axenix

Скрытая цена ИИ-революции

Производственный процесс создания программного обеспечения в эпоху искусственного интеллекта

Андрей Толстов
Максим Ким

Москва, май 2026

Искусственный интеллект радикально изменил производственные процессы создания программного обеспечения, но за впечатляющими цифрами роста продуктивности скрывается парадокс: ИИ ускоряет генерацию кода в 2–3 раза, но одновременно замедляет верификацию на 90%[1,2]. Узкое место перемещается с написания кода на его проверку.

Одновременно наблюдается системный мировой кадровый кризис, который большинство компаний ещё не осознало. Занятость начинающих программистов в США упала на 20% за три года (2022–2025) [3], найм начинающих разработчиков сократился на 25%[4]. Компании экономят, автоматизируя рутину и сокращая найм начинающих ИТ-специалистов, но одновременно уничтожают тот слой работы, через который они учились ремеслу.

Эта проблема имеет не линейный характер — это каскад. Не нанимая начинающих, мы гарантируем отсутствие средних разработчиков в 2027-м, старших — в 2031-м, архитекторов — в 2036-м. Проблема, созданная решениями 2025–2026 годов, проявится катастрофой в 2032–2036 годах, когда поколение X выйдет на пенсию, а заменить их будет некем.

Данное исследование посвящено анализу механизма кризиса, его масштаба, степени влияния на ключевые бизнес-метрики и выявление путей решения назревающей проблемы для всего технологического сообщества.

Ключевой вывод исследования — проблема кроется не в ИИ-инструментарии как таковом, а в том, как компании трансформируют производственные процессы создания программного обеспечения и ролевую модель продуктовых команд.

Содержание

Проблема, которую мы создаём сегодня	**4**
Производственный процесс создания ПО: где ИИ помогает и где возникает проблема	**5**
Механизм проблемы: от автоматизации к кризису	**6**
Влияние ИИ на ключевые бизнес-метрики	**9**
Возможные пути решения проблемы	**12**
Заключение: три сценария будущего	**16**
Источники и литература	18

Проблема, которую мы создаём сегодня

Сегодня компании по всему миру вкладывают в ИИ-инструменты огромные средства. Цифры впечатляют: 84% разработчиков используют ИИ[5], 41% кода генерируется искусственным интеллектом[5]. Продуктивность разработки удваивается[6], но ожидаемого эффекта нет — время вывода продукта на рынок не сокращается, а со временем скорость может даже снижаться. Причина заключается в том, что ИИ ускоряет генерацию кода в 2–3 раза, но одновременно замедляет его верификацию. Узкое место перемещается с написания кода на его проверку. Старшие разработчики перегружены и вместо архитектурного проектирования тратят 70% времени на проверку кода, сгенерированного ИИ[9].

Статистика показывает, что в настоящий момент происходит структурное изменение мирового рынка труда в области ИТ. Занятость начинающих программистов (22–25 лет) в США упала на 20% за три года (2022–2025)[3]. Вакансий для начинающих разработчиков стало на 25% меньше[4]. Безработица выпускников компьютерных специальностей достигла 6,1% против 4,3% общей безработицы[7].

Автоматизируя рутину при помощи ИИ, мы уничтожаем тот слой работы, через который начинающие разработчики учились ремеслу. Не нанимая начинающих сегодня, мы гарантируем отсутствие архитекторов через десять лет. А это означает, что через 5–10 лет скорость разработки упадёт катастрофически — контролировать работу ИИ и проверять результаты его работы будет просто некому.

Как следствие технический долг будет стремительно накапливаться. Исследование GitClear, проанализировавшее 211 миллионов изменённых строк кода, показывает, что дублирование кода увеличилось в 4 раза, доля рефакторинга упала с 25% в 2021 году до менее 10% в 2024 году[8].

В рамках данного исследования мы выявим оптимальные способы решения назревающей проблемы и выработаем баланс между краткосрочной оптимизацией квартальных показателей и долгосрочной устойчивостью ИТ-ландшафта.

Производственный процесс создания ПО: где ИИ помогает и где возникает проблема

Полный производственный процесс создания программного обеспечения включает множество взаимосвязанных этапов, каждый из которых критически важен для конечного результата. Сначала идёт сбор и анализ требований — понимание того, что именно нужно создать. Затем проектирование — разработка архитектуры будущей системы, выбор технологий, описание компонентов и их взаимодействия. Следующий этап — непосредственно разработка, написание кода, который реализует спроектированное решение. После этого тестирование — проверка того, что созданное работает корректно и соответствует требованиям. Затем развёртывание — перенос готового решения в промышленную эксплуатацию. И наконец, эксплуатация и поддержка — обеспечение стабильной работы системы, исправление ошибок, добавление новых возможностей.

Искусственный интеллект действительно даёт впечатляющее повышение эффективности на ряде этапов. ИИ-инструменты помогают обработать накопленные корпоративные знания, проанализировать существующий код, выявить паттерны — работа, которая раньше занимала неделю, теперь выполняется за день. Автоматическая генерация технической документации, описаний API, комментариев к коду экономит недели усилий команды. Создание тест-кейсов и тестирование: ИИ автоматически генерирует сценарии тестирования, покрытие тестами вырастает с ~70% до ~85%. Непосредственно генерация кода ускоряется в 2–3 раза[2]. На этих этапах эффект очевиден.

При драматическом ускорении отдельных этапов процесса общая продуктивность и Time to market — время от идеи до релиза продукта — может не расти, а в ряде случаев даже снижаться. Это означает, что существует этап, где происходит замедление, которое полностью «съедает» весь выигрыш от ускорения других этапов.

Этот проблемный этап — верификация и проверка качества кода. Дело в том, что ИИ-код требует почти в два раза больше времени на проверку. Старшие разработчики перегружены верификацией автоматически сгенерированного кода вместо архитектурного проектирования. Узкое место переместилось с написания кода на его проверку, и это имеет далеко идущие последствия для всей ИТ-индустрии.

Именно на этом проблемном этапе сконцентрирован основной кадровый кризис. Проверка ИИ-кода требует экспертизы старших специалистов, которых катастрофически не хватает. Одновременно автоматизация рутинного программирования уничтожает точки входа для начинающих специалистов — те самые задачи низкой сложности, на которых они учились ремеслу и постепенно вырастали до уровня экспертов. Это разрывает конвейер роста от начинающего разработчика к архитектору — путь, который в среднем занимал 10–15 лет непрерывного практического опыта традиционной «ручной» разработки.

Механизм проблемы: от автоматизации к кризису

Проблема имеет чёткую причинно-следственную цепочку, где каждый этап запускает следующий. Понимание этого механизма критически важно, потому что эта проблема имеет не линейный характер — это каскад взаимосвязанных эффектов.

Пять шагов к катастрофе

2023 – 2024

Автоматизация рутины

ИИ автоматизирует задачи низкой сложности — написание типовых функций, создание тестов, документирование кода. Именно эти задачи традиционно выполняли начинающие разработчики. Когда GitHub Copilot генерирует функцию валидации за секунды, когда ChatGPT пишет тесты по описанию, когда ИИ автоматически создаёт документацию API — рабочих мест для начинающих становится меньше.

