УДК 004.4
DOI: 10.26102/2310-6018/2025.51.4.009

Автоматизация программирования: от первых компиляторов до генеративного искусственного интеллекта

А.М. Бершадский, Ю.И. Евсеева✉, А.А. Гудков

Пензенский государственный университет, Пенза, Российская Федерация

Резюме. Стремительное развитие средств автоматизации программирования является ключевым фактором цифровой трансформации общества. Целью работы является комплексный анализ эволюции инструментов автоматизации, включая языки программирования высокого уровня, структурное и объектно-ориентированное программирование, интегрированные среды разработки, low-code/no-code платформы и большие языковые модели. В исследовании рассматриваются принципы работы генеративного искусственного интеллекта, его возможности и ограничения, а также специфика российских решений в данной области. Особое внимание уделено вызовам, связанным с широким внедрением автоматизации: проблемам интеллектуальной собственности, безопасности сгенерированного кода, трансформации роли программиста и адаптации образовательных программ. Сделан вывод о формировании новой парадигмы совместной работы человека и искусственного интеллекта в разработке программного обеспечения. Практическая значимость работы заключается в предоставлении разработчикам и руководителям структурированной информации для принятия решений о внедрении инструментов автоматизации, выборе технологий и оценке связанных с ними рисков.

Ключевые слова: автоматизация программирования, генеративный искусственный интеллект, большие языковые модели, история программирования, интегрированные среды разработки, low-code/no-code, DevOps, машинное обучение.

Для цитирования: Бершадский А.М., Евсеева Ю.И., Гудков А.А. Автоматизация программирования: от первых компиляторов до генеративного искусственного интеллекта. *Моделирование, оптимизация и информационные технологии.* 2025;13(4). URL: https://moitvivt.ru/ru/journal/pdf?id=2031 DOI: 10.26102/2310-6018/2025.51.4.009

Automation of programming: from the first compilers to generative artificial intelligence

A.M. Bershadsky, Yu.I. Evseeva✉, A.A. Gudkov

Penza State University, Penza, the Russian Federation

Abstract. The rapid development of automation tools for programming is a key factor in the digital transformation of society. The purpose of this work is a comprehensive analysis of the evolution of automation tools, including high-level programming languages, structured and object-oriented programming, integrated development environments, low-code/no-code platforms and large language models. The study examines the principles of operation of generative artificial intelligence, its capabilities and limitations, as well as the specifics of Russian solutions in this area. Particular attention is paid to the challenges associated with the widespread introduction of automation: problems of intellectual property, security of generated code, transformation of the programmer's role and adaptation of educational programs. A conclusion is made about the formation of a new paradigm of joint work of humans and artificial intelligence in software development. The practical significance of the work is to provide developers and managers with structured information for making decisions on the implementation of automation tools, the choice of technologies and the assessment of associated risks.

Keywords: programming automation, generative artificial intelligence, large language models, history of programming, integrated development environments, low-code/no-code, DevOps, machine learning.

For citation: Bershadsky A.M., Evseeva Yu.I., Gudkov A.A. Automation of programming: from the first compilers to generative artificial intelligence. *Modeling, Optimization and Information Technology*. 2025;13(4). (In Russ.). URL: https://moitvivt.ru/ru/journal/pdf?id=2031 DOI: 10.26102/2310-6018/2025.51.4.009

Введение

Программное обеспечение сегодня пронизывает все сферы жизни – от смартфонов до космических технологий. За каждым цифровым решением стоит труд программистов, который с годами усложняется, требуя новых подходов к организации работы. Автоматизация программирования – создание инструментов, позволяющих делегировать компьютерам рутинные и сложные аспекты разработки программного обеспечения – становится ключевым фактором цифровой эволюции.

Разработка программных систем традиционно требует значительных интеллектуальных и временных ресурсов, что актуализирует задачи математического моделирования и оптимизации процессов создания программного обеспечения. В условиях цифровой экономики, где программные решения определяют конкурентоспособность предприятий и скорость внедрения инноваций, оптимизация процессов разработки становится критически важным фактором экономического развития.

