УДК 004.4

Бадертдинова Елена Радитовна, доктор технических наук, ФГБОУ ВО «Казанский национальный исследовательский технологический университет», г. Казань

Совгачев Александр Андреевич, магистрант, ФГБОУ ВО «Казанский национальный исследовательский технологический университет», г. Казань

ТЕХНИЧЕСКИЙ ДОЛГ В ЭПОХУ ГЕНЕРАТИВНОГО ИИ: ВЛИЯНИЕ AI-СГЕНЕРИРОВАННОГО КОДА НА КАЧЕСТВО И СОПРОВОЖДАЕМОСТЬ ПРОГРАММНЫХ СИСТЕМ

Аннотация

Статья исследует технический долг как системный риск, возникающий при интенсивном использовании инструментов генеративного ИИ в разработке программного обеспечения. На основе нарративного обзора ключевых эмпирических работ 2022–2025 годов выявлены и систематизированы три измерения AI-индуцированного технического долга: деградация структурного качества кода (феномен code churn), накопление уязвимостей безопасности и рост нагрузки на сопровождение программных систем. В статье предложена авторская концептуализация механизмов накопления AI-долга, включающая дефицит контекстуального понимания, эффект самоуверенности при ревью, дублирование кода и утрату неявного знания. Разработана матрица профилей риска, позволяющая дифференцировать сценарии использования GenAI по степени долговой нагрузки. Статья является продолжением работы авторов об эволюции парадигмы вайб-кодинга и расширяет её фокус с краткосрочных эффектов продуктивности на долгосрочные последствия для инженерного качества программных систем.

Annotation

This paper examines technical debt as a systemic risk arising from the intensive use of generative AI tools in software development. Based on a narrative review of key empirical studies from 2022–2025, three dimensions of AI-induced technical debt are identified and systematized: structural code quality degradation (the code churn phenomenon), accumulation of security vulnerabilities, and increased software maintenance burden. The paper proposes an original conceptualization of AI debt accumulation mechanisms, encompassing contextual understanding deficits, overconfidence effects in code review, code duplication, and tacit knowledge loss. A risk profile matrix is developed to differentiate GenAI usage scenarios by degree of debt exposure. This paper extends the authors' prior work on the vibe coding paradigm, shifting focus from short-term productivity effects to long-term consequences for the engineering quality of software systems.

Ключевые слова: технический долг, генеративный искусственный интеллект, AI-сгенерированный код, сопровождаемость программного обеспечения, качество кода, code churn, безопасность программных систем, когнитивная нагрузка, вайб-кодинг, устойчивая разработка.

Keywords: technical debt, generative artificial intelligence, AI-generated code, software maintainability, code quality, code churn, software security, cognitive load, vibe coding, sustainable software development.

Введение

Генеративный ИИ стал одним из наиболее быстро принятых инструментов в истории программной инженерии. По данным опросов Stack Overflow Developer Survey (2024), более 60 % профессиональных разработчиков используют или планируют использовать AI-ассистенты в своей рабочей практике. По заявлениям компании GitHub, Copilot принят более чем в 90 % компаний из списка Fortune 100. Скорость внедрения, однако, порождает вопрос, который задаётся значительно реже: что происходит с программными системами после того, как код написан при помощи ИИ?

Ранние исследования 2022–2024 годов в основном фиксировали краткосрочные выгоды: ускорение выполнения задач, рост количества pull

requests, снижение когнитивной нагрузки при решении рутинных задач [8, 6, 16]. Однако уже к 2025 году в академической литературе начали появляться данные об иных, менее очевидных последствиях — накоплении технического долга, росте нагрузки на code review и систематическом снижении сопровождаемости кода, созданного с участием AI-инструментов [15, 4].

Методология обзора

Источники отбирались методом целенаправленной выборки (purposive sampling) через поиск в базах arXiv, ACM Digital Library, Google Scholar и Semantic Scholar по запросам «AI-generated code technical debt», «generative AI code quality», «LLM code security», «software maintainability AI assistant», «code churn generative AI». Критерии включения: наличие эмпирических данных о качестве или сопровождаемости кода, созданного с участием GenAI-инструментов; период публикации 2022–2026 годов; доступность полного текста. Работы, посвящённые исключительно технике генерации кода без анализа последствий для качества кодовой базы, исключались. Настоящий обзор является нарративным и не претендует на исчерпывающее покрытие всего корпуса литературы по теме.