2024 – 2025

Экономическая оптимизация

Компании видят простую арифметику: старший разработчик с ИИ-ассистентом выполняет работу 2-3 начинающих. Экономия составляет 2–2,5 млн руб. в год на каждом не нанятом специалисте. Принимается очевидное решение: не нанимаем начинающих, инвестируем в ИИ-инструменты для опытных разработчиков. Это рациональное решение с точки зрения квартальных показателей. Проблема в том, что последствия проявятся не сразу.

2026 – 2027

Разрыв обучающего конвейера

Начинающие не нанимаются → не получают опыта → не вырастают в средних разработчиков. Через три года компании начнут искать специалистов с 3-5 годами опыта для усиления команд. Их нет. Тех, кого должны были нанять начинающими в 2023–2024 годах, не наняли. Они не получили опыт. Конвейер кадров разорван.

2028 – 2031

Дефицит старших специалистов

Средних разработчиков нет → некому расти в старшие. Существующие старшие разработчики становятся дефицитным ресурсом. Зарплаты растут до критических значений — более миллиона рублей в месяц. Компании начинают переманивать друг у друга оставшихся специалистов. Война за таланты достигает критической точки. Но талантов действительно нет — их просто не успели вырастить.

2032 – 2036

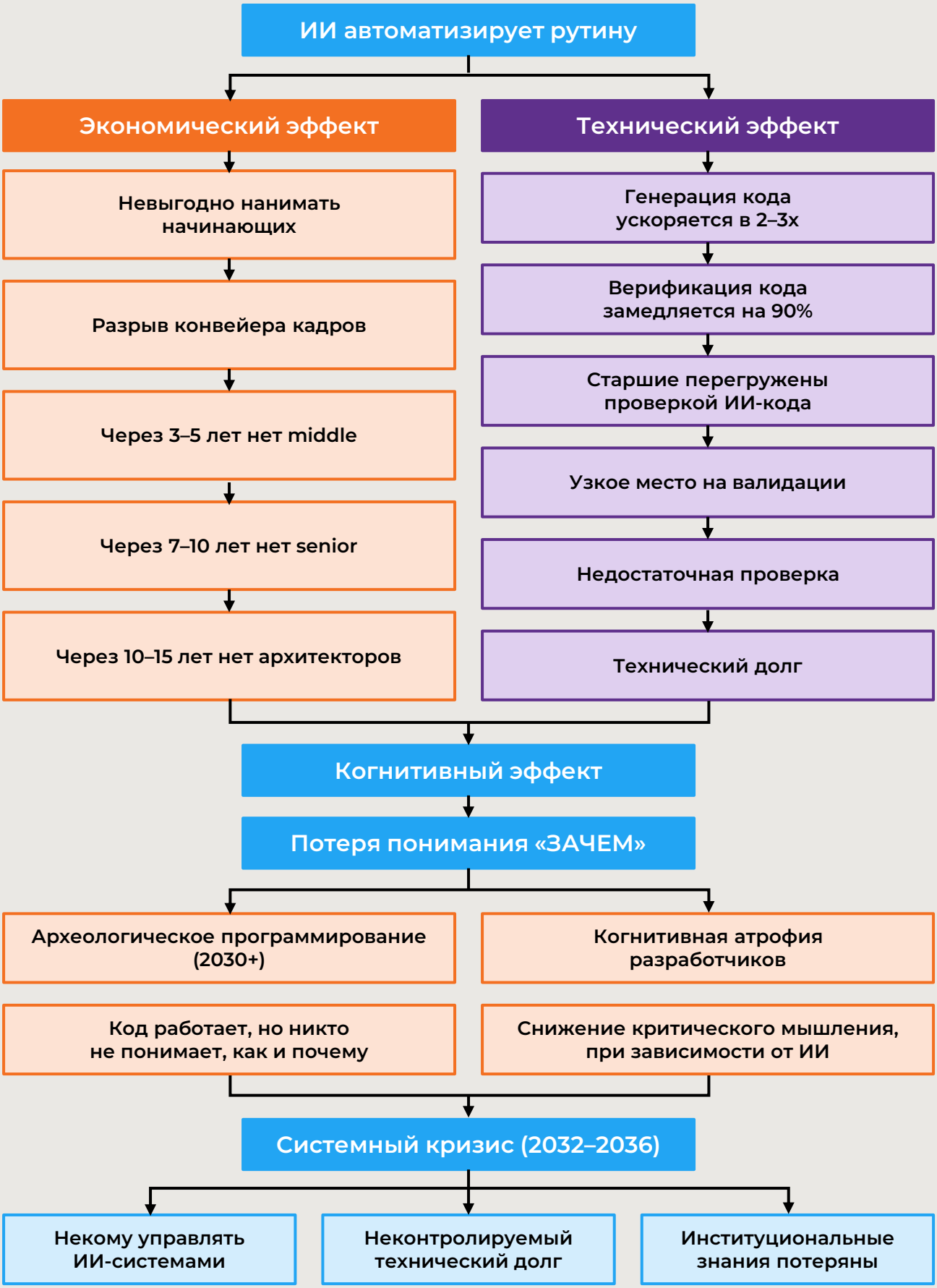
Катастрофа архитектурных компетенций

Поколение X выходит на пенсию. Архитекторы, определявшие технические стандарты последние 20 лет, уходят из индустрии. Заменить их некем. Старшие разработчики 2032 года — это средние разработчики 2027 года, которых не было, потому что их не наняли начинающими в 2024 году.

ИТ-системы, построенные на ИИ, остаются без людей, способных ими управлять. Критические решения принимаются специалистами без достаточного опыта. Институциональные знания — почему система спроектирована именно так, какие компромиссы были приняты, где находятся архитектурные риски — исчезают безвозвратно.

Системная природа кризиса

Это не линейная проблема с единственной причиной — это система взаимоусиливающих эффектов. Диаграмма показывает три параллельных процесса деградации:



Экономический эффект (левая ветвь): автоматизация рутины делает невыгодным найм начинающих, что разрывает конвейер роста кадров на 10–15 лет вперёд.

Технический эффект (центральная ветвь): ускорение генерации кода создаёт узкое место на верификации, перегружает старших специалистов, приводит к недостаточной проверке и росту технического долга.

Когнитивный эффект (правая ветвь): потеря понимания «зачем» система спроектирована именно так делает невозможным осмысленный рефакторинг и приводит к археологическому программированию. Исследования показывают измеримое снижение критического мышления при зависимости от ИИ [12]. Когда разработчик всегда может спросить у ИИ «как это сделать», он перестаёт спрашивать себя «а зачем это делать именно так». Появляется феномен археологического программирования. Код работает. Тесты проходят. Метрики производительности в норме. Но когда нужно внести изменение или исправить редкий баг, команда обнаруживает: никто не понимает, почему система спроектирована именно так. Архитектор, принимавший эти решения, вышел на пенсию три года назад. Документация описывает «что делает код», но не «зачем такой дизайн был выбран».

Эти три процесса усиливают друг друга. Нехватка кадров означает недостаточную верификацию. Технический долг требует архитекторов, которых нет. Когнитивная атрофия ухудшает качество проверки. Это не точечная проблема — это системный кризис, требующий системного ответа.