Автоматизация программирования представляет собой комплекс технологий для сокращения ручного труда при создании, тестировании и поддержке программных систем. Это концепция, направленная на повышение продуктивности, качества кода и скорости разработки. От первых компиляторов до современных генеративных моделей искусственного интеллекта (ИИ) – эволюция средств автоматизации отражает стремление к повышению эффективности интеллектуального труда.

История разработки программного обеспечения прошла несколько ключевых этапов: программирование на машинном языке, появление первых языков высокого уровня (Fortran, COBOL) в 1950-х годах, структурное программирование 1970-х и объектно-ориентированный подход 1980-х–1990-х годов, современные интегрированные среды разработки и методологии (Agile, DevOps). Сегодня мы входим в эру, где искусственный интеллект становится партнером разработчика, способным генерировать код и предвидеть потенциальные проблемы.

Цель исследования – проследить историю развития средств автоматизации программирования, проанализировать современное состояние технологий и обозначить перспективные направления их эволюции. Рассмотрим технические аспекты автоматизации, ее влияние на профессию программиста и общество в целом, а также уделим внимание существующим ограничениям и путям их преодоления.

Данная статья может быть полезна разработчикам программного обеспечения, специалистам в области информационных технологий, студентам, руководителям проектов и менеджерам по продукту, которые интересуются развитием и текущим состоянием автоматизации в создании программного обеспечения. Рассмотренные в статье вопросы помогут им оценить, как автоматизация влияет на скорость разработки программного обеспечения, надежность программного кода, на сам процесс разработки, а также понять сопутствующие проблемы, такие как безопасность кода, этические вопросы и необходимость адаптации системы образования.

Эволюция средств разработки программного обеспечения

Эра языков высокого уровня (1950-е годы). В 1950-х годах произошел первый значительный прорыв в разработке программного обеспечения – появление языков

программирования высокого уровня. До этого программисты были вынуждены работать непосредственно с машинным кодом или ассемблером, что требовало глубокого понимания архитектуры конкретных компьютеров и делало процесс разработки чрезвычайно трудоемким.

Появление первых языков высокого уровня стало революционным изменением в истории программирования, кардинально упростив процесс создания программ. Fortran (FORmula TRANslation), разработанный в 1957 году командой IBM под руководством Джона Бэкуса, позволил ученым и инженерам выражать математические формулы и алгоритмы на языке, близком к естественному математическому обозначению, что значительно ускорило разработку научных программ. COBOL (COmmon Business-Oriented Language), появившийся в 1959 году, был создан специально для бизнес-приложений и обработки данных, что существенно расширило область применения компьютеров в коммерческом секторе и открыло эру автоматизации бизнес-процессов. ALGOL (ALGOrithmic Language) стал важной вехой в развитии языков программирования, введя блочную структуру и другие фундаментальные концепции, которые повлияли на архитектуру многих последующих языков и заложили основы современного программирования.

Первые компиляторы трансформировали процесс программирования, преобразуя код высокого уровня в машинные инструкции. Компилятор для Fortran, созданный командой Бэкуса, стал первым эффективным компилятором, генерирующим машинный код, сравнимый по производительности с рукописным ассемблером. Появление компиляторов позволило значительно повысить производительность программистов и абстрагироваться от особенностей конкретных машин.

Изобретение Fortran и других языков высокого уровня стало революционным скачком в автоматизации программирования, сменив сложное и подверженное ошибкам кодирование на ассемблере. Компилятор автоматизировал критически важные операции: распределение памяти, генерацию машинного кода и сложную оптимизацию (вычисление выражений, работу с регистрами). Качественный скачок выразился в значительном росте производительности труда программистов, повышении надежности и эффективности кода, а также в «демократизации» программирования – язык стал доступен ученым и инженерам, не обладающим глубокими познаниями в аппаратном устройстве конкретной ЭВМ. Fortran и другие инновационные языки программирования того времени перевели фокус с управления машиной на формулировку алгоритмов.

Тем не менее, процесс разработки оставался трудоемким: код писался на перфокартах, компилировался в пакетном режиме, а отладка осуществлялась через анализ дампов памяти при сбоях программы.