Цель статьи — синтезировать эмпирические данные 2022–2025 годов о влиянии AI-сгенерированного кода на качество и сопровождаемость программных систем, предложить концептуализацию механизмов накопления AI-индуцированного технического долга и разработать практически применимую матрицу профилей риска.

Технический долг в контексте генеративного ИИ

Понятие «технический долг» было введено Уордом Каннингемом в 1992 году как метафора для объяснения накапливающихся последствий проектных решений, принятых в пользу скорости в ущерб качеству [Cunningham, 1992]. В оригинальной трактовке технический долг — это подразумеваемая стоимость переработки, которая возникает при выборе краткосрочного «быстрого» решения вместо правильного долгосрочного. Метафора процентов здесь существенна: долг сам по себе не смертелен, но чем дольше

он не погашается, тем дороже обходится каждое последующее изменение системы.

В академической литературе сложилась разветвлённая таксономия технического долга. Различают архитектурный долг (несовместимые с общей архитектурой решения), кодовый долг (плохо структурированный, продублированный или чрезмерно сложный код), тестовый долг (недостаточное покрытие и низкое качество тестов), документационный долг (отсутствие или неактуальность описания системы) и долг безопасности (известные уязвимости, не устранённые своевременно). Каждый из этих типов имеет свои метрики: cyclomatic complexity, code churn, maintainability index, CVSS-оценки уязвимостей и другие.

Ключевая особенность классического техдолга — он возникает в результате сознательного или несознательного выбора разработчика. Разработчик знает или мог знать о более качественном решении, но принял компромисс: ради дедлайна, ради прототипа, ради ограниченности ресурсов. Это разграничение окажется принципиальным при анализе AI-индуцированного долга.

Специфика AI-индуцированного технического долга

AI-сгенерированный технический долг обладает принципиально иной природой по сравнению с классическим. Разработчик, принимающий предложение AI-ассистента, как правило, не принимает сознательного решения «срезать угол». Напротив, он может быть убеждён, что получает оптимальное решение — ведь оно сгенерировано мощной моделью, прошедшей обучение на миллиардах строк кода. Технический долг здесь возникает не как результат компромисса, а как побочный эффект делегирования: разработчик передаёт задачу инструменту, не располагающему полным контекстом системы.

Авторы настоящего обзора предлагают определить AI-индуцированный технический долг следующим образом: это подразумеваемая стоимость переработки и дополнительного сопровождения, возникающая вследствие

структурного, семантического или архитектурного несоответствия AI-сгенерированного кода долгосрочным требованиям программной системы — несоответствия, которое не было намеренно заложено разработчиком, но стало следствием ограниченного контекстуального понимания генерирующей модели.

Liu et al. (2026) в масштабном эмпирическом исследовании реальных репозиториев GitHub вводят это явление в академический дискурс, описывая феномен «невидимого долга»: AI-сгенерированный код компилируется без ошибок, проходит синтаксические проверки, нередко корректно выполняет целевую функцию. Тесты зелёные. Ревью пройдено. Но структурная деградация уже заложена — и проявится она лишь тогда, когда команда попытается изменить, расширить или отладить этот фрагмент системы [5].

Смежная концептуализация предложена Storey (2026), которая расширяет понятие технического долга до «тройной модели задолженности» (Triple Debt Model): помимо классического технического долга, генеративный ИИ порождает когнитивный долг (erosion of shared team understanding — постепенное разрушение общего понимания системы в команде) и долг намерений (intent debt — отсутствие явного обоснования принятых решений). Последние два типа долга напрямую связаны с природой AI-генерации и существенно усложняют долгосрочное сопровождение [14].

GitClear (2024) операционализирует этот феномен через метрику code churn — долю кода, переписанного в течение двух недель после первоначального создания. Рост churn является косвенным, но надёжным индикатором заложенного долга: код, который нужно переписывать сразу после написания, не был оптимальным изначально. Именно эта метрика становится центральной для понимания масштаба AI-индуцированного долга на уровне реальных кодовых баз [4].