Визуализация коллапса конвейера

Чтобы понять масштаб катастрофы, посмотрим на эволюцию распределения разработчиков по уровням квалификации:

	2020 год До ИИ-революции, здоровый конвейер	**2026 год** После ИИ-революции, разрыв конвейера	**2031 год** коллапс конвейера, если ничего не менять
Архитекторы	**10%** Опыт 10-15+ лет	**10%** Стареют, уходят	**5%** Массовый уход на пенсию
Senior	**20%** Опыт 7-10 лет	**25%** Перегружены верификацией	**15%** Дефицит, $500K зарплаты
Middle	**30%** Опыт 3-7 лет	**35%** Растёт, но медленно	**25%** Нехватка, рост невозможен
Junior	**40%** Опыт 0-3 года	**30%** Найм упал на 25%	**55%** Попытка восстановить найм

Проблема очевидна: junior 2031 года не станут senior 2031 года. Им нужно 7–10 лет опыта. Senior 2031 года — это middle 2027 года, которых не было, потому что их не наняли junior в 2024 году.

Влияние ИИ на ключевые бизнес-метрики

Чтобы понять реальное влияние ИИ на производственные процессы создания программного обеспечения, нужно смотреть не только на скорость генерации кода. Ключевые метрики подтверждают обозначенный выше парадокс — ИИ ускоряет генерацию, но создаёт новые узкие места на этапе верификации и долгосрочного сопровождения кода.

Бизнес-метрики

Скорость вывода продукта на рынок

От идеи до релиза — ключевая метрика конкурентоспособности. Результаты впечатляют: Альфа-Банк сообщает об удвоении продуктивности[9], Сбербанк в декабре 2025 года выпустил полноценный релиз продукта, где весь код был написан ИИ-агентами[6]. Для задач с чёткими требованиями и паттернами ИИ даёт кратное ускорение без потери качества.

Стоимость разработки

Краткосрочная экономика выглядит привлекательно: старший разработчик с ИИ-экосистемой выполняет работу 2–3 начинающих, экономия в России составляет в среднем 2–2.5 млн руб. в год на каждом не нанятом специалисте, с учётом налогов. Компании, которые правильно выбирают задачи для автоматизации, видят снижение стоимости разработки на 20–30%.

Но скрытые затраты растут быстрее: технический долг растёт в 4 раза по дублированию кода[8], доля переделок выросла с 5,5% до 7,9% — рост на 43%[8].

Долгосрочная перспектива ещё мрачнее. При критическом дефиците зарплаты архитекторов могут вырасти в 3–4 раза, превращая экономию 2 млн руб. в год сегодня в необходимость платить 15–25 млн руб. в год одному архитектору завтра.

Удовлетворённость пользователей

Парадокс верификации означает, что новые функции могут выходить не быстрее, а медленнее. Одновременно 45% ИИ-кода содержит уязвимости[13], доля переделок выросла на 43%[8] — функции чаще оказываются неработоспособными и требуют срочных исправлений.

Растущий технический долг означает, что исправление ошибки в одном месте может сломать функциональность в другом. Пользователи сталкиваются с регрессиями — ранее работавшая функция внезапно перестаёт работать после обновления.

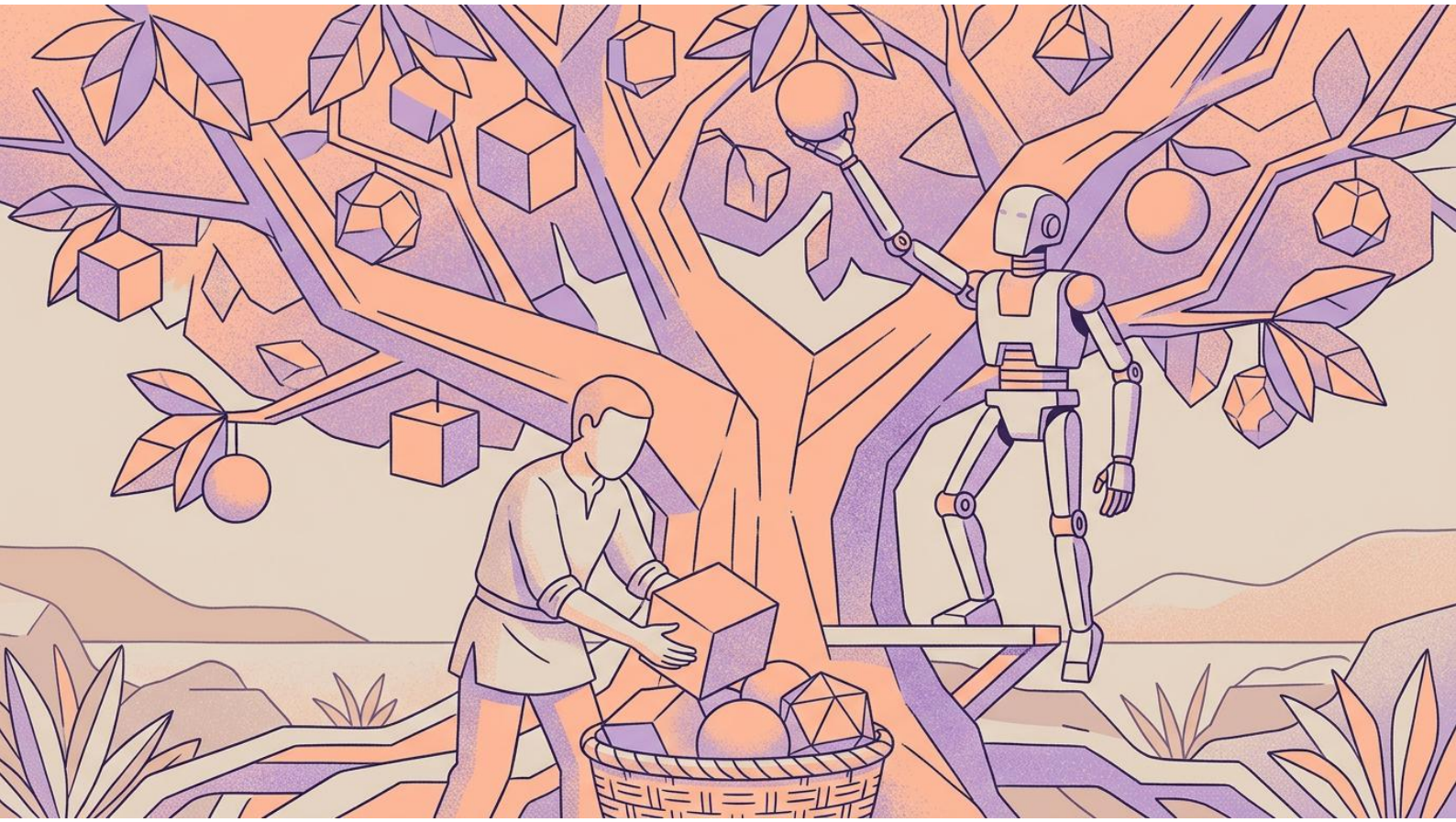
Текучесть кадров

Рынок труда поляризуется, и текучесть растёт на обоих концах спектра — но по разным причинам.