Структурное программирование (1970-е годы). В 1970-е годы индустрия программирования столкнулась с «кризисом программного обеспечения» – проекты становились все сложнее, сроки и бюджеты не соблюдались, качество программ оставляло желать лучшего. Концепции и методологии структурного программирования, предложенные такими учеными как Эдсгер Дейкстра, Дональд Кнут и Тони Хоар, стали ответом на вызовы неструктурированного кода того времени. Дейкстра в своей знаменитой статье «Go To Statement Considered Harmful» (1968) положил начало движению против бессистемного использования операторов перехода (GOTO), что привело к появлению принципов структурного программирования, включающих использование логических конструкций (последовательность, выбор, повторение) вместо произвольных переходов. Концепции модульности и пошаговой детализации (top-down design) кардинально изменили подход к проектированию программ, делая их более понятными и управляемыми. Язык Pascal, разработанный Никлаусом Виртом в 1970 году, был создан специально для обучения структурному программированию и

быстро приобрел популярность в академических кругах, став практическим воплощением этих теоретических принципов.

Появление интегрированных сред разработки (Integrated Development Environment, IDE) и первых средств отладки значительно облегчило процесс разработки. Первые интегрированные среды разработки, такие как Maestro I для JOVIAL (начало 1970-х), начали объединять редактор кода, компилятор и другие инструменты. Turbo Pascal от Borland стал одной из первых популярных IDE, предлагающих быструю компиляцию и интегрированные средства разработки.

Структурное программирование стало ответом на кризис программного обеспечения 1960-х годов, вызванный неконтролируемой сложностью и ненадежностью кода из-за злоупотребления оператором goto. Его суть – в строгом ограничении управления потоком команд тремя базовыми конструкциями: последовательностью, ветвлением и циклом, что обеспечило предсказуемость логики. Ключевой принцип – нисходящее проектирование («сверху вниз») с разбиением на модули (функции) и локальными переменными. Это существенно повысило читаемость кода, упростило отладку и тестирование. Структурное программирование позволило масштабировать разработку, заложив основы модульности для объектно-ориентированного программирования и превратив создание программ из искусства в инженерную дисциплину.

Объектно-ориентированное программирование (1980–1990-е годы). 1980-е и 1990-е годы ознаменовались переходом к объектно-ориентированной парадигме программирования, которая предлагала более естественный способ моделирования реального мира в программах.

Языки объектно-ориентированного программирования стали доминировать в индустрии благодаря нескольким ключевым разработкам. Smalltalk, разработанный в исследовательском центре Xerox PARC, был первым полностью объектно-ориентированным языком и средой программирования, представив фундаментальные концепции объектов, классов и сообщений. C++, созданный Бьярном Страуструпом в 1983 году как расширение языка C, объединил эффективность C с концепциями объектно-ориентированного подхода, став одним из самых влиятельных языков программирования в истории. Java, выпущенный компанией Sun Microsystems в 1995 году, предложил революционную кроссплатформенную среду выполнения на основе виртуальной машины (Java Virtual Machine) и быстро стал стандартом для корпоративных приложений, окончательно утвердив объектно-ориентированный подход как основную парадигму современного программирования.

Объектно-ориентированное программирование существенно изменило процесс разработки, позволив преодолеть ограничения процедурных подходов за счет моделирования реального мира как системы взаимодействующих сущностей — объектов. Его ядро составляют четыре принципа: инкапсуляция (объединение данных и методов в классе с защитой внутреннего состояния), наследование (создание иерархий для повторного использования кода), полиморфизм (единый интерфейс для разных реализаций) и абстракция (сокрытие сложности за упрощенными моделями). Это позволило справиться со сложностью разработки больших систем, повысить эффективность разработки за счет повторного использования компонентов и повысить надежность благодаря изоляции данных. Объектно-ориентированное программирование трансформировало программирование из написания алгоритмов в конструирование взаимодействующих объектов.

CASE-средства (Computer-Aided Software Engineering) и визуальное программирование расширили арсенал инструментов разработчика. CASE-инструменты, такие как Rational Rose, позволили автоматизировать аспекты

проектирования и разработки программного обеспечения. Язык моделирования UML (Unified Modeling Language), стандартизированный в 1997 году, стал универсальным языком для визуализации, спецификации и документирования программных систем.

Зрелость интегрированных сред разработки (конец 1990-х – начало 2000-х).

