УДК 004.032.6

А.Ю. Савицкий, канд. воен. наук,
А.С. Герасимов (БГУИР, г. Минск)

РАЗВИТИЕ АЛГОРИТМИЧЕСКИХ ПОДХОДОВ ПОД ВЛИЯНИЕМ ГЕНЕРАТИВНОГО ИСКУССТВЕННОГО ИНТЕЛЕКТА И ЕГО ПОСЛЕДСТВИЯ ДЛЯ ПРОГРАММНОЙ ИНЖЕНЕРИИ

Ключевая задача программной инженерии всегда состояла в преобразовании системных требований, эффективные и надёжные алгоритмы, воплощённые в коде. Однако с появлением генеративного искусственного интеллекта (ГИИ), способного синтезировать программный код по естественно-языковым описаниям, сами основы этого процесса подвергаются пересмотру.

Технологии вроде GitHub Copilot, основанные на крупных языковых моделях (Large Language Models, LLM), эволюционируют от инструментов автодополнения к концепции активных участников разработки, способных выполнять многошаговые задачи [1]. Этот сдвиг знаменует собой фундаментальное изменение подходов, когда акцент деятельности постепенно перемещается от написания алгоритмов к их точной спецификации и внедрения в более крупные системы [2]. Согласно прогнозам, на 2026 год, развитие будет определяться не отдельными моделями, а целыми системами и агентами, способными к автономному планированию и выполнению рабочих процессов [3]. Данная трансформация ставит перед дисциплиной программной инженерии ряд новых вопросов, касающихся изменения ролевой модели разработчика, переопределения жизненного цикла ПО и возникновения уникальных рисков. В этой статье анализируется влияние ГИИ на эволюцию алгоритмических подходов и рассматриваются ключевые последствия для методологий, практик и образования в области программной инженерии.

Эволюция роли разработчика

Внедрение генеративного ИИ в жизненный цикл разработки ПО приводит к глубокому перераспределению профессиональных задач и необходимых компетенций. Если в 2022-2024 годах доминировали инструменты уровня «умного автодополнения» (поколение 1, например, GitHub Copilot), то трендом 2026 года становятся автономные агенты (поколение 3), которые могут самостоятельно брать задачу, писать код, запускать тесты и создавать запросы на объединение [1]. Эта динамика наглядно показывает постепенное смещение роли человека с тактического написания строк кода на деятельность стратегического характера. Как отмечают эксперты, разработчик превращается в

рецензента и создателя, в то время как механическую работу выполняют ИИ-агенты [4].

В новом подходе от инженера требуются принципиально иные ключевые компетенции. Прежде всего, это способность к алгоритмизации на высоком уровне абстракции, что означает умение декомпозировать сложную бизнес-проблему не на функции и классы, а на чёткие, контекстуально полные спецификации для ИИ-агента. Разработчик теперь формулирует не то, как решить задачу, а что должно быть решено и в каких граничных условиях. Не менее важным становится управление контекстом, поскольку эффективное взаимодействие с ИИ превращается в новый вид интерфейса между замыслом и его реализацией. Эксперты подчёркивают, что в 2026 году успех будет зависеть от способности управлять контекстом проекта и точно доносить задачи до модели [5]. Одновременно резко возрастает значимость критического мышления и верификации. Поскольку ИИ может генерировать правдоподобный, но содержащий ошибки, неоптимальный или даже небезопасный код, фундаментальной обязанностью разработчика становится тщательная проверка, тестирование и анализ предлагаемых решений [6]. Наконец, в условиях использования многоагентных систем, где различные ИИ-агенты специализируются на отдельных задачах, роль инженера трансформируется в роль руководителя, который проектирует взаимодействия и рабочие процессы между этими автономными компонентами. Этот подход, становится ключевым навыком, смещая разработчика от написания кода к управлению командами агентов [7].

Таким образом, алгоритмическое мышление не утрачивает актуальности, а поднимается на мета-уровень, фокусируясь на проектировании систем и процессов в целом, что созвучно общим трендам трансформации инженерного труда под влиянием ИИ [8].