Старшие разработчики выгорают от рутинной верификации. Правильно внедрённый ИИ должен освобождать от рутины, но реальность иная — старшие перегружены проверкой ИИ-кода. Это не то, ради чего люди становились архитекторами. Выгорание — прямой путь к уходу из компании, что ещё больше обостряет дефицит высококлассных ИТ-специалистов.

Начинающим разработчикам сложнее войти в профессию. Те немногие, кого нанимают, сталкиваются с когнитивной атрофией — зависимость от ИИ снижает критическое мышление[12]. Они не получают фундаментального опыта написания кода, который формирует архитектурное мышление. Результат — высокая текучесть среди начинающих, которые уходят в другие отрасли или перепрыгивают между компаниями в поисках реального роста.

Средний слой разработчиков просто не формируется. Некому заменить выгорающих старших специалистов, некому брать на себя архитектурные решения, некому наставлять начинающих.



Метрики качества кода

Качество кода напрямую влияет на стоимость и скорость разработки. Дублирование кода увеличивает трудозатраты на исправление ошибок и внесение изменений. Отсутствие рефакторинга приводит к накоплению технического долга — отложенных проблем структуры кода, которые замедляют последующую разработку.

Дублирование и отказ от переиспользования кода

В 2025 году частота блоков кода с пятью и более дублирующимися строками выросла в 4 раза[8]. Одновременно копирование кода превысило его переиспользование — разработчики стали чаще копировать готовые решения, чем выделять повторяющуюся логику в общие модули.

Экономический эффект: когда одна и та же функциональность реализована в десяти местах, исправление ошибки требует десяти правок вместо одной. Исследования показывают, что 57,1% ошибок в дублированном коде возникают из-за несогласованных изменений копий — разработчик исправил баг в восьми местах из десяти, два пропустил, ошибка вернулась в продуктив. Добавление новой возможности требует модификации всех копий, что многократно увеличивает трудозатраты и риск регрессий.

Рост доли переделок: «выброшенная работа»

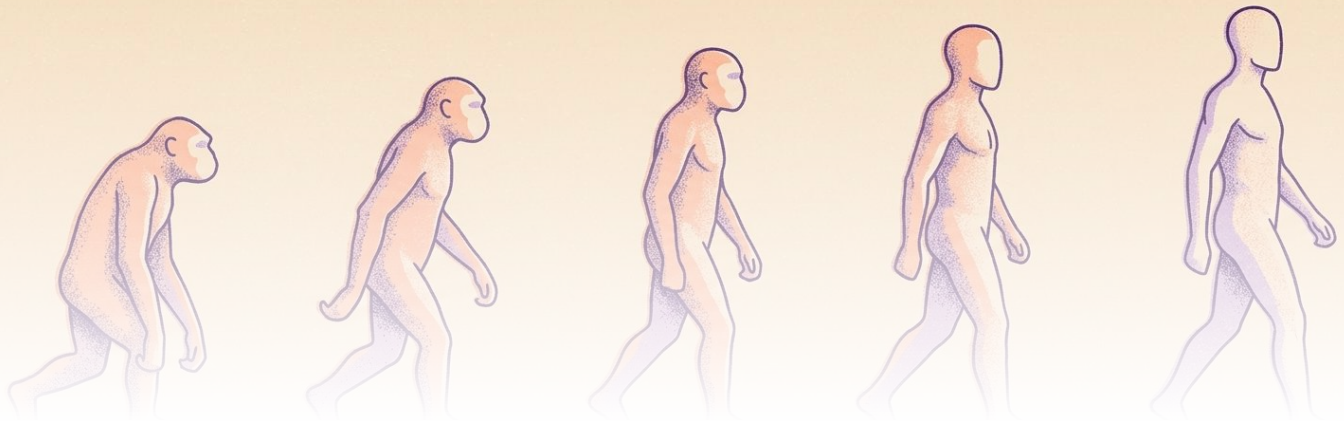
Доля переделок — процент кода, который переписывается в течение короткого времени после создания — выросла с 5,5% (2020) до 7,9% (2025), рост на 43%[8]. ИИ генерирует код быстро, но значительная часть оказывается недостаточно качественной и требует быстрой переработки. Это прямые потери: усилия на написание, проверку и интеграцию кода, который через две недели выбрасывается и переписывается заново.

Обвал рефакторинга и накопление технического долга

Доля рефакторинга в общих изменениях упала с 25% (2021) до менее 10% (2025)[8]. Рефакторинг — это улучшение структуры кода без изменения его поведения: выделение дубликатов в общие функции, упрощение сложной логики, приведение кода к стандартам. Когда разработчики перестают рефакторить, они генерируют новый код поверх старого, не улучшая структуру.

Каждое следующее изменение становится дороже предыдущего. Добавление простой функции в хорошо структурированный код занимает 2–3 дня. В коде без рефакторинга та же функция может занять 2–3 недели — нужно разобраться в запутанной структуре, найти все места дублирования, убедиться, что изменение не сломает что-то неожиданное.

GitClear ввела метрику «совокупный дефицит рефакторинга» (CRD) — насколько часто команды откладывают глубокую очистку кода в пользу поверхностных правок. Репозитории с активным использованием ИИ показали CRD на 34% выше, чем традиционные кодовые базы[8]. Команды, использующие ИИ, накапливают технический долг на треть быстрее. Технический долг начисляет «проценты» — чем дольше откладываешь, тем дороже она обходится.



Возможные пути решения проблемы

Мнения лидеров индустрии разделились радикально. Marc Benioff (CEO Salesforce) в январе 2025 заявил: «В прошлом году Salesforce не нанял ни одного нового инженера». Matt Garman (CEO Amazon Web Services) называет сокращение найма начинающих «одной из самых глупых идей, которые я слышал»[16].

Промежуточный подход: начинающих нанимают, но их роль трансформирована. Они учатся управлять ИИ-агентами, верифицировать ИИ-код, работать с автоматизированными инструментами. Это требует более высокого уровня входных компетенций, чем традиционное «написать функцию валидации email».

Проблема очевидна. Но есть ли решение, кроме массового найма начинающих разработчиков? Ответ положительный — команды со строгими процессами верификации, автоматическими инструментами обнаружения дубликатов, обязательным рефакторингом, архитектурными ревью и переосмыслением традиционных ролей показывают значительно лучшие результаты.

Системный аналитик 2026: новая реальность

Роль системного аналитика радикально меняется под влиянием ИИ. То, что раньше занимало дни работы — сбор требований, описание use cases, создание диаграмм, написание технических заданий — теперь автоматизируется или выполняется в разы быстрее.

ИИ автоматизирует:

- Создание документации по требованиям
- Генерацию SQL-запросов для анализа данных
- Построение визуализаций и дашбордов
- Создание диаграмм процессов и архитектуры[17]

Что остаётся аналитику? Именно то, что не могут делать ИИ-инструменты: понимание бизнес-контекста, коммуникация со стейкхолдерами, синтез противоречивых требований, архитектурное мышление.