Структурная деградация кодовой базы

Эмпирические исследования последних лет указывают на связь между использованием генеративного ИИ и ухудшением отдельных показателей

качества программного кода. Согласно данным GitClear (2024), после массового распространения AI-ассистентов наблюдается устойчивый рост метрики code churn — доли кода, переписываемого вскоре после первоначального создания. Одновременно снижается доля изменений, связанных с рефакторингом и архитектурным улучшением системы [4].

Схожие результаты получены в исследованиях качества AI-сгенерированного кода. Paul, Zhu и Bayley (2025) выявили существенный рост числа code smells по сравнению с эталонными реализациями, а Sabra et al. (2025) подтвердили наличие дефектов качества и проблем безопасности в результатах нескольких популярных LLM [7, 11].

Дополнительное подтверждение дают данные Xu et al. (2025), согласно которым увеличение объёма создаваемого кода не всегда сопровождается повышением его сопровождаемости. Авторы отмечают рост нагрузки на процесс ревью и увеличение количества последующих доработок, необходимых для интеграции AI-сгенерированных решений в существующие проекты [15].

В совокупности результаты исследований позволяют предположить, что генеративный ИИ способен ускорять создание программного кода, однако часть полученного выигрыша компенсируется дополнительными затратами на исправление структурных недостатков и поддержание качества кодовой базы.

Безопасность как измерение технического долга

Уязвимости безопасности представляют собой одну из наиболее затратных форм технического долга, поскольку их последствия могут проявляться спустя значительное время после внедрения программного решения. В отличие от структурных недостатков кода, которые обычно обнаруживаются в процессе сопровождения, проблемы безопасности нередко остаются незамеченными до момента эксплуатации или проведения специализированного аудита.

Исследования Perry et al. (2023) показали, что использование AI-ассистентов способно увеличивать вероятность появления небезопасных

программных конструкций. При этом разработчики, работавшие с AI-инструментами, демонстрировали более высокий уровень уверенности в корректности полученного результата, что снижало критичность последующей проверки [9]. Схожие выводы представлены в работе Sandoval et al. (2023), где было установлено, что генеративные модели могут воспроизводить распространённые уязвимые паттерны программирования, включая ошибки обработки пользовательского ввода и механизмы небезопасной аутентификации [12].

Дополнительное подтверждение данной тенденции содержится в исследовании Sabra et al. (2025), выявившем наличие дефектов безопасности в коде, сгенерированном несколькими современными языковыми моделями [11].

Причины подобных проблем во многом связаны с особенностями обучения LLM на больших массивах публичного программного кода. Модели воспроизводят статистически наиболее распространённые решения, среди которых присутствуют как безопасные, так и потенциально уязвимые практики разработки. При недостатке контекста модель не всегда способна корректно определить, какой вариант является предпочтительным в конкретной ситуации.

С точки зрения технического долга такие уязвимости формируют отложенные издержки. На этапе внедрения система может функционировать корректно, однако в дальнейшем организация вынуждена нести дополнительные расходы на аудит безопасности, устранение обнаруженных проблем и минимизацию последствий возможных инцидентов.

Сопровождаемость и нагрузка на команду

Третье измерение AI-индуцированного технического долга связано с влиянием генеративного ИИ на процессы сопровождения программных систем и взаимодействие внутри команды разработки. В отличие от структурных дефектов или уязвимостей безопасности, данный тип долга

проявляется через увеличение затрат времени на проверку, понимание и доработку кода.

По данным Xu et al. (2025), активное использование AI-инструментов сопровождается ростом нагрузки на опытных участников проектов. Авторы отмечают увеличение времени, затрачиваемого на code review и последующие доработки, поскольку AI-сгенерированные решения требуют дополнительной проверки на соответствие архитектуре и стандартам проекта [15].

Схожие результаты были получены Rush et al. (2025) в эксперименте с опытными разработчиками open-source проектов. Несмотря на ожидания повышения производительности, существенная часть времени расходовалась на анализ, верификацию и корректировку предложений AI-ассистентов, что в ряде случаев приводило к снижению итоговой эффективности работы [10].

Вместе с тем влияние генеративного ИИ не является однозначно негативным. Исследование Brandebusemeyer et al. (2025) показывает, что результаты во многом зависят от сценария использования инструмента. При наличии понятных правил взаимодействия с AI и сохранении контроля со стороны разработчика влияние на когнитивную нагрузку и сопровождаемость оказывается существенно ниже [2].