Изменение алгоритмических подходов и возникновение новых классов задач

Проникновение ГИИ в программную инженерию порождает новые области для применения и развития алгоритмических подходов, выходящие далеко за рамки традиционного кодирования. Одной из таких областей стала алгоритмизация взаимодействия в многоагентных средах. Проектирование устойчивых и эффективных протоколов для ИИ-агентов превращается в сложную алгоритмическую задачу, требующую решения проблем распределения задач, избежания конфликтов, обеспечения консистентности данных и достижения глобальных целей. Для поддержки таких систем возникают новые стандарты, такие как Model Context Protocol (MCP), который становится ключевым для подключения агентов к инструментам и данным [3].

Этот сдвиг указывает на переход от конкурентной борьбы на уровне моделей ИИ к конкуренции на уровне целых систем и платформ [7].

Другим вызовом является то, что генеративный ИИ, часто оптимизированный для быстрого достижения локальных целей, может способствовать накоплению специфического «агентного техдолга», выражающегося в создании избыточно сложного, плохо структурированного или содержащего скрытые уязвимости кода. Исследования указывают на рост метрик, подобных Code Churn (текучесть кода), когда разработчики, увлечённые лёгкостью генерации, многократно переписывают одни и те же участки, не добавляя ценности, а также на риск «разработки через копипаст» (Copy-Paste Driven Development) [9]. Противодействие этому требует разработки новых алгоритмов статического и динамического анализа, способных выявлять паттерны, характерные для ИИ-генерируемого кода, а также алгоритмов автоматической переработки.

Параллельно наблюдается развитие формальных методов спецификации, поскольку растёт потребность в языках и методах, которые могли бы однозначно описывать требования для ИИ-агентов, минимизируя неопределённость и риск ошибок. Передовые практики смещаются от «вейб-кодинга» к «Протоколу Цель-Валидация» (Objective-Validation Protocol), где пользователь определяет цели и проверяет результаты, а агенты автономно исполняют задачи [5]. Это стимулирует интерес к интеграции методов контрактного программирования и формальной верификации в процесс промпт-инжиниринга (взаимодействия с ИИ). Всё чаще успешная работа с ИИ начинается с создания детального плана и спецификации, что позволяет и разработчику, и агенту находиться «на одной странице» [6].

Последствия для образования и индустриальных практик

Трансформация алгоритмических подходов под влиянием генеративного ИИ требует адекватного и сбалансированного ответа как от системы образования, так и от индустриальных стандартов. В образовательных программах акцент должен постепенно смещаться с механического запоминания синтаксиса конкретных языков к углублённому пониманию фундаментальных принципов. Структуры данных, алгоритмическая сложность, архитектурные паттерны и методологии проектирования становятся важнее, чем когда-либо, поскольку только прочное усвоение этих основ позволяет будущим специалистам эффективно направлять и критически оценивать работу ГИИ [2]. Параллельно с этим возникает необходимость в интеграции новых дисциплин, связанных с основами работы LLM, промпт-инжинирингом, инженерией контекста и этическими аспектами применения подобных систем в профессиональной деятельности.

На уровне индустриальных методологий также необходима адаптация. Стандартные процессы разработки, такие как Agile и DevOps, должны быть пересмотрены с учётом включения ИИ-агентов в качестве полноправных участников рабочих процессов. Это касается всех этапов. На этапе планирования задач требуется новая чёткость формулировок. При проведении анализа проверяется не только итоговый код, но и исходные промпты с контекстом, а также соответствие результатов архитектурным решениям. Процедуры тестирования и непрерывной интеграции должны быть усилены для автоматизированной проверки ИИ-генерируемого кода [1]. Особую важность приобретает управление знаниями и контекстом организации, что напрямую влияет на эффективность работы интеллектуальных агентов [5].