Освободившееся время аналитики тратят на то, что действительно важно: глубокое погружение в бизнес-контекст, выявление скрытых требований через интервью со стейкхолдерами, синтез противоречивых потребностей разных отделов. ИИ взял на себя рутину — аналитик получил возможность заниматься стратегическими задачами. Компании, которые инвестировали в переобучение аналитиков, видят рост качества требований и снижение доли переделок на 30–40%.

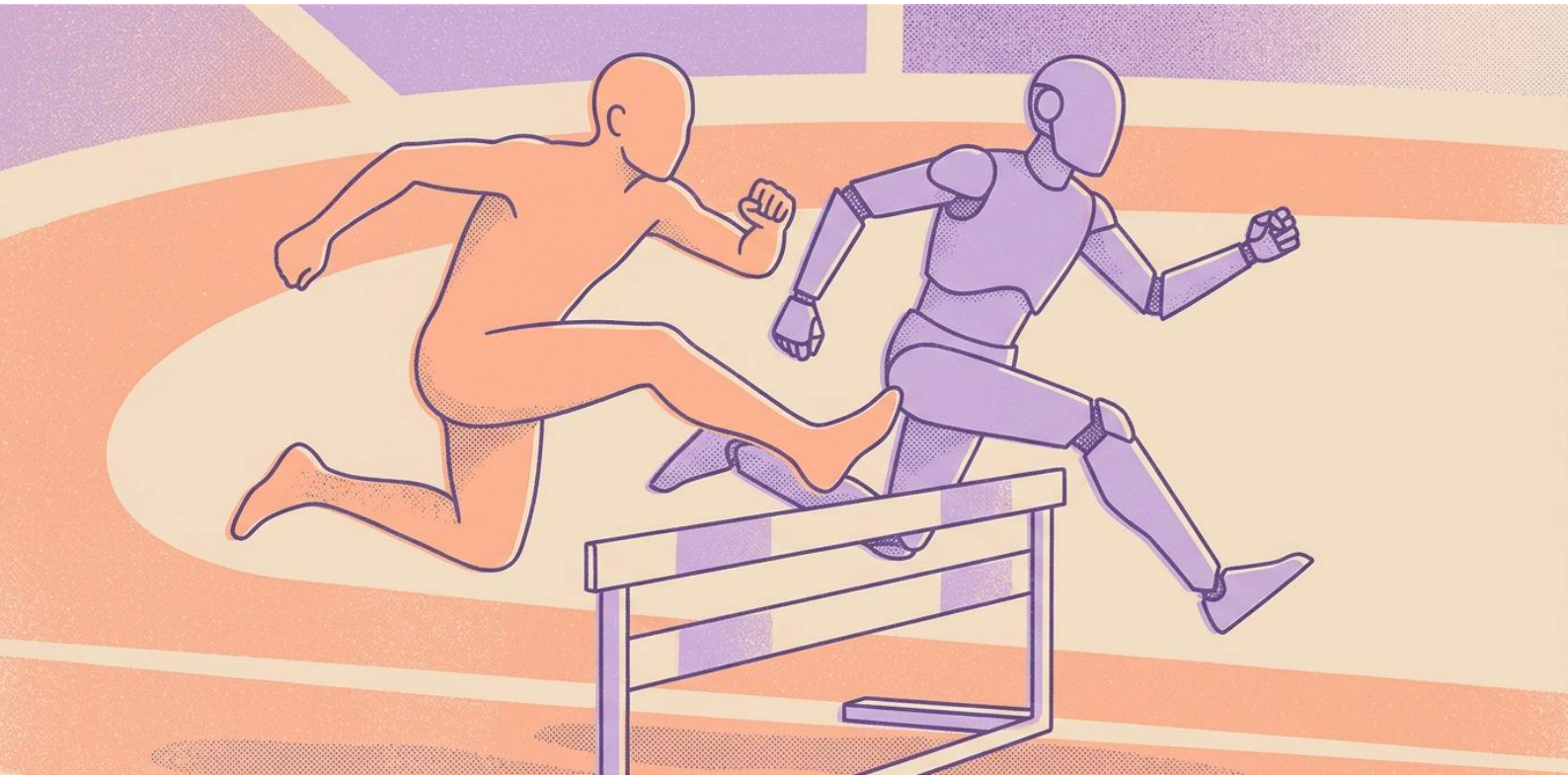
Карьерный путь: от бизнес-аналитика к архитектору

Традиционно существовало два основных карьерных пути в архитектуру:

Путь 1: Junior Developer → Middle → Senior → Solution Architect → Enterprise Architect. Длительность: 10–15 лет. Начало с написания кода, постепенный рост к архитектурным решениям.

Путь 2: Business Analyst → Business Architect → Enterprise Architect. Длительность: 7–12 лет. Начало с анализа бизнес-требований, переход к техническому проектированию.

Исследования показывают: около 60% бизнес-архитекторов начинали как бизнес-аналитики[18]. Этот путь работает, но требует дополнительных компетенций.

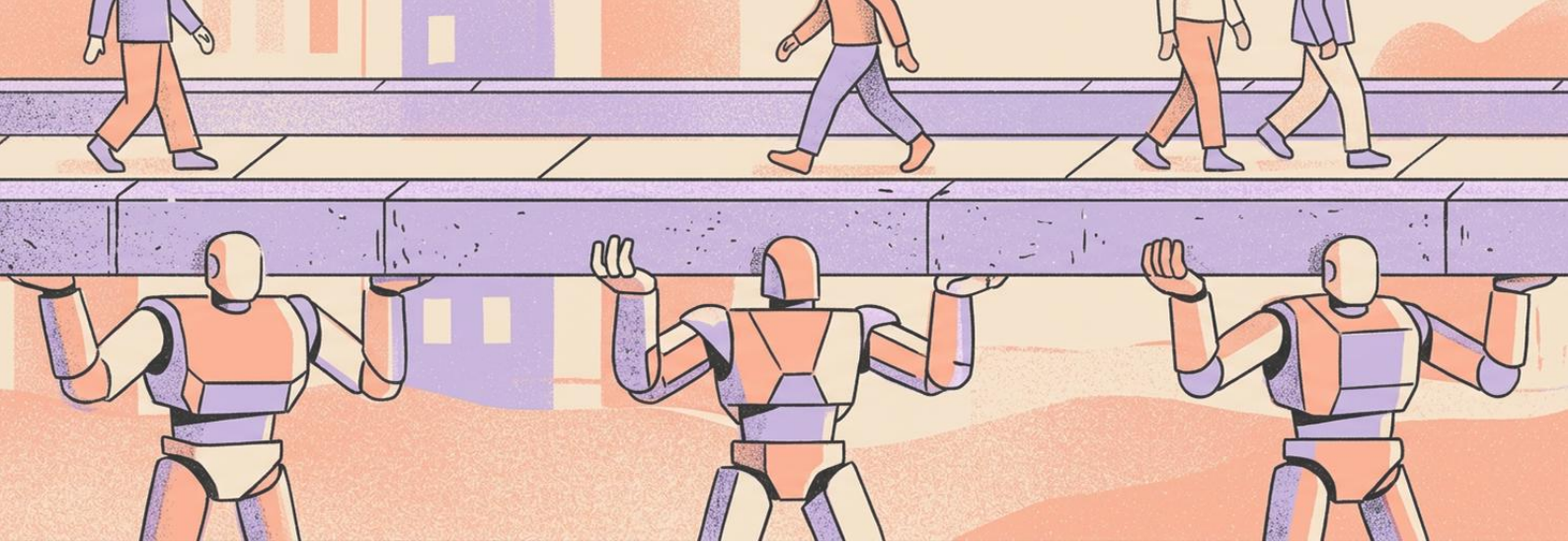


Проблема перехода: технический барьер

Критическая проблема пути «аналитик → архитектор»: технический барьер. Традиционно бизнес-аналитик фокусировался на понимании бизнес-требований, работе со стейкхолдерами, описании процессов. Он не имел глубокого понимания технологий, языков программирования, паттернов проектирования[19].