Таким образом, современные исследования указывают на то, что преимущества генеративного ИИ в разработке следует оценивать не только через скорость создания кода, но и через последующие затраты на его поддержку, ревью и интеграцию в существующую кодовую базу.

Дефицит контекстуального понимания и архитектурный долг

Одной из ключевых причин накопления AI-индуцированного технического долга является ограниченность контекста, доступного языковой модели. Несмотря на способность анализировать значительные объёмы кода, современные LLM не обладают полным представлением о программной системе, её архитектурных ограничениях и накопленных проектных решениях. В результате сгенерированные фрагменты могут корректно решать локальную задачу, но противоречить общей архитектуре проекта. Подобные

несоответствия приводят к росту архитектурного долга и необходимости последующих доработок, что косвенно подтверждается наблюдаемым увеличением показателей code churn [4].

Эффект самоуверенности и деградация ревью

Дополнительным фактором выступает изменение поведения разработчиков при взаимодействии с AI-инструментами. Результаты Perry et al. (2023) показывают, что использование AI-ассистентов может повышать субъективную уверенность в корректности предложенного решения [9]. Снижение критичности оценки приводит к тому, что потенциальные проблемы не выявляются на этапе ревью и переносятся на более поздние стадии жизненного цикла программного обеспечения. Вследствие этого возрастает объём повторных доработок и исправлений [15].

Дублирование кода и потеря когерентности кодовой базы

Генеративные модели формируют решения на основе доступного контекста и не всегда способны учитывать существующие реализации аналогичной функциональности внутри проекта. Это создаёт предпосылки для появления дублирующегося кода и параллельных реализаций схожих механизмов. Подобные практики усложняют сопровождение системы, увеличивают риск несогласованных изменений и способствуют накоплению структурного технического долга. Рост показателей design smells, зафиксированный в ряде исследований, может рассматриваться как одно из проявлений данного процесса [7].

Утрата неявного знания и долг намерений

Важная часть знаний о программной системе существует в неявной форме и связана с пониманием причин принятых архитектурных и проектных решений. При активном использовании AI-инструментов существует риск постепенной утраты такого контекста, поскольку модель способна воспроизводить готовые решения без объяснения их происхождения и ограничений. В работе Storey (2026) данное явление рассматривается через концепцию «долга намерений», возникающего при потере информации о

мотивах и предпосылках проектных решений [14]. В долгосрочной перспективе это может затруднять развитие системы и снижать устойчивость процессов сопровождения.

Профили риска: дифференцированный подход

Данные исследований убедительно демонстрируют, что AI-индуцированный технический долг не является неизбежным следствием применения GenAI-инструментов. Brandebusemeyer et al. (2025) зафиксировали «оптимальный режим» использования, при котором риск минимален. Rush et al. / METR (2025) показали, что наибольшее замедление фиксируется именно у опытных разработчиков на сложных задачах — тогда как junior на рутинных задачах демонстрируют преимущества AI-ассистирования без значимых потерь качества. Это позволяет предложить матрицу профилей риска, дифференцирующую сценарии по степени долговой нагрузки.

Характеристика	**Низкий риск**	**Средний риск**	**Высокий риск**
Тип задачи	Рутинная, изолированная, хорошо специфицированная (рефакторинг, документация, написание тестов)	Умеренная сложность, частично задокументированные требования, работа в рамках одного модуля	Архитектурная, межкомпонентная, с неявными требованиями или ограничениями безопасности
Уровень разработчика	Senior, глубокое знание системы и архитектуры — верифицирует	Mid-level, знание отдельных модулей — верификация частичная	Junior, ограниченный контекст системы — принимает AI-предложения с

	AI-предложения критически		минимальной проверкой
Глубина code review	Строгое, с архитектурным контекстом; специальное внимание к AI-сгенерированным фрагментам	Стандартное, без специального протокола для AI-кода	Минимальное, автоматизированное или формальное
Зрелость кодовой базы	Зрелая, хорошо документированная, высокое тестовое покрытие, устоявшиеся паттерны	Умеренная зрелость, частичное покрытие тестами	Новая база без устоявшихся паттернов или легаси с высоким существующим долгом
Характеристика	**Низкий риск**	**Средний риск**	**Высокий риск**
Режим использования AI	Дополняющий: AI как черновик, разработчик сохраняет авторскую ответственность; один канал взаимодействия	Смешанный: без чёткого протокола, ситуативное использование нескольких режимов	Замещающий: AI как основной автор, минимальная верификация; интенсивное комбинирование режимов
Ключевые риски	Минимальные; локальные стилистические расхождения	Дублирование, неоптимальные паттерны, умеренный rework	Архитектурный долг, уязвимости безопасности, высокий code

			churn, нагрузка на core contributors
Опорные исследования	Brandebusemeyer et al. (2025) — оптимальный режим	Xu et al. (2025) — умеренная нагрузка	Perry et al. (2023), GitClear (2024), Rush/METR (2025)