Кроме того, широкое внедрение ГИИ выводит на первый план комплекс этических, управленческих и экономических вопросов. Речь идёт об определении авторства кода, распределении ответственности за возможные ошибки, обеспечении безопасности данных (например, через использование локальных моделей или Enterprise Trust Layer [10]) и трансформации профессиональных ролей на рынке труда. Индустрия переходит от фазы экспериментов к фазе измерения реальной отдачи от инвестиций и экономической эффективности [8]. Таким образом, перед программной инженерией стоит задача разработки концепций и лучших практик для безопасного, этичного и одновременно эффективного использования генеративных технологий, что требует междисциплинарного подхода и стратегического управления изменениями.

Таким образом развитие алгоритмических подходов под влиянием генеративного искусственного интеллекта представляет собой не кризис дисциплины, а её закономерную и глубокую эволюцию. Фундаментальное алгоритмическое и системное мышление не только не теряет своей актуальности, но и становится важнейшим дифференцирующим навыком современного инженера. Однако область его применения закономерно смещается с уровня непосредственной реализации на уровень проектирования, создания спецификаций и осуществления контроля. Актуальными проблемами программной инженерии сегодня становятся разработка методологий эффективного симбиоза человека и ИИ, создание инструментов для управления качеством и безопасностью ИИ-генерируемых продуктов, а также адаптация образовательных стандартов и индустриальных практик к новым подходам. Успешное будущее профессии напрямую связано со способностью инженеров выступать в роли создателей интеллектуальных систем, где глубокое понимание классических принципов алгоритмизации служит надёжной основой для управления мощью генеративного

искусственного интеллекта и направления её на решение сложных практических задач. Как отмечают аналитики, в 2026 году фокус смещается с самих моделей на системы и результаты их работы [3], и именно программные инженеры, обладающие соответствующими мета-навыками, будут определять, какие результаты будут достигнуты.

ЛИТЕРАТУРА

1. Акерман Д. Code AI 2026: Почему 'просто Copilot' уже недостаточно и как нейросети переписывают профессию программиста // Mypl.pro. – 2025. – 2 декабря. – URL: https://mypl.pro/blog/code-ai-2026-why-just-copilot-is-no-longer-enough-and-how-neural-networks-rewrite-the-profession-of-a-programmer (дата обращения: 13.01.2026).

2. Топ-10 AI-помощников для программирования: обзор инструментов и как с ними работать // GoIT. – 2026. – URL: https://goit.global/ua-ru/articles/ai-dlia-programmirovaniya/ (дата обращения: 13.01.2026).

3. The trends that will shape AI and tech in 2026 // IBM Think. – 2026. – 1 января. – URL: https://www.ibm.com/think/news/ai-tech-trends-predictions-2026 (дата обращения: 13.01.2026).

4. Число пользователей GitHub Copilot превысило 20 миллионов // Habr. – 2025. – URL: https://habr.com/ru/companies/bothub/news/932668/ (дата обращения: 13.01.2026).

5. Osmani A. My LLM coding workflow going into 2026 // Medium. – 2026. – URL: https://medium.com/@addyosmani/my-llm-coding-workflow-going-into-2026-52fe1681325e (дата обращения: 13.01.2026).

6. Large Language Models: What You Need to Know in 2026 // Hatchworks. – 2025. – 23 декабря. – URL: https://hatchworks.com/blog/gen-ai/large-language-models-guide/ (дата обращения: 13.01.2026).

7. Signals for 2026 // O'Reilly Radar. – 2026. – URL: https://www.oreilly.com/radar/signals-for-2026/ (дата обращения: 13.01.2026).

8. Топ-7 трендов ИИ, за которыми стоит следить в 2026 году // CometAPI. – 2026. – 8 января. – URL: https://www.cometapi.com/ru/top-7-ai-trends-to-watch-in-2026/ (дата обращения: 13.01.2026).

9. Top 9 Large Language Models as of January 2026 // Shakudo. – 2026. – URL: https://www.shakudo.io/blog/top-9-large-language-models (дата обращения: 13.01.2026).

10. 30 of the best large language models in 2026 // TechTarget. – 2025. – 3 декабря. – URL: https://www.techtarget.com/whatis/feature/12-of-the-best-large-language-models (дата обращения: 13.01.2026).УДК