Системные аналитики находятся на стыке бизнеса и технологий — они уже работают с техническими требованиями, понимают взаимосвязи между системами, описывают интеграции.
Но и для них технический барьер остаётся значительным: одно дело описать, как должна работать система, и совсем другое — принять решение о том, какую архитектуру использовать, как обеспечить масштабируемость, где находятся узкие места по производительности [20].

Архитектор решения (Solution Architect) принимает технические решения, которые требуют понимания: производительности баз данных, масштабируемости систем, безопасности архитектур, интеграционных паттернов. Аналитик без технического бэкграунда не может принимать эти решения обоснованно[20].



ИИ как технический мост

Но ИИ меняет это уравнение. Сегодня системный аналитик может:

- Генерировать прототипы кода через ChatGPT/GitHub Copilot для проверки технической осуществимости
- Анализировать существующую кодовую базу через ИИ-помощников, понимая архитектуру без глубокого знания языка программирования
- Проверять производительность решений через автоматические инструменты профилирования
- Оценивать безопасность архитектуры через ИИ-ревьюеров кода
- Экспериментировать с технологиями без многолетнего опыта разработки

ИИ снижает технический барьер входа. Аналитику больше не нужно 5 лет писать код на Java, чтобы понять, как спроектировать микросервисную архитектуру. Он может сгенерировать прототип, проверить его работоспособность, оценить производительность — и всё это за дни, а не годы.

Этот подход уже показывает результаты. Аналитики с 5–7 годами опыта, прошедшие программы технической подготовки, успешно переходят на позиции технических аналитиков и младших архитекторов. Они приносят критически важное понимание бизнес-контекста, которого часто не хватает разработчикам, выросшим исключительно в коде. ИИ не просто снижает барьер входа — он делает возможным альтернативный путь роста, который раньше требовал десятилетия опыта.

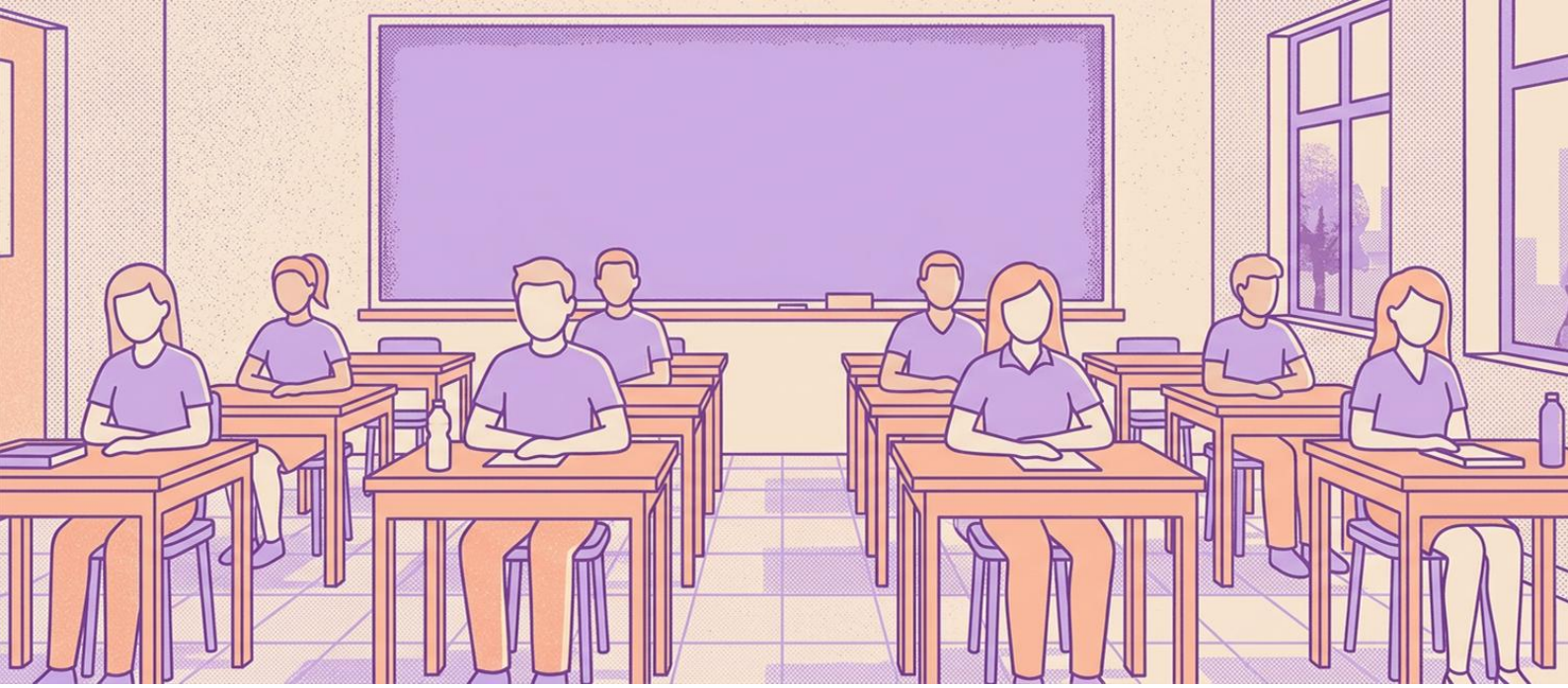
Преимущества пути Аналитик → Архитектор в эпоху ИИ

Бизнес-контекст: Аналитик понимает «зачем», а не только «как». Это критическое преимущество в мире, где ИИ генерирует «как» автоматически, но не понимает бизнес-смысла.

Коммуникационные навыки: Аналитик привык переводить бизнес-язык в технический и обратно. Архитектору эта компетенция критически важна.

Системное мышление: Аналитик видит процессы end-to-end, понимает взаимосвязи между системами, может синтезировать требования разных стейкхолдеров.

Меньше когнитивных искажений от ИИ: Junior developer, который с первого дня пишет код через ChatGPT, может не понять фундаментальные принципы. Аналитик, пришедший в технологии через бизнес-требования, сохраняет критическое мышление.



Что нужно изменить для реализации этого пути

1. Программы обучения для аналитиков

Добавить технические компетенции в программы подготовки системных аналитиков:

- Основы программирования (понимание принципов, не обязательно глубокое владение языками)
- Архитектурные паттерны и стили (микросервисы, event-driven, CQRS)
- Инфраструктура и DevOps (понимание CI/CD, контейнеризации, облачных платформ)
- Работа с ИИ-инструментами для разработки (промптинг, верификация ИИ-кода)

2. Карьерные треки в компаниях

Создать формализованный путь: System Analyst → Technical Analyst → Solution Architect → Enterprise Architect. Определить ключевые вехи и требуемые компетенции на каждом этапе.