К концу 1990-х и началу 2000-х годов интегрированные среды разработки достигли зрелости, предлагая беспрецедентные возможности для повышения производительности программистов.

Ведущие IDE этого периода установили новые стандарты разработки, кардинально изменив подход к созданию программного обеспечения. Visual Studio от Microsoft, впервые выпущенная в 1997 году, объединила различные инструменты Microsoft для разработки программного обеспечения в единую мощную среду, поддерживающую множество языков программирования и ставшую стандартом для разработки под платформы Windows. Eclipse, появившаяся в 2001 году с открытым исходным кодом, быстро стала популярной платформой для разработки на Java и других языках благодаря своей расширяемой архитектуре на основе плагинов, что позволило создать вокруг нее целую экосистему инструментов. IntelliJ IDEA, выпущенная в том же году, стала революционной разработкой в отношении интеллектуальных функций IDE, предлагая высококачественную контекстно-зависимую помощь разработчикам.

Автоматизация рутинных задач стала ключевым направлением развития сред разработки. Интеллектуальное автодополнение кода, автоматический рефакторинг, инструменты статического анализа, встроенная документация и автоматизированное тестирование стали неотъемлемой частью процесса разработки.

Эти инновации значительно повысили продуктивность: автоматизация рутинных задач (сборка, отладка) сократила время разработки, а встроенные инструменты анализа кода снизили количество ошибок. Однако рост сложности IDE требует от разработчиков больше времени на их освоение.

Современные подходы и технологии автоматизации

ИИ-помощники в программировании. Современные виртуальные помощники программиста значительно повышают продуктивность разработчиков. Например, GitHub Copilot, использующий в качестве основного компонента интеллектуального агента OpenAI Codex, анализирует контекст файлов с исходным кодом и генерирует предложения кода. Особенно эффективно инструмент справляется с написанием стандартных функций, API-вызовов и шаблонного кода. Tabnine использует глубокое обучение для предсказания и предложения следующих строк кода, адаптируясь к индивидуальному стилю программирования разработчика.

В России также развиваются собственные решения в области ИИ-помощников для программирования: SourceCraft Code Assistant от Яндекса предлагает автодополнение и генерацию кода с учетом специфики российского рынка разработки, а GigaCode от Сбербанка использует технологии машинного обучения для анализа корпоративного кода и предоставления контекстных предложений, адаптированных под внутренние стандарты разработки крупных российских компаний.

Современные ИИ-помощники совершили рывок в программировании, трансформируя разработку в коллаборацию человека и нейросети. Они анализируют контекст всего проекта, генерируют код по описанию, автоматически исправляют ошибки и предлагают оптимизацию. При этом нейросети не заменяют инженерное мышление – они усиливают его, позволяя разработчику фокусироваться на более сложных проблемах. Однако их внедрение порождает новые вызовы, в частности,

повышенный риск уязвимостей в сгенерированном коде. Использование ИИ-помощников требует критической проверки предлагаемых ими решений.

Low-code и No-code платформы. Low-code и No-code решения упрощают разработку программного обеспечения, делая создание приложений доступным для широкого круга пользователей без глубоких технических знаний. Например, с помощью Microsoft Power Apps можно создавать бизнес-приложения, используя визуальный интерфейс с минимальным программированием и интеграцией с экосистемой Microsoft Office и облачными сервисами. Инструменты Bubble и Webflow стали революцией в веб-разработке. Bubble позволяет создавать полнофункциональные веб-приложения с базами данных без написания кода, а Webflow генерирует на основе визуального дизайна код на HTML, CSS и JavaScript.

В России активно развиваются собственные low-code платформы: «1С:Предприятие» традиционно предоставляет визуальные инструменты для создания бизнес-приложений, Comindware Platform от российской компании Comindware специализируется на автоматизации рабочих процессов и управлении документооборотом, а платформа «Мегаплан» позволяет создавать CRM-системы и инструменты управления проектами через визуальный конструктор, адаптированный под специфику российского бизнеса.

Low-code/no-code платформы демократизировали разработку, позволив бизнес-пользователям без навыков программирования создавать приложения через визуальные интерфейсы. Это сократило время разработки для типовых решений, например, создание отчетов или управление взаимоотношениями с клиентами. К недостаткам такого подхода можно отнести ограниченную гибкость при реализации сложных алгоритмов, а также проблемы безопасности при недостаточном контроле. Возможным прогнозом на будущее является применение гибридных моделей, где low-code служит для быстрого прототипирования, а ИИ-генерация кода восполняет пробелы в гибкости.

