИ-ассистент, GitHub Copilot, разработка программного обеспечения, взаимодействие человека с компьютером, паттерны взаимодействия, промпт-инжиниринг, генерация кода, рефакторинг, отладка.

Введение. В современную разработку программного обеспечения (ПО) все активнее интегрируются инструменты на основе искусственного интеллекта (ИИ). Значительное распространение получают ИИ-ассистенты кодирования, такие как GitHub Copilot, AWS CodeWhisperer, Tabnine. Эти ИИ-ассистенты позволяют значительно ускорить процесс разработки за счет автоматической генерации кода, автодополнения, помощи в рефакторинге и отладке [1]. Однако, несмотря на потенциальные преимущества, эффективное использование ИИ-ассистентов остается задачей, требующей системного подхода. Проблема заключается в том, что взаимодействие разработчика с ИИ-ассистентом часто носит нерегламентированный характер. Отсутствие устоявшихся методик и паттернов взаимодействия приводит к следующим проблемам: нерелевантность или ошибочность предложений кода, сложность формулирования точных запросов (промптов), проблемы интеграции и последующей верификации сгенерированного кода. Это может не только нивелировать ожидаемый прирост производительности, но и привести к снижению качества ПО и появлению новых уязвимостей [2].

Целью данного исследования является изучение особенностей взаимодействия разработчиков с ИИ-ассистентами и формализация эффективных паттернов этого взаимодействия для оптимизации ключевых этапов разработки: написание, рефакторинг и отладка кода.

В соответствии с поставленной целью были определены следующие задачи:

1. Анализ текущих практик использования ИИ-ассистентов разработчиками разного уровня квалификации.
2. Выявление основных проблем, возникающих при взаимодействии "разработчик – ИИ-ассистент".
3. Определение эффективных стратегий формулирования запросов, верификации и интеграции кода.
4. Формализация выявленных эффективных стратегий в виде паттернов взаимодействия.
5. Предварительная оценка потенциального влияния предложенных паттернов на процесс разработки.

Анализ существующих практик и проблем. Было проведено анкетирование и серия неформальных интервью с разработчиками ПО (n=35) различного уровня (Junior, Middle, Senior), использующих ИИ-ассистенты в своей работе. Анализ показал, что ИИ-ассистенты применяются для широкого спектра задач: от генерации шаблонного кода и автодополнения (основное использование у Junior-разработчиков) до помощи в реализации сложных алгоритмов, рефакторинге и поиске неочевидных ошибок (характерно для Middle/Senior). Несмотря на активное использование ИИ-ассистентов, были выявлены типичные проблемы (см. таблицу 1).

Таблица 1 – Основные проблемы взаимодействия с ИИ-ассистентами

Проблема	Частота упоминания, %	Характерные комментарии
Неточность/нерелевантность предложений	75	«Часто предлагает не то, что нужно», «Код не учитывает контекст проекта»
Трудности с формулированием запросов (промптов)	60	«Непонятно, как правильно спросить, чтобы получить нужный результат»
Необходимость тщательной проверки кода	85	«Никогда не принимаю код без проверки», «Иногда генерирует код с ошибками/уязвимостями»
Сложности интеграции предложенного кода	40	«Стиль кода отличается», «Нужно дорабатывать, чтобы вписать в архитектуру»
«Черный ящик» / Непонимание логики ИИ	30	«Не всегда понимаю, почему он предложил именно это»

Результаты указывают на необходимость выработки структурированных подходов к взаимодействию с ИИ-ассистентами, которые позволят минимизировать указанные проблемы.

Эффективные стратегии взаимодействия. В результате анализа проблем и примеров эффективного использования ИИ-ассистентов были определены стратегии, которые могут сделать взаимодействие разработчика с этими ИИ-ассистентами более продуктивным и безопасным. Качество результата работы ИИ напрямую зависит от качества запроса, который ему задает человек. Поэтому одним из важнейших навыков становится промпт-инжиниринг – умение грамотно формулировать задачи для ИИ. Эффективные запросы отличаются четкостью и однозначностью постановки задачи. Например, вместо общей фразы «напиши функцию для пользователя» результативнее будет конкретный запрос типа «напиши функцию Python get_user_by_id(user_id: int) с использованием SQLAlchemy для получения пользователя из БД по ID». ИИ-ассистент функционирует эффективнее, если ему предоставить релевантный контекст: окружающий код, комментарии с описанием требований, указание на используемые библиотеки и фреймворки. Получение оптимального результата не всегда возможно с первой попытки, особенно для сложных задач. В таких случаях эффективен итеративный подход, когда разработчик рассматривает взаимодействие как диалог: анализирует первый ответ ИИ и формулирует уточняющий запрос, корректируя предложения или добавляя новые детали. Эти наблюдения подтверждают важность четких запросов и управления ожиданиями, что также отмечается в исследованиях пользовательского опыта и юзабилити подобных ИИ-ассистентов [3].