3. Менторство и shadowing

Дать аналитикам возможность работать рядом с архитекторами, участвовать в технических решениях, постепенно брать на себя больше ответственности.

4. Проектная ротация

Переводить перспективных аналитиков на проекты с высокой технической сложностью, где они могут учиться архитектурному мышлению.

Риски и ограничения

Этот путь не панацея. Не все аналитики хотят или могут стать архитекторами. Технический барьер, хотя и снижен ИИ, не исчез полностью — базовое понимание технологий остаётся критически важным.

Но в мире, где традиционный путь «junior → senior → architect» разорван сокращением найма начинающих разработчиков, путь «аналитик → архитектор» может стать важной альтернативой для восполнения дефицита архитектурных компетенций.

Заключение: три сценария будущего

Мы стоим на развилке. Решения, принимаемые компаниями сегодня — нанимать или не нанимать начинающих, инвестировать или не инвестировать в альтернативные пути обучения, сохранять или разрушать конвейер кадров — определят, какой из трёх сценариев реализуется к 2030–2035 годам

1. ИИ-центричная разработка (2027-2030)

Индустрия принимает новую реальность и адаптируется осознанно. Компании инвестируют в гибридные команды: архитекторы работают с ИИ-агентами как с младшими членами команды. Продолжается найм начинающих специалистов, но их роль трансформирована: они учатся верифицировать ИИ-код с первого дня, параллельно изучая фундаментальные принципы программирования.

Одновременно развивается альтернативный путь через системных аналитиков. Компании создают формализованные карьерные треки: System Analyst → Technical Analyst → Solution Architect. ИИ-инструменты снижают технический барьер, позволяя аналитикам постепенно принимать архитектурные решения.

Ключевые характеристики:

- Сохраняется найм на всех уровнях, но с изменёнными требованиями
- Программы обучения перестраиваются: меньше синтаксиса, больше архитектурного мышления
- Зарплаты остаются на разумном уровне благодаря балансу спроса и предложения
- ИТ-индустрия здорова и устойчива

2. Архитектурная элита (2028-2033)

Дефицит старших специалистов становится критическим. Годовые зарплаты архитекторов взлетают до 20, 30, 50 млн руб. и выше. Формируется узкая элитарная прослойка, управляющая ИИ-системами. Барьер входа в профессию становится непреодолимым для большинства начинающих ИТ-специалистов.

Ключевые характеристики:

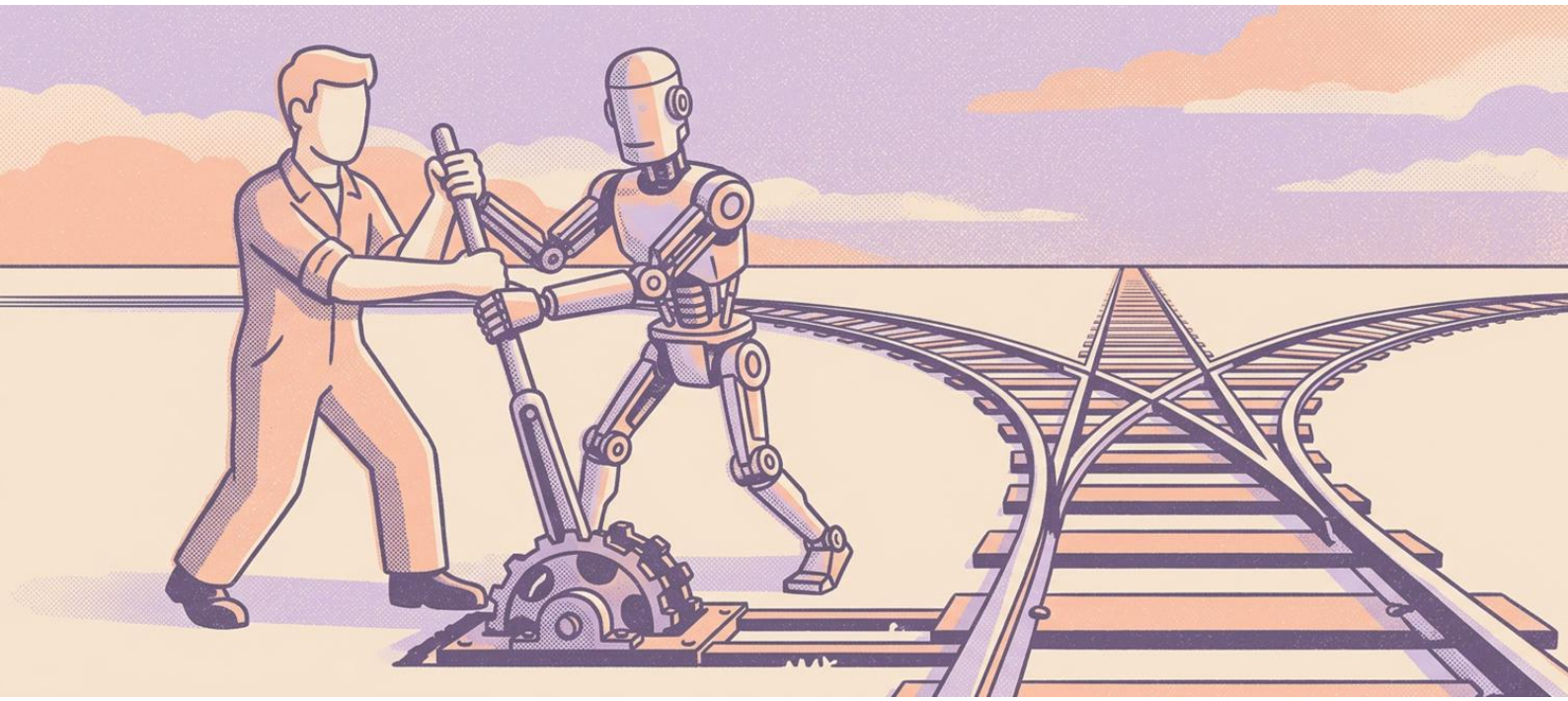
- Критический дефицит старших специалистов
- Неустойчивые зарплаты ИТ-специалистов создают давление на бизнес-модели
- Огромное неравенство — архитекторы становятся новой бизнес-аристократией
- Социальная мобильность практически исчезает
- Технологические компании превращаются в феодальные структуры

3. Обесценивание профессии (2030+)

ИИ настолько хорош в генерации и верификации кода, что большинство разработки превращается в рутину, автоматизируемую ИИ. Спрос на разработчиков падает радикально. Остаются только узкие специалисты для критических систем (финансы, здравоохранение, авиация) и регулируемых отраслей, где передача кода в облачные ИИ-сервисы запрещена.

Ключевые характеристики:

- Массовая безработица среди разработчиков среднего уровня
- Небольшие анклавы элитных специалистов в критических отраслях
- Разработка ПО больше не путь к среднему классу
- Молодые люди не идут в ИТ из-за неясных карьерных перспектив



Какой сценарий реализуется?

