Лукьянчук Никита Сергеевич

Ведущий специалист, Санкт-Петербургский политехнический университет Петра Великого, Россия, Санкт-Петербург

СОЗДАНИЕ ПРИЛОЖЕНИЙ С ПОМОЩЬЮ ИСКУССТВЕННОГО ИНТЕЛЛЕКТА

Аннотация

Статья посвящена исследованию влияния искусственного интеллекта на разработку приложений. Рассматриваются возможности ИИ в задачах генерации программного кода, проектирования пользовательских интерфейсов, тестирования и сопровождения программных систем, а также в автоматизации DevOps-практик. Отдельное внимание уделяется трансформации роли разработчика, распространению no-code и low-code платформ и ограничениям, связанным с безопасностью и качеством решений, созданных с использованием интеллектуальных технологий.

Ключевые слова: искусственный интеллект, разработка приложений, генерация кода, программная инженерия, машинное обучение, DevOps, тестирование программного обеспечения, no-code, low-code, автоматизация разработки.

Текст статьи

За последние годы искусственный интеллект занял заметное место в цифровой экономике и в программной инженерии. В разработке приложений это особенно видно: привычные методы программирования постепенно дополняются инструментами, которые работают на основе ИИ. Он уже не ограничивается анализом данных. Сейчас такие системы могут генерировать

код, помогать проектировать интерфейсы, запускать тесты и участвовать в развёртывании приложений. Из-за этого сам процесс создания программных продуктов стал быстрее и в ряде случаев проще в организации. Раньше разработка приложения почти всегда держалась на команде: программисты, дизайнеры, тестировщики, специалисты по инфраструктуре. Теперь часть этих задач берут на себя ИИ-ассистенты и платформы no-code и low-code. Они автоматизируют отдельные этапы работы, особенно там, где задачи можно формализовать. Например, генеративные модели уже умеют превращать текстовое описание вроде «сделать сервис для учёта задач с напоминаниями» в рабочий код. Облачные платформы идут дальше и позволяют собирать мобильные и веб-приложения с минимальным участием ручного программирования. Это снижает порог входа: участвовать в создании приложений могут не только разработчики, но и люди без технической подготовки.

Создание приложений с участием искусственного интеллекта уже нельзя описывать как набор строго последовательных шагов. Раньше всё выглядело довольно линейно: анализ требований, проектирование, кодирование, тестирование, запуск. Сейчас эта схема местами сохраняется, но часть операций берут на себя интеллектуальные системы, и границы между этапами становятся размытыми. Один из самых заметных сдвигов связан с генерацией кода. Языковые модели работают с текстовым описанием задачи и переводят его в программный код — Python, JavaScript, Java и другие языки. Разработчик формулирует задачу обычными словами: например, «сервис учёта задач с напоминаниями и фильтрацией». Система выдаёт заготовку решения, которую можно дорабатывать. Качество таких фрагментов растёт за счёт обучения на открытых репозиториях, где собраны миллионы примеров реального кода. При этом ИИ не сводится к простому «копированию» готовых решений. Он комбинирует разные подходы, иногда довольно неожиданные, и подстраивает их под конкретную задачу. Это ускоряет прототипирование.

Особенно заметно это на ранних стадиях проекта, когда продукт ещё нестабилен и многое меняется буквально на ходу. Архитектура приложений тоже постепенно вовлекается в этот процесс. Системы анализируют требования и предлагают структуру: как разделить клиентскую и серверную части, какие базы данных использовать, как организовать взаимодействие модулей. В сложных распределённых системах это снижает риск ошибок, которые потом дорого исправлять. Отдельный пласт — интерфейсы. UX/UI-дизайн раньше опирался на длительные исследования поведения пользователей. Сейчас модели машинного обучения обрабатывают большие массивы данных о взаимодействии с приложением и выявляют закономерности. Например, какие элементы интерфейса перегружены, а какие почти не используются. На основе этого предлагаются изменения компоновки экрана. Есть и другой сценарий: интерфейс формируется по текстовому описанию. Пользователь описывает приложение, а система предлагает макет — расположение элементов, визуальный стиль, структуру навигации. Входной порог для разработки здесь заметно ниже, потому что часть дизайнерской работы фактически автоматизируется. Тестирование программного обеспечения также меняется. Вместо ручного составления большого количества сценариев ИИ может генерировать тест-кейсы, имитировать поведение пользователей и искать потенциальные сбои. Иногда анализ строится на статистике предыдущих проектов, где уже фиксировались типичные ошибки. Это помогает заранее выделять проблемные участки кода. В отладке ИИ работает с логами приложения. Он находит причины сбоев и предлагает варианты исправлений. В некоторых случаях речь идёт не только о подсказках, но и о прямых правках кода, которые затем проверяются разработчиком. Скорость реакции на ошибки при таком подходе заметно выше. В DevOps-процессах ИИ применяется для управления нагрузкой и ресурсами. Облачные платформы анализируют поведение системы и распределяют вычислительные мощности в зависимости от текущих условий. Если нагрузка растёт, ресурсы перераспределяются заранее, а не после