Другой важный аспект – верификация и интеграция предложенного ИИ кода. Требуется постоянное проведение анализа логики, корректности, безопасности и производительности предложенного кода. Этому способствуют существующие инструменты разработки: статические анализаторы, линтеры и тесты (модульные, интеграционные). Код, генерируемый ИИ-ассистентами, в большинстве случаев требует последующей модификации. Почти всегда требуется его адаптация под стиль кодирования, принятый в проекте, под существующие архитектурные решения и конкретные требования задачи, что подчеркивает влияние инструментов ИИ-генерации на устоявшиеся процессы разработки и необходимость внимательного отношения к их результатам [4].

Использование возможностей современных IDE, таких как подсветка синтаксиса в предложениях ИИ, навигация по коду, инструменты сравнения версий, также способствует более эффективному рабочему процессу.

Формализация паттернов взаимодействия. Для систематизации полученных данных и упрощения применения эффективных стратегий использован подход, основанный на паттернах взаимодействия. По аналогии с паттернами проектирования в ПО, паттерны взаимодействия описывают проверенное, повторяемое решение типичной проблемы, возникающей при работе с ИИ-ассистентом. Рассмотрим три таких паттерна, выявленных в ходе исследования.

Первый предложенный паттерн, «Контекстно-ориентированный запрос», решает распространенную проблему генерации ИИ-ассистентом синтаксически верного, но не соответствующего контексту проекта кода (например, использование нерелевантных библиотек или нарушение принятого стиля). Это часто происходит из-за обучения моделей на обширных, но гетерогенных наборах данных, где отсутствует специфика конкретной кодовой базы. Решение заключается в том, чтобы перед формулированием основного запроса на генерацию кода явно предоставить ИИ необходимую контекстную информацию, тем самым направляя процесс генерации и повышая релевантность результата. Контекст может быть эффективно передан несколькими способами: через подробные комментарии в коде, предшествующие запросу и описывающие требования, ожидаемое поведение и окружение; через включение в сам промпт сигнатур (или даже

тел) релевантных функций, классов или структур данных, с которыми должен взаимодействовать генерируемый код; либо используя специальные возможности интегрированной среды разработки (IDE), многие из которых позволяют явно указать ассистенту файлы или выделенные фрагменты кода, которые следует учитывать при обработке запроса. Пример эффективного промпта с контекстом представлен на рисунке 1.

```
Context: Use Pydantic model 'UserData' defined above.
Input: json_string (str)
Output: UserData instance or None if parsing fails
Task: generate a Python function parse_user_data(json_string:
str) -> UserData | None:
```

Рисунок 1 – Пример промта с контекстом

Второй паттерн – «Итеративное Уточнение». Данный паттерн находит применение при решении с помощью ИИ сложных или слабо формализованных задач (например, при разработке комплексных алгоритмов, выполнении значительного рефакторинга или генерации кода по нечетким требованиям), где первый ответ ИИ-ассистента часто оказывается неполным или содержит логические неточности. Решение в таких случаях состоит в том, чтобы рассматривать взаимодействие не как однократный запрос-ответ, а как последовательный диалог или процесс совместной доработки. Получив первоначальный вариант кода от ИИ, разработчик проводит его критический анализ и, вместо полного отказа, формулирует уточняющий запрос. Ключевым аспектом здесь является предоставление конкретной и конструктивной обратной связи: явное указание на обнаруженные ошибки или логические просчеты, предложение альтернативных подходов или библиотек для использования, добавление ранее не учтенных ограничений или функциональных требований. Этот цикл «запрос – анализ – уточняющий запрос с обратной связью» повторяется итеративно до тех пор, пока не будет получен результат, полностью удовлетворяющий разработчика и соответствующий всем поставленным требованиям (см. рисунок 2).

```
Запрос: // Переработай код, чтобы он стал эффективнее
Анализ ответа ИИ: "Стало лучше, но используется N+1 запрос к БД".
Уточняющий запрос: // переработай предыдущее решение, чтобы избежать
N+1 проблемы, возможно используя "selectinload"
```