DevOps и автоматизация развертывания. DevOps – это методология и культура разработки программного обеспечения, которая объединяет команды разработки (Development) и эксплуатации (Operations) для ускорения и улучшения процесса создания, тестирования и развертывания приложений. Основные принципы DevOps включают автоматизацию процессов, непрерывную интеграцию и доставку (CI/CD – Continuous Integration/Continuous Deployment), мониторинг и быструю обратную связь. Цель – сократить время от написания кода до его попадания в эксплуатацию, при этом повысив качество и надежность программного обеспечения.

Современные инструменты CI/CD представляют собой автоматизированные системы, которые значительно ускоряют цикл разработки за счет автоматического тестирования, сборки и развертывания кода при каждом изменении в репозитории. Jenkins остается одним из самых популярных решений с открытым исходным кодом для непрерывной интеграции и доставки, предоставляя гибкую систему плагинов для настройки процессов разработки. GitHub Actions и GitLab CI тесно интегрированы с соответствующими платформами управления кодом, позволяя разработчикам настраивать автоматизацию прямо в репозитории через YAML-файлы. Концепция «Инфраструктура как код» (IaC) стала стандартом в DevOps-практиках, позволяя управлять серверами и облачными ресурсами через код вместо ручной настройки. Terraform позволяет декларативно описывать инфраструктуру для различных облачных провайдеров, обеспечивая воспроизводимость и версионирование инфраструктуры, а Ansible автоматизирует конфигурацию и управление серверами через простые файлы конфигурации.

В России активно развиваются собственные DevOps-решения: платформа «Крок» предоставляет CI/CD инструменты для российских компаний с возможностью

развертывания в отечественных дата-центрах, система «GitVerse» от VK объединяет управление кодом с инструментами непрерывной интеграции, а решение «Selectel CI/CD» позволяет автоматизировать развертывание приложений в российской облачной инфраструктуре с соблюдением требований по локализации данных.

Автоматизация тестирования и анализа качества. Современное тестирование программного обеспечения включает несколько уровней проверки качества кода. Модульное тестирование проверяет отдельные компоненты в изоляции, интеграционное тестирование проверяет взаимодействие между модулями, а end-to-end тестирование имитирует полные пользовательские сценарии. Для этих целей используются различные фреймворки: Jest для JavaScript-приложений, pytest для Python и JUnit для Java. Автоматизированное тестирование пользовательского интерфейса выполняется с помощью таких инструментов, как Selenium или Playwright, позволяющих имитировать действия пользователя в браузере.

Технология статического анализ кода позволяет выявлять потенциальные проблемы без запуска программы. Например, Pylint анализирует Python-код на соответствие стандартам. Подобные системы помогают обнаруживать ошибки и уязвимости безопасности на раннем этапе разработки.

Из отечественных решений следует отметить разработку PVS-Studio, которая предоставляет статический анализ для C, C++, C# и Java, выявляя серьезные ошибки, включая переполнения буферов и утечки памяти. Svace от ИСП РАН находит критические ошибки в коде и используется крупными российскими IT-компаниями. Российские технологические гиганты, такие как Яндекс, Mail.ru Group и Kaspersky, разработали собственные внутренние инструменты тестирования, некоторые из которых стали доступны как коммерческие продукты или open-source решения.

Автоматизация тестирования и анализа качества трансформировала контроль качества из ручных проверок в непрерывный интеллектуальный процесс. Внедрение инструментов статического анализа и автоматизированного тестирования в CI/CD-конвейеры позволило выявлять больше дефектов на этапе разработки, сократив время выхода новых версий приложений и предотвращая критические сбои в них. Однако сохраняются вызовы: высокая стоимость поддержки автоматического тестирования, риск ложноположительных срабатываний статического анализа и сложность автоматизации исследовательского тестирования.