Это зависит от сегодняшних решений, принимаемых ключевыми игроками ИТ-индустрии.

Выбор за нами. Но время действовать — сейчас. В 2031 году будет поздно. Нельзя мгновенно вырастить архитекторов, когда они понадобятся. Это требует 10–15 лет непрерывного опыта. Конвейер кадров, разорванный в 2024 году, проявится катастрофой в 2032–2036.

Те, кто сокращают найм ради квартальных показателей, столкнутся с кризисом компетенций, когда будет уже поздно что-то исправлять. Сегодняшняя экономия — завтрашний кризис. Индустрия должна сделать выбор между краткосрочной оптимизацией и долгосрочной устойчивостью. Этот выбор определит, будет ли у нас через десять лет здоровая ИТ-индустрия или феодальная структура с небольшой элитарной прослойкой, владеющей критическими компетенциями.

Компании, сохраняющие баланс между краткосрочной оптимизацией и долгосрочными инвестициями в персонал, получат конкурентное преимущество через 5–10 лет. Те, кто инвестируют в новые модели обучения — через трансформацию роли системных аналитиков, через AI-native образовательные программы, через гибридные карьерные треки — будут иметь устойчивые команды, способные управлять ИИ-системами.

Источники и литература

1. Faros AI Research (2025). +90% время верификации. Analysis of 10,000+ developers across 1,255 teams. "PR review time increases 91%". Cited in: Latent Space, 'How to Kill the Code Review'. https://www.latent.space/p/reviews-dead
2. xGeeks (2026). AI Writes Code 3x Faster. Your Team Spends 2x Fixing It. Анализ 7 production проектов (200+ коммитов, 150,000 строк): "AI-assisted development is 2-3x faster than manual". https://blog.xgeeks.com/ai-writes-code-faster/
3. Brynjolfsson, E., Chandar, B., & Chen, R. (2025). Canaries in the Coal Mine? Six Facts about the Recent Employment Effects of Artificial Intelligence. Stanford Digital Economy Lab / ADP payroll data https://digitaleconomy.stanford.edu/publications/canaries-in-the-coal-mine/
4. SignalFire State of Talent Report 2025. Найм начинающих разработчиков сократился на 25% https://signalfire.com/blog/signalfire-state-of-talent-report-2025/
5. Stack Overflow Developer Survey (2025). AI vs Gen Z: 84% разработчиков используют ИИ ежедневно, 41% кода сгенерирован ИИ. https://stackoverflow.blog/2025/12/26/ai-vs-gen-z/
6. vc.ru (2025). Альфа-Банк повысил продуктивность разработчиков в два раза с помощью новой ИИ-платформы. https://vc.ru/ai/1862916
7. Federal Reserve Bank of New York (2025). 6,1% безработица CS graduates vs 4,3% общая. https://www.cio.com/article/4062024/demand-for-junior-developers-softens-as-ai-takes-over.html
8. GitClear (2025). AI Copilot Code Quality Research: 211 млн строк, 4x дубликаты, рефакторинг 25%→<10%. https://www.gitclear.com/ai_assistant_code_quality_2025_research
9. Habr Сбербанк (2025). GigaCode внутри Сбербанка: 14 000 разработчиков, 45%→69% AI-кода, декабрьский релиз без ручного кода. https://habr.com/ru/companies/sberbank/articles/1009838/
10. METR (2025). Measuring the Impact of Early-2025 AI on Experienced Open-Source Developer Productivity. Randomized controlled trial, 16 developers, 246 tasks. https://metr.org/blog/2025-07-10-early-2025-ai-experienced-os-dev-study/
11. CodeConductor (2026). Future of Junior Developers in AI: Экономия $148K/год на не нанятом специалисте. https://codeconductor.ai/blog/future-of-junior-developers-ai/
12. Harvard Gazette (2025). Is AI dulling our minds? MIT Media Lab study: excessive reliance on AI contributes to cognitive atrophy. https://news.harvard.edu/gazette/story/2025/11/is-ai-dulling-our-minds/
13. ClackyAI (2025). Code Review Checklist for AI-Generated Code: 45% содержит уязвимости. https://clacky.ai/blog/code-review-checklist-ai-generated-code
14. SignalFire Tech Hiring Report (2024). -25% найм начального уровня в топ-15 UK компаниях. https://signalfire.com/blog/tech-hiring-report-2024/
15. Indeed & Handshake (2025). Internship Data: стажировки -11%, tech-стажировки -30%, заявки +7%. https://www.indeed.com/lead/internship-trends-2025
16. CodeConductor (2026). CEO Quotes: Marc Benioff (Salesforce) "no new engineers 2025", Matt Garman (AWS) "одна из самых глупых идей". https://codeconductor.ai/blog/future-of-junior-developers-ai/
17. Yandex Practicum (2025). Системный аналитик: автоматизация документации, SQL, визуализации. https://practicum.yandex.ru/blog/kto-takoy-sistemnyi-analitik/
18. Process Street (2018). How to Move from Business Analyst to Business Architect: 60% архитекторов были аналитиками. https://www.process.st/business-analyst-business-architect/
19. ModernAnalyst (2024). From Business Analyst to Application Architect: требуется глубокое понимание ПО, языков программирования. https://www.modernanalyst.com/Resources/Articles/tabid/115/ID/6466/
20. Bridging the Gap (2011). Is Solution Architect a Good Career Path: аналитик не имеет технического бэкграунда для технических решений. https://www.bridging-the-gap.com/help-a-ba-is-solution-architect-a-good-career-path/

Скрытая цена ИИ-революции

Производственный процесс создания программного обеспечения в эпоху искусственного интеллекта

Мы готовы поделиться выводами экспертов и помочь адаптировать материалы публикации под задачи вашего бизнеса. Вы можете связаться с командой экспертов AXENIX по всем вопросам через электронную почту: research@axenix.pro.

Больше о практике стратегического консалтинга AXENIX — на нашем сайте.

Все исследования и обзоры AXENIX на нашем сайте.

Авторы исследования

Максим Ким
Директор практики стратегического консалтинга

Андрей Толстов
Старший менеджер практики стратегического консалтинга

Экспертная поддержка

Сергей Шаламов
Старший менеджер практики стратегического консалтинга

О компании AXENIX

Компания AXENIX (экс-Аксенчер) более 30 лет оказывает на российском рынке услуги в сфере ИТ-консалтинга и цифровой трансформации бизнеса. В числе ключевых направлений — разработка ПО, внедрение платформенных решений, информационная безопасность, искусственный интеллект и ИТ-аутсорсинг.

Компания реализует проекты для крупнейших предприятий страны, включая финансовый сектор, промышленность, металлургию и ритейл. Накопленная за годы работы экспертиза легла в основу собственных методологий и продуктов AXENIX.

Одним из ключевых преимуществ компании является способность вести проекты полного цикла — от разработки стратегии до внедрения и сопровождения ИТ-решений.

В настоящее время в компании работают более 2,8 тыс. сотрудников. Офисы и центры разработки расположены в Москве, Санкт-Петербурге, Твери, Ростове-на-Дону и Краснодаре.

axenix.pro