Рисунок 2 – Пример промта с уточнением

Третий паттерн – «Цикл Предложение-Верификация-Интеграция». Данный паттерн предназначен для минимизации рисков (функциональных ошибок, уязвимостей, снижения производительности и качества кода), связанных с интеграцией непроверенного кода, сгенерированного ИИ-ассистентом. При получении любого нетривиального кодового предложения от ИИ-ассистента необходимо применение регламентированного рабочего цикла, представленного на рисунке 3. Процесс начинается с получения предложения от ИИ, за которым следует обязательный этап верификации. На этом этапе выполняется всесторонняя оценка предложенного кода: проверяется корректность реализованной логики, отсутствие явных уязвимостей, потенциальное влияние на производительность и соответствие стандартам проекта, включая архитектурные ограничения. Для этого эффективно использовать как ручной анализ, позволяющий оценить читаемость и общую адекватность предложенного решения, так и инструментальные средства, такие как: статические анализаторы, SAST-инструменты, профилировщики, существующие модульные тесты. По результатам верификации принимается взвешенное решение о приемлемости кода. В случае отрицательного решения код отклоняется, и разработчик может перейти к уточнению запроса (согласно Паттерну 2) или выбрать другой подход к решению задачи. При успешной верификации, следующим этапом, выполняемым при необходимости, является адаптация кода. Даже корректный код может требовать модификаций для лучшей интеграции с существующей кодовой базой или для полного соответствия стилю кодирования, принятому в команде или проекте. Только после этапов проверки и необходимой адаптации код интегрируется в основную кодовую базу, например, через систему контроля версий. Завершающим этапом цикла является финальное тестирование. Его цель – убедиться, что интеграция не только добавила новую функциональность, но и не нарушила работу существующей части системы, то есть не вызвала регрессий. Сбой на данном этапе требует незамедлительной отладки или, возможно, отката внесенных изменений для сохранения стабильности продукта. Успешное прохождение финального тестирования завершает применение данного цикла для конкретного предложения ИИ.