Таблица 1 - матрица профилей риска накопления AI-индуцированного технического долга

Представленная матрица носит эвристический характер и предназначена для практической оценки рисков. Её практическая ценность состоит в том, что она позволяет командам идентифицировать наиболее уязвимые точки своего AI-рабочего процесса и адресно применять защитные меры. Сценарий «junior-разработчик + межкомпонентная задача + минимальное ревью + замещающий режим AI» является максимально рискованным с точки зрения накопления долга. Сценарий «senior-разработчик + рутинная задача + строгое ревью + дополняющий режим AI» — практически безопасным [2, 10].

Стратегии управления AI-индуцированным техническим долгом

Снижение рисков, связанных с использованием генеративного ИИ, требует сочетания организационных, технических и компетентностных мер. На организационном уровне важную роль играет адаптация процессов code review к особенностям AI-сгенерированного кода. Проверка должна включать анализ архитектурной согласованности решений, поиск дублирования логики и дополнительную оценку потенциальных рисков безопасности [2].

На техническом уровне эффективными инструментами являются автоматизированный статический анализ кода, quality gates и мониторинг показателей качества программного обеспечения. Особое значение приобретает контроль метрики code churn, которая может использоваться как индикатор накопления AI-индуцированного технического долга [4].

Дополнительное значение имеют практики фиксации архитектурных решений и причин их принятия. Использование Architectural Decision Records позволяет сохранить контекст проектирования и снижает риск возникновения «долга намерений», описанного Storey (2026) [14].

Наконец, важным фактором остаётся уровень подготовки разработчиков. Эффективное применение AI-инструментов требует не только навыков работы с ними, но и способности критически оценивать полученные результаты, выявлять архитектурные несоответствия и своевременно обнаруживать потенциальные дефекты [10, 15].

Таким образом, управление AI-индуцированным техническим долгом должно рассматриваться как часть общей стратегии обеспечения качества программного обеспечения, а не как отдельная задача, связанная исключительно с использованием генеративного ИИ.

Заключение

Генеративный ИИ не генерирует технический долг автоматически и неизбежно. Однако он создаёт принципиально новые механизмы его накопления, которые не покрываются существующими практиками управления качеством — и именно поэтому требуют отдельного осмысления.

На основе нарративного обзора ключевых эмпирических работ 2022–2026 годов авторы выделили три документированных измерения AI-индуцированного технического долга. Первое — структурная деградация кодовой базы: GitClear (2024) фиксирует удвоение code churn за период массового внедрения AI-ассистентов, Paul et al. (2025) измеряют рост code smells на 63 % в среднем — оба показателя являются прямыми индикаторами накопления структурного долга [4, 7]. Второе — уязвимости безопасности: Perry et al. (2023) и Sandoval et al. (2023) демонстрируют систематическую тенденцию к появлению небезопасных паттернов в AI-сгенерированном коде — усугублённую эффектом самоуверенности ревьюеров [9, 12]. Третье —

нагрузка на сопровождение: Xu et al. (2025) и Rush et al. / METR (2025) независимо фиксируют рост времени на review и rework, снижение оригинальной продуктивности опытных разработчиков примерно на 19 % [15, 10].

Авторская концептуализация механизмов накопления AI-долга выделяет четыре взаимосвязанных процесса: дефицит контекстуального понимания (архитектурный долг), эффект самоуверенности (деградация ревью), дублирование кода (потеря когерентности) и утрата неявного знания (долг намерений по Storey, 2026) [14]. Разработанная матрица профилей риска позволяет командам дифференцировать сценарии использования GenAI по степени долговой нагрузки и адресно применять меры управления.