перегрузки. Внутри самих приложений ИИ активно используется для работы с данными пользователей. Системы фиксируют действия, анализируют поведение и формируют персонализированный контент. В сервисах электронной коммерции это выражается в подборе товаров, в медиа — в рекомендациях фильмов или музыки. Логика проста: чем точнее профиль поведения, тем релевантнее выдача. Параллельно развиваются no-code и low-code платформы. Они позволяют собирать приложения без глубокого погружения в программирование. Пользователь задаёт логику на уровне описания задачи, а система формирует интерфейс, базу данных и базовую функциональность. Для малого бизнеса и стартапов это часто становится рабочим инструментом, а не экспериментом. Ограничения при этом никуда не исчезают. Сгенерированный код может содержать ошибки или неэффективные решения. Иногда он работает, но плохо масштабируется. Поэтому контроль со стороны разработчика остаётся обязательным этапом, даже если большая часть кода создана автоматически. Отдельная проблема связана с безопасностью. Облачные ИИ-сервисы требуют передачи данных, и это создаёт дополнительные риски. Плюс к этому — уязвимости, которые могут появляться в автоматически сгенерированном коде и не всегда очевидны на первый взгляд. Есть и вопрос интерпретации решений моделей. Многие из них работают как «чёрный ящик»: видно вход и выход, но не всегда понятно, почему система пришла именно к такому результату. В прикладной разработке это создаёт сложности, особенно в областях, где цена ошибки высокая. Развитие технологий продолжается без явного замедления. Постепенно формируется подход, при котором ИИ охватывает весь цикл разработки — от идеи до поддержки продукта после релиза. Роль разработчика смещается в сторону контроля, настройки и принятия решений в спорных случаях. ИИ уже встроился в процесс создания приложений и продолжает менять его логику — не резко, но последовательно, слой за слоем. Параллельно меняются подходы к управлению проектами. Раньше заметная часть времени уходила на повторяющиеся операции: подготовку шаблонного

кода, первичную проверку решений, настройку отдельных компонентов. Сейчас значительную долю этой работы выполняют интеллектуальные инструменты. За счёт этого команды быстрее переходят от формулировки идеи к рабочему прототипу и могут раньше начать проверку продукта на практике. Изменяются и профессиональные требования к разработчикам. Само по себе владение языком программирования уже не гарантирует высокой эффективности. Всё большее значение приобретает способность формулировать запросы к ИИ-системам, оценивать качество полученных решений и выявлять ошибки, которые модель не смогла распознать. На первый план выходит не скорость написания кода, а качество инженерных решений и понимание того, как отдельные компоненты будут вести себя в реальной эксплуатации. По этой причине взаимодействие человека и интеллектуальных систем всё чаще рассматривается как совместная работа. Искусственный интеллект генерирует варианты реализации, предлагает исправления и автоматизирует рутинные действия. Окончательное решение остаётся за разработчиком. Именно он определяет архитектуру приложения, оценивает риски, контролирует производительность системы и отвечает за её соответствие поставленным требованиям. Особенно заметны такие изменения в небольших командах. Для создания первой версии продукта теперь нередко требуется меньше ресурсов, чем несколько лет назад. Задачи, которые раньше распределялись между несколькими специалистами, частично автоматизируются. Это сокращает затраты на запуск проекта и позволяет быстрее проверять бизнес-гипотезы, не вкладывая значительные средства на ранних этапах разработки. В крупных компаниях эффект проявляется иначе. Здесь ИИ чаще используют для работы с уже существующими системами, которые развиваются годами и содержат большие объёмы кода. Интеллектуальные инструменты помогают находить повторяющиеся фрагменты, анализировать зависимости между модулями и обнаруживать ошибки ещё до того, как они повлияют на работу приложения. Для сложных

корпоративных систем подобная поддержка становится частью повседневной инженерной практики.

Использование искусственного интеллекта при создании приложений уже заметно меняет саму логику программной инженерии. Он ускоряет разработку, уменьшает объём ручной работы и снижает стоимость отдельных этапов. Но речь не о том, что разработчики исчезают из процесса. Скорее, часть рутинных задач уходит к инструментам, которые работают на базе ИИ, и это меняет распределение ролей в команде. Дальше тенденция, скорее всего, будет развиваться в сторону более сложных генеративных систем. Они уже сейчас умеют собирать фрагменты приложений, а со временем могут доходить до уровня полноценных решений, собранных почти целиком на основе описания задачи. Хотя даже в этом сценарии остаётся слой, который нельзя просто «автоматизировать». Кто-то должен проверять архитектуру, следить за безопасностью, разбираться в том, как всё это работает под нагрузкой, и брать ответственность за результат.

Список литературы:

1. Habr. Без кода и программистов: как ИИ меняет low-code [Электронный ресурс]: https://habr.com/ru/companies/lanit/articles/837224/ (дата обращения: 14.05.2026).
2. Habr. GPT как основа low-code платформ: разработка без программистов [Электронный ресурс]: https://habr.com/ru/articles/965322/ (дата обращения: 14.05.2026).
3. BPMS.ru. Семь платформ low-code с функциями искусственного интеллекта [Электронный ресурс]: https://bpms.ru/post/20231111-low-code-ai/ (дата обращения: 14.05.2026).
4. Yandex Cloud. No-code и low-code разработка без кода и программирования [Электронный ресурс]: https://yandex.cloud/ru/blog/lowcode-nocode (дата обращения: 14.05.2026).
5. ITWeek. Инструменты разработки с помощью ИИ vs low-code/no-code [Электронный ресурс]: https://www.itweek.ru/themes/detail.php?ID=229744 (дата обращения: 14.05.2026).
6. Ailibri. Нейросети для лёгкого создания приложений и сайтов [Электронный ресурс]: https://ailibri.com/low-code-no-code/ (дата обращения: 14.05.2026).