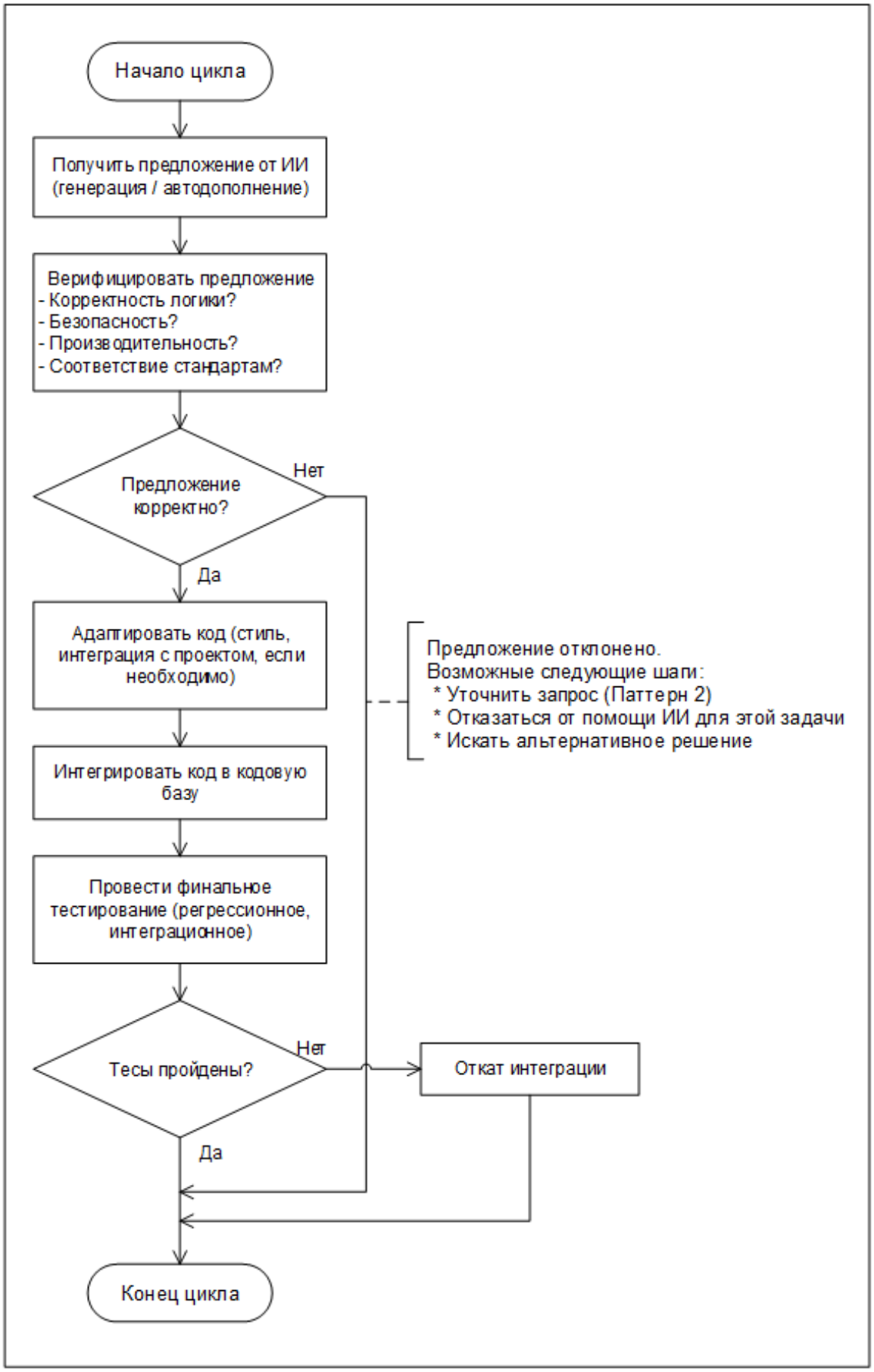

Рисунок 3 – Схема цикла «Предложение-Верификация-Интеграция»

Заключение. ИИ-ассистенты кодирования являются важной частью инструментальных средств современного разработчика. Однако реализация их потенциала требует разработки методик эффективного взаимодействия. В ходе исследования были проанализированы текущие практики использования ИИ-ассистентов и связанные с этим проблемы, выделены стратегии и формализованы три паттерна взаимодействия: «Контекстно-ориентированный запрос», «Итеративное Уточнение» и «Цикл Предложение-Верификация-Интеграция». Применение данных паттернов позволяет структурировать взаимодействие с ИИ-ассистентами, что способствует лучшему пониманию принципов эффективной работы с ИИ-ассистентом и его ограничений и ведет к росту продуктивности и улучшению качества ПО. В дальнейшем необходимо выявление и формализация большего числа паттернов, разработка инструментов для поддержки этих паттернов в IDE, а также изучение влияния ИИ-ассистентов и паттернов взаимодействия на командную работу и процессы обучения разработчиков.

Список использованных источников:

1. Ziegler, M. Programming with AI Assistance: A Comparative Study of GitHub Copilot and Human Programmers / M. Ziegler, L. G. Szafron // Proceedings of the 44th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '22). – Association for Computing Machinery, New York, NY, USA, 2022. – P. 288–297.

2. Pearce, H. Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions / H. Pearce [et al.] // 2022 IEEE Symposium on Security and Privacy (SP). – San Francisco, CA, USA, 2022. – P. 754-768.

3. Vaithilingam, P. Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models / P. Vaithilingam, T. D. LaToza, K. Moran // Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23). – Association for Computing Machinery, New York, NY, USA, 2023. – Article 715. – P. 1–17.

4. Bird, C. Don't Touch My Code! Examining the Effects of AI-Based Code Generation on Software Development / C. Bird, D. S. Sarkar, A. Mastropaolo // IEEE Software. – 2023. – Vol. 40, № 3. – P. 58-65.

UDC 004.89

RESEARCH AND FORMALIZATION OF INTERACTION PATTERNS "DEVELOPER – AI ASSISTANT" FOR OPTIMIZING THE PROCESS OF WRITING, REFACTORING, AND DEBUGGING CODE

Bocharov A.N., student

Belarusian State University of Informatics and Radioelectronics[1], Minsk, Republic of Belarus

Skiba I.G. – Assistant

Annotation. The article addresses the relevant issue of interaction between software developers and intelligent coding assistants (AI assistants). Practices and challenges associated with using GitHub Copilot, AWS CodeWhisperer, and Tabnine are analyzed. An analysis is presented, and interaction patterns are proposed for optimizing the processes of code writing, refactoring, and debugging. The potential impact of implementing these patterns on developer productivity and software product quality is assessed.

Keywords. Artificial intelligence, AI assistant, GitHub Copilot, software development, human-computer interaction, interaction patterns, prompt engineering, code generation, refactoring, debugging.