Связь с предшествующей статьёй авторов [1] принципиальна: парадигма вайб-кодинга — максимального доверия AI-инструменту при минимальной верификации его вывода — является крайним проявлением всех четырёх механизмов одновременно. Чем полнее делегирование и чем менее критичен разработчик — тем выше риск накопления AI-индуцированного долга [1].

Ограничения обзора

Рассмотренные исследования проводились преимущественно на англоязычной аудитории и в американских или западноевропейских организационных контекстах, что ограничивает прямую экстраполяцию выводов на другие регионы. Большинство эмпирических работ сосредоточены на open-source проектах или крупных корпоративных средах; малый и средний бизнес, а также ранние стартапы остаются менее изученными. Publication bias мог привести к недопредставленности нейтральных или отрицательных результатов в части исследований. Наконец, быстрое развитие самих AI-инструментов означает, что данные 2022–2023 годов могут не вполне отражать поведение frontier-моделей 2025–2026 годов.

Список литературы

1. Brandebusemeyer C., Schimmer T., Arnrich B. Developers' experience with generative AI: Mixed-methods field study : arXiv preprint. 2025. URL: https://arxiv.org/abs/2512.19926 (дата обращения: 26.03.2026).
2. Cunningham W. The WyCash Portfolio Management System // ACM SIGPLAN OOPS Messenger. 1992. Vol. 4, № 2. P. 29–30. DOI: 10.1145/157710.157715.
3. GitClear. Coding on Copilot: 2023 Data Suggests AI Copilots Erosion of Code Quality (Debt Behind the AI Boom). 2024. URL: https://www.gitclear.com/coding_on_copilot_data_shows_ais_downward_pressure_on_code_quality (дата обращения: 08.04.2026).
4. Liu Y., Widyasari R., Zhao Y., Irsan I.C., Lo D. Debt Behind the AI Boom: A Large-Scale Empirical Study of AI-Generated Code in the Wild : arXiv preprint. 2026. URL: https://arxiv.org/abs/2603.28592 (дата обращения: 08.04.2026).
5. Noy S., Zhang W. Experimental evidence on the productivity effects of generative artificial intelligence // Science. 2023. Vol. 381, № 6654. P. 187–192. DOI: 10.1126/science.adh2586.
6. Paul D.G., Zhu H., Bayley I. Investigating The Smells of LLM Generated Code : arXiv preprint. 2025. URL: https://arxiv.org/abs/2510.03029 (дата обращения: 08.04.2026).
7. Peng S., Kalliamvakou E., Cihon P., Demirer M. The impact of AI on developer productivity: Evidence from GitHub Copilot : arXiv preprint. 2023. URL: https://arxiv.org/abs/2302.06590 (дата обращения: 26.03.2026).
8. Perry N., Srivastava M., Kumar D., Boneh D. Do Users Write More Insecure Code with AI Assistants? // Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS). 2023. DOI: 10.1145/3576915.3623157.

9. Rush A., Becker J. et al. Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity : arXiv preprint. 2025. URL: https://arxiv.org/abs/2507.09089 (дата обращения: 08.04.2026).
10. Sabra A., Schmitt O., Tyler J. Assessing the Quality and Security of AI-Generated Code: A Quantitative Analysis : arXiv preprint. 2025. URL: https://arxiv.org/abs/2508.14727 (дата обращения: 08.04.2026).
11. Sandoval G. et al. Lost at C: A User Study on the Security Implications of Large Language Model Code Assistants // Proceedings of the 32nd USENIX Security Symposium. 2023. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/sandoval (дата обращения: 08.04.2026).
12. Stack Overflow. Developer Survey 2024. 2024. URL: https://survey.stackoverflow.co/2024/ (дата обращения: 08.04.2026).
13. Storey M.-A. From Technical Debt to Cognitive and Intent Debt: Rethinking Software Health in the Age of AI : arXiv preprint. 2026. URL: https://arxiv.org/abs/2603.22106 (дата обращения: 08.04.2026).
14. Xu F. et al. AI-assisted programming may decrease the productivity of experienced developers by increasing maintenance burden : arXiv preprint. 2025. URL: https://arxiv.org/abs/2510.10165 (дата обращения: 26.03.2026).
15. Ziegler A. et al. Measuring GitHub Copilot's impact on productivity // Communications of the ACM. 2024. Vol. 67, № 3. P. 54–63.