Генеративные модели искусственного интеллекта в программировании

Обзор технологии и существующие решения. Генеративный искусственный интеллект – это класс моделей машинного обучения, которые способны создавать новые данные (текст, изображения, музыку, код, видео), статистически схожие с данными из обучающей выборки. В отличие от дискриминативных моделей, которые только распознают и классифицируют существующую информацию, генеративные модели учатся аппроксимировать распределение вероятностей обучающих данных, чтобы затем генерировать из этого распределения новые данные. Например, языковая модель, обученная на текстовом корпусе, может генерировать связный текст; модель, обученная на изображениях лиц, может синтезировать фотореалистичный портрет несуществующего человека; а модель, обученная на музыкальных произведениях, может создать новую композицию в заданном стиле. Ключевая особенность — эти системы не воспроизводят существующие примеры из обучающей выборки, а генерируют

принципиально новые объекты, комбинируя изученные статистические закономерности способами, которых не было в исходных данных.

Революционная архитектура нейросетей-трансформеров, представляющих собой один из видов генеративных моделей, была представлена компанией Google в 2017 году в работе «Attention Is All You Need» [1] и кардинальным образом изменила обработку естественного языка. Трансформеры отличаются от предыдущих архитектур тем, что используют механизм внимания (attention) для прямого установления связей между любыми элементами входной последовательности символов. Вместо последовательной обработки слов (как в рекуррентных нейронных сетях), трансформер «видит» все слова одновременно и для каждого слова вычисляет веса важности всех остальных слов в контексте. Это позволяет модели эффективно улавливать длинные зависимости (например, связывать местоимение с существительным через десятки слов), работать параллельно на всех позициях (что ускоряет обучение) и достигать лучшего понимания контекста.

Существуют специализированные модели нейросетей-трансформеров для решения задач разработки программного обеспечения. AlphaCode от DeepMind [2] фокусируется на решении алгоритмических задач соревновательного программирования, CodeGen от Salesforce [3] представляет семейство моделей с открытым исходным кодом, а Codex от OpenAI стал основой GitHub Copilot [4]. CodeLlama от Meta, StarCoder от Hugging Face и CodeT5 от Salesforce предлагают уникальные особенности и специализацию для различных аспектов программирования [5].

GPT-4 и ChatGPT стали прорывом в автоматизации программирования, поддерживая более 40 языков программирования и способные генерировать код на основе описания функциональности, отлаживать существующий код и объяснять сложные алгоритмы [6]. Claude от Anthropic выделяется способностью обрабатывать особенно длинные контекстные окна (более 200 тыс. токенов), что позволяет анализировать и модифицировать масштабные программные проекты с пониманием архитектурных паттернов [7]. Gemini от Google имеет глубокую интеграцию в экосистему разработки Google с прямым доступом к поисковой системе и облачным сервисам [8].

В России активно развиваются собственные языковые модели для программирования. Yandex GPT и GigaChat от Сбербанка адаптированы для работы с российскими стандартами разработки и поддерживают специфические требования отечественного информационно-технологического сектора. Модель ruGPT-3 от Сбербанка и языковые модели от МТС демонстрируют способности в генерации и анализе кода, особенно эффективно работая с документацией, составленной на русском языке.

Практические применения генеративных моделей в разработке. Современные генеративные технологии активно внедряются в процессы создания программного обеспечения. Эти инструменты помогают разработчикам быстрее создавать рабочие прототипы, преобразуя текстовые описания задач в код приложений. Искусственный интеллект сегодня способен справляться с программистскими задачами разного уровня сложности — от базовых учебных примеров до комплексных алгоритмических решений. Благодаря этому такие системы становятся полезными как для обучения программированию, так и для решения практических задач в работе.

Особенно заметны достижения ИИ в области переноса программного кода между разными платформами и языками программирования. Автоматизация таких процессов позволяет командам разработчиков экономить значительные ресурсы при смене технологического стека. Нейросети также демонстрируют хорошие результаты при

работе с системами управления данными – они могут создавать эффективные запросы к базам данных и улучшать производительность существующих решений.

Анализ программного кода и создание документации представляют еще одну перспективную сферу использования ИИ-помощников. Такие системы могут исследовать большие проекты, находить типовые решения в архитектуре, составлять подробные описания функций и модулей, а также автоматически формировать техническую документацию. Эти возможности особенно востребованы при сопровождении действующих программных продуктов и адаптации новых сотрудников к проектам, поскольку ИИ может оперативно разъяснить логику работы и связи между различными компонентами системы.

Проблемы и вызовы автоматизации программирования

Современные инструменты автоматизации программирования обучаются на огромных массивах существующего кода, следствием чего является возникновение определенных этических и юридических вопросов. Проблемы интеллектуальной собственности приобретают актуальность, когда системы искусственного интеллекта генерируют код, существенно напоминающий исходные обучающие примеры.

Сгенерированный языковой моделью код часто содержит скрытые проблемы, которые могут остаться незамеченными при поверхностном обзоре. Исследования показывают, что даже наиболее продвинутые системы могут генерировать решения, содержащие логические ошибки, неоптимальные алгоритмы или уязвимости безопасности [9]. Автоматизация многократно усиливает влияние существующих проблем программирования. Создается риск системных уязвимостей, когда одна и та же ошибка внедряется в тысячи независимых систем [10].

Усиливающаяся автоматизация ставит новые серьезные вопросы перед системой образования в сфере информационных технологий. Традиционный подход, в основе которого лежит овладение синтаксисом языков программирования, может потерять актуальность. Новое поколение разработчиков рискует сформировать поверхностное понимание программирования, полагаясь на автоматические решения без глубокого понимания принципов их работы. Создается опасность «иллюзии компетентности».

Тенденция к чрезмерному доверию результатам работы автоматизированных систем также представляет собой проблемы. Психологические исследования выявили феномен «автоматизированного уклона», когда люди склонны считать компьютерные решения более точными и надежными [9].

Нахождение баланса между автоматизацией и человеческим контролем требует тщательного анализа конкретных контекстов разработки. Для критических систем необходим более консервативный подход с многоуровневой верификацией.

Перспективы развития и будущие тенденции

Ожидается дальнейшее развитие генеративных моделей с улучшенными возможностями понимания контекста и генерации более качественного кода. Интеграция различных типов ИИ (текстовых, визуальных, аудио) может привести к созданию мультимодальных систем разработки.

Профессия программиста претерпевает фундаментальные изменения. Возникает потребность в новых ролях: специалисты по верификации автоматически сгенерированного кода, эксперты по интеграции человеческого и искусственного интеллекта, инженеры по разработке естественно-языковых инструкций для ИИ – «промптов».

Образовательная система должна адаптироваться, смещая акцент с написания кода на формирование критического мышления, системного проектирования и понимания ограничений автоматизированных систем.

Формируется новая парадигма программирования, где человеческий интеллект и автоматизированные системы выступают как взаимодополняющие компоненты. Развиваются коллаборативные модели, требующие новых инструментов визуализации и верификации.

Ожидается появление более специализированных инструментов автоматизации для конкретных доменов: медицины, финансов, науки о данных, что повысит эффективность разработки в этих областях.

Заключение

Автоматизация программирования прошла длительный путь от первых компиляторов 1950-х годов до современных генеративных систем искусственного интеллекта. Каждый этап эволюции приносил революционные изменения в способы создания программного обеспечения, существенно повышая продуктивность разработчиков и снижая барьеры входа в профессию.

Современные крупные языковые модели представляют собой качественно новый этап автоматизации, способный не только генерировать код, но и анализировать, оптимизировать и документировать программные решения. Однако вместе с беспрецедентными возможностями эти технологии привносят серьезные вызовы: этические и юридические вопросы, проблемы качества и безопасности, необходимость переосмысления образовательных подходов.

Рассмотренные в статье вопросы могут оказаться полезными для разработчиков программного обеспечения при оценке влияния автоматизации на продуктивность, качество кода и процессы разработки, а также при принятии решений о внедрении инструментов автоматизации и подготовке к будущим изменениям в индустрии.

Ключевым выводом исследования является то, что будущее автоматизации программирования лежит не в полной замене человеческого труда машинным, а в создании эффективных коллаборативных систем, где каждый компонент — человеческий интеллект и искусственный интеллект – вносят уникальный вклад в процесс разработки.

Для успешной адаптации к новой реальности индустрия должна решить ряд критических задач: разработать надежные методы верификации автоматически генерируемого кода, создать эффективные образовательные программы для новых компетенций, выработать этические стандарты использования искусственного интеллекта в разработке и найти оптимальный баланс между автоматизацией и человеческим контролем.

Автоматизация программирования продолжит трансформировать не только индустрию разработки программного обеспечения, но и – более широко – способы человеческого взаимодействия с цифровыми технологиями. Понимание этих процессов становится критически важным для всех участников цифровой экономики.

СПИСОК ИСТОЧНИКОВ / REFERENCES

1. Vaswani A., Shazeer N., Parmar N., et al. Attention Is All You Need. In: *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems (NIPS 2017), 04–09 December 2017, Long Beach, CA, USA*. 2017. P. 5998–6008.

2. Li Yu., Choi D., Chung Ju., et al. Competition-Level Code Generation with AlphaCode. *Science*. 2022;378(6624):1092–1097. https://doi.org/10.1126/science.abq1158
3. Nijkamp E., Pang B., Hayashi H., et al. CodeGen: An Open Large Language Model for Code with Multi-Turn Program Synthesis. arXiv. URL: https://doi.org/10.48550/arXiv.2203.13474 [Accessed 14th July 2025].
4. Chen M., Tworek J., Jun H., et al. Evaluating Large Language Models Trained on Code. arXiv. URL: https://doi.org/10.48550/arXiv.2107.03374 [Accessed 14th July 2025].
5. Rozière B., Gehring J., Gloeckle F., et al. Code Llama: Open Foundation Models for Code. arXiv. URL: https://doi.org/10.48550/arXiv.2308.12950 [Accessed 14th July 2025].
6. Moussiades L., Zografos G. OpenAI's GPT4 as Coding Assistant. arXiv. URL: https://doi.org/10.48550/arXiv.2309.12732 [Accessed 6th August 2025].
7. Peng Yu., Wan Ju., Li Yi., Ren X. COFFE: A Code Efficiency Benchmark for Code Generation. arXiv. URL: https://doi.org/10.48550/arXiv.2502.02827 [Accessed 14th July 2025].
8. Anil R., Borgeaud S., Alayrac J.-B., et al. Gemini: A Family of Highly Capable Multimodal Models. arXiv. URL: https://doi.org/10.48550/arXiv.2312.11805 [Accessed 14th July 2025].
9. Torka S., Albayrak S. Optimizing AI-Assisted Code Generation. arXiv. URL: https://doi.org/10.48550/arXiv.2412.10953 [Accessed 14th July 2025].
10. Daniotti S., Wachs J., Feng X., Neffke F. Who Is Using AI to Code? Global Diffusion and Impact of Generative AI. arXiv. URL: https://doi.org/10.48550/arXiv.2506.08945 [Accessed 14th July 2025].

ИНФОРМАЦИЯ ОБ АВТОРАХ / INFORMATION ABOUT THE AUTHORS

Бершадский Александр Моисеевич, доктор технических наук, профессор, профессор кафедры «Системы автоматизированного проектирования», Пензенский государственный университет, Российская Федерация.
e-mail: bam@pnzgu.ru

Alexander M. Bershadsky, Doctor of Engineering Sciences, Professor, Professor at the Department of Computer-Aided Design Systems, Penza State University, the Russian Federation.

Евсеева Юлия Игоревна, кандидат технических наук, доцент, доцент кафедры «Системы автоматизированного проектирования», Пензенский государственный университет, Российская Федерация.
e-mail: shymoda@mail.ru

Yulia I. Evseeva, Candidate of Engineering Sciences, Docent, Associate Professor at the Department of Computer-Aided Design Systems, Penza State University, the Russian Federation.

Гудков Алексей Анатольевич, кандидат технических наук, доцент, доцент кафедры «Системы автоматизированного проектирования», Пензенский государственный университет, Российская Федерация.
e-mail: alexei-ag@yandex.ru

Alexei A. Gudkov, Candidate of Engineering Sciences, Docent, Associate Professor at the Department of Computer-Aided Design Systems, Penza State University, the Russian Federation.

Статья поступила в редакцию 17.07.2025; одобрена после рецензирования 11.09.2025; принята к публикации 29.09.2025.

The article was submitted 17.07.2025; approved after reviewing 11.09.2025; accepted for publication 29.